Research report 98

THE BLACKBOARD ARCHITECTURE: A DEFINITION AND ITS IMPLICATIONS

Iain D Craig

(RR98)

Abstract

The blackboard architecture for problem solving has been considered as a more-or-less informal construct for a considerable period of time: this paper attempts to rectify the matter by presenting a more rigorous definition. Due to the controversy surrounding the status and role of the concept of opportunism in blackboard systems, an argument is presented to show that it is not a fundamental property of the architecture — it merely results from the memory-based control component of implementations. Finally, the issues of modularity and generality of the architecture are addressed. The findings, here, to suggest that the architecture suffers from severe problems.

Department of Computer Science
University of Warwick
Coventry
CV4 7AL, UK

March 1987

Iain D. Craig

Department of Computer Science,

University of Warwick,

Coventry CV4 7AL, UK

1. Introduction

In this paper, I will attempt a more or less rigourous definition of the blackboard architecture. Such an exercise is quite obviously required, for there is no adequate definition of what constitutes a blackboard system in the publically available literature, although there is a brief published definition by Nii (Nii, 1986a) and there is a Stanford Technical Report by Hayes-Roth (Hayes-Roth, 1983) devoted to the subject. The comments by Nii are very brief and, perhaps, rather biased towards the demands of signal processing applications; the Hayes-Roth paper is not generally available, although it discusses the architecture in considerably more detail than (Nii, 1986a). The definition proposed by Nii suffers from some significant problems, as I will argue below.

The blackboard architecture is considered to be a general model for problem solving. As such, it cannot be described or defined purely in terms dependent upon one particular class of problem or upon one psychological theory. In this paper, I will attempt a definition which is completely independent of application domain needs. It will be seen (and argued) that some implementations of the blackboard architecture violate some of the constraints which I will require; others will add features that I do not see as being central. As Barbara Hayes-Roth notes (Hayes-Roth, 1983, p. 2), the blackboard architecture is, basically, an informal construct which is frequently adapted to the needs of a specific application or to the needs of a particular psychological theory. Without a standard against which to compare systems and models, there is very little sense in which one can present a system as being a blackboard system, nor is there much meaningfully to be gained in comparing and contrasting other problem solving architectures with the blackboard architecture.

The definition proposed by Barbara Hayes-Roth is very similar to the one I shall present here. The definition presented here differs in one major respect from the one proposed by Penny Nii. That difference is the emphasis which Nii places on opportunism. Nii cites opportunism as a defining characteristic of blackboard systems. Indeed, she claims it as *the* fundamental property of the architecture. I do not agree with this and may be viewed as almost taking exception to this position. The reason for this difference must rest on the view one has of the architecture: my purpose, here, is to consider the blackboard architecture as a general model of problem solving. This amounts to a view that the issues must reside in the theoretical properties of an architecture and its adequacy as an explanatory device. Nii, on the other hand, is very much more concerned with Knowledge Engineering, and seems to view the blackboard architecture as a device with which to construct complex applications in domains which require considerable power from the problem solver. Although theoretical considerations are important in Knowledge Engineering, the focus in that enterprise is the construction of working systems which perform to specification. This, of course, does not suggest that I am solely concerned with Cognitive Modelling: until a series of experiments along the lines of those reported by Anderson in e.g., (Anderson, 1983), have been conducted, it will not be possible to argue for or against the blackboard architecture in terms of its adequacy as a theory within which to account for high-level cognition.

The structure of this paper is as follows. I will begin by giving a a general definition of the blackboard architecture. This definition will serve for the remainder of the paper, although it might be seen to exclude some systems (notably CHRYSALIS (Terry, 1983)). During the presentation of the definition, I will contrast blackboard systems with other architectures such as production systems: this will be done by pointing out those features of blackboard systems which are radically different from other architectures. I will then move on to the question of opportunism: Nii cites opportunism as being one of the key defining features of the architecture. I will argue against the position taken by Nii and will argue:

(i) that opportunism is only *one* aspect of the control behaviour and that it is not crucial in blackboard systems;

(ii) that opportunism has dangerous consequences, and

(iii) that opportunism is really not one but many related control phenomena.

