AI Magazine Volume 7 Number 2 (1986) (© AAAI)

PART ONE

The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures

H. Penny Nii

Knowledge Systems Laboratory, Computer Science Department, Stanford University,
701 Welch Road, Building C, Palo Alto, California 94304

Abstract

The first blackboard system was the HEARSAY-II speech understanding system (Erman *et al.*, 1980) that evolved between 1971 and 1976. Subsequently, many systems have been built that have similar system organization and run-time behavior. The objectives of this article are (1) to define what is meant by "blackboard systems" and (2) to show the richness and diversity of blackboard system designs. The article begins with a discussion of the underlying concept behind all blackboard systems, the blackboard model of problem solving. In order to bridge the gap between a model and working systems, the blackboard framework, an extension of the basic blackboard model is introduced, including a detailed description of the model's components and their behavior. A model does not come into existence on its own, and is usually an abstraction of many examples. In Section 2 the history of ideas is traced, and the designs of some application systems that helped shape the blackboard model are detailed. Part 2 of this article which will appear in the next issue of *AI Magazine,* describes and contrasts some blackboard systems and discusses the characteristics of application problems suitable for the blackboard method of problem solving.

Blackboard Model of Problem Solving

Historically, the blackboard model arose from abstracting features of the HEARSAY-II speech-understanding system developed between 1971 and 1976.[1] HEARSAY-II understood a spoken speech query about computer science abstracts stored in a data base. It "understood" in the sense that it was able to respond to spoken commands and queries about the database. From an informal summary description of the HEARSAY-II program, the HASP system was designed and implemented between 1973 and 1975. The domain[2] of HASP was ocean surveillance, and its task[3] was the interpretation of continuous passive sonar data. HASP, as the second example of a blackboard system, not only added credibility to the claim that a blackboard approach to problem solving was general, but it also demonstrated that it could be abstracted into a robust model of problem solving. Subsequently, many application programs have been implemented whose solutions were formulated using the blackboard model. Because of the different characteristics of the application problems and because the interpretation of the blackboard model varied, the design of these programs differed considerably. However, the blackboard model of problem solving has not undergone any substantial changes in the last ten years.

A *problem-solving model* is a scheme for organizing reasoning steps and domain knowledge to construct a solution to a problem. For example, in a *backward-reasoning model,* problem solving begins by reasoning backward from a goal to be achieved toward an initial state (data). More specifically, in a *rule-based backward reasoning model* knowledge is organized as "if-then" rules, and modus ponens inference steps are applied to the rules from a goal rule back to an initial-state rule (a rule that looks at the input data). An excellent example of this approach to problem solving is the MYCIN program (Shortliffe, 1976). In a

[1] This document is a part of a retrospective monograph on the AGE project currently in preparation.

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

[2] Domain refers to a particular area of discourse, for example, chemistry.

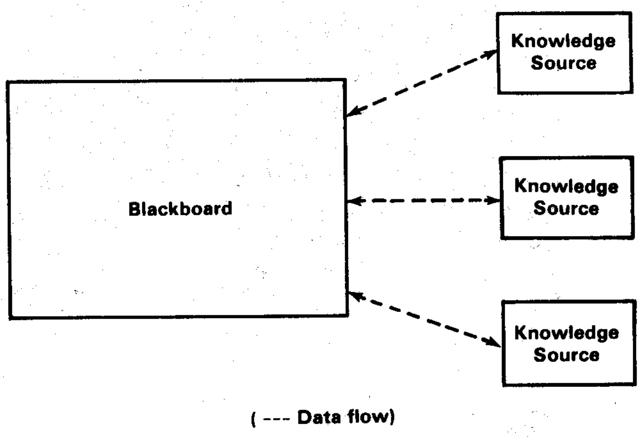
[3] Task refers to a goal-oriented activity within the domain, for example, to analyze the molecular composition of a compound.

forward-reasoning model, however, the inference steps are applied from an initial state toward a goal. The OPS system exemplifies such a model (Forgy & McDermott, 1977). In an *opportunistic reasoning model,* pieces of knowledge are applied either backward or forward at the most "opportune" time. Put another way, the central issue of problem solving deals with the question, "What pieces of knowledge should be applied when and how?" A problem-solving model provides a conceptual framework for organizing knowledge and a strategy for applying that knowledge.

The blackboard model of problem solving is a highly structured special case of opportunistic problem solving. In addition to opportunistic reasoning as a knowledge-application strategy, the blackboard model prescribes the organization of the domain knowledge and all the input and intermediate and partial solutions needed to solve the problem. We refer to all possible partial and full solutions to a problem as its *solution space.*

In the blackboard model, the solution space is organized into one or more application-dependent hierarchies.[4] Information at each level in the hierarchy represents partial solutions and is associated with a unique vocabulary that describes the information. The domain knowledge is partitioned into independent modules of knowledge that transform information on one level, possibly using information at other levels, of the hierarchy into information on the same or other levels. The knowledge modules perform the transformation using algorithmic procedures or heuristic rules that generate actual or hypothetical transformations. Opportunistic reasoning is applied within this overall organization of the solution space and task-specific knowledge; that is, which module of knowledge to apply is determined dynamically, one step at a time, resulting in the incremental generation of partial solutions. The choice of a knowledge module is based on the solution state (particularly, the latest additions and modifications to the data structure containing pieces of the solution) and on the existence of knowledge modules capable of improving the current state of the solution. At each step of knowledge application, either forward- or backward-reasoning methods can be applied.[5]

The blackboard model is a relatively complex problem-solving model prescribing the organization of knowledge and data and the problem-solving behavior within the overall organization. This section contains a description of the basic blackboard model. Variations and extensions are discussed in subsequent sections.



There is a global database called the blackboard, and there are logically independent sources of knowledge called the knowledge sources. The knowledge sources respond to changes on the blackboard. Note that there is no control flow; the knowledge sources are self-activating.

The Blackboard Model

Figure 1

The Blackboard Model

The blackboard model is usually described as consisting of three major components (see Figure 1):

The knowledge sources. The knowledge needed to solve the problem is partitioned into *knowledge sources,* which are kept separate and independent.

The blackboard data structure. The problem-solving state data are kept in a global database, the *blackboard.* Knowledge sources produce changes to the blackboard that lead incrementally to a solution to the problem. Communication and interaction among the knowledge sources take place solely through the blackboard.

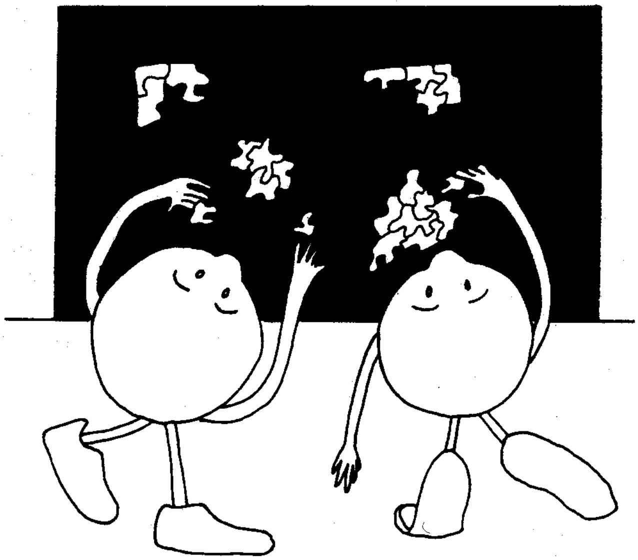
Control. The knowledge sources respond opportunistically to changes in the blackboard.[6]

The difficulty with this description of the blackboard model is that it only outlines the organizational principles. For those who want to build a blackboard system, the model does not specify how it is to be realized as a computational entity; that is, the blackboard model is a conceptual entity, not a computational specification. Given a problem to be solved, the blackboard model provides enough guidelines for sketching a solution, but a sketch is a long way from a working system. To design and build

[4] The hierarchy can be an abstraction hierarchy, a part-of hierarchy, or any other type of hierarchy appropriate for solving the problem.

[5] There are various other ways of categorizing reasoning methods, for example, event driven, goal driven, model driven, expectation driven, and so forth. Without getting into the subtle differences between these methods, it is safe to say that any one of these methods can be applied at each step in the reasoning process.

[6] There is no control component specified in the blackboard model. The model merely specifies a general problem-solving behavior. The actual locus of control can be in the knowledge sources, on the blackboard, in a separate module, or in some combination of the three. (The need for a control component in blackboard systems is discussed later.)



Solving Jigsaw Puzzles

Figure 2

a system, a detailed model is needed. Before moving on to adding details to the blackboard model, we explore the implied behavior of this abstract model.

