Blackboard Systems

Daniel D. Corkill
Blackboard Technology Group, Inc.

Blackboard systems are not new technology. The first blackboard system, the Hearsay-II speech understanding system [1], was developed nearly twenty years ago. While the basic features of Hearsay-II remain in today's blackboard systems, numerous advances and enhancements have been made as a result of experience gained in using blackboard systems in widely varying application areas.

Unlike most AI problem-solving techniques that implement formal models, the blackboard approach was designed as a means for dealing with ill-defined, complex applications. Unconstrained by formal requirements, researchers and developers have had considerable flexibility in inventing and applying advanced techniques to blackboard architectures. However, the lack of formal specifications has also contributed to confusion about blackboard systems and their proper place in the AI problem-solving toolkit.

This article describes the characteristics and potential of blackboard systems. I'll discuss what a blackboard system is (and is not) and why the use of blackboard-based problem solving is only now emerging from the academic and research laboratory. Finally, I'll discuss whether you should consider using the blackboard approach for your applications and how to get started using a blackboard approach.

1 The Blackboard Metaphor

Blackboard-based problem solving is often presented using the following metaphor:

> *Imagine a group of human specialists seated next to a large blackboard. The specialists are working cooperatively to solve a problem, using the blackboard as the workplace for developing the solution.*
>
> *Problem solving begins when the problem and initial data are written onto the blackboard. The specialists watch the blackboard, looking for an opportunity to apply their expertise to the developing solution. When a specialist finds sufficient information to make a contribution, she records the contribution on the blackboard, hopefully enabling other specialists to apply their expertise. This process of adding contributions to the blackboard continues until the problem has been solved.*

This simple metaphor captures a number of the important characteristics of blackboard systems, each of which is described separately below.

Independence of expertise (I think, therefore I am.)

The human specialists in the metaphor were not trained to work solely with that specific group of specialists. Our metaphorical specialists learned their expertise in vastly different situations. Some specialists have years of work experience, others recently received academic degrees, and still others are outside consultants brought in specifically for this particular problem. Each

This is the original, unabridged version of the article that appeared in *AI Expert* 6(9):40-47, September 1991.

specialist is a self-contained expert on some aspects of the problem and can contribute to the solution independently of the particular mix of other specialists in the room.

Blackboard systems also have this functional modularization of expertise. Each knowledge module (called a *knowledge source*, or simply a KS) is a specialist at solving certain aspects of the overall problem. No KS requires other KSs in making its contribution. Once it finds the information it needs on the blackboard, it can proceed without any assistance from other KSs. Furthermore, without changing any other KSs, additional KSs can be added to the blackboard system, poorer performing KSs can be enhanced, and inappropriate KSs can be removed. KSs perform relatively large computations, reflecting the processing required to implement their specialty.

Rule-based systems are also modular, but at the level of individual rules. Unlike the large-grained scope of KSs, the small size of each rule prevents full independence. A pair of rules that implement iteration by using a counter value and a termination rule is an example of two rules that cannot be designed independently or removed individually without affecting the performance of the other rule.

Diversity in problem-solving techniques (I don't think like you do.)

There are vast differences in how human experts think about and solve problems. Yet, these differences do not prevent our metaphorical group of specialists from solving the problem.

In blackboard systems, the internal representation and inferencing machinery used by each KS is similarly hidden from direct view. The blackboard approach views each KS as a black box in which the internal workings are invisible from the outside. It does not matter if one KS is a forward-chaining rule-based system, another uses a neural network approach, another uses a linear-programming algorithm, and still another is a procedural simulation program. Each of these diverse approaches can make its contributions within the blackboard framework.

Flexible representation of blackboard information (If you can draw it, I can use it.)

Our metaphorical human specialists could use any intelligible doodles when adding their contributions to the blackboard. They might use formulas, diagrams, sentences, checklists, and numerous circles and arrows.

Representational flexibility is similarly important in blackboard systems. The blackboard model does not place any prior restrictions on what information can be placed on the blackboard. One blackboard application might use assertional blackboard data and require that consistency be maintained. Another application might allow incompatible alternatives to be maintained on the blackboard, with each alternative available for opportunistic[1] exploration of the solution.

Common interaction language (What'd you say?)

While flexible representation of blackboard information is important, there must also be a common understanding of the representation of the information placed on the blackboard in order for the specialists to interact. The formulas, diagrams, sentences, and checklists must be understood by all specialists who need to access the information. If our metaphorical specialists consisted of specialists of differing nationalities, the use of different languages on the blackboard would hamper or even prohibit sufficient interaction to solve the problem.

Similarly, KSs in blackboard systems must be able to correctly interpret the information recorded on the blackboard by other KSs. Private jargon shared by only a few KSs limits the flexibility of applying other KSs on that information. In practice, there is a trade off between

[1] Opportunistic is a buzzword intimately associated with blackboard systems. It means having the control flexibility to perform the most appropriate problem-solving action at each step in the solution process.

the representational expressiveness of a specialized representation shared by only a few KSs and a fully general representation understood by all KSs. Finding the proper balance is an important aspect of blackboard-application engineering.

Positioning metrics (You could look it up.)

If the problem being solved by our human specialists is complex and the number of their contributions made on the blackboard begins to grow, quickly locating pertinent information becomes a problem. A specialist should not have to scan the entire blackboard to see if a particular item has been placed on the blackboard by another specialist.

