Collaborating Software

Blackboard and Multi-Agent Systems & the Future

Daniel D. Corkill Department of Computer Science University of Massachusetts Amherst, Massachusetts 01003 corkill@cs.umass.edu

ABSTRACT

For the past quarter century, AI researchers have used the paradigm of collaborating software systems to tackle large and difficult problems. Blackboard systems were the first attempt at integrating "cooperating" software modules. The goal was to achieve the flexible, brainstorming style of problem solving exhibited by a group of diverse human experts working together to address problems that no single expert could solve alone. Multi-agent systems research is revisiting the collaborating-software paradigm from an agent-centric orientation. Again the goal is to achieve effective collaboration with a group of independent software entities, but in a way that appears to be markedly different from the approach taken in blackboard systems.

In this paper, I compare and contrast these two approaches. Examining collaborating software from both perspectives provides insights into the nature of collaboration, reveals unresolved problems in integrating disparate contributions, and underscores issues in coordinating collaborative activities.

Keywords

Collaborating software, blackboard and multi-agent systems

1. COLLABORATING SOFTWARE

For the past quarter century, AI researchers have used the paradigm of *collaborating software systems* to tackle large and difficult problems. The collaborating-software paradigm is an effective divide-and-conquer approach to the development and maintenance of large and complex software applications. In this approach, a number of smaller, independently developed and maintained software modules are applied in concert to form the overall system. The individual details and complexity associated with each module is encapsulated within the module, and computations that require the capabilities of multiple modules involve collaboration among modules.

Blackboard systems were the first attempt made by AI researchers at integrating "cooperating" software modules [5, 13, 23]. The goal was to achieve the flexible, brainstorming style of problem solving exhibited by a group of diverse human experts working together to address problems

The author uses Common Lisp for the majority of his research.

that no single expert could solve alone. The resulting technology enabled applications that are among the most advanced and capable AI systems that have been developed. Multi-agent systems research is revisiting this collaboratingsoftware paradigm from an agent-centric orientation. Again the goal is to achieve effective collaboration with a group of independent software entities, but in a way that appears to be markedly different from the approach taken in blackboard systems.

Collaborating software involves the integration and coordination of relatively independent, self-contained software systems that are able to work together effectively on their own. Although the names are nearly identical, the goals and purposes of *"collaborating software"* are very different from "collaboration software," where the software is used to facilitate the interaction among human participants rather than to provide an automated environment where software—and potentially human—entities work together in order to perform complex activities. Collaborative-engineering environments, virtual conferencing facilities, whiteboarding software, etc. Examples of collaborating software include blackboard systems and some multi-agent systems.

For the past few years, I have been using "collaborating software" to delineate a broad area of study relating to the automated support of collaboration among human and/or software entities. A number of research areas address issues relevant to collaborating software (Figure 1). These areas include blackboard systems¹ and some aspects of multiagent systems, as well as programming in the large (PIL) or "megaprogramming," component-based software engineering, distributed-object systems, computer-supported cooperative work (CSCW) and collaboration environments, and research into effective human collaboration and biological societies. In this figure, the large oval indicates the space of collaborating-software research.

1.1 Collaborating-Software Challenges

There are six key challenges involved in creating effective collaborating-software systems:

- 1. **Representation** getting software modules to understand one another
- 2. Awareness making modules aware when something relevant to them occurs

This work was supported, in part, under DARPA/IXO contract MDA-972-02-C-0028. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the University of Massachusetts or the U.S. government.

¹In the field of software architectures, blackboard-style systems are called *repositories* [32, 31].



Figure 1: Collaborating-Software Research Areas

- 3. **Investigation** helping modules to quickly find information related to their current activities
- 4. **Interaction** creating modules that are able to use the concurrent work of others while working on a shared task²
- 5. **Integration** combining results produced by other modules
- 6. **Coordination** getting modules to focus their activities on the right things at the right time

How these challenges are addressed has a significant effect on application performance and scalability, and each of these collaborating-software challenges will be considered in more detail shortly.

Because blackboard systems are fully within the space of collaborating software and were, historically, the first attempt at creating collaborating software modules, we next look closely at the architectural techniques developed for blackboard systems, how they address the six collaboratingsoftware challenges, and why these techniques work so effectively.

2. BLACKBOARD SYSTEMS

In my opinion, the blackboard framework is the most general and flexible knowledge-system architecture. — Edward A. Feigenbaum^[15]

Blackboard systems were first developed in the 1970s to solve complex signal-interpretation problems in systems such as Hearsay-II [25, 14] and, shortly thereafter, HASP/SIAP [28]. Since the days of those early blackboard-system applications, the blackboard approach has been viewed as an ideal candidate for tackling difficult, ill-structured problems in a wide range of application areas. Given this reputation, it is surprising that blackboard systems are not widely understood in depth, are rarely mentioned in modern AI texts, and are even considered by some to be outdated technology (generally without mention of a replacement-technology candidate). Given this, it is important to look into the details of blackboard systems and the basis for their reputation.

2.1 What is a blackboard system?

²In many situations, concurrent interaction can be replaced with interleaved executions.



Figure 2: Connecting Modules Together

A traditional way of combining a set of diverse software modules is to connect them according to their data-flow requirements (such as the five modules shown in Figure 2(a)). When appropriate, the modules can appear multiple times in the communication graph, but the connections are predetermined and direct. This approach can work well when both the module set and the appropriate communications among modules are static. When the specific modules are subject to change and/or when the ordering of modules cannot be determined until specific data values become known at execution time, the inflexibility of direct interaction becomes unwieldy. From a system-building perspective, direct interaction promotes the use of private communication protocols between modules. Such specialized protocols can be made succinct and efficient, but changes to the communication graph or the addition of a new module can require changes to a number of individual communication protocols.