The discussion of opportunism is one of the novel aspects of this paper. From this discussion of opportunism, I will consider the generality of the blackboard architecture. Again, there is little discussion of methodology in the literature, so I hope to make some small rectification of this fault by grounding some arguments in methodological considerations. The discussion of generality will indicate some substantial problems with the blackboard architecture. These problems are concerned with the separation of knowledge into neatly encapsulated chunks and the relationships between them. The discussion will also indicate that the complexity of the blackboard architecture can sometimes be overwhelming, a fact which seems to suggest limitations to the claim that blackboard is a *general* model of problem solving. This fault is already evidenced, I believe, by the fact that one would not normally consider the blackboard architecture for some classes of problem (although if one's focus is cognitive modelling, one might consider something similar -- this point will be extended below).

1.1. Acknowledgements

This paper is the result of many long conversations with Ted Briscoe and Peter Harper on the general features of the blackboard architecture. David Wilson suggested that I write a paper along these lines and acted as a sounding board while I developed the approach to opportunism presented in section three.

2. Basic Definition

In this section, I will present the basic definition of the blackboard architecture. Readers familiar with the blackboard architecture will see that it is a fairly conventional definition which concentrates on the usual features.

For my purposes, the blackboard architecture has four defining elements:

(i) entries, which are intermediate results generated during problem solving;

(ii) knowledge sources, which are independent, event-driven, processes which produce entries;

(iii) the blackboard, which is a structured global database which mediates knowledge source interactions and organises entries; and

(iv) an intelligent control mechanism which decides if and when particular knowledge sources should generate entries and record them on the blackboard.

These elements interact to produce a problem solving style which is "incremental and opportunistic" (Hayes-Roth, 1983, p. 4) in character. Partial solution islands emerge in different structural partitions of the blackboard, one entry at a time (in a serial implementation). New islands appear and existing ones grow as a result of the addition of entries. Eventually, mutually supportive partial solutions merge to form a complete solution.

I will describe each of the architectural elements in more detail.

2.1. Entries

Entries are intermediate results generated during problem solving. Entries may include both elements of the problem solution and information considered important for generating solution elements. Depending upon the problem domain, entries may include perceptions, goals, judgements, beliefs, hypotheses, decisions, expectations and so on. In HEARSAY-II (Erman, 1975), the domain was continuous speech understanding, so its entries represented hypothetical interpretations of the speech signal. For OPM (Hayes-Roth, 1979), the task was to generate an errand plan, so its entries were decisions about the entries and the organisation of the plan. For PROTEAN (Hayes-Roth, 1984; Hayes-Roth, 1985a), the task is the determination of protein structure: its entries represent hypothesised atom configurations and constraints between hypotheses.*

Entries may contain representations of relationships to any other structures found useful or beneficial by the user. The relationships (frequently referred to as *links*) connect entries together into solution islands. In HEARSAY-II, the hypothesis "The first word in the utterance is 'get'" might have a *supports* relationship to another hypothesis, "The first phrase in the utterance is 'get me'". In OPM, the decision "From Stanford, travel east on University Boulevard" may have an *elaborates* relationship to another decision "Go from Stanford to The Good Earth". PROTEAN entries are related by relationships expressing constraints between structural entities: an entry on the Secondary-Anchor level has constraints indicating the position

* The development of PROTEAN continues as I write. This accounts for the tense change over the last three sentences: HEARSAY-II and OPM are completed projects.

of the secondary anchor (part of a protein structure) relative to atoms in other pieces of protein secondary structure.

Entries are represented as structures of arbitrary complexity. They are frequently represented as sets of attribute/value pairs which describe an entry's semantic content and its relationships to other entries. Sometimes, entry history (in terms of the time it was created and by whom, and its subsequent modifications) is recorded (as it is in, for example, (Craig, 1986)), as well as other information considered useful by the user.

To make the typical structure of an entry clear, I now give an example entry from the blackboard control model (Hayes-Roth, 1985b). The entry is generated by the PROTEAN system (Hayes-Roth, 1984) and resides on the Focus level of the control blackboard. The entry is first described in tabular English form and then in symbolic form (the latter was printed from a runtime blackboard display). I will then explain some of the details of the entry so that the reader may gain a better understanding of the role of the entry and the range and type of information it contains.