One solution is to subdivide the blackboard into regions, each corresponding to a particular kind of information. This approach is commonly used in blackboard systems, where different levels, planes, or multiple blackboards are used to group related objects.

Similarly, ordering metrics can be used within each region, to sort information numerically, alphabetically, or by relevance. Advanced blackboard-system frameworks provide sophisticated multidimensional metrics for efficiently locating blackboard objects of interest.

Efficient retrieval is needed to support the use of the blackboard as a group memory for contributions generated by earlier KS executions. An important characteristic of the blackboard approach is the ability to integrate contributions for which relationships would be difficult to specify by the KS writer in advance.

For example, a KS working on one aspect of the problem may put a contribution on the blackboard that does not initially seem relevant or immediately interesting to any other KS. Only until much later, when substantial work on other aspects of the problem has been performed, is there enough context to realize the value of the early contribution. By retaining these contributions on the blackboard, the system can save the results of these early problem-solving efforts, avoiding recomputing them later (when their importance is understood). Additionally, the system can notice when promising contributions placed on the blackboard remain unused by other KSs and possibly choose to focus problem-solving activity on understanding why they did not fit with other contributions.

Locating previously generated contributions of interest is dependent upon the context of other information being used by a KS. This makes a simple pattern-matching specification of the specific contributions difficult and computationally inefficient. Many contributions placed on the blackboard may never prove useful, and maintaining the state of numerous, partially completed patterns is expensive. Therefore, an important characteristic of blackboard systems is enabling an executing KS to quickly and efficiently inspect the blackboard to see if relevant information is present.

The developers of the original Hearsay-II system recognized that rule-like condition specifications of KS interest would be ineffective. Instead, they opted for a combination of simple triggering-condition specifications to be followed by a more detailed procedural examination of the blackboard before activating the KS for execution.

Event-based activation (Is anybody there?)

In the metaphor, specialists do not interact directly. Each specialist watches the blackboard, looking for an opportunity to contribute to the solution. Such opportunities arise when an event occurs (a change is made to the blackboard) that enables the specialist to act. Blackboard events include the addition of some new information to the blackboard, a change in some existing information, or the removal of existing information. Some specialists may also respond to external events, such as receiving a telephone call, noticing it is lunch time, and so on.

KSs in blackboard systems are similarly triggered in response to blackboard and external events. Rather than having each KS scan the blackboard (as in the metaphor), each KS informs

the blackboard system about the kind of events in which it is interested. The blackboard system records this information and directly considers the KS for activation whenever that kind of event occurs.

Need for control (It's my turn.)

What if most of the human specialists respond to an event and all rush to the blackboard simultaneously? Some means of ordering their contributions is needed. (A single piece of chalk is a simple control strategy, but one that favors the swiftest rather than the most appropriate specialist.)

A manager, separate from the individual specialists, can be used to restore civility at the blackboard. The manager's job is to consider each specialist's request to approach the blackboard in terms of what the specialist can contribute and the effect that the contribution might have on the developing solution. The manager attempts to keep problem solving on track, to insure that all crucial aspects of the problem are receiving attention, and to balance the stated importance of different specialist's contributions.

Blackboard systems have a similar approach to controlling KSs. A control component that is separate from the individual KSs is responsible for managing the course of problem solving. The control component can be viewed as a specialist in directing problem solving, by considering the overall benefit of the contributions that would be made by triggered KSs. When the currently executing KS activation completes, the control component selects the most appropriate pending KS activation for execution.

Importantly, the control component must be able to make its selection among pending KS executions without posessing the expertise of the individual KSs. Without such a separation, the modularity and independence of KSs would be lost. Therefore, the control component must be able to ask for estimates from triggered KSs in making its control decisions.

When a KS is triggered, the KS uses its expertise to evaluate the quality and importance of its contribution. Each triggered KS informs the control component of the quality and costs associated with its contribution, without actually performing the work to compute the contribution. Instead, each KS generates estimates of the computations that would be generated by using fast, low-cost, approximations developed by the KS writer. These estimates are of the form, "If I am executed, I'll generate contributions of this type, with these qualities, while expending these resources." The control component uses these estimates to decide how to proceed.

Incremental solution generation (Step by step, inch by inch...)

In the metaphor, the solution is generated incrementally as each specialist adds contributions to the blackboard. No single specialist can solve the problem. Instead, specialists refine and extended one another's contributions, building the solution incrementally.

Blackboard systems also operate incrementally. KSs contribute to the solution as appropriate, sometimes refining, sometimes contradicting, and sometimes initiating a new line of reasoning. Blackboard systems are particularly effective when there are many steps toward the solution and many potential paths involving those steps. By opportunistically exploring the paths that are most effective in solving the particular problem, a blackboard system can significantly outperform a problem solver that uses a predetermined approach to generating a solution.

Now that we've considered the metaphor in detail, let's restate the blackboard model of problem solving.