Another approach is to use *indirect* and *anonymous* communication among modules via an intermediary, such as a blackboard data repository (Figure 2(b)). In this approach, all processing paths are possible, and the choice among paths can be made dynamically by a separate "moderator" mechanism that selects among the possible paths. The information placed on the blackboard is public, available to all modules, control mechanisms, newly added modules, and monitoring and debugging tools. Indirection reduces the number of communication interfaces that must be supported among highly collaborating modules.

A blackboard system consists of three main components (Figure 3):

• **Knowledge sources** (KSs) are independent computational modules that together contain the expertise needed to solve the problem. KSs can be widely diverse in their internal representation and computational techniques and are *anonymous* in that they do not interact directly with one another or know what other specific KSs are present in the system.



Figure 3: Blackboard-System Components

of problem-solving resources. The control component is separate from the individual KSs.³

Although all blackboard systems have these three main components, their mere presence does not create the powerful framework described in Feigenbaum's quote. Let's consider each of these three main components in more detail.

2.2KSs

Blackboard systems use a functional modularization of expertise. Each KS is a specialist at solving certain aspects of the overall application and is separate and independent of all other KSs. A KS does not require 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 making changes to 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.

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 it might be able to contribute to the problem-solving process is known as a triggering condition.

At an abstract level, a blackboard system may appear to be very similar to a rule-based system: the blackboard system's blackboard and the rule-based system's working memory; the blackboard system's KSs and the rule-based system's production rules; event-based triggering of KSs and of rules; anonymous interaction of KSs and rules; and so on. Historically and operationally, however, blackboard-systems and rule-based systems are very different, especially in the size and scope of rules versus the size and complexity of KSs and in the relatively small number of large-grained control decisions that are made by a blackboard system versus the large number of fine-grained conflict-resolution decisions made by a rule-based system. With regard to knowledge granularity, KSs are substantially larger and more complex than each isomorphic rule in an expert system. While expert systems work by firing a rule in response to stimuli, a blackboard system works by executing an entire KS in response to an event. Each KS can be arbitrarily complex and internally different from one another. In particular, a single KS in a blackboard system could be implemented as a complete rule-based system [6].

KSs are not the active "agents" in a blackboard system. Instead, KS activations (historically also called KS instances) are the active entities competing for computational 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 occur that trigger the same KS. In such cases, control decisions involve choosing among 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). Taking this distinction one step further, KSs are static repositories of knowledge while KS activations are the active "agents" that are created in response to each triggering context. These KS-activation "agents" remain alive only until the KS activation is executed or is canceled prior to execution.

The blackboard 2.3

decisions

of problem solving

and the expenditure

The blackboard is a shared data 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 abstraction. Although this organization of blackboard data is often useful to the developer and user of the system, the principal reason is to make locating appropriate information on the blackboard more efficient. If the problem being solved is complex and the number of contributions placed on the blackboard becomes large, quickly locating pertinent information becomes a problem. A *KS execution*⁴ should not have to scan the entire blackboard to see if appropriate items have been placed on the blackboard by another KS execution.

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 organize information numerically, alphabetically, or by relevance. Advanced blackboard-system frameworks provide rich positional metrics for efficiently locating blackboard objects of interest [7].

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

³In some blackboard systems, the control component itself is implemented using a blackboard approach (involving control KSs and blackboard areas devoted to control).

⁴The term "KS execution" is shorthand for the execution of a KS activation."



Figure 4: Classic Blackboard-System Control Cycle

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 blackboard control component 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.

Typically, 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.⁵ 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 a KS to efficiently inspect the blackboard to see if relevant information is present.

2.4 Control component

In a blackboard system, a separate control mechanism, sometimes called the *control shell*, directs the problem-solving process by allowing KSs to respond opportunistically to changes made to the blackboard. A blackboard system uses an incremental reasoning style: the solution to the problem is built one step at a time.

In a classic blackboard-system control approach, the currently executing KS activation generates events as it makes changes on the blackboard (Figure 4). These events are main-



Figure 5: Separation (Encapsulation) of Control Knowledge

tained (and possibly ranked) until the executing KS activation is completed. At that point, the control shell uses the events to trigger and potentially activate KSs. The KS activations are ranked, and the most appropriate KS activation is selected for execution. This KS-execution cycle continues indefinitely (for continuous applications) or until the problem is solved (for single-solution-based applications).

It is important that the control component in a blackboard system is able to make its selection among pending KS activations without possessing the detailed expertise of the individual KSs. Without such a separation, the modularity and independence of KSs would be lost. If specific knowledge of all the KSs had to be included within the control shell, it would have to be modified every time a KS was added or removed from the system. On the other hand, we do not want KSs to be making autonomous control decisions—in a blackboard system, control decisions are made by the control shell.

The solution is to separate control knowledge into generic, overall control knowledge contained in the control shell and detailed KS-specific control knowledge packaged with each KS. Then, whenever the control shell needs KS-specific control information, it asks the individual KSs for these estimates on how the KS will behave. This separation of control knowledge is shown in Figure 5. When a KS is triggered, the control shell passes the triggering context to the KS, which uses its KS-specific control knowledge to estimate factors such as the quality, importance, cost, and likelihood of successfully making potential contributions. This estimate is determined without actually performing the work to compute the contributions. Instead, each KS generates estimates of the contributions that would be generated by using fast, low-cost, approximations developed by the KS writer. These estimates are of the form, "If this activation is selected for execution, I estimate it will generate contributions of this type, with these qualities, while expending these resources." The KS returns these estimates to the control shell which uses them in deciding how to proceed.