Name	"Focus" + number
Goal	Predicate or function to rate potential actions (result in range 0 - 100)
Criterion	Predicate to evaluate an expiration condition
Weight	Goal importance (value in range 0 - 1)
Rationale	Reason for Goal
Creator	Action that created this decision
Source	Information that triggered the Creator
Type	Role in the control plan, e.g., "Strategic", "Solution-Based"
Status	Function in control plan, e.g., "Operative", "Inoperative"
First-Cycle	First operative cycle number
Last-Cycle	Last operative cycle number

The entry actually appears as the following. This will give the reader an idea of the genuine structure of entries on the PROTEAN control blackboard.

```
Name        Focus1
Goal        (Eq KS-Type 'Anchor)
Criterion   (for Element in ($Level 'Secondary-Anchor)
                    always
                    ($Find-One ((Level-Is 'Secondary)(Copies Element))))
Weight      0.8
Rationale   "Develop a comprehensive set of partial solutions
             before deciding which ones to refine at the blob level"
Creator     Chosen-Action5
Source      Strategy1
Type        Strategic
Status      Operative
First-Cycle 6
Last-Cycle  20
```

The interpretation of Focus1 is as follows.

The Goal of this entry, Focus1, contains a predicate which finds all triggered anchoring KSs: i.e., KSs which perform an anchoring operation -- anchoring, in PROTEAN, is the process of attaching one piece of protein structure to another via the establishment of constraints between the structures. Focus1's Criterion says that a mapping should be established between entries on the domain blackboard levels Secondary-Anchor and Secondary (Secondary-Anchor is the highest abstraction level; Secondary comes immediately below it): the mapping is establishes a correspondence between Secondary-Anchor entries and Secondary entries such that the lower entry is a copy of the higher (in simpler terms, the entry at the Secondary-Anchor level has been dicovered to be a piece of secondary structure which has been incorporated into a partial solution). When such a correspondence has been derived, Focus1 may expire -- that is, change its Status to "Inoperative": it will no longer take part in control reasoning. Focus1's Rationale requires that partial solutions generated at the Secondary level should be preferred for refinement down to blob and atomic levels: effectively, this requires that the partial solutions chosen for refinement should be properly constrained. The Weight of Focus1 is used to determine the priority assigned to KSARs considered for execution during problem solving cycles 6 through 20: the Weight is used by the blackboard control model in determining priority. During cycles 6 through 20, the control regime partially defined by Focus1 is Operative and Focus1 was created as a result of a control decision at the Strategy level of the control blackboard. The Creator was the fifth KSAR to be executed during the execution of the system.*

* The information in the Creator, Source and Type slots is explained only with reference to the blackboard control model. This is an example of how information contained in entries depends, for its interpretation, on a global understanding of the system, its architecture and the details of the application domain. A similar point can be made about the slots which refer to the PROTEAN problem solving blackboard.

Despite the implementation specificity and detail of this example, it can be seen that entries contain large quantities of information. Focus1 contains information about the kind of decision it represents (the Goal and Criterion slots), about its history (Creator and Source slots) and about its role in the solution process (Rationale and Weight). Focus1 also contains relationships to other entries: in particular, the Creator slot refers to a Chosen-Action (an entry recording the KSAR selected for execution on any cycle) and to a Strategy decision (the value Strategy1 in the Source slot). This example entry also shows the variety of information which may legally be contained by entries: the attribute values range from single symbols to pieces of implementation-language code. The interpretations of these items is performed by Knowledge Sources which trigger on entries and by users of the system (the Rationale value is used during explanation, for example).

2.2. Knowledge Sources

Knowldge Sources are the processes which generate or modify blackboard entries. Hayes-Roth describes them as "cognitive processes". Knowledge Sources contain the procedural representation of knowledge employed by a system to solve problems. They also contain information which assists the system's control component in making decisions about which KS to execute.

The structure of a KS is usually described as being in two parts: a condition and an action. The condition describes the circumstances under which a Knowledge Source can contribute to the problem solving process. Usually, the condition requires the existence of previously generated entries and can be viewed as a predicate on those entries. The action of a Knowledge Source generates new entries to be posted on the blackboard or else modifies the contents of existing entries. The two-part structure or condition-action format corresponds to the two stages of KS operation:

(i) triggering, during which a KS's condition is evaluated; and

(ii) execution, during which a KS's action is performed.

Knowledge Sources are basically event-driven because only those KSs whose conditions have evaluated to *true* can execute their actions.