2 The Blackboard Model of Problem Solving

A blackboard system consists of three components (Figure 1):

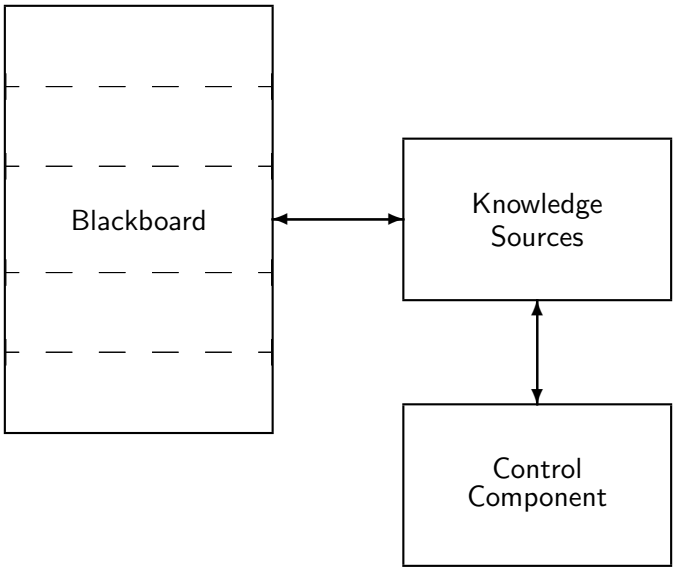

Figure 1: Basic Components of the Blackboard Model

- Knowledge sources (KSs) are independent modules that contain the knowledge needed to solve the problem. KSs can be widely diverse in representation and inference techniques.
- The blackboard is a global database containing input data, partial solutions, and other data that are in various problem-solving states.
- A control component makes runtime decisions about the course of problem solving and the expenditure of problem-solving resources. The control component is separate from the individual KSs. In some blackboard systems, the control component itself is implemented using a blackboard approach (involving control KSs and blackboard areas devoted to control).

Let's consider each component in more detail (Figure 2).

KSs

Each KS is separate and independent of all other KSs. A KS needs no knowledge of the expertise, or even the existence, of the others; however, it must be able to understand the state of the problem-solving process and the representation of relevant information on the blackboard.

Each KS knows the conditions under which it can contribute to the solution and, at appropriate times, attempts to contribute information toward solving the problem. This knowledge that each KS has about when to contribute to the problem-solving process is known as a *triggering condition*.

KSs are much larger grained than the individual rules used by expert systems. While expert systems work by firing a rule in response to stimuli, a blackboard system works by firing an entire knowledge module, or KS, such as an expert system; a neural net or fuzzy logic routine; or a procedure.

Unlike our metaphor, KSs are not the active agents in a blackboard system. Instead, KS activations (sometimes called KS instances) are the active entities competing for execution resources. A KS activation is the combination of the KS knowledge and a specific triggering context. The distinction between KSs and KS activations is important in applications where numerous events trigger the same KS. In such cases, control decisions involve choosing among

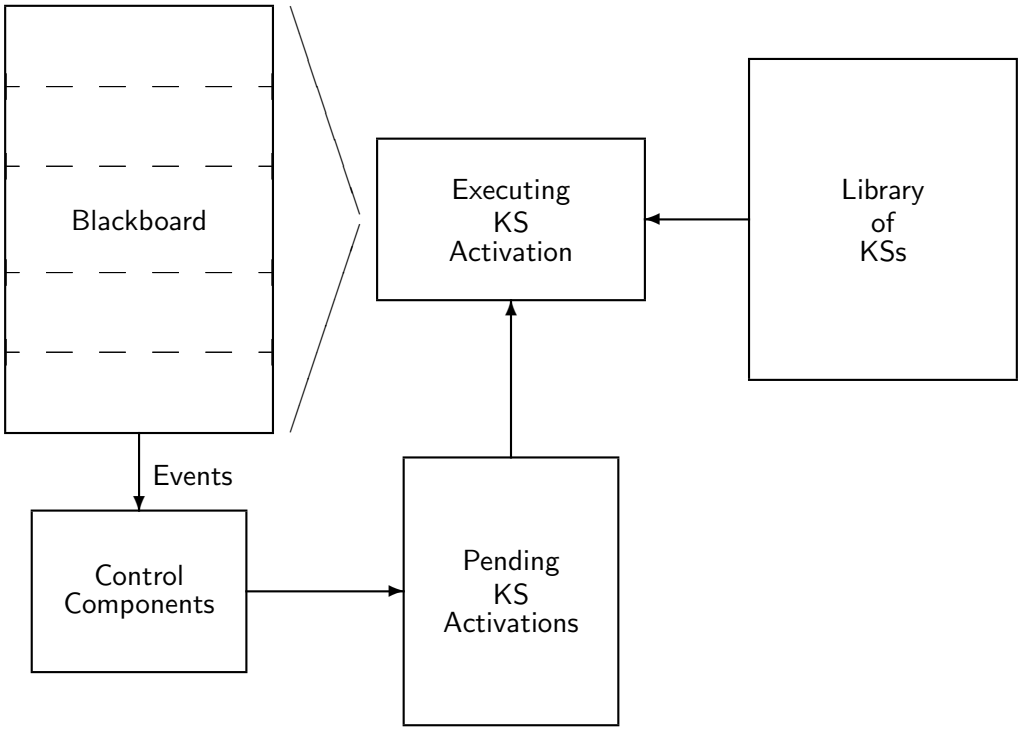

Figure 2: Basic Blackboard System

particular applications of the same KS knowledge (focusing on the appropriate data context), rather than among different KSs (focusing on the appropriate knowledge to apply). KSs are static repositories of knowledge, KS activations are the active processes.