Let us consider a hypothetical problem of a group of people trying to put together a jigsaw puzzle. Imagine a room with a large blackboard and around it a group of people each holding over-size jigsaw pieces. We start with volunteers who put their most "promising" pieces on the blackboard (assume it's sticky). Each member of the group looks at his pieces and sees if any of them fit into the pieces already on the blackboard. Those with the appropriate pieces go up to the blackboard and update the evolving solution. The new updates cause other pieces to fall into place, and other people go to the blackboard to add their pieces. It does not matter whether one person holds more pieces than another. The whole puzzle can be solved in complete silence; that is, there need be no direct communication among the group. Each person is self-activating, knowing when his pieces will contribute to the solution. No *a priori* established order exists for people to go up to the blackboard. The apparent cooperative behavior is mediated by the state of the solution on the blackboard. If one watches the task being performed, the solution is built incrementally (one piece at a time) and opportunistically (as an opportunity for adding a piece arises), as opposed to starting, say, systematically from the left top corner and trying each piece.

This analogy illustrates quite well the blackboard problem-solving behavior implied in the model and is fine for a starter. Now, let's change the layout of the room in such a way that there is only one center aisle wide enough for one person to get through to the blackboard. Now, no more than one person can go up to the blackboard at one time, and a monitor is needed, someone who can see the group and can choose the order in which a person is to go up to the blackboard. The monitor can ask all people who have pieces to add to raise their hands. The monitor can then choose one person from those with their hands raised. To select one person, some criteria for making the choice is needed, for example, a person who raises a hand first, a person with a piece that bridges two solution islands (that is, two clusters of completed pieces), and so forth. The monitor needs a strategy or a set of strategies for solving the puzzle. The monitor can choose a strategy before the puzzle solving begins or can develop strategies as the solution begins to unfold. In any case, it should be noted that the monitor has broad executive power. The monitor could, for example, force the puzzle to be solved systematically from left to right; that is, the monitor has the power to violate one essential characteristic of the original blackboard model, that of opportunistic problem solving.

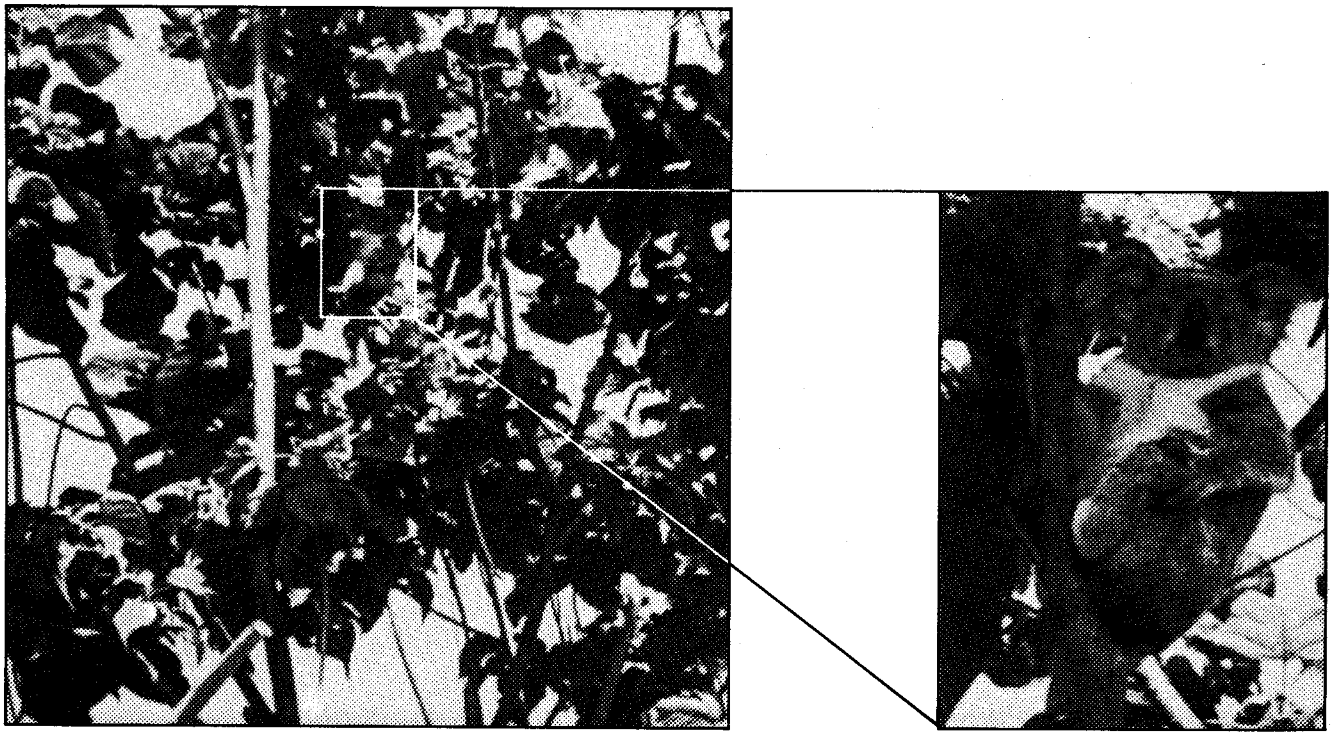
The last analogy, though slightly removed from the original model, is a useful one for computer programmers interested in building blackboard systems. Given the serial nature of most current computers, the conceptual distance between the model and a running blackboard system is a bit far, and the mapping from the model to a system is prone to misinterpretation. By adding the constraint that solution building physically occur one step at a time in some order determined by the monitor (when multiple steps are possible and desirable), the blackboard model is brought closer to the realities inherent in serial-computing environments.[7]

Although the elaborate analogy to jigsaw puzzle solving gives us additional clues to the nature of the behavior of blackboard systems, it is not a very good example for illustrating the organization of the blackboard or for the partitioning of appropriate knowledge into knowledge sources. To illustrate these aspects of the model, we need another example. This time let us consider another hypothetical problem, that of finding koalas in a eucalyptus forest (see Figure 3).

Imagine yourself in Australia. One of the musts if you are a tourist is to go and look for koalas in their natural habitat. So, you go to a koala preserve and start looking for them among the branches of the eucalyptus trees. You find none. You know that they are rather small, grayish creatures which look like bears.[8] The forest is dense,

[7] The serialization of the blackboard model is useful only because we tend to work on uniprocessor computers. We are currently conducting research on concurrent problem-solving methods. A starting point for the work is the pure blackboard model. One can see, at least conceptually, much parallelism inherent in the model. The problem is how to convert the model into an operational system that can take advantage of many (100s to 1000s) processor-memory pairs.

[8] More details at this descriptive level would be considered factual



Courtesy, San Diego Zoo.

Courtesy, San Diego Zoo.

Finding Koalas

Figure 3

however, and the combination of rustling leaves and the sunlight reflecting on the leaves adds to the difficulty of finding these creatures, whose coloring is similar to their environment.[9] You finally give up and ask a ranger how you can find them. He gives you the following story about koalas: "Koalas usually live in groups and seasonally migrate to different parts of the forest, but they should be around the northwest area of the preserve now. They usually sit on the crook of branches and move up and down the tree during the day to get just the right amount of sun.[10] If you are not sure whether you have spotted one or not, watch it for a while; it will move around, though slowly."[11] Armed with the new knowledge, you go back to the forest with a visual image of exactly where and what to look for. You focus your eyes at about 30 feet with no luck, but you try again, and this time focus your eyes at 50 feet, and suddenly you do find one. Not only one, but a whole colony of them.[12]

Let's consider one way of formulating this problem along the lines of the blackboard model. Many kinds of knowledge can be brought to bear on the problem: the color and shape of koalas, the general color and texture of the environment (the noise characteristics), the behavior of the koalas, effects of season and time of the day, and so on. Some of the knowledge can be found in books, such as *Handbook of Koala Sizes and Color* or *Geography of the Forest.* Some knowledge is informal—the most likely places to find koalas at any given time or the koalas' favorite resting places. How can these diverse sources of knowledge be used effectively? First, we need to decide what constitutes a solution to the problem. Then, we can consider what kinds of information are in the data, what can be inferred from them, and what knowledge might be brought to bear to achieve the goal of finding the koalas.

knowledge and can be used as a part of a prototypical model of koalas.

[9] The signal-to-noise ratio is low.

[10] This is knowledge about the prototypical behavior pattern of koalas. The ranger suggests a highly model-driven approach to finding them.

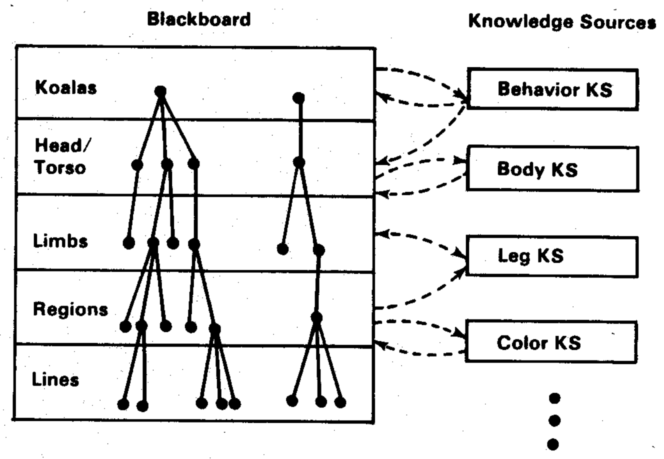
[11] This is a method of detection as well as confirmation.