3. BLACKBOARD SYSTEMS AS COLLAB-ORATING SOFTWARE

To better understand why blackboard systems have been such effective problem-solving architectures, and provide insights into their limitations, we consider how the blackboard approach addresses each of the six key collaborating-software challenges.

3.1 Representation

The structure of information on the blackboard is at the heart of the blackboard-system approach. In principle, the blackboard representation should not be based on any specific set of KSs. Instead, the design of the blackboard representation should stem directly from the characteristics of

⁵The developers of the original Hearsay-II blackboard system recognized that complex, conjunctive trigger-condition specifications of KS interest would be inefficient as well as unsuited to highly parameterized patterns involving positioning in time and other real-valued point and interval (or region) attributes. Instead, they opted for a combination of simple triggering-condition specifications to be followed by a more detailed procedural examination of the blackboard before fully activating the KS [22]. Although this Hearsay-II design decision was made a few years before the development of specialized pattern-matching algorithms such as RETE [17], the choice remains appropriate for almost all blackboard-system applications due to the large ratio of partially completed patterns to fully completed ones.

the application and the goal of allowing any potential KS to make contributions toward a solution. In practice, however, the design of the blackboard representations are not fully divorced from a general sense of the kind of KSs that will be used in the application, and experience has demonstrated that choices made in the blackboard representation can have a major effect on system performance and complexity.

The KSs in a blackboard application must be able to correctly interpret the information recorded on the blackboard by other KSs. Additionally, the control shell may also need to understand aspects of blackboard data in order to make strategic focus-of-attention decisions. However, all aspects of blackboard data do not need to be understandable by all KSs. Many KSs only use data from one or two blackboard levels as input and only make modifications at a single blackboard level. Similarly, KSs may only operate on a few classes of blackboard objects. In a very practical sense, this characteristic means that portions of the blackboard may be relevant to only a few KSs and could be specialized to the interaction requirements among those KSs. Yet, private jargon shared by only a few KSs limits the flexibility of applying other KSs on that information in the future. In practice, there is a trade-off between the representational expressiveness of a specialized representation shared by only a few KSs and a fully general representation understood by all KSs. Determining the proper balance between a general and specialized representation is an important aspect of blackboard-application engineering.

Having KSs suggest possibilities is an important activity in blackboard-system applications (and collaborating software in general). When human problem-solvers brainstorm together, "thinking aloud" by one participant often stimulates ideas in others in a highly synergistic fashion. However, the amount of such sharing needs to be appropriate to the problem-solving context. If participants withhold ideas until they are highly confident in their veracity, potential stimulating suggestions may be lost. On the other hand, if everyone shares every harebrained idea, there will be so much distraction that progress toward a solution will be impeded [26]. When it comes to implementing the blackboard representation, consideration must be given to efficiently supporting the aggressive sharing of contributions among KSs. A highly efficient implementation can support more aggressive sharing of lower confidence (but possibly relevant) contributions.

There is a second dimension of sharing that involves the degree of detail in the contributions that are placed on the blackboard. At one extreme in this detail dimension, KSs only put on the blackboard the minimal amount of information needed to convey the results of their work to other KSs. All specific details of how the results were generated and similar processing-context information remain unshared. At the other extreme, a detailed representation of the relationship between inputs, interim partial work product, the knowledge procedures used, etc., are placed onto the blackboard along with the basic KS results. This level of detail allows other KSs (and control and explanation tools) to critique and integrate the results based on the processing activities that were performed by the KS. It is pointless and inefficient to put detailed information onto the blackboard that is never used, but not sharing information also limits possibilities. Again, determining the right level of sharing is an important engineering decision

Any architectural mechanisms that increase the cost of placing information onto the blackboard must be carefully evaluated in light of the potential performance decrease that would result from aggressive sharing. For example, high-use blackboard data is typically memory resident in high-performance blackboard applications. Using a *main memory* repository rather than a traditional database allows KS contributions to be made at memory rather than disk speed. Similar considerations apply to the persistence of blackboard data. Maintaining full persistence of highly dynamic and transient blackboard data can significantly lower blackboard-system performance. Instead, strategies for maintaining only important or non-reconstructible blackboard data can be used to reduce disk bandwidth requirements.

In the basic blackboard-system control cycle, only a single KS activation is executing at any time. This KS execution runs to completion (or termination by the control shell) before another KS execution begins. To further simplify the architecture in this basic control cycle, only the executing KS is allowed to make changes to the blackboard while it is executing. This requirement eliminates the need to incorporate complex blackboard locking or transaction mechanisms that would slow down blackboard operations [16, 4].

3.2 Awareness

In a blackboard application, KSs are triggered in response to specific types of blackboard (and external) events that indicate that the KS may be able to contribute to problem solving. Rather than having KSs continually poll the blackboard, the control shell is told about the kind of events in which each KS is interested. This is typically called *registering* the KS. The control shell maintains this triggering information and directly considers the KS for activation whenever that kind of event occurs. To be efficient, this triggering information is provided to the low-level blackboard repository accessor routines which only notify the control shell of events for which any KS is currently registered. KS triggering can be made highly efficient when the registration involves only simple, disjunctive trigger events.

3.3 Investigation

When a KS is triggered by one or more events, it must often look on the blackboard for other information that is related to these events. This search for associated data involves: 1) computing approximate attribute values for the kind of blackboard objects that are relevant to computations stemming from these triggering events, and then 2) finding those objects on the blackboard. For example, a KS that is triggered by the sudden movement of an unfriendly unit toward a friendly position might look on the blackboard for related movement of other unfriendly units that could indicate the initiation of an orchestrated threat. Units of interest would be unfriendly, within some radius of the friendly position, and may have recently changed their movements. The identity (names) of these units of interest are not known, nor are they linked to the unit whose change triggered the event or to units at the friendly position. The units of interest can only be determined by the approximate values of some of their attributes. The importance of such proximity-based associative retrieval to locate relevant objects that have been placed on the blackboard by other KSs is often overlooked in casual discussions of blackboard systems.