The blackboard

The blackboard is a global structure that is available to all KSs and serves as:

- a community memory of raw input data; partial solutions, alternatives, and final solutions; and control information
- a communication medium and buffer
- a KS trigger mechanism.

Blackboard applications tend to have elaborate blackboard structures, with multiple levels of analysis or abstraction.

Occasionally, a system containing subsystems that communicate using a global database is incorrectly presented as a blackboard system. (A set of FORTRAN routines using `COMMON` is an extreme example of this view). True blackboard systems involve closely interacting KSs and a separate control mechanism.

Control component

An explicit control mechanism directs the problem-solving process by allowing KSs to respond opportunistically to changes on the blackboard database. On the basis of the state of the blackboard and the set of triggered KSs, the control mechanism chooses a course of action.

A blackboard system uses an incremental reasoning style: the solution to the problem is built one step at a time. At each step, the system can:

- execute any triggered KS

- choose a different focus of attention, on the basis of the state of the solution.

Under a typical control approach, the currently executing KS activation generates events as it makes contributions to the blackboard. These events are maintained (and possibly ranked) until the executing KS activation is completed. At that point, the control components use the events to trigger and activate KSs. The KS activations are ranked, and the most appropriate KS activation is selected for execution. This cycle continues until the problem is solved.

Blackboard systems support a variety of control mechanisms and algorithms, so a choice of opportunistic control techniques is available to the application developer.

3 Why Use the Blackboard Problem-Solving Approach?

The blackboard model offers a powerful problem-solving architecture that is suitable in the following situations.

- Many diverse, specialized knowledge representations are needed. KSs can be developed in the most appropriate representation for the data they are to handle. For example, one KS might be most naturally written as a rule-based system while another might be written as a neural-net or fuzzy-logic routine.
- An integration framework is needed that allows for heterogeneous problem-solving representations and expertise. For example, a blackboard is an excellent framework for combining several separately established diagnostic systems.
- The development of an application involves numerous developers. The modularity and independence provided by large-grained KSs in blackboard systems allows each KS to be developed and tested separately. The software-engineering benefits of this approach apply during design, implementation, testing, and maintenance of the application.
- Uncertain knowledge or limited data inhibits absolute determination of a solution. The incremental approach of the blackboard system will still allow progress to be made.
- Multilevel reasoning or flexible, dynamic control of problem-solving activities is required in an application.

The blackboard approach has been applied in numerous areas, including the following:

- sensory interpretation
- design and layout
- process control
- planning and scheduling
- computer vision
- case-based reasoning
- knowledge-based simulation
- knowledge-based instruction
- command and control
- symbolic learning
- data fusion

In each of these applications, the scope of the problem to be solved was the prime factor in selecting a blackboard approach. That is, deciding whether to use a blackboard approach should be based on the problem-solving requirements discussed above, rather than the specific application area.

4 Why So Few Blackboard Applications?

One measure of success of a technology is its routine use in applications. In 1989, Lee Erman, one of the original Hearsay-II designers, conjectured that the lack of widespread use of blackboard technology stems from the following.

- The advantages of blackboard systems do not scale down to simple problems; they are only worth pursuing for complex applications. The history of blackboard-system use outside of research labs is too short to see many fielded applications.

- A blackboard system is useful for prototyping an application, but, once developed and understood, the application can be reimplemented without the blackboard structure or opportunistic control machinery.

Dr. Erman's first conjecture is proving valid. A number of applications are nearing completion and will be deployed within the next year.

His second conjecture has also applied to some situations. Applications have been built using a blackboard system, and in the process, the developers learned enough about how to build their application (using the blackboard model) to discover that the application did not require the capabilities of blackboard systems. (Use of a blackboard approach, however, facilitated this discovery process.) On the other hand, there are applications in which the complexity requires dynamic control of problem solving or the maintainability remains significantly enhanced by the KS modularity of blackboard systems.

I would add a few of my own reasons for the slow adoption of blackboard systems into the mainstream of AI applications:

- lack of commercial software designed specifically for building blackboard applications
- the myth that blackboard applications are too slow or too hard to develop
- a shortage of application developers with experience building blackboard applications.

These reasons are all related to the historical need to build each blackboard application, including the blackboard machinery, from scratch. This forced a researcher or developer to commit to the entire effort, if a blackboard approach was to be used. Outside of research laboratories, the level of commitment required to choose a blackboard approach rather than alternative AI techniques was simply too great, in many cases. Pressures to get a prototype blackboard application working has also led to shortcuts in implementing blackboard machinery, which hampers application performance. Finally, because relatively few blackboard applications were developed, the number of experienced developers remained small. Fortunately, the need to build everything from scratch has passed, and these factors are quickly disappearing.

Despite these historical difficulties, a number of blackboard applications have been developed in diverse problem areas since Hearsay-II. Many of them are described in the two books listed in the Additional Reading box.

One blackboard application that is in daily use is the Pontecello Burden Adviser (PBA) jointly developed by FMC and Cimflex Teknowledge. Started in 1985, this system assists the operator of a phosphorus manufacturing furnace in understanding the process's state, and it recommends actions to the operator for increasing quality and yield. Phosphorus manufacture is a dynamically unstable process, with slowly changing process states that cannot be predicted analytically. Inappropriate furnace parameter settings are hard to catch in time, and there are large delays in the response of the process to parameter changes.