[12] This koala problem has a long history. It was invented by Ed Feigenbaum (after his trip to Australia) and myself in 1974, during the time when we were not allowed to write about the HASP project. The primary objective of this example was to illustrate the power of model-directed reasoning in interpreting noisy data. A paper was written about it but has been collecting dust, a victim of our distaste for writing about hypothetical problems. I resurrect it here because it's hard to come up with a good example that does not require specialized domain knowledge.

Think of the solution to this problem as a set of markings on a series of snapshots of the forest. The markings might say, "This is certainly a koala because it has a head, body, and limbs and because it has changed its position since the last snapshot"; "This might be a koala, because it has a blob that looks like a head"; "These might be koalas because they are close to the one we know is a koala and the blobs could be heads, legs, or torsos." The important characteristics of the solution are that the solution consists of bits and pieces of information, and it is a reasoned solution with supporting evidence and supporting lines of reasoning.

Having decided that the solution would consist of partial and hypothetical identifications, as well as complete identifications constructed from partial ones, we need a solution-space organization that can hold descriptions of bits and pieces of the koalas. One such descriptive framework is a part-of hierarchy. For each koala, the highest level of description is the koala itself, which is described on the next level by head and body; the head is described on the next level by ears, nose, and eyes; the body is described by torso, legs, and arms; and so on. At each level, there are descriptors appropriate for that level: size, gender, and height on the koala level, for example. Each primitive body part is described on the lower levels in terms of geometric features, such as shapes and line segments. Each shape has color and texture associated with it as well as its geometric descriptions (see Figure 4). In order to identify a part of the snapshot as a koala, we need to mark the picture with line segments and regions. The regions and pieces of lines must eventually be combined, or synthesized, in such a way that the description of the constructed object can be construed as some of the parts of a koala or a koala itself. For example, a small, black circular blob could be an eye, but it must be surrounded by a bigger, lighter blob that might be a head. The more pieces of information one can find that fit the koala description, the more confident we can be. In addition to the body parts that support the existence of a koala, if the hypothesized koala is at about 30 to 50 feet above ground, we would be more confident than if we found the same object at 5 feet.

The knowledge needed to fill in the koala descriptions falls into place with the decision to organize the solution space as a part-of abstraction hierarchy. We would need a color specialist, a shape specialist, a body-part specialist, a habitat specialist, and so forth. No one source of knowledge can solve the problem; the solution to the problem depends on the combined contributions of many specialists. The knowledge held by these specialists is logically independent. Thus, a color specialist can determine the color of a region without knowing how the shape specialist determined the shape of the region. However, the solution of the problem is dependent on both of them. The torso specialist does not have to know whether the arm specialist checked if an arm had paws or not (the torso specialist probably doesn't even know about paws), but each specialist must rely on the other specialists to supply the information each one needs. Cooperation is achieved by assuming that whatever information is needed is supplied by someone else.



The koalas in the scene are described as a part-of hierarchy. Specialist knowledge modules contribute information about what they "see" to help in the search for koalas.

Koalas: Blackboard Structure and Knowledge Sources

Figure 4

The jigsaw puzzle and the koala problems illustrate the organization of information on the blackboard database, the partitioning of domain knowledge into specialized sources of knowledge, and some of the characteristic problem-solving behavior associated with the blackboard model.[13] Neither of these, however, answers the questions of how the knowledge is to be represented, or of what the mechanisms are for determining and activating appropriate knowledge. As mentioned earlier, problem-solving models are conceptual frameworks for formulating solutions to problems. The models do not address the details of designing and building operational systems. How a piece of knowledge is represented, as rules, objects, or procedures, is an engineering decision. It involves such pragmatic considerations as "naturalness," availability of a knowledge representation language, and the skill of the implementers, to name but a few.[14] What control mechanisms are needed depends on the complexity and the na-

[13] As in the jigsaw problem, the problem-solving behavior in the koala problem would be opportunistic. As new pieces of evidence are found and new hypotheses generated, appropriate knowledge sources analyze them and create new hypotheses.

[14] The blackboard model does not preclude the use of human knowledge sources. Interesting interactive and symbiotic expert systems can be built by integrating human expertise during run time.

ture of the application task. We can, however, attempt to narrow the gap between the model and operational systems. Now, the blackboard model is extended by adding more details to the three primary components in terms of their structures, functions, and behaviors.

The Blackboard Framework

Applications are implemented with different combinations of knowledge representations, reasoning schemes, and control mechanisms. The variability in the design of blackboard systems is due to many factors, the most influential one being the nature of the application problem itself. It can be seen, however, that blackboard architectures which underlay application programs have many similar features and constructs. (Some of the better known applications are discussed in Part 2.) The blackboard framework is created by abstracting these constructs.[15] The blackboard framework, therefore, contains descriptions of the blackboard system components that are grounded in actual computational constructs. The purpose of the framework is to provide design guidelines appropriate for blackboard systems in a serial-computing environment.[16] Figure 5 shows some modifications to Figure 1 to reflect the addition of system-oriented details.

The Knowledge Sources. The domain knowledge needed to solve a problem is partitioned into knowledge sources that are kept separate and independent.

The objective of each knowledge source is to contribute information that will lead to a solution to the problem. A knowledge source takes a set of current information on the blackboard and updates it as encoded in its specialized knowledge.

The knowledge sources are represented as procedures, sets of rules, or logic assertions. To date most of the knowledge sources have been represented as either procedures or as sets of rules. However, systems that deal with signal processing either make liberal use of procedures in their rules or use both rule sets and procedurally encoded knowledge sources.

The knowledge sources modify only the blackboard or control data structures (that also might be on the blackboard), and only the knowledge sources modify the blackboard. All modifications to the solution state are explicit and visible.

Each knowledge source is responsible for knowing the conditions under which it can contribute to a solution. Each knowledge source has *preconditions* that indicate the condition on the blackboard which must exist before the body of the knowledge source is *activated.*[17]

The Blackboard Data Structure. The problem-solving state data are kept in a global database, the *blackboard.* Knowledge sources produce changes to the blackboard that lead incrementally to a solution, or a set of acceptable solutions, to the problem. Interaction among the knowledge sources takes place solely through changes on the blackboard.

The purpose of the blackboard is to hold computational and solution-state data needed by and produced by the knowledge sources. The knowledge sources use the blackboard data to interact with each other indirectly.

The blackboard consists of objects from the solution space. These objects can be input data, partial solutions, alternatives, and final solutions (and, possibly, control data).

The objects on the blackboard are hierarchically organized into levels of analysis. Information associated with objects (that is, their properties) on one level serves as input to a set of knowledge sources, which, in turn, place new information on the same or other levels.

The objects and their properties define the vocabulary of the solution space. The properties are represented as attribute-value pairs. Each level uses a distinct subset of the vocabulary.[18]

The relationships between the objects are denoted by named links.[19] The relationship can be between objects on different levels, such as "part-of" or "in-support-of," or between objects on the same level, such as "next-to" or "follows."

The blackboard can have multiple blackboard panels. That is, a solution space can be partitioned into multiple hierarchies.[20]

The data on the blackboard are hierarchically organized. The knowledge sources are logically independent, self-selecting modules. Only the knowledge sources are allowed to make changes to the blackboard. Based on the latest changes to the information on the blackboard,

[15] There is an implicit assumption that systems can be described at various levels of abstraction. Thus, the description of the framework is more detailed than the model and less detailed than a specification (a description from which a system can be built). Here, they are called the model, framework, and specification levels.

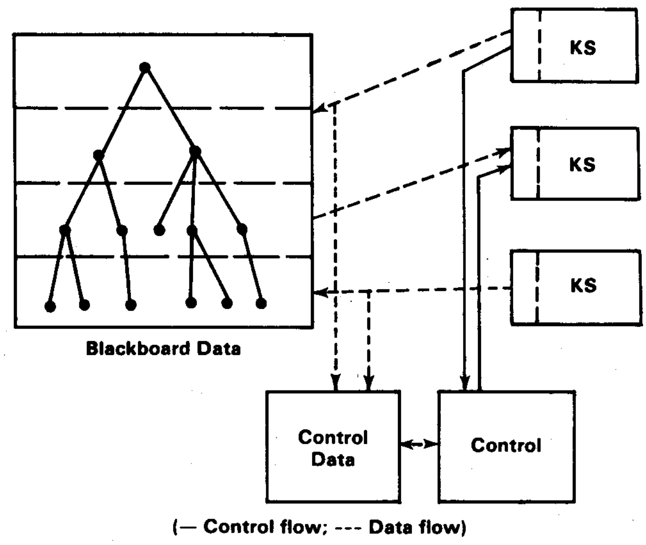
[16] One can view the blackboard framework as a prescriptive model; that is, it prescribes what must be in a blackboard system. However, it must be kept in mind that application problems often demand extensions to the framework, as can be seen in the examples in Part 2.