Most KS executions in a blackboard system involve the following steps (Figure 6):

1. the control shell is notified of an event of interest to the



Figure 6: KS Activities

KS

- 2. this triggering context is used to activate the KS
- 3. the KS uses the triggering context to determine the ranges of attribute values that are relevant to the triggering context and looks on the blackboard to see what additional blackboard objects have attributes within those ranges
- 4. the KS uses the retrieved objects and the triggering context information to perform its computations
- 5. the results of this computation are written onto the blackboard

In this sequence, step 3 is the step associated with investigation.

Associative retrieval of blackboard data is central to the blackboard paradigm. It is used to facilitate the indirect and anonymous communication among KSs by allowing them to look for relevant information on the blackboard rather than receiving the information directly from other KSs. Objects on the blackboard often have significant latency between the time they are placed on the blackboard and the time they are determined to be relevant for use by another KS. If it were not for this latency between creation and use, the blackboard could be compiled away into direct calls among KSs by a configuration-time compiler, and we would be back to the directly connected modules of Figure 2(a). This latency in blackboard objects supports the use of the blackboard as a both a short-term scratchpad and a longer-term global memory for the KSs. Objects are held on the blackboard to be used when and if they are needed by the KSs. It is support for the temporal separation of creation and use that provides blackboard systems with such flexibility in ordering KS executions. In order to obtain this same flexibility without a shared blackboard, each KS module would have to maintain its own copy of objects received from other modules. Furthermore, whether the memory is globally shared (on the blackboard) or private (within a KS), an efficient means of scanning for remembered objects is required.

3.4 Interaction

Blackboard systems *prohibit* direct interaction among modules, as all communication is done via the blackboard. Traditional blackboard systems have only a single control thread and execute only one KS activation at a time; once execution is started, the KS activation runs to completion or until it is aborted by the control shell. This means that all interaction among KS activations is serial, is unidirectional (from earlier to later executions), can have unbounded latency, and is indirect via the blackboard. This severe restriction on interaction greatly simplifies the development of blackboard applications, but in certain situations this restriction can be a significant collaborating-software limitation.

Assume that KS A and B both are interested in the same event and can both do some initial work without interacting with one another. However, the initial work of A is needed for B to complete its work and vice versa. In this situation, the blackboard-application designer must artificially split A into 2 KSs, APRE and APOST, and similarly, B is split into BPRE and BPOST. Once APRE completes, BPOST can begin and, similarly, once BPRE completes, APOST can begin. If a lot of interaction is required, this KS-splitting approach can result in a large number of artificial KS fragments. Alternatively, the same iterative form of interaction can be achieved by creating KSs that are able to jump into later computations, based on the information present on the blackboard. In this case, multiple KS executions are still required to support the serial interaction, but the number of KSs present in the system does not need to be increased.

Parallel and distributed blackboard-system extensions of the classic, single-threaded blackboard architecture allow true concurrent KS executions, and this raises another important interaction issue. If the KSs are to remain anonymous and indirect in their interaction, then all interaction must still occur via changes to the blackboard. Executing KSs must be able to notice and respond to changes made to the blackboard during their execution to support such indirect interaction.⁶ We could also extend the KS model to allow for direct communication among co-executing KS activations. However, this is a major departure from the blackboard-system model, and it is problematic because of the uncertainty about which KS activations will be executing concurrently at any moment.

3.5 Integration

Integration and representation are closely linked in blackboard-system applications. The representation choices that are made not only effect the ability of KSs to use the results of others, but also how KS results are combined. In a blackboard application, integration of results involves three major activities: relationship management, attribute merging, and value propagation.

The need for *relationship management* occurs when a KS execution wants to create a new object on the blackboard and the semantics of the blackboard representation requires that the relationship between the new object and some existing objects be represented. A simple example of this is the creation of a higher-level object as a result of identifying a set of lower-level *supporting* objects, such as creating a platoon object based on a set of individual unit objects. If this synthesis activity is performed by a single KS execution, the relationship between the new platoon object and the set of supporting unit objects can be easily represented by also creating support

⁶This is a special case of the general problem of blackboard changes occurring during KS execution.

links⁷ that connect the objects. Such support links explicitly maintain the relationship between the objects on the blackboard.

Now, let's assume that a new individual unit object is detected that should be added to the set of supporting units for the platoon object. If the "platoon synthesis" KS is registered to be triggered on the creation of individual unit objects, when it executes it might again want to create a platoon object based on the new, slightly larger set of support objects. If no platoon object had yet been created on the blackboard, the system could simply create the new object and all the support links as before. In this case, however, there already exists a platoon object that is *semantically identical* to the new platoon objects to be placed onto the blackboard. So, instead, a new support link is added between the existing platoon object and the newly created, individual unit object.

What blackboard-system component should be responsible for maintaining these relationships? One approach makes each KS responsible for this. When an executing KS wants to create a new blackboard object, it must first check to see if a semantically equivalent object already exists (which involves a blackboard retrieval). If one is found, the KS modifies the relationships of the existing object instead of creating a new object and relationships. This approach requires that each KS writer perform this check and that the semantics of equivalency are consistent across all KSs.

Another approach is to make equivalent-object checking and relationship management part of the unit-creation operation. In this case, the KS would ask to create a new platoon unit with links to the support units and the blackboard itself would perform the required bookkeeping. This latter approach begins to move the blackboard from a passive repository to a more active entity with application knowledge about what constitutes equivalency and how to handle duplicate creation requests.