Prior to PBA, manufacturing effectiveness was solely dependent on the operator's experience in diagnosing and controlling the process. Quality and yield differed markedly among different operators, and the most expert operators were constantly on call. PBA performs real-time situation assessment, simulation, diagnosis, and action planning to produce its recommendations. PBA also determines sampling rates and preprocessing parameters to obtain a level of sensory information appropriate to the current situation.

FMC has not disclosed the development costs and return on investment associated with PBA. However, the cost savings are described as significant, and they plan to use PBA at the other three furnaces at the site. Additional benefits provided by PBA include: improved training for new operators, more consistent interpretation and control of the process across multiple operators, and improved record-keeping of all recommendations and actions for future analysis.

Additional Reading

Robert S. Engelmore and Anthony Morgan, editors. *Blackboard Systems*. Addison-Wesley, 1988. A comprehensive collection of historical blackboard papers from mid-1970 into early 1987.

V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, editors. *Blackboard Architectures and Applications*. Academic Press, 1989. A cohesive collection of papers (primarily from the 1987 and 1988 AAAI Blackboard Workshops) presenting the state of blackboard-system work at that time.

The following papers highlight some recent research activity involving blackboard systems:

Norman Carver, Zarko Cvetanvic, and Victor Lesser. "Sophisticated cooperation in FA/C distributed problem-solving systems." In *Proceedings of the National Conference on Artificial Intelligence*, pages 191–198, Anaheim, California, July 1991.

Keith S. Decker, Victor R. Lesser, and Robert C. Whitehair. "Extending a blackboard architecture for approximate processing." *The Journal of Real-Time Systems*, 2(1):47–79, 1990.

Keith Decker, Alan Garvey, Marty Humphrey, and Victor Lesser. "Effects of parallelism on blackboard system scheduling." In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 15–21, Sydney, Australia, August 1991.

Barbara Hayes-Roth, Richard Washington, Rattikorn Hewett, Michael Hewett, and Adam Seiver. "Intelligent monitoring and control." In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 243–249, Detroit, Michigan, August 1989.

Susan E. Lander, Victor R. Lesser, and Margaret E. Connell. "Knowledge-based conflict resolution for cooperation among expert agents." In D. Sriram, R. Logher, and S. Fukuda, editors, *Computer-Aided Cooperative Product Development*, pages 183–198, Springer Verlag, 1991.

5 Should You Consider Using a Blackboard Approach?

Should you consider using a blackboard system for your next application? The answer is no, if:

- you can easily represent all knowledge in a framework with which you are already familiar
- the application does not need to make dynamic control decisions
- the completed application will not be combined with other systems.

Often, however, applications that initially appear simple evolve into more ambitious systems as they are better understood and as early successes trigger more ambitious requirements. Therefore, you may want to consider using a blackboard approach for a simple application if you suspect it may grow in the future (or at least keep the possibility of moving to a blackboard system in mind during initial application development).

6 Alternatives to Blackboard Systems

Blackboard systems have no equal in their combination of capabilities. There are approaches, however, that provide some of the capabilities of blackboard systems. Use of rule sets and

object-oriented, method-based inference are two examples of techniques that provide some of these capabilities.

Rule sets

Some rule-based shells provide rule sets as a means of modularizing the complexity and control of rule-based systems. In such shells, rules are partitioned into rule sets, and a rule becomes a candidate for execution only when its rule set is active.

Use of rule sets provides modularity to a large rule-based application. Because a change to a rule is isolated from other rule sets, rule sets help overcome some of the problems of unexpected rule firing. Rule sets provide some of the modularity of blackboard KSs. However, unlike KSs in a blackboard system, rule sets do not provide diverse representation and inference engines for each module, since all rules sets share a common representation and inference engine.

Decisions about which rule set should be active mirror the choice among KS executions in a blackboard system. As with deciding to execute a particular KS activation in a blackboard system, a decision to activate a particular rule set can change the problem-solving behavior for a significant period of time. Large-grained control decisions involving the activation of a rule set approximate the selection among KS executions, but the control behavior is limited to what can be achieved with the rule-based inference engine.

Method-based inference

The popularity of object-oriented languages encourages what can be called method-based inference. In this approach, problem-solving occurs in response to actions applied to objects. As is the case with rule-based and blackboard systems, method-based knowledge is applied in response to events (in this case, actions on the objects). Proponents of pure object-oriented representations argue that the object-based decomposition of knowledge is more understandable and manageable than a functional decomposition. Others argue that a mixture of object-based and functional representations are appropriate for complex applications.

The blackboard approach is neutral in this object-based versus functional viewpoint. KSs can be triggered in response to object-based events in the same manner as methods are invoked. As with object-oriented languages, whether methods are encapsulated with their associated objects is a property of the particular blackboard system.

A potential problem with the unrestricted use of method-based inference is controlling the spreading activation of methods in response to events. Method-based inference is an integral expense associated with modifying an object. By default, this expense cannot be controlled.

Many blackboard systems provide a smooth integration of method-based and KS-based inference. Where method-based computations have low cost and do not need to be controlled, normal object-oriented techniques can be used. When the propagation of the effects of modifying an object needs to be weighed against other potential actions, the same computations can be defined as a KS, which competes with other KSs for computational resources.