[17] One can view a knowledge source as a large rule. The major difference between a rule and a knowledge source is the grain size of the knowledge each holds. The condition part of this large rule is called the knowledge source precondition, and the action part is called the knowledge source body.

[18] Many times, the names of the attributes on different levels are the same, for example, "type." Often these are shorthand notations for "type-of-x-object" or "type-of-y-object." Sometimes they are duplications of the same attribute used for convenience sake.

[19] A relationship is a special kind of property.

[20] This feature was first used in the CRYSALIS system. The rationale for introducing multiple panels is discussed in Part 2.



The Blackboard Framework

Figure 5

a control module selects and executes the next knowledge source.

Control. The knowledge sources *respond opportunistically* to changes in the blackboard.

There is a set of control modules that monitor the changes on the blackboard and decide what actions to take next.

Various kinds of information are made globally available to the control modules. The information can be on the blackboard or kept separately. The control information is used by the control modules to determine the focus of attention.

The focus of attention indicates the next thing to be processed. The focus of attention can be either the knowledge sources (that is, which knowledge sources to activate next) or the blackboard objects (that is, which solution islands to pursue next) or a combination of both (that is, which knowledge sources to apply to which objects).[21]

The solution is built one step at a time. Any type of reasoning step (data driven, goal driven, model driven, and so on) can be applied at each stage of solution formation. As a result, the sequence of knowledge source invocation is dynamic and opportunistic rather than fixed and preprogrammed.

Pieces of problem-solving activities occur in the following iterative sequence:

1. A knowledge source makes change(s) to blackboard object(s). As these changes are made, a record is kept in a global data structure that holds the control information.
2. Each knowledge source indicates the contribution it can make to the new solution state. (This can be defined *a priori* for an application or dynamically determined.)
3. Using the information from points 1 and 2, a control module selects a focus of attention.
4. Depending on the information contained in the focus of attention, an appropriate control module prepares it for execution as follows:
   a. If the focus of attention is a knowledge source, then a blackboard object (or sometimes, a set of blackboard objects) is chosen to serve as the context of its invocation (knowledge-scheduling approach).
   b. If the focus of attention is a blackboard object, then a knowledge source is chosen which will process that object (event-scheduling approach).
   c. If the focus of attention is a knowledge source and an object, then that knowledge source is ready for execution. The knowledge source is executed together with the context, thus described.

Criteria are provided to determine when to terminate the process. Usually, one of the knowledge sources indicates when the problem-solving process is terminated, either because an acceptable solution has been found or because the system cannot continue further for lack of knowledge or data.

Problem-Solving Behavior and Knowledge Application. The problem-solving behavior of a system is determined by the knowledge-application strategy encoded in the control modules. The choice of the most appropriate knowledge-application strategy is dependent on the characteristics of the application task and on the quality and quantity of domain knowledge relevant to the task.[22] Basically, the acts of choosing a particular blackboard region and choosing a particular knowledge source to operate on that region determine the problem-solving behavior. Generally, a knowledge source uses information on one level as its input and produces output information on another level. Thus, if the input level of a particular knowledge source is on the level lower (closer to data) than its output

[21] Any given system usually employs one of the three approaches, not all.

[22] It might be said that this is a hedge, that there should be a knowledge-application strategy or a set of strategies built into the framework to reflect different problem-solving behaviors. It is precisely this lack of doctrine, however, that makes the blackboard framework powerful and useful. If an application task calls for two forward-reasoning steps followed by three backward-reasoning steps at some particular point, the framework allows for this. This is not to say that a system with built-in strategies cannot be designed and built. If there is a knowledge-application strategy "generic" to a class of applications, then it might be worthwhile to build a skeletal system with that particular strategy.

level, then the application of this knowledge source is an application of bottom-up, forward reasoning.

Conversely, a commitment to a particular type of reasoning step is a commitment to a particular knowledge-application method. For example, if we are interested in applying a data-directed, forward-reasoning step, then we would select a knowledge source whose input level is lower than its output level. If we are interested in goal-directed reasoning, we would select a knowledge source that put information needed to satisfy a goal on a lower level. Using the constructs in the control component, one can make any type of reasoning step happen at each step of knowledge application.[23]

How a piece of knowledge is stated often presupposes how it is to be used. Given a piece of knowledge about a relationship between information on two levels, that knowledge can be expressed in top-down or bottom-up application forms. These can further be refined. The top-down form can be written as a goal, an expectation, or as an abstract model of the lower-level information. For example, a piece of knowledge can be expressed as a conjunction of information on a lower level needed to generate a hypothesis at a higher level (a goal), or it can be expressed as information on a lower level needed to confirm a hypothesis at a higher level (an expectation), and so on. The framework does not presuppose nor does it prescribe the knowledge-application, or reasoning, methods. It merely provides constructs within which any reasoning methods can be used. Many interesting problem-solving behaviors have been implemented using these constructs (some of them are discussed in Part 2).

Perspectives

The organizational underpinnings of blackboard systems have been the primary focus. The blackboard framework is a system-oriented interpretation of the blackboard model. It is a mechanistic formulation intended to serve as a foundation for system specifications. In problem-solving programs, we are usually interested in their performance and problem-solving behavior, not their organization. We have found, however, that some classes of complex problems become manageable when they are formulated along the lines of the blackboard model. Also, interesting problem-solving behavior can be programmed using the blackboard framework as a foundation. Even though the blackboard framework still falls short of being a computational specification, given an application task and the necessary knowledge, it provides enough information so that a suitable blackboard system can be designed, specified, and built. (Some examples of complex problems with interesting problem-solving behavior are discussed in Part 2. The examples show that new constructs can be added to the blackboard framework as the application problems demand, without violating the guidelines contained in it.)[24]

There are other perspectives on the blackboard model. The blackboard model is sometimes viewed as a model of general problem solving (Hayes-Roth, 1983). It has been used to structure cognitive models (McClelland & Rumelhart, 1981; Rumelhart & McClelland, 1982; and Hayes-Roth *et al.*, 1979); the OPM system, (described in Part 2) simulates the human planning process. Sometimes the blackboard model is used as an organizing principle for large, complex systems built by many programmers. The ALVan project (Stentz & Shafer, 1985) takes this approach.

Summary

The basic approach to problem solving in the blackboard framework is to divide the problem into loosely coupled subtasks. These subtasks roughly correspond to areas of specialization within the task (for example, there are human specialists for the subtasks). For a particular application, the designer defines a solution space and knowledge needed to find the solution. The solution space is divided into analysis levels of partial or intermediate solutions, and the knowledge is divided into specialized knowledge sources that perform the subtasks. The information on the analysis levels is globally accessible on the blackboard, making it a medium of interaction between the knowledge sources. Generally, a knowledge source uses information on one level of analysis as its input and produces output information on another level. The decision to employ a particular knowledge source is made dynamically using the latest information contained in the blackboard data structure (the current solution state). This particular approach to problem decomposition and knowledge application is very flexible and works well in diverse application domains. One caveat, however: How the problem is partitioned into subproblems makes a great deal of difference to

[23] The control component of the framework is extensible in many directions. In the BB-1 system (Hayes-Roth, 1983), the control problem is viewed as a planning problem. Knowledge sources are applied according to a problem-solving plan in effect. The creation of a problem-solving plan is treated as another problem to be solved using the blackboard approach.

[24] What about statements such as "Fortran Common is a blackboard" or "Object-oriented systems are blackboard systems?" All I can say is the potential for a thing is not that thing itself. With some effort, one can design and build a blackboard system in Fortran, and the common area is a good candidate for storing blackboard data. However, one also needs to design knowledge sources that are self-selecting and self-contained and control modules that determine the focus of attention and manage knowledge source application. The blackboard framework is a problem-solving framework. It is not a programming language, although an instance of the framework can have a blackboard language associated with it. It is also not a knowledge representation language, although one can use any knowledge representation language for the knowledge sources and the blackboard. Why can't I get away with placing a hunk of ground beef, a can of tomato sauce, a box of spaghetti, and bottles of seasoning in a pile, and call the pile a spaghetti dinner or, better yet, linguine a la pesta rosa? It would certainly simplify my life.

the clarity of the approach, the speed with which solutions are found, the resources required, and even the ability to solve the problem at all.

In order to discuss the details of various blackboard systems, it is helpful to trace the intellectual history of the blackboard concepts. Aside from being interesting in itself, it explains the origins of ideas and reasons for some of the differences between blackboard system designs. The reasons often have no rational basis but have roots in the "cultural" differences between the research laboratories that were involved in the early history of blackboard systems.