The second integration activity is *attribute merging*. Using our same example, assume that the belief in a platoon object is based on the number, types, and relative positions of supporting units that are linked to it. When the second KS execution wants to generate a duplicate platoon object, the platoon's belief attribute needs to be changed in addition to the link attributes. As with relationship management, we can have the KS execution determine the new belief value or we can have the blackboard object-creation routines do it automatically. The problem with the latter approach is that the knowledge required by the blackboard grows with the complexity of determining merged values. We certainly do not want to end up duplicating much of the knowledge used by KSs in computing new blackboard objects in automatic blackboard-integration routines.

Belief values are a special attribute that we will revisit shortly when we consider principled integration. However, there can be many other attribute values that may differ among objects that are considered equivalent. Perhaps the location of our platoon object does not have to be precisely identical for the old and new objects to be considered equivalent. The addition of the new supporting unit may result in a slightly more accurate estimate of the overall position attribute used to represent the location of the platoon. Now when the old and new platoon objects are merged, the location attribute also needs to be changed to reflect the improved position estimate. Clearly, as the number of object attributes that need to be appropriately merged grows, the complexity of work required by either every KS execution or the shared blackboard-representation maintenance routines also grows.

The third integration activity is value propagation. Assume that the belief associated with the platoon object is a function of the beliefs of its supporting units and their spatial locations relative to one another. Assume a field report is received that contains a confirmed sighting of one of the supporting units of the platoon and that the KS execution that processes this information increases the belief value of the supporting unit. We would like this increased belief value to propagate to the platoon value, increasing our belief in it as well. Again, we could make this propagation be the responsibility of the executing KS or an activity of a more active blackboard repository. Similarly, suppose yet another KS execution, using different sensor data than was available to earlier KS executions, is able to compute a more accurate position for one of the support units and changes the position attribute of that support unit. We would like this new position value to be used to update the position attribute of the platoon unit and, potentially, the platoon unit's belief value if the new location of the supporting unit affects the belief calculation.

Historically, blackboard systems have handled these integration activities in a very ad hoc manner. Some applications placed the responsibility for these activities with the executing KSs. This required substantial discipline on the part of KS writers to maintain semantic consistency across KSs. Other applications placed this responsibility with the blackboard, risking duplication of KS knowledge and the potential for inconsistency if the way that the KS performed its activities was changed significantly. By careful modularization and sharing of code among KSs and the "active" blackboard, it is possible to reduce this duplication and risk. Finally, some applications dealt with value propagation by simply triggering and executing KSs again if important attributes used in their contributions changed. In this case, a re-executed KS simply replaced its original contributions with the latest version. Each of these approaches worked well enough in specific situations and, when used with care, allowed complex blackboard-system applications to be built.

Recent growth in the use of principled graphical models, such as Bayesian networks, in AI systems [29, 30, 34, 33, 20] have made the ad hoc confidence and belief values that have been used in most blackboard applications seem particularly unprincipled. These ad hoc belief values were involved in everything from making control decisions to determining solutions and the system's confidence in them. How to best incorporate more principled integration of KS results remains an open blackboard-system research issue, particularly in relation to the amount of sharing discussed earlier.

The main motivation for moving to a principled representation of blackboard data is to make the integration of disparate KS results well founded. This can only be achieved by creating accurate models of how these results are generated and relate to one another. Note that such models of result integration are <u>not</u> identical to probabilistic graphical models of the underlying domain. This is a subtle, but important, distinction—models used for integrating KS results reflect properties of the collaborating-software system and its behavior rather than only properties of the problem domain. For example, conditional-independence assumptions in

⁷A link is a bidirectional pointer between two objects.



Figure 7: Separation (Encapsulation) of Integration Knowledge

application-level models that might be used within KSs to determine their results stem from domain properties: the nodes and the relationships among them relate to understanding of the domain in which the application is operating. Conditional independence in graphical result-integration models, however, are based on knowledge of how the results produced by a KS were obtained: the specific data pedigrees and algorithms used by the KS. If two KSs trigger on the same event and produce similar results using different computational approaches, how independent are the results? Are the two results redundant⁸ (with no added certainty in the results) or are the two techniques complementary (in the sense that each has the potential to make mistakes on certain data values, but these mistakes are fully independent of one another)? In the latter case, the integration model needs to reflect the additional certainty produced by the corroborating KS execution.

A principled domain model can be used to determine the effect that obtaining particular observations will have on the current assessment of the problem. Such power-ofinformation reasoning is very useful in making control decisions among KSs that can produce various observations. Similarly, a principled result-integration model is also useful in making control decisions, as it can be used to determine the potential of a KS increasing the confidence in an attribute value. If both models are used together, principled powerof-information and power-of-reasoning control decisions can be made.

Creating a principled integration model for a blackboard application requires close analysis of how the KSs in the application operate in conjunction with one another. Yet, such an analysis is antithetical with the independent and anonymous KS philosophy of blackboard systems, as any changes in the set of KSs used in the application are likely to require changes in the principled result-integration model. The model may even need to be changed if the computations used in a KS are modified. This close coupling is not surprising, as integrating KS results really requires understanding their pedigree. These issues have always been present in blackboard systems, they have only been hidden in the use of ad hoc integration techniques or ignored in favor of higher order issues. From a practical standpoint, however, how we maintain the consistency of the result-integration model with the current KS set is an important issue. Just as KS-specific control expertise is developed and maintained with each KS, it is important to develop KS-specific models of result generation that can be incorporated into an overall result-integration model when the KS is added to the system (Figure 7). Such a capability remains to be developed, but it is an important research goal in enabling principled result integration in applications, that will have many KS changes throughout their lifetimes.