Knowledge Sources represent independent knowledge-possessing entities. They are permitted to

interact *only* by the generation or modification of entries: that is, communication between KSs must always be via the blackboard and not, say, via additional tables or via shared variables. KSs influence each other indirectly whenever the execution of a KS's action generates or modifies entries which satisfy or partially the condition of another Knowledge Source.

These points are illustrated by the HEARSAY-II KS, MOW, and the OPM Knowledge Source, FIRST-LEG.

MOW can be described as follows.

KS-NAME: MOW
CONDITION:
There are previously generated syllable hypotheses
ACTION
Hypothesise words that comprise sequential subsets
of the syllable hypothesis. Adjust their beginning
and end times. Rate their credibilities.

MOW is invoked when other HEARSAY-II KSs hypothesise sequences of syllables which are likely to be present in the speech signal. When executed, MOW first hypothesises all words in its vocabulary which contain subsequences of the hypothesised syllables. Because of noise in the speech signal and because of ambiguity in the interpretation of syllables, MOW usually hypothesises more than one word for any given syllable sequence. MOW then adjusts the beginning and end times in the speech signal of each hypothesised word and rates each word's credibility against the syllable data.

FIRST-LEG can be summarised as follows.

KS-NAME: FIRST-LEG
CONDITION:
Two tasks, t1 and t2, have been planned in temporal
order t1, t2.
ACTION:
Plan the first leg of the route t1-t2 as:
start = t1.
leg1 = the direction-on-street from t1 which minimises
the angular disparity from the straight line connecting
t1 and t2.
end-of-leg1 = the first intersection encountered on leg1.

FIRST-LEG is invoked when other OPM KSs decide to perform two tasks, t1 and t2, in the temporal order <t1, t2>. When executed, FIRST-LEG establishes t1 as the starting point for the route from t1 to t2. The KS then enumerates the paths starting from t1 in all directions on all streets in the town. This set of

paths usually includes at least two paths in each of two directions on t1's street; it will include additional paths if t1 is on a street intersection. FIRST-LEG determines which of these paths minimises the angular disparity from the straight line connecting t1 and t2** and plans that path as the first leg of the route. It then identifies the first street intersection encountered on the planned route and makes that intersection the end of the first leg (this is because another choice will usually have to be made to connect the intersection to t2).

These examples illustrate the general point that Knowledge Sources typically transform entries at one level of abstraction into entries at another level. Some KSs, MOW is an example, operate in a basically bottom-up manner. Bottom-up KSs aggregate several lower-level entries into a smaller number of higher-level entries: MOW aggregates syllables into words. FIRST-LEG exemplifies top-down KSs. Top-down KSs expand higher-level entries into a larger number of lower-level entries: FIRST-LEG expands task sequences into route segments. Other Knowledge Sources operate within a single level of the blackboard (for example, KSs which attempt to find a temporal ordering over entries representing plan segments), or between different blackboard panels (OPM contains many such KSs, as does CHRYSALIS). The blackboard architecture can combine Knowledge Sources embodying different general inference mechanisms in a single problem solving system. It should be noted that by 'general inference mechanisms', I mean *only* characterisations of the type just given: I do not intend characterisations such as model-driven, data-driven, and so on. These properly belong to more *intensional* descriptions of the architecture in its application to particular problems: the definition I am attempting here is intended to be *extensional*.

The representation of Knowledge Sources is basically that of condition-action pairs. The condition component requires that there be at least one previously generated entry with particular attributes and values and that the evaluation of the condition yield exactly one of *true* or *false*. (One may assume that there is some process external to the blackboard system which creates and posts one or more entries to start problem solving.) The action component generates one or more new entries or modifies existing entries already on the blackboard: the action component may also combine new entry generation with existing

** OPM represents the town as a two-dimensional coordinate array. Each object in the town (building, park, etc.) is represented as a coordinate in the array. Street are represented as a sequence of points, each point representing the streets which intersect the given street.

entry modification. Beyond these specifications, KSs may be viewed as 'black boxes' of arbitrary internal complexity which are applied to user-determined partial solution representations.

2.3. Blackboard

The blackboard is a global database which contains all entries generated by all Knowledge Sources applied during the problem solving process. The blackboard serves two purposes. Firstly, it mediates all Knowledge Source interactions in the following sense. Although Knowledge Sources may not communicate directly with each other, they influence each other indirectly by placing and responding to entries on the blackboard. An entry posted by the execution of one Knowledge Source may satisfy the condition of another KS. Second, the blackboard organises and collects all partial and complete solutions generated for the solution of the problem under attack. The solutions on the blackboard comprise configurations of related entries on the blackboard. The contents of the blackboard are at least notionally inspectable by any KS: the blackboard makes entries public.