Evolution of Blackboard Architectures

> Metaphorically we can think of a set of workers, all looking at the same blackboard: each is able to read everything that is on it, and to judge when he has something worthwhile to add to it. This conception is just that of Selfridge's Pandemonium [19]: a set of demons, each independently looking at the total situation and shrieking in proportion to what they see that fits their natures... (Newell, 1962.)

Prehistory

The above quotation is the first reference to the term blackboard in the AI literature. Newell was concerned with the organizational problems of programs that existed at the time (for example, checker-playing programs, chess-playing programs, theorem-proving programs), most of which were organized along a generate-and-test search model[25] (Newell, 1969). The major difficulty in these programs was rigidity. He notes:

> A program can operate only in terms of what it knows. This knowledge can come from only two sources. It can come from assumptions [or] it can come from executing processes ... either by direct modification of the data structure or by testing ... but executing processes take time and space [whereas] assumed information does not have to be stored or generated. Therefore the temptation in creating efficient programs is always to minimize the amount of generated information, and hence to maximize the amount of stipulated information. It is the latter that underlies most of the rigidities.

In one example, Newell discusses an organization to synthesize complex processes by means of sequential flow of control and hierarchically organized, closed subroutines. Even though this organization had many advantages (isolation of tasks, space saving by coding nearly identical tasks once, and so on), it also had difficulties. First, conventions required for communication among the subroutines often forced the subroutines to work with impoverished information. Second, the ordered subroutine calls fostered the need for doing things sequentially. Third, and most importantly, it encouraged programmers to think of the total program in terms of only one thing going on at a time. However, in problem solving there are often many possible things to be processed at any given time (for example, exploring various branches of a search tree), and relatively weak and scattered information is necessary to guide the exploration for a solution (for example, observations noticed while going down one branch of a search tree could be used when going down another branch). The primary difficulties with this organization, then, were inflexible control and restricted data accessibility. It is within this context that Newell notes the difficulties "might be alleviated by maintaining the isolation of routines, but allowing all the subroutines to make use of a common data structure." He uses the blackboard metaphor to describe such a system.

The blackboard solution proposed by Newell eventually became the production system (Newell & Simon, 1972), which in turn led to the development of the OPS system (Forgy & McDermott, 1977). In OPS the subroutines are represented as condition-action rules[26], and the data are globally available in the working memory. One of the many "shrieking demons" (those rules whose "condition sides" are satisfied) is selected through a conflict-resolution process. The conflict-resolution process emulates the selection of one of the loudest demons, for example, one that addresses the most specific situation. OPS does reflect the blackboard concept as stated by Newell and provides for flexibility of control and global accessibility to data. However, the blackboard systems as we know them today took a slightly more circuitous route before coming into being.

In a paper first published in 1966 (later published in Simon, 1977), Simon mentions the term blackboard in a slightly different context from Newell. The discussion is within the framework of an information-processing theory about discovery and incubation of ideas:

> In the typical organization of a problem-solving program, the solution efforts are guided and controlled by a hierarchy or "tree" of goals and subgoals. Thus, the subject starts out with the goal of solving the original problem. In trying to reach this goal, he generates a subgoal ... If the subgoal is achieved, he may then turn to the now-modified original goal. If difficulties arise in achieving the subgoal, sub-subgoals may be created to deal with them ... we would specify that the goal tree be held in some kind of temporary memory, since it is a dynamic structure, whose function is to guide

[25] "Generate" is a process that produces each element of the solution space one at a time. The "test" process determines if the generated element satisfied some conditions of predicates in the task domain.

[26] See Davis & King, 1977, for an overview of production systems.

> search, and it is not needed when the problem solution has been found ... In addition, the problem solver is noticing various features of the problem environment and is storing some of these in memory ... What use is made of [a feature] at the time it is noted depends on what subgoal is directing attention at that moment ... over the longer run, this information influences the growth of the subgoal tree ... I will call the information about the task environment that is noticed in the course of problem solution and fixated in permanent (or relatively long-term) memory the "blackboard."

Although Newell's and Simon's concerns appear within different contexts, the problem-solving method they were using was the goal-directed, generate-and-test search method. They encountered two common difficulties: the need for previously generated information during problem solving and flexible control. It was Simon who proposed the blackboard ideas to Raj Reddy and Lee Erman for the HEARSAY project.[27]

Although the blackboard metaphor was suggested by Simon to the HEARSAY designers, the final design of the system, as might be expected, evolved out of the needs of the speech-understanding task. Such system characteristics as hierarchically organized analysis levels on the blackboard and opportunistic reasoning, which we now accept as integral parts of blackboard systems, were derived from needs and constraints that were different from Newell's and Simon's. One of the key notions attributable to the speech-understanding problem was the notion of the blackboard partitioned into analysis levels. This is a method of using and integrating different "vocabularies," as mentioned earlier, in problem solving. In most problem-solving programs of the time, such as game-playing and theorem-proving programs, the problem space had a homogeneous vocabulary. In the speech-understanding problem, there was a need to integrate concepts and vocabularies used in describing grammars, words, phones, and so on.

There are two interesting observations to be made from early history. First, the early allusions to a blackboard are closely tied to search methodologies, and, not surprisingly, the use of generate-and-test search is evident in HEARSAY-II. Second, although the HEARSAY-II blackboard system was designed independently from the OPS system, there are, as we might expect, some conceptual similarities. For example, the scheduler in HEARSAY-II is philosophically and functionally very similar to the conflict-resolution module in OPS, which, in turn, is a way of selecting one of the shrieking demons.

The HASP system, which has its own intellectual history, does not focus so much on search techniques as on knowledge-application techniques. Put another way, HASP was built in a culture that had a tradition of using problem-solving approaches that focused on applying large amounts of situation-specific knowledge rather than on applying a weak method (generate-and-test) using general knowledge about the task.[28] The methodology used to select and apply knowledge in HASP is, therefore, quite different philosophically from the one reflected in the HEARSAY-II scheduler. These and other differences are elaborated on in the next section.[29] Next is examined another branch of a history that influenced the design of HEARSAY-II, the speech-understanding task.

The HEARSAY Project

Although a blackboard concept was documented in AI literature as early as 1962 by Newell, it was implemented as a system a decade later by people working on a speech-understanding project. The first article on the HEARSAY system appeared in *IEEE Transactions on Audio and Electroacoustics* in 1973 (Reddy *et al.*, 1973).[30] There, the authors described the limitations of extant speech-recognition systems and proposed a model that would overcome the limitations. To summarize, the article stated that although the importance of context, syntax, semantics, and phonological rules in the recognition of speech was accepted, no system had been built that incorporated these ill-defined sources of knowledge. At the same time, the authors' previous work indicated (1) that the limitation of syntax-directed methods of parsing from left to right had to be overcome; (2) that parsing should proceed both forward and backward from anchor points; and (3) that because of the lack of feedback in a simple part-of hierarchical structure, the magnitude of errors on the lower level propagated multiplicatively up the hierarchy; that is, minor errors in the signal level, for example, became major errors on a sentence level.

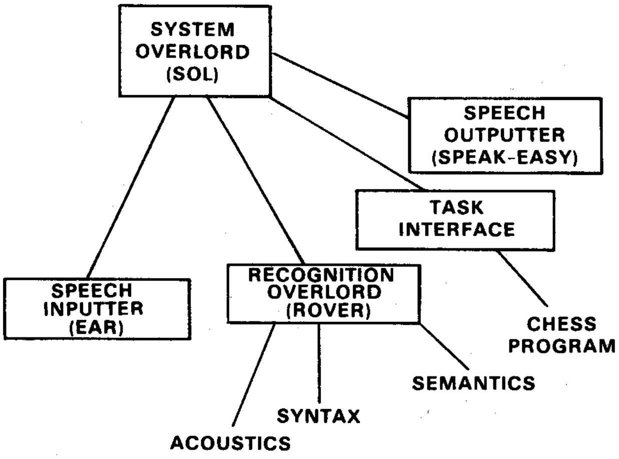
The system architecture described in the Reddy article, later to be known as the HEARSAY-I architecture, was based on a model that addressed the following requirements: (1) the contribution of each source of knowledge (syntax, semantics, context, and so on) to the recognition of speech had to be measurable; (2) the absence of one or more knowledge sources should not have a crippling effect on the overall performance; (3) more knowledge sources should improve the performance; (4) the system must permit graceful error recovery; (5) changes in performance

[27] These historical notes are communications from Herbert Simon.

[28] Most OPS expert systems use strong knowledge, but this came about later (*ca.* 1979).

[29] There is no denying that there are cultural differences in the AI laboratories; they foster different styles, methods, and lines of research. Whatever the research topic, the intellectual effort tends to follow the line of least resistance, and adopts the styles and methods at the researcher's own laboratory. For example, the work of Newell and Simon on general problem solving has a great deal of influence on much of the work at Carnegie-Mellon University. On the other hand, the work of Feigenbaum and Buchanan on applications of domain-specific knowledge influences the work at their Stanford University laboratory.