Finally, the degree that results are shared in a blackboard

application has a direct relation to the complexity of the result-integration models. The integration model need only address results produced by KSs that are placed onto the blackboard, so there is a tension between limited sharing (which reduces the size of the integration model) and aggressive sharing (with a complex integration model). Principled result integration adds yet another consideration to degreeof-sharing design decisions.

3.6 Coordination

The last collaborating-software challenge is running the right KSs on the right data at the right time. The *opportunistic* control that is the hallmark of blackboard systems is highly flexible, responsive, and generally efficient. During each control cycle, a traditional blackboard system makes a single, instantaneous choice of the best KS activation to execute and, if new conditions warrant, the system can focus its attention on a new area as early as the next cycle.⁹ As discussed earlier, executing only one KS activation at a time also greatly simplifies the architecture. Nevertheless, even achieving effective single-threaded control in a complex blackboard application can be challenging.

At any given moment, a blackboard application rarely lacks choices among a large number of potential KS activations to execute. These choices stem from multiple inputs arriving into the system, combinatoric ways in which this data can be combined and used, and, in many applications, multiple KSs that can be applied to the same data (Figure 8). This results in a large and dynamic space of possible KS executions, of which only a small fraction can be pursued. Because blackboard systems operate incrementally, poor choices early on can result in triggering a large number of inappropriate downstream KS executions in response to the results generated by a single "inappropriate" KS execution. Agenda-based control uses a utility-based rating computed for each KS activation to select the best activation to execute in each cycle. This rating incorporates the estimates of what the activation will do if executed (from the KS-specific control knowledge) and more global requirements, such as parts of the solution that need attention (from global control information maintained by the control shell). Unselected activations remain on the agenda, potentially to be executed in the future.

While the activations that are queued on the agenda await execution, the state of the blackboard and of overall problem solving is being changed by other KS executions. This results in a *queue-latency problem* where the information associated with the KS activation becomes inconsistent with the current situation (Figure 9). One naive solution to this problem is to re-rate all KS activations on every cycle. However, since the number of pending KS activations can become large, this is not an efficient solution, particularly if the re-rating of each KS activiation involves searching the blackboard for changes relevant to the activation. Blackboard systems using this approach have needed to artificially limit the number of activations held pending or to re-rate only the topmost activa-

⁸Certainly reflecting a control problem.

⁹Refocusing can be made even more responsive by allowing the control shell to abort the executing KS activation in response to critical events. Although it is possible to merely suspend the KS execution, the potential changes to the blackboard that might occur while the KS is suspended can make resumption difficult. In many cases, rerunning the KS execution—if it is still relevant—is preferable to resumption.



Although developed independently, KSs have implicit dataflow relationships among themselves that determine the amount of potential concurrency (control choice) available in the KS structure. A linearly-ordered ("pipelined") KS structure can only support data concurrency (a). If the ordering contains incomparable KSs, additional concurrency is possible among the incomparable KSs (b). Typically, the KS structure is dependent upon the blackboard data and may contain multiple minimal and maximal elements (c). In such structures a number of KS paths are possible, and not all paths need be initiated or completed.

Figure 8: Linearly- and Partially-ordered KS Structures

tions, under the assumption that the other ratings would not have changed too drastically. Other systems have organized the agenda in much the same way as the blackboard, so that the control shell could quickly identify pending KS activations that might be affected by changes on the blackboard. Eventbased re-triggering of pending KS activations is an example of this strategy.

Sometimes blackboard changes make execution of the KS activation unnecessary. Such obviation events result in removal of the activation from the agenda.¹⁰ Obviation can be performed on every cycle, either by scanning each pending KS activation or by the use of obviation events, or it can be deferred until each KS activation is about to be executed. In the latter case, a revalidation step is performed in conjunction with selecting the highest rated KS activation from the agenda. Revalidation can update information that has been maintained with the activation and, if appropriate, instruct the control shell to obviate execution of the activation and select the next KS activation from the agenda. Delaying obviation in favor of revalidation can be useful if the cost of dealing with obviation on each cycle is high but the cost of maintaining undetected obviated KS activations (re-rating, etc.) does not significantly add to the cost of maintaining the agenda.

In addition to the queue-latency problem, simple agendabased control techniques can introduce unwelcomed depthfirst bias to opportunistic control. Consider the agenda shown in Figure 10. KS activations of A and B have the same rating



Figure 9: The Queue-Latency Problem

with C close behind. From a control standpoint, these can be considered equally valid choices to be executed next. If A is selected and executed, its results may trigger a number of other KS activations, such as X, Y, and Z, potentially at higher ratings than B and C. If B had been selected instead of A, it might have triggered X' and Y', again with ratings much higher than A and C and potentially even higher than the ratings of X, Y, and Z. To be fair, and to make our control decisions as informed as possible, we should execute A, B, and C before executing any of the KS activations that are triggered by them.



This is one simple example of some of the problems that result from making instantaneous, history and purpose free, control decisions, and this problem was observed in the original Hearsay-II blackboard system [22]. A number of advanced blackboard-system control techniques were developed during the 1980s and early 1990s that provided improvements in the ability to reason about alternative control decisions, maintain continuity of purpose and adherence to plans, and modify control strategies in

response to application conditions. Some of these efforts include: integrating data-directed and goal-directed control [9, 27], control planning [21, 1], uncertainty-reduction planning [3], and design-to-time scheduling [18, 10]. Many of these efforts are surveyed in [2].

4. MULTI-AGENT SYSTEMS

Multi-agent systems (MAS) are another technology source that could be used to build collaborating-software applications. Rather than use a blackboard system as the basis for combining KSs, we could make each KS an agent in a MAS (Figure 11(a)). In many respects, having each KS be an agent seems closer to the "cooperating experts" style of problem solving that gave rise to blackboard systems. In contrast with blackboard systems, MAS research has emphasized:

¹⁰Some blackboard implementations simply delete obviated KS activations, others retain them in an obviated KSA list in case the conditions change to make them desirable again [21]. In practice, it is generally more efficient to create a new KS activation in response to changing conditions than to maintain obviated activations for reuse.