7 Getting Started with Blackboard Systems

The benefits of blackboard systems do not scale down to toy problems. To really get a feel for blackboard systems, you must undertake a serious application head on.

For example, implementing the monkeys and bananas problem as a blackboard system will not demonstrate any advantages over implementing it using a rule-based system. A toy blackboard application suggests that there aren't significant reasons for considering a blackboard approach, or even worse, implies that the blackboard approach is simple and that the toy approach could be used to develop a serious application. On the other hand, a trivial

blackboard system and application can illustrate some of the basic principles of blackboard systems and serve as an example of what not to do when implementing a serious application.

Accompanying this article is a listing of a trivial blackboard system and a simple application built using it. The simple blackboard (SBB) system is written in Common Lisp. It allows KSs to be implemented as Common Lisp functions. KSs can be triggered in response to the creation of particular blackboard objects (represented as structures). The blackboard is represented as a simple list of objects (a hash table would be a better choice!), and no retrieval capabilities have been provided. The control component supports only one scheduling approach: last-in, first-out.

The example SBB application is equally trivial. Three KSs are defined: one to generate integer values, one to compute the squares of generated integer values, and one to print the values of the generated squares. Obviously, a blackboard system is not needed for this application!

If you are interested in experimenting with this application:

- consider how the performance would change if the order of KS definitions is changed in the file
- rewrite the generate-integers KS to generate prime numbers instead
- rewrite the control loop to be first-in, first-out or even support a constant rating for each KS, by giving generate-integers a higher rating than generate-squares and print-squares.

Since SBB is not for serious applications, how should you go about developing a real blackboard application? There are four approaches.

Build from scratch

In the early days of blackboard systems, this was the only option. First, you would carefully read all information you could locate about blackboard systems, and then you would start coding. The early blackboard papers did not discuss implementation details, so you were left to your own programming intuition in building an effective system.

This approach is the source of the inaccurate view that blackboard systems must be either complex to develop or slow in performance. (Some of the slowest were not much more sophisticated than SBB!) Developers worrying about performance discovered that a lot of specialized machinery needed to be coded before they could start building their applications. On the other hand, developers that built a simple blackboard framework found that their applications performed poorly.

This situation mirrored the early days of rule-based systems, where every implementer wrote his own inference engine from scratch (again, typically poorly!). Today, there is rarely any rationale for building an inference engine from scratch, and the same holds for blackboard systems.

Use university research software

Beginning with Penny Nii's AGE skeletal blackboard framework that was developed at Stanford University from 1977–1982 [2], academic researchers have built tools for their own blackboard system research. Most notable of recent academic research tools are Barbara Hayes-Roth's BB1 system [3] (which can be licensed from Stanford) and my own UMass GBB framework [4] (available from the University of Massachusetts at Amherst). These systems have the advantages of low-cost (even free!) and complete source code. They have the disadvantages of limited documentation and support.

Use in-house expertise

If you work for a company with an AI laboratory, its researchers there may be building blackboard systems by using university software or they may possibility have developed an

internal blackboard tool.

Use commercial tools

The fastest way to get started with blackboard technology is to purchase blackboard tools and expertise from a commercial vendor. Just as expert system technology evolved into an industry during the mid-1980s, blackboard system technology is now becoming available commercially. A commercial tool allows you to immediately begin building your blackboard application, and a vendor that provides training and consulting services can offer assistance if you need it.

8 Generality, Flexibility, and System Engineering

The blackboard approach has been termed the most general and flexible architecture for building knowledge-based systems. In this article, we have discussed some of the basic characteristics and capabilities of blackboard systems. We have also indicated some of the freedom that blackboard systems provide developers in structuring their applications.

Generality and flexibility are a double-edged sword. Because blackboard systems leave many choices open to developers, appropriate choices have to be made for each application. Identifying appropriate KSs, determining the structure of the blackboard and the objects needed, selecting a control approach, determining control knowledge, etc., all must be determined when developing an application. For someone developing a blackboard application for the first time, these choices may be intimidating. However, for an experienced blackboard application developer, these same choices present opportunities for tailoring a high-performance approach to the problem.

For the novice developer, the flexibility of the blackboard architecture allows an incremental approach to building the application. Many decisions can be deferred until an improved understanding of blackboard systems and the characteristics of the application is gained. A small set of KSs and blackboard objects can be developed to get a prototype application running, and blackboard system tools can help speed this process. Further, by using a simple control approach, choices of the most appropriate control techniques can be deferred until the need for specific capabilities becomes apparent. Once the prototype is operational, its performance and proficiency become driving forces in enhancing the initial KSs, in developing additional KSs, and in enhancing control decision making.

The generality and flexibility of blackboard systems are also being applied and extended by researchers working in such areas as real-time AI, distributed and parallel AI, cooperating-agent systems, and mixed-paradigm systems. In many of these efforts, sophisticated control components are being developed to improve the ability of the blackboard system to control problem-solving activities. Papers describing these efforts are not as accessible as the two books on blackboard systems, but I have listed a few examples in the Additional Readings box.

AI applications are becoming increasingly more complex and ambitious, and the ability to integrate diverse techniques (expert systems, neural networks, fuzzy logic, case-based reasoning, etc.) into an intelligent whole is becoming more important. Blackboard systems provide a structure for building these applications. They are likely to be in your future.