[30] The manuscript was delivered to IEEE on April 30, 1972.



Overview of the HEARSAY-I System*
Figure 6

requirements, such as increased vocabulary size or modifications to the syntax or semantics, should not require major modifications to the model. The functional diagram of the HEARSAY-I architecture is shown in Figure 6, and its behavior is summarized as follows:

> The EAR module accepts speech input, extracts parameters, and performs some preliminary segmentation, feature extraction, and labeling, generating a "partial symbolic utterance description." The recognition overlord (ROVER) controls the recognition process and coordinates the hypothesis generation and verification phases of various cooperating parallel processes. The TASK provides the interface between the task being performed and the speech recognition and generation (SPEAK-EASY) parts of the system. The system overlord (SOL) provides the overall control for the system.

From Figure 7, which illustrates the recognition process, one can glean the beginnings of an organization of a blackboard system. Note how the overlord (ROVER) controlled the invocation of activities. The beginnings of the scheduler, as well as the knowledge sources, are apparent, as they became incorporated in HEARSAY-II.

> Since the different recognizers are independent, the recognition overlord needs to synchronize the hypothesis generation and verification phases of various processes.... Several strategies are available for deciding which subset of the processes generates the hypotheses and which verify. At present this is done by polling the processes to decide which process is most confident about generating the correct hypothesis. In voice chess, [The task domain for HEARSAY-I was chess moves] where the semantic source of knowledge is dominant, that module usually generates the hypotheses. These are then verified by the syntactic and acoustic recognizers. However, when robust acoustic cues are present in the incoming utterance, the roles are reversed with the acoustic recognizer generating the hypotheses."

"Knowledge sources are activated in a lock-step sequence consisting of three phases: poll, hypothesize, and test." (Hayes-Roth *et al.*, 1979.) During the polling phase, the overlord queries the knowledge sources to determine which ones have something to contribute to that region of the sentence hypothesis which is "in focus" and with what level of "confidence".[31] In the hypothesizing phase, the most promising knowledge source is activated to make its contribution. Finally, in the testing phase, knowledge sources evaluate the new hypotheses.

Some of the difficulties encountered in HEARSAY-I can be attributed to the way in which the solution to the application task was formulated, and other difficulties arose from the design of the system. The problem was formulated to use the hypothesize-and-test paradigm only on the word level, that is, the blackboard only contained a description at the word level. This meant that all communication among the knowledge sources was limited to sharing information at the word level. This formulation caused two major difficulties. First, it becomes difficult to add nonword knowledge sources and to evaluate their contributions. Second, the inability to share information contributed by nonword knowledge sources caused the information to be recomputed by each knowledge source that needed it. In other words, the difficulty lay in trying to force the use of a single vocabulary when multiple vocabularies were needed.

The architectural weaknesses of HEARSAY-I, as stated by its designers, lay in (1) the lock-step control sequence that limited "parallelism,"[32] (2) the lack of provision to express relationships among alternative sentence hypotheses, and (3) the built-in problem-solving strategy that made modifications awkward and comparisons of different strategies impossible (Lesser *et al.*, 1974). To overcome these difficulties, information (in the multiple vocabularies needed to understand utterances) used by all the knowledge sources was uniformly represented and made globally accessible on the blackboard in HEARSAY-II. In addition, a scheduler dynamically selected and activated the appropriate knowledge sources. (In Part 2 the design of the HEARSAY-II system is described in detail.)

*From (Reddy *et al.*, 1973a).

[31] The poll portion of the poll, hypothesize, and test is also very characteristic of OPS and HEARSAY-II. This construct takes on a totally different form in HASP and other subsequent systems.

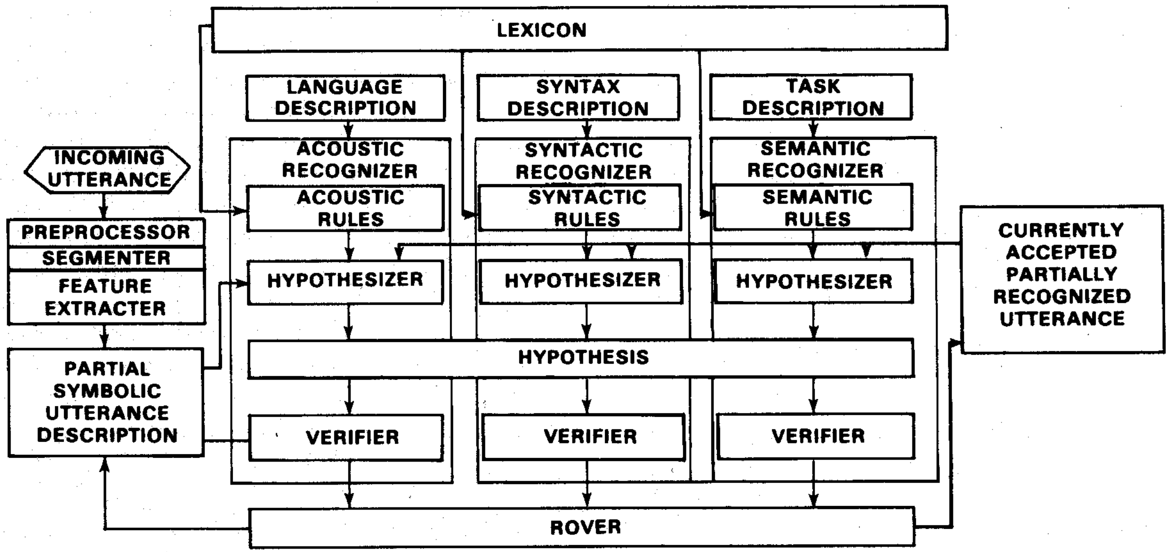
[32] The term parallelism was used quite early in the project even though at that time the system ran on uniprocessors. Later (*ca.* 1976), experiments with parallel executions were conducted on the C.mmp system (Fennell & Lesser, 1977).

During the time that HEARSAY-II was being developed, the staff of the HASP project was looking for an approach to solve its application problem. The search for a new methodology came about because the plan-generate-and-test problem-solving method that was successful for interpreting mass-spectrometry data in the DENDRAL program (Lindsay *et al.*, 1980) was found to be inappropriate for the problem of interpreting passive sonar signals. In the history of blackboard systems, HASP represents a branching point in the philosophy underlying the design of blackboard systems. Generally, later systems can be thought of as modifications of, or extensions to, either the HEARSAYlike or HASPlike designs.

The HASP Project

The task of HASP was to interpret continuous sonar signals passively collected by hydrophone arrays monitoring an area of the ocean. Signals are received from multiple arrays, with each array consisting of multiple hydrophones. Each array has some directional resolution. Imagine a large room full of plotters, each recording digitized signals from the hydrophones. Now, imagine an analyst going from one plotter to the next trying to discern what each one is hearing, and then integrating the information from all the plots in order to discern the current activity in the region under surveillance. This interpretation and analysis activity goes on continuously day in and day out. The primary objective of this activity is to detect enemy submarines. The objective of the HASP project was to write a program that "emulated" the human analysts, that is to incorporate, in a computer program, the expertise of the analysts, especially their ability to detect submarines.[33] The HASP problem was chosen to work on because it appeared to be similar to the DENDRAL problem, a signal interpretation problem for which there were experts who could do the job. The system designers were confident that the problem-solving approach taken in DENDRAL would work for HASP. What was DENDRAL's task, and what was its approach? To quote from Feigenbaum (1977), the task was

> to enumerate plausible structures (atom-bond graphs) for organic molecules, given two kinds of information: analytic instrument data from a mass spectrometer and a nuclear magnetic resonance spectrometer; and user-supplied constraints on the answers, derived from any other source of knowledge (instrumental or contextual) available to the user.
>
> DENDRAL's inference procedure is a heuristic search that takes place in three stages, without feedback: *plan-generate-and-test.*
>
> *Generate* is a generation process for plausible structures. Its foundation is a combinatorial algorithm that can produce all the topologically legal candidate structures. Constraints supplied by the user or by the *Plan* process prune and steer the



Details of the Recognition Process*

Figure 7

*From (Reddy *et al.*, , 1973b).

[33] This was in 1973 before the term expert system was coined. The only expert system in existence at the time was DENDRAL, and MYCIN was on its way.

> generation to produce the plausible set and not the enormous legal set.
>
> *Test* refines the evaluation of plausibility, discarding less worthy candidates and rank-ordering the remainder for examination by the user... It evaluates the worth of each candidate by comparing its predicted data with the actual input data... Thus, test selects the 'best' explanation of the data.
>
> *Plan* produces direct (that is, not chained) inference about likely substructures in the molecule from patterns in the data that are indicative of the presence of the substructure. In other words, Plan worked with combinatorially reduced abstracted sets to guide the search in a generally fruitful direction.