(c) With a Blackboard & Control (Manager) Agent



(b) With a Blackboard Agent



(d) Full-Fledged Blackboard Agents



- distribution (no central data repository)
- autonomy (local control)
- interaction (communication and representation)
- coordination (achieving coherence in local control decisions)
- organization (emergent organizational behavior)

The majority of MAS research has targeted homogeneous agent systems and loosely coupled heterogeneous systems with multiple providers of functionality. Outside of agentbased design applications [24], there has been limited research with closely collaborating, functional agents—our "brainstorming experts" analogy.

The interacting expert agents in Figure 11(a) do not have a blackboard in their meeting room! Without a blackboard, each agent must not only decide what results to share, but what other agents to share them with (either via direct communication with other agents or broadcast to everyone). Similarly, each agent must decide what results it has received from other agents—and what results it has produced locally should be remembered for later use. If agents need to remember very much, some sort of repository (a private "blackboard") will be needed at each agent. Finally, each agent must decide what it should be doing, using only its local view of problem solving activities. We could simplify things for the agents by providing them with simple rules for what and who they should communicate with, but if these become too static our desired flexibility in brainstorming interactions becomes the directly connected interaction structure we saw in Figure 2(a).

Perhaps there is a reason that every meeting room has a whiteboard in it... A meeting room without a whiteboard (or similar technology) is just not as effective. We can add a special "blackboard" agent to our set of agents to provide a similar capability (Figure 11(b)). Now our agents can communicate directly with one another or interact indirectly via the blackboard agent. In fact, we can eliminate the need for each agent to maintain its own repository of information by having the blackboard agent provide this service to the group. Such functional centralization would have been harshly criticized by early MAS researchers, but the elimination of redundant work by centralization is evident in federated multiagent architectures, where specialized agents serve as matchmakers, facilitators, or brokers to eliminate the need for all agents to perform these activities [11, 19, 12, 35]. To further simplify communication decisions by our agents, we can



Figure 12: Collaborating-Software Design Space

require all agent communication to be with the blackboard agent. To ensure that agents are notified when useful information is given to the blackboard agent, we will also have our agents register their desires with the blackboard agent. When such information arrives, the blackboard agent will notify the appropriate agents of its arrival.

At this point, our agents still have to make local control decisions about what they should be doing. However, much of the information needed to make these decisions is now located at the blackboard agent. We can have our agents delegate their local control decisions to the blackboard agent, which now also becomes a manager agent that tells the other agents what to do (Figure 11(c)). While co-locating the control decision making with the data required for these decisions makes sense, the resulting MAS is very divorced from the strong emphasis on local autonomy of traditional MASs. In fact, a closer look at Figure 11(c) shows that we have essentially implemented the KS interface in a traditional blackboard system using MAS technology!

5. THE FUTURE

Traditional multi-agent and blackboard systems can be viewed as two diverse points in the collaborating-software design space (Figure 12). Traditional blackboard-system research has concentrated on closely collaborating problemsolving techniques with a single thread of activity operating in a centralized setting. The KS activations in a blackboard system can be considered as one-shot, stateless agents that are created in response to an activation event and remain alive only until they have been executed. MAS research has concentrated on long-lived agents that collaborate concurrently in a distributed environment. Although the highperformance, closely collaborating capabilities of blackboard systems is an important aspect of collaborating-software applications, distribution and concurrency requirements will also require the use of technologies from other parts of the collaborating-software design space. What will be important is using the appropriate technology in the right context.

In general, MAS technologies support loosely coupled, distributed agents. Therefore using a MAS framework as the

core foundation for closely collaborating software is not an appropriate choice. High-performance blackboard applications rely on fast interactions between the blackboard repository, KSs, and control components. Adding agent-interaction costs at this level would be a serious mistake. Nevertheless, MAS technologies have their place with large-scale, distributed applications. Consider Figure 11(d) in which entire blackboard systems have been made into agents. This is a simple example of an architecture that provides flexibility in grouping closely interacting entities together in an agent-based environment. If each blackboard system in Figure 11(d) had only a single KS and only one of the systems served as the shared blackboard, then Figure 11(d) is essentially the same as Figure 11(c). On the other hand, if all agents have all the KSs but they only focus on nearby data in a distributed setting, then Figure 11(d) becomes a homogeneous distributed application. By placing only closely interacting KSs in the same agent, this same architecture supports a more function-based organization [8].

A quarter-century of blackboard-system experience and more than a decade of MAS development have produced a strong baseline of collaborating-software technologies. Yet, much more research remains in developing highperformance, generic collaborating-software capabilities that span the entire design space of Figure 12 and in developing the accompanying expertise for matching design choices to application settings. Further advances in meeting the six collaboration challenges (representation, awareness, investigation, interaction, integration, and coordination) are needed to enable the next generation of complex, collaborative-software applications.