Daniel Corkill is a Senior Research Computer Scientist in the Department of Computer and Information Science at the University of Massachusetts in Amherst. He is also a founder of Blackboard Technology Group, Inc., a firm specializing in blackboard-system software and services.

References

[1] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[2] H. Penny Nii and Nelleke Aiello. AGE (Attempt to GEneralize): A knowledge-based program for building knowledge-based programs. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 645–655, Tokyo, Japan, August 1979.

[3] Alan Garvey, Michael Hewett, M. Vaughan Johnson, Robert Schulman, and Barbara Hayes-Roth. *BB1 User Manual*. Knowledge Systems Laboratory, Departments of Medical and Computer Science, Stanford, California 94305, Common Lisp edition, October 1986. (Published as Working Paper KSL 86-61, Knowledge Systems Laboratory, Departments of Medical and Computer Science, Stanford University, Stanford, California 94305.).

[4] Daniel D. Corkill, Kevin Q. Gallagher, and Kelly E. Murray. GBB: A generic blackboard development system. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1008–1014, Philadelphia, Pennsylvania, August 1986. (Also published in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, pages 503–518, Addison-Wesley, 1988.).

A The Simple Blackboard System

```
;;;; -*- Mode:Common-Lisp; Package:user; Base:10 -*-

(in-package :user)

;;; ---------------------------------------------------------------------------
;;;
;;; The Simple Blackboard (SBB) system and a trivial application.  Although
;;; this application demonstrates basic aspects of a real blackboard
;;; application, the SBB system is *far* too simple and inefficient to be used
;;; for any serious work.
;;;
;;; Placed in the Public Domain by Daniel D. Corkill, 1991.
;;;
;;; ---------------------------------------------------------------------------

(defvar *blackboard* nil

  "*BLACKBOARD*

Contains the blackboard database, which is simply a list of blackboard objects
stored by name.  (A particularly poor choice!)")

;;; ---------------------------------------------------------------------------

(defvar *events* nil

  "*EVENTS*

Contains the events generated during each KSA execution.  The events are
buffered until execution of the KSA is completed.  Then the events are
processed by the control components.")

;;; ---------------------------------------------------------------------------

(defvar *agenda* nil

  "*AGENDA*

Contains the list of activated KSs awaiting execution.")

;;; ---------------------------------------------------------------------------

(defvar *creation-event-kss* nil

  "*CREATION-EVENT-KSS*

Contains KS specifications for all KSs that are interested in blackboard-object
creation events (which is the only type of events supported).  The
define-creation-ks macro manages this list.")

;;; ---------------------------------------------------------------------------

(defvar *trace-level* 3

  "*TRACE-LEVEL*

Contains the current trace level.  Trace levels from 0 (none) to 3 (highest)
are supported.")
```

```
;;; ---------------------------------------------------------------------------

(defstruct (ks-spec (:conc-name "KS-SPEC."))

  "KS-SPEC (Structure)

Contains the information about a KS needed by the control machinery."

  object-types      ; a list of object types of interest
  ks-function)      ; the name of the function implementing the KS

;;; ---------------------------------------------------------------------------

(defstruct (bb-object (:type list)
          (:conc-name "BB-OBJECT.")
          ;; We define our own constructor below
          (:constructor nil))

  "BB-OBJECT (List)

Contains the name and data of a blackboard object."

  name           ; the name of the object
  data)          ; the object data

;;; ---------------------------------------------------------------------------

(defun reset-bb-system ()

  "RESET-BB-SYSTEM

Resets the system to the initial state."

  (setf *blackboard* nil)
  (setf *events* nil)
  (setf *agenda* nil))

;;; ---------------------------------------------------------------------------

(defun undefine-all-kss ()

  "UNDEFINE-ALL-KSS

Removes all KS definitions."

  (setf *creation-event-kss* nil))

;;; ---------------------------------------------------------------------------

(defun signal-creation-event (bb-object)

  "SIGNAL-CREATION-EVENT bb-object

Signals that 'bb-object' has been created."

  (push `(creation-event ,(bb-object.data bb-object)) *events*))

;;; ---------------------------------------------------------------------------

(defun make-bb-object (name data)
```

```
  "MAKE-BB-OBJECT name data

Makes 'object' a blackboard object with name 'name' and signals a creation
event.  'Name' must be a symbol (a very poor choice)."

  (let ((bb-obj (list name data)))
   (when (> *trace-level* 2)
    (format t "~&~5tCreating ~a object: ~a~%" (type-of data) bb-obj))
   (push bb-obj *blackboard*)
   (signal-creation-event bb-obj)
   bb-obj))

;;; ---------------------------------------------------------------------------

(defun get-bb-object (name)

  "GET-BB-OBJECT name

A trivial means of retrieving a blackboard object by name. 'Name' must be a
symbol.  This function is not used in the application below."

  (bb-object.data (find name *blackboard* :key #'bb-object.name :test #'eq)))

;;; ---------------------------------------------------------------------------

(defun creation-event (bb-object)

  "CREATION-EVENT bb-object

Control component code for processing a creation event. Determines which KSs
are interested in the event and adds them to the agenda.  Does not evaluate
the relative importance of activated KSs."

  (let ((bb-object-type (type-of bb-object)))