If some of the words in this description were replaced, the plan-generate-and-test approach seemed appropriate for the HASP tasks:

> *Generate* plausible ship candidates and their signal characteristics. *Test* by comparing the predicted signals with the real signals. *Plan* by selecting types of ships that could be in the region of interest. The Plan phase would use intelligence reports, shipping logs, and so on.

The system designers had already talked with the analysts and had read their training manuals. They knew the necessary knowledge could be represented as rules, a form of domain knowledge representation that had proven its utility and power in DENDRAL. Difficulties were encountered immediately; some of these were:

1. The input data arrived in a continuous stream, as opposed to being batched like in DENDRAL. The problem of a continuous data stream was solved by processing data in time-framed batches.
2. The analysis of the activities in the ocean had to be tracked, and updated over time. Most importantly, past activities played an important role in the analysis of current activities.
3. There were numerous types of information that seemed relevant but remote from the interpretation process, for example, the average speeds of ships.

To address the second problem, it was immediately clear that a data structure was needed which was equivalent to a "situation board" used by the analysts; the data structure was called the Current Best Hypothesis (CBH). CBH reflected the most recent hypothesis about the situation at any given point in time. This could serve as the basis for generating a "plan"; that is, the CBH could be used as a basis for predicting the situation to be encountered in the next time frame. The prediction process could also utilize and integrate the variety of information mentioned in item 3. The predicted CBH would then be used (1) to verify that the interpretation from the previous time frame was correct (2) to reduce the number of alternatives generated during past time frames,[34] and (3) to reduce the number of new signals not accounted for in the predicted CBH that needed to be analyzed in full. CBH was thought of as a cognitive "flywheel" that maintained the continuous activities in a region of ocean between time frames. The initial design, a modified version of DENDRAL, was sketched out in December of 1973 (see Figure 8).

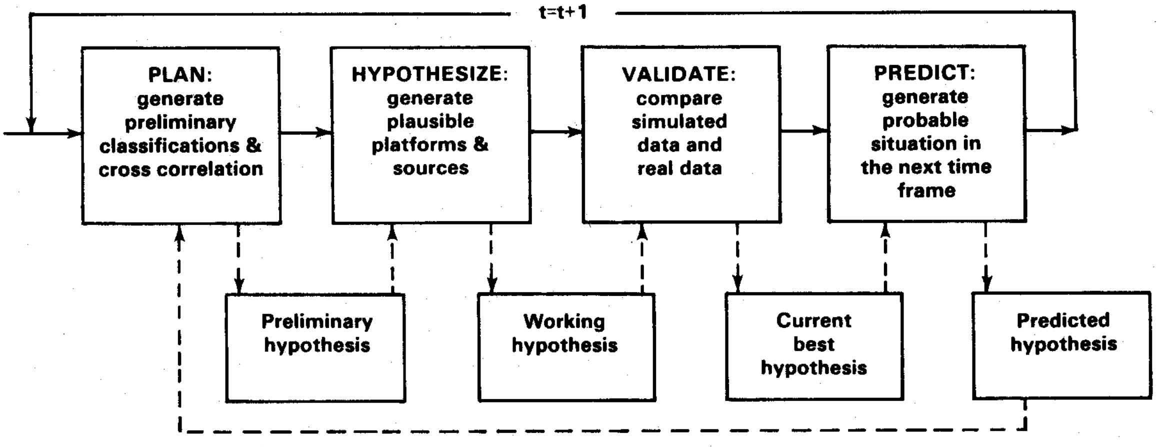
Then there came the bad news. There was no plausible generator of the solution space, and there was no simulator to generate the signals of hypothesized platforms. The bad news had a common root; given a platform, there was a continuum of possible headings, speeds, and aspects relative to an array. Each parameter, in addition to variations in the water temperature, depth, and so on, uniquely affected the signals "heard" at an array. Consequently, there was a continuum of possibilities in the solution space as well as for the simulator to simulate. The designers tried to limit the number of possibilities, for example, by measuring the headings by unit degrees, but this left an enormous search space. Moreover, there was not enough knowledge to prune the space to make the generate-and-test method practical. The DENDRAL approach was abandoned. Then the HEARSAY-II approach was learned of. The description of the approach produced enough of a mental shift in the way the HASP problem was viewed that a new solution could be designed. It should be noted in passing that HEARSAY-II in fact had generators and used them. It was the idea of fusing uncertain and partial solutions to construct solutions, combined with "island driving,"[35] that intrigued the designers.

The sonar analysts solved the problem piecemeal. They first identified a harmonic set in the signals. The "accounted-for" signals were then "subtracted" from the data set. Then another harmonic set would be formed with the remaining data, and so on until all the signals were accounted for.[36] Each harmonic set implied a set of possible sources of sound (for example, a propeller shaft), which in turn implied a set of possible ship types from which the sounds could be emanating. Certain signal characteristics directly implied platform types, but this type of diversion from the incremental analysis was very rare. What the hu-

[34] There was only one solution hypothesis. However, some attributes, for example, platform type, could have alternative values.

[35] *Island driving* is a problem solving strategy. A relatively reliable partial hypothesis is designated as an "island of certainty," and the hypothesis building pushes out from this solution island in many directions. This is sometimes called a "middle-out" strategy. There can be many islands of certainty driving the problem-solving process.

[36] As easy as it sounds, the task of harmonic set formation was a very difficult one, given noisy and missing data, and could produce large combinatorial possibilities. Addressing this problem became one of the major concerns in HASP.


t=t+1
PLAN: generate preliminary classifications & cross correlation
HYPOTHESIZE: generate plausible platforms & sources
VALIDATE: compare simulated data and real data
PREDICT: generate probable situation in the next time frame
Preliminary hypothesis
Working hypothesis
Current best hypothesis
Predicted hypothesis

(— Control flow; --- Data flow)

There was one global data structure that contained a hypothesis about the situation. After each of the plan, hypothesize, validate, and predict phases, the hypothesis changed states. These states were called the preliminary hypothesis, the working hypothesis, the current best hypothesis, and the predicted hypothesis.

HASP Design Based on DENDRAL

Figure 8

man analysts were doing was what might be called logical induction and synthesis.[37] Hypotheses were synthesized from pieces of data using a large amount of domain-specific knowledge that translated information in one form to information in another form, that is, transformed a description in one vocabulary to one in another vocabulary. For example, a set of frequencies was transformed into a set of possible ship parts (for example, a shaft or a propeller) by using knowledge of the form, "If the harmonic set consists of …, then it is likely due to …" The partial solutions thus formed were then combined using other knowledge to construct acceptable solutions.

The analysts were also strongly model driven.· There were common shipping lanes used by merchant ships traveling from one port to another. These platforms usually maintained a steady speed and heading. This and similar knowledge served to constrain the number of possible partial solutions. For example, if a hypothetical surface platform traveled across a shipping lane, then the possibility that it might be a merchant ship could be eliminated. In this example, a model of ship movements was able to aid in the platform classification process. Moreover, knowledge about the characteristics of platforms was used to combine lower-level, partial solutions. Suppose a platform type was hypothesized from an acoustic source, for example, a propeller shaft. Knowledge about the platform type (a model) was then used to look for other acoustic sources (for example, engine) belonging to that platform. This type of top-down, model-driven analysis was used as often as the bottom-up signal analysis.

Once it was clear that interpretation in HASP, as in HEARSAY, was a process of piecemeal generation of partial solutions that were combined to form complete solutions, the HEARSAY-II system organization could be exploited. The CBH was partitioned into levels of analysis corresponding to the way analysts were used to thinking (that is, signals, harmonic sets, sources, and ship types). The rule-based knowledge gathered for the purposes of pruning and guiding the search process was organized into sets of rules (knowledge sources) that transformed information on one level to information on another level.[38]

Nothing is as easy as it appears. There were many differences between the speech and the sonar signal understanding tasks that drove the HASP system architecture in a different direction from HEARSAY-II. The use of the blackboard as a situation board that evolves over time has already been mentioned. This is somewhat equivalent to asking a speaker to repeat his utterance over and over again while moving around, and having the interpretation improve with each repeated utterance as well as being able to locate the speaker after each utterance. After each utterance, the CBH would reflect the best that the system could do up to that point. It was also mentioned that sets of rules, as opposed to procedures in HEARSAY-II, were used to represent knowledge sources. Rules were chosen because they were used in DENDRAL and in MYCIN.[39]

[37] An interesting article on this point is " A More Rational View of Logic." by Alex P. Pentland and Martin A. Fischler; it appeared in the Winter, 1983 issue of the *AI Magazine.*

[38] It is interesting to note that many of the pieces of knowledge intended for pruning purposes could be converted into inductive knowledge. For example, a pruning rule that read "If a signal is coming from outside the normal traffic lane, then its source could not be cargo or cruise ships" could be used directly for reducing alternatives or could be converted to read "…, then its source is either military ships or fishing boats." One can hold the view that this is not surprising, because knowledge is knowledge, and what counts is how and when it's used.