6. **REFERENCES**

- [1] N. Carver and V. Lesser. A new framework for sensor interpretation: Planning to resolve sources of uncertainty. In *Proceedings of the National Conference on Artificial Intelligence*, pages 724–731, Anaheim, California, July 1991.
- [2] N. Carver and V. Lesser. The evolution of blackboard control architectures. Expert Systems with Applications, special issue on the blackboard paradigm and its application, 7(1):1–30, Jan. 1994.
- [3] N. Carver and V. Lesser. The DRESUN testbed for research in FA/C distributed situation assessment: Extensions to the model of external evidence. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95), pages 33–40, Menlo Park, California, June 1995. AAAI Press.
- [4] D. D. Corkill. Design alternatives for parallel and distributed blackboard systems. In V. Jagannathan, R. Dodhiawala, and L. S. Baum, editors, *Blackboard Architectures and Applications*, pages 99–136. Academic Press, 1989.
- [5] D. D. Corkill. Blackboard systems. *AI Expert*, 6(9):40– 47, Sept. 1991.
- [6] D. D. Corkill. Embedable problem-solving architectures: A study of integrating OPS5 with UMass GBB. *IEEE Transactions on Knowledge and Data Engineering*, 3(1):18–24, Mar. 1991.
- [7] D. D. Corkill, K. Q. Gallagher, and P. M. Johnson. Achieving flexibility, efficiency, and generality in blackboard architectures. In *Proceedings of the National*

Conference on Artificial Intelligence, pages 18–23, Seattle, Washington, July 1987. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 451–456, Morgan Kaufmann, 1988.)

- [8] D. D. Corkill and S. E. Lander. Agent organizations. Object Magazine, 8(4):41–47, Apr. 1998. (An extended version of this article was published as "Diversity in Agent Organizations," technical report, Blackboard Technology, Amherst, Massachusetts, 1998.)
- [9] D. D. Corkill, V. R. Lesser, and E. Hudlická. Unifying data-directed and goal-directed control: An example and experiments. In *Proceedings of the National Conference on Artificial Intelligence*, pages 143–147, Pittsburgh, Pennsylvania, Aug. 1982.
- [10] D. D. Corkill, Z. B. Rubinstein, S. E. Lander, and V. R. Lesser. Live-representation process management. In 5th International Conference on Enterprise Information Systems, volume 8, pages 202–208, Angers, France, Apr. 2003.
- [11] M. R. Cutkosky, R. S. Engelmore, R. E. Fikes, M. R. Genesereth, T. R. Gruber, W. S. Mark, J. M. Tenenbaum, and J. C. Weber. PACT: An experiment in integrating concurrent engineering systems. *Computer*, 26(1):28–38, Jan. 1993.
- [12] K. Decker, K. Sycara, and M. Williamson. Middleagents for the Internet. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, Aug. 1997.
- [13] R. S. Engelmore and A. Morgan, editors. Blackboard Systems. Addison-Wesley, 1988.
- [14] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.
- [15] E. A. Feigenbaum. Book foreword. In R. S. Engelmore and A. Morgan, editors, *Blackboard Systems*, pages v–viii. Addison-Wesley, 1988.
- [16] R. D. Fennell and V. R. Lesser. Parallelism in Artificial Intelligence problem solving: A case study of Hearsay II. *IEEE Transactions on Computers*, C-26(2):98–111, Feb. 1977. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 106-119, Morgan Kaufmann, 1988.)
- [17] C. L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, 19:17–37, 1982.
- [18] A. Garvey and V. R. Lesser. Design-to-time scheduling and anytime algorithms. *SIGART Bulletin*, 7(2):16–19, 1996.
- [19] M. R. Genesereth and S. P. Ketchpel. Software agents. Communications of the ACM, Special Issue on Intelligent Agents, 37(7):48–53, July 1994.
- [20] Glymour and Cooper, editors. *Computation, Causation and Discovery*. MIT Press, 1999.
- [21] B. Hayes-Roth. A blackboard architecture for control. Artificial Intelligence, 26(3):251–321, July 1985. (Also published in Readings in Distributed Artificial Intelligence, Alan H. Bond and Les Gasser, editors, pages 503–540, Morgan Kaufmann, 1988.)
- [22] F. Hayes-Roth and V. R. Lesser. Focus of attention in the Hearsay-II system. In *Proceedings of the Fifth*

International Joint Conference on Artificial Intelligence, pages 27–35, Cambridge, Massachusetts, Aug. 1977.

- [23] V. Jagannathan, R. Dodhiawala, and L. S. Baum, editors. *Blackboard Architectures and Applications*. Academic Press, 1989.
- [24] S. E. Lander. Issues in multiagent design systems. IEEE Expert: Intelligent Systems and their Applications, 12(2):18–26, March-April 1997.
- [25] V. R. Lesser and L. D. Erman. A retrospective view of the Hearsay-II architecture. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 790–800, Cambridge, Massachusetts, Aug. 1977.
- [26] V. R. Lesser and L. D. Erman. Distributed interpretation: A model and experiment. *IEEE Transactions on Computers*, C-29(12):1144–1163, Dec. 1980. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 120–139, Morgan Kaufmann, 1988.)
- [27] V. R. Lesser, R. C. Whitehair, D. D. Corkill, and J. A. Hernandez. Goal relationships and their use in a blackboard architecture. In V. Jagannathan, R. Dodhiawala, and L. S. Baum, editors, *Blackboard Architectures and Applications*, pages 9–26. Academic Press, 1989.
- [28] H. P. Nii, E. A. Feigenbaum, J. J. Anton, and A. J. Rockmore. Signal-to-symbol transformation: HASP/SIAP case study. *AI Magazine*, 3(2):23–35, Spring 1982.
- [29] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 1988.
- [30] J. Pearl. *Causality: Models, Reasoning and Inference.* Cambridge University Press, 2000.
- [31] M. Shaw and P. Clements. A field guide to boxology: Preliminary classication of architectural styles for software systems. In COMPSAC '97 International Computer Software and Applications Conference, pages 6–13, Aug. 1997.
- [32] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
- [33] B. Shipley. *Cause and Correlation in Biology*. Cambridge University Press, 2000.
- [34] Spirtes, Glymour, and Scheines. *Causation, Prediction and Search*. MIT Press, 2nd edition, 2001.
- [35] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. *Special Joint Issue of Autonomous Agents and MAS*, 7(1 and 2), July 2003.