The blackboard may have a user-determined internal structure which defines important relationships between entries.* The relationships might be, for example, generalisation, aggregation, support, or temporal sequencing. The relationships may be combined in various ways. The blackboard focusses Knowledge Source activity. It is usual for a Knowledge Source condition to refer to previously generated entries in a particular region of the blackboard, while its action may alter another blackboard region. A Knowledge Source need not consider entries in regions of the blackboard which are of no concern to its condition or which are not mentioned in its action. Viewed this way, the blackboard is a framework for organising, interrogating and generating entries.

The blackboard may have additional structure: in fact, it may have any additional structure desired by the user. There are, however, two dimensions of particular importance to the structure of the blackboard: these are the vertical and horizontal dimensions. The vertical dimension separates and distinguishes entries at different levels of abstraction. Entries at a given abstraction level may elaborate or support entries at the next level or at some other level in the hierarchy. The horizontal dimension represents intervals in

* It is a matter of fact that most, if not all, blackboard implementations impose internal blackboard structure.

the solution. Horizontal intervals may represent temporal, spatial, conceptual or other groupings: indeed, in some cases, the horizontal dimension may be extended to cover many interacting groupings. It is usual for the horizontal dimension to represent some form of temporal ordering on the solution process. Together, these dimensions define an aggregation structure (hierarchy).

The structure of the OPM blackboard is thus:

Outcomes
Designs
Procedures
Operations

The vertical dimension represents tentative decision types collected into an abstraction hierarchy. The abstraction relation may be described as generating a *refines-to* relationship between adjacent abstraction levels. The horizontal dimension of OPM's blackboard represents time points in the emerging plan. Each level of abstraction offers a different specification of the plan. The highest level represents the outcomes the plan should achieve; the designs level represents designs for the general spatial/temporal layout of the plan (e.g., do tasks in the S.W. corner of the town); procedures represent the sequencing of individual tasks (e.g., pick up the medicine for the dog before buying a newspaper); operations represent the performance of individual tasks and make inter-task transitions. Operations are aggregated into procedures, procedures into designs and designs into outcomes. Because the errand planning problem requires the linearisation of plan components with time, the blackboard structure distinguishes temporal intervals within the plan.

2.4. Control Mechanism

During problem solving, many different Knowledge Sources may be triggered simultaneously by entries on the blackboard. In a serial implementation of the blackboard architecture, only one Knowledge Source may execute at any one time (in a problem solving cycle, that is). In parallel implementations, more than one KS may execute. In both cases, an intelligent control mechanism determines which of the currently triggered KSs to execute at any time. It might be thought that parallel implementations could permit execution of all triggered Knowledge Sources, but this is, however, not the case: I will be discussing this when I examine the issue of opportunism in the next section.

In HEARSAY-II, OPM, BB1 (Hayes-Roth, 1985b, Hayes-Roth, 1986), HEARSAY-III (Balzer,

1980; Erman, 1981) and NBB (Craig, 1987), control is operationalised as an agenda-based scheduler. On each problem solving cycle, the agenda lists all pending Knowledge Sources (that is, KSs whose conditions are satisfied by entries or entry configurations on the blackboard). The scheduler rates these pending KSs and selects one (or more, in the case of HEARSAY-III) for execution. Control can be operationalised in other ways: for example, HASP/SIAP (Feignebaum, 1982) represents control in terms of stack-like event lists which are used by a more-or-less algorithmic control mechanism; CHRYSALIS (Terry, 1983) makes control decisions in terms of hierarchically organised control rules and an operational semantics very similar to procedure call. The fundamental differences between the various operationalisations of the control mechanism are none too important for it can be shown (e.g., (Hayes-Roth, 1985b)) that that the agenda-based mechanism can be used to implement other mechanisms.*