[39] This is a good example of the cultural influence. No other repre-

This choice of knowledge representation had a great influence in simplifying the HASP scheduler. The following characteristics influenced the final design of HASP.

Events: The concept of events is inherent in the HASP problem. For example, a certain type of frequency shift in the signal would be an event that implied the ship was changing its speed. An appearance or disappearance of a signal would be an event that implied a new ship was on the scene or a known ship was getting out of the range of the sensors, or it implied an expected behavior of certain types of ships. This inherent task characteristic made it natural for the HASP system to be an event-based system; that is, an occurrence of a particular event implied that new information was available for some *a priori* determined knowledge source to pursue. The goals of the task dictated what events were significant and what were not. This, in turn, meant that the programmer (the knowledge engineer of today) could *a priori* decide what changes in the blackboard, that is, events, were significant for solving the problem (as opposed to the system noticing every change). Furthermore, the only time a knowledge source needed to be activated was when some events occurred that it knew about. These task characteristics, together with the use of a rule-based knowledge representation, helped redefine and simplify the task of the scheduler in the sense that each piece of knowledge was more or less self-selecting for any given event.[40]

Temporal events: In HEARSAY-II "time" meant the sequence in which the words appeared in a spoken sentence. Based on the sequence of words, one could predict or verify the appearance of another set of words later or earlier in the sequence. In HASP time had different connotations. In one sense, time was similar to the separate utterances in the hypothetical repetitive utterances problem mentioned earlier. There was information redundancy, as well as new and different information (no two utterances sound exactly the same), as time went on. Redundancy meant that the system was not pressed to account for every piece of data at each time frame. It could wait to see if a clearer signal appeared later, for example. Also, time meant that the situation at any time frame was a "natural" consequence of earlier situations, and such information as trends and temporal patterns (both signal and symbolic) that occur over time could be used. One of the most powerful uses of time in this sense was the generation and use of expectations of future events.

Multiple input streams: Aside from the digitized data from many hydrophones, HASP had another kind of input—reports. Reports contained information gathered from intelligence or normal shipping sources. These reports tended to use descriptions similar to those used on the ship level on the blackboard (CBH). Whereas the ordinary data came in at the bottom level for both HEARSAY and HASP, HASP had another input "port" at the highest level. Given the input at this level, the system generated the kinds of acoustic sources and acoustic signatures it expected in the future based on information in its taxonomic knowledge base. This type of model-based expectation was one of the methods used to "fuse" report data with signal data.

Explanation: The purpose of explanation is to understand what is going on in the system from the perspective of the user and the programmer. Because the needs of the users are different from those of the programmers, explanation can take on many forms. Explanation for the user was especially important in HASP, because there was no way to test the correctness of the answer. The only way to test the performance of the system was to get human analysts to agree that the system's situation hypotheses and reasoning were plausible. CBH, with its network of evidential support, served to justify the elements of the hypothesis and their hypothetical properties. It served to "explain" the relationships between the signal data and its various levels of interpretation. The explanation of the reasoning, that is, "explaining" which pieces of knowledge had been applied under what circumstances, was made possible by "playing back" the executed rules.[41]

There were many other differences, but these characteristics had the most impact on the design of the eventual system. The list serves to illustrate how strongly the task characteristics influence blackboard architectures. (The details of the organization of the HASP system are explained in Part 2.)

Since Hearsay-II and HASP, there have been a variety of other programs whose system designs are rooted in the blackboard model. These programs include applications in the area of interpretation of electron density maps of protein crystals (Terry, 1983), planning (Hayes-Roth *et al.*, 1979), scene analysis (Nagao & Matsuyama, 1980), and signal understanding and situation assessment (Spain, 1983; & Williams, 1985). There are other applications currently being built in the areas of process control, very large-scale integration (VLSI) design, crisis management, image understanding, and signal interpretation. Many applications in the Defense Advanced Research Project Agency's (DARPA) Strategic Computing Program in the areas of military battle management, a pilot's associate, and autonomous vehicles utilize the blackboard model. To date, there is no commonly agreed upon architecture for

sentation was even considered.

[40] The relationship between events within the task and events in the system are discussed in Part 2.

[41] In MYCIN and other similar rule-based programs, explanation consists of a playback of rule firings. In HASP the ordinary method of playback turned out to be useful only to programmers for debugging purposes. For the user, the rules were either too detailed or were applied in a sequence (breadth first) that was hard for them to understand. In HASP the explanation was generated from an execution history with the help of "explanation templates" that selected the appropriate rule activities in some easy-to-understand order.

blackboard systems. Rather, there are more or less strict interpretations of the model. The blackboard framework that distills the common constructs and features of many blackboard systems has been introduced. In Part 2, documented systems that follow the intent and spirit of the blackboard model are described.

References

Davis, Randall & King, Jonathan (1977) An overview of production systems. In E.W. Elcock & D. Michie (Eds.), *Machine intelligence 8: Machine representation of knowledge.* New York: John Wiley.

Erman, D. L.. Hayes-Roth, F., Lesser, V. R., & Reddy, D. Raj. (1980) The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Survey* 12:213–253.

Feigenbaum, Edward A. (1977) The art of artificial intelligence: I. Themes and case studies of knowledge engineering. IJCAI 5: 1014–1029.

Fennel, Richard D. & Lessor, Victor R. (1977) Parallelism in AI problem solving: A case study of HEARSAY-II. *IEEE Transactions on Computers,* 98–111.

Forgy, C. & McDermott, J. (1977) *OPS, a domain-independent production system language.* IJCAI 5: 933-939.

Hayes-Roth, Barbara. (1983) The blackboard architecture: A general framework for problem solving?. Tech. Rep. HPP-83-30, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Hayes-Roth, Barbara. (1985) Blackboard architecture for control. *Journal of Artificial Intelligence* 26:251–321.

Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S., & Cammarata, S. (1979) *Modelling planning as an incremental, opportunistic process.* IJCAI 6: 375-383.

Lesser, Victor R., Fennell, Richard D.,Erman, Lee D., & Reddy, D. Raj. (1974) Organization of the HEARSAY-II speech understanding system. In *IEEE Symposium on Speech Recognition,* pp. 11-M2–21-M2.

Lindsay, R., Buchanan, B. G., Feigenbaum, E. A., & Lederberg, J. (1980) *Applications of artificial intelligence for organic chemistry: The Dendral project.* New York, McGraw-Hill.

McClelland J. L. & Rumelhart D.E. (1981) An interactive activation model of context effects in letter perception: Part 1, an account of basic findings. *Psychological Review* 88:375–407.

Nagao, Makoto, & Matsuyama, Takashi. (1980) *A structural analysis of complex aerial photographs.* New York: Plenum Press.

Newell, Allen. (1962) Some problems of basic organization in problem-solving programs. In M. C. Yovits, G. T. Jacobi, & G. D. Goldstein (Eds.), *Conference on Self-Organizing Systems.* Washington, D. C.: Spartan Books, 393–423.

Newell, Allen. (1969) *Heuristic programming: Ill-structured problems. Progress in operations research.* New York: John Willey, III:360-414.

Newell, Allen & Simon, Herbert A. (1972) *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Reddy, D. Raj, Erman, Lee D., & Richard B. Neely. (1973) A model and a system for machine recognition of speech. *IEEE Transactions on Audio and Electroacoustics* AU-21:229–238.

Reddy, D. Raj, Erman, Lee D., & Neely, Richard B. (1973) *The HEARSAY speech understanding system: An example of the recognition process.* IJCAI 3: 185–193.

Rumelhart, D. E. & McClelland, J. L. (1982) An interactive model of context effects in letter perception: Part 2, The enhancement effect and some tests and extensions to the model. *Psychological Review* 89: 60–94.

Selfridge, Oliver G. (1959) *Pandemonium: A paradigm for learning.* Proceedings of the Symposium on the Mechanization of Thought Processes, 511–529.

Shortliffe, Edward H. (1976) *Computer-based medical consultation: MYCIN.* New York: American Elsevier.

Simon, Herbert A. (1977) Scientific discovery and the psychology of problem solving. In *Models of discovery.* Boston, Mass: D. Reidel Publishing Company.

Spain, David S. (1983) *Application of artificial intelligence to tactical situation assessment.* Proceedings of the 16th EASCON 83. September.

Stentz, Anthony & Shafer, Steve. (1985) Module programmer's guide to local map builder for ALVan. Technical Report, Carnegie-Mellon University Computer Science Department, August.

Terry, Allan (1983) The CRYSALIS Project: Hierarchical control of production systems. Tech. Rep. HPP-83-19, Stanford University, Heuristic Programming Project.

Williams, Mark A. (1985) Distributed, cooperating expert systems for signal understanding. *Seminar on AI Applications to Battlefield* 3.4-1–3.4-6.