   (dolist (ks-spec *creation-event-kss*)
    (when (find bb-object-type (ks-spec.object-types ks-spec))
     (let* ((ks (ks-spec.ks-function ks-spec))
            (ksa `(,ks ,bb-object)))
      (when (> *trace-level* 1)
       (format t "~&~5tActivating ~a~%" ksa))
      (push ksa *agenda*))))))

;;; ---------------------------------------------------------------------------

(defun control-loop ()

  "CONTROL-LOOP

A trivial control loop.  No opportunistic control is performed -- simply
last-in, first-out scheduling.

The loop terminates when the agenda is empty."

  (loop
   ;; process events:
   (dolist (event *events*)
    (eval event))
   (setf *events* nil)

   ;; check for stopping condition:
   (unless *agenda*
```

```
    (format t "~2&Agenda is empty. Stopping.~%")
    (return-from control-loop (values)))

   ;; run the top KSA:
   (let ((ksa (pop *agenda*)))
     (when (> *trace-level* 0)
       (format t "~&~5tRunning: ~a~%" ksa))
     ;; Note that use of eval is a very poor choice here:
     (eval ksa))))

;;; ---------------------------------------------------------------------------

(defmacro define-creation-ks (ks obj-types arglist &body body)

  "DEFINE-CREATION-KS ks obj-types arglist &body body

Defines KSs interested in creation events.  'KS' must be a symbol and is the
name to be given to the created KS function.  'Obj-types' is a list of the
types of objects for which creation events are of interest to the KS.  'Arglist'
and 'body' are as per normal Common Lisp functions."

  `(progn
     ;; remove any existing definitions:
     (setf *creation-event-kss*
         (delete ',ks *creation-event-kss* :key #'ks-spec.ks-function))

     ;; add the new definition:
     (push (make-ks-spec :object-types ',obj-types
                         :ks-function ',ks)
         *creation-event-kss*)

     ;; define the function:
     (defun ,ks ,arglist
       ,@body)))

;;; ---------------------------------------------------------------------------
;;;
;;;  A simple "blackboard" application that generates integer values,
;;;  computes their squares, and prints the squares.
;;;
;;; ---------------------------------------------------------------------------

(defparameter *stop-value* 25

  "*STOP-VALUE*

Specifies the last integer generated by the generate-integers KS.")

;;; ---------------------------------------------------------------------------

(defstruct (integer-object
          (:conc-name "INTEGER-OBJECT.")
          (:print-function
           (lambda (object stream depth)
             (declare (ignore depth))
             (let ((*print-structure* nil))
               (format stream "#<integer-object ~D>"
                     (integer-object.value object))))))

  "INTEGER-OBJECT (Structure)
```

```
A blackboard object containing a generated integer."

  value
  square)

;;; ---------------------------------------------------------------------------

(defstruct (square-object
         (:conc-name "SQUARE-OBJECT.")
         (:print-function
          (lambda (object stream depth)
            (declare (ignore depth))
            (let ((*print-structure* nil))
              (format stream "#<square-object ~D>"
                    (square-object.value object))))))

  "SQUARE-OBJECT (Structure)

A blackboard object containing a squared integer."

  value
  integer)

;;; ---------------------------------------------------------------------------
;;;
;;; The KS Definitions:
;;;
;;; Note: Because the SBB control scheme implements a simple LIFO ordering and
;;;     the KSs interested in a single type of event are activated in the
;;;     order in which they appear in the *creation-event-kss* list,
;;;     changing the order of definitions below will change the behavior of
;;;     the application.
;;;
;;; ---------------------------------------------------------------------------

(define-creation-ks compute-squares (integer-object) (bb-obj)

  "COMPUTE-SQUARES bb-obj

Defines a KS that computes the square of its 'bb-obj'
argument.

This KS is interested only in integer-object creation
events."

  (let* ((value (integer-object.value bb-obj))
       (square-obj (make-square-object :value (* value value))))
    (make-bb-object (gensym) square-obj)
    (setf (square-object.integer square-obj) bb-obj)
    (setf (integer-object.square bb-obj) square-obj)))

;;; ---------------------------------------------------------------------------

(define-creation-ks generate-integers (integer-object) (bb-obj)

  "COMPUTE-SQUARES bb-obj

Defines a KS that creates a new integer-object with a value that is 1 larger
than its 'bb-obj' argument.  Creation stops when the value exceeds *stop-value*.

This KS is interested only in integer-object creation events."
```

```
  (when (< (integer-object.value bb-obj) *stop-value*)
   (make-bb-object
    (gensym)
    (make-integer-object :value (1+ (integer-object.value bb-obj))))))

;;; ---------------------------------------------------------------------------

(define-creation-ks print-squares (square-object) (bb-obj)

  "PRINT-SQUARES bb-obj

Defines a KS that prints the value of the created square-object (contained in
the 'bb-obj' argument).

This KS is interested only in square-object creation events."

  (format t "~&** Square: ~d~%" (square-object.value bb-obj)))

;;; ---------------------------------------------------------------------------

(defun run-application ()

  "RUN-APPLICATION

The top-level application function that runs (and reruns) the simple
application."

  (reset-bb-system)
  (make-bb-object (gensym) (make-integer-object :value 1))
  (control-loop))

;;; ---------------------------------------------------------------------------
;;;  End of File
;;; ---------------------------------------------------------------------------
```