The control mechanism's level of intelligence reflects its understanding of the system's Knowledge Sources and their applicability to particular aspects of the the current problem and its understanding of the development of a solution. The scheduler can exhibit a range of behaviours based upon its level of understanding. At one extreme, the scheduler may adopt one, well-defined, strategy. It might, for example, adopt a strategy to proceed top-down through the abstraction levels, executing KSs which are known to produce entries at the next abstraction level lower in the hierarchy from the current one. Alternatively, the scheduler can integrate multiple strategies: for example, a mixed strategy might execute both bottom-up and top-down KSs, preferring that strategy which prescribes the smaller number of triggered Knowledge Sources -- that is, preferring the strategy which more effectively reduces the search space. The scheduler can also deviate from the prescriptions of the current strategy, adopting arbitrary criteria: that is, behaving in an opportunistic fashion. Some opportunistic strategies (it is quite possible to have opportunistic *strategies:* indeed, more than one may be active at any time) might prescribe the execution of KSs which have recently triggered or might prescribe the execution of KSs known to generate particularly important entries. Intelligent schedulers are able also to alter the control strategies adopted as a dynamic response to progress in problem solving. It is a characteristic of the blackboard architecture that there is a capacity for opportunistic behaviour (a stronger claim is that opportunistic behaviour is, somehow, pre-disposed, but I intend

* There are reasons other than those mentioned by Hayes-Roth in (Hayes-Roth, 1985b) for prefering agenda-based scheduling. The reasons are made clear when I discuss opportunism in the next section.

to argue against this stronger claim as being based only upon empirical and not upon conceptual issues).

A clear example of multiple strategy scheduling is implemented by the HEARSAY-II scheduler. The HEARSAY-II scheduler operates a two-phase strategy. In the first phase, a bottom-up strategy is employed. Signal parameters are progressively aggregated into lexical items. It begins by scheduling all KSs which have triggered on the Parameter level, then schedules all KSs triggering on the Segment level. Next it schedules all Knowledge Sources which have triggered on the Syllable level and which generate word hypotheses on the Word or Lexical level of the blackboard.

During the second phase of HEARSAY-II's activity, the scheduler engages in a more opportunistic mode of operation. On each cycle, the scheduler calculates a priority for each pending KSAR. A KSAR is a Knowledge Source Activation Record: KSARs record information about the conditions which caused a KS to trigger. KSARs are the unit of control in HEARSAY-II and other blackboard systems. The scheduler rates each KSAR against three criteria:

(i) the local effects of the proposed activity. This includes the nature of the hypotheses which would be generated, the cumulative credibility of the generated hypotheses and of their predecessors;

(ii) the global effects of the activity in terms of co-operative and competitive relationships between the hypotheses it would generate and the existing hypotheses on the blackboard; and,

(iii) the computational resources which would be required by the proposed activity.

On each cycle, the scheduler executes the KSAR with the highest priority.

2.5. Summary

The blackboard architecture can be summarised as a list of eleven points. These points are similar to those cited by Hayes-Roth (Hayes-Roth, 1983).

1. Problem solving activity generates a set of intermediate results which are represented as objects with attributes and values. The objects are called entries.

2. Entries may have user-specified relationships with other entries.

3. All entries have the relational attributes: abstracts/refines and adjacent-to. These attributes define the vertical and horizontal structure of the blackboard.

4. All entries are recorded in a global database called the blackboard.

5. The blackboard structure includes partitions for different levels of abstraction and solution intervals.

6. The blackboard may have additional, user-specified, structure.

7. Independent knowledge-representing processes, called Knowledge Sources, generate, modify and record entries on the blackboard.

8. Each Knowledge Source has a condition and an action. The condition matches a hypothetical configuration of entries on the blackboard, performs computation and is a predicate. The action performs computation and generates blackboard modifications.

9. Only triggered Knowledge Sources (those KSs whose conditions evaluate to *true*) can be executed.

10. An intelligent scheduler determines which triggered KS(s) should execute their actions.

11. The scheduler can base its decisions on user-determined criteria such as the characteristics of the triggered KS, the utility of the proposed action, information about the general blackboard state, characteristics of the problem or information about previous control decisions.

It is generally believed that the criteria listed above exhaustively define the blackboard architecture. It is possible to extend the properties of a system and still correctly refer to it as a blackboard system. For example, the BB1 (Hayes-Roth, 1984; Hayes-Roth, 1985b; Hayes-Roth, 1986) and NBB (Craig, 1987) systems introduce extra KS components. The two major introductions are obviation conditions and a distinction between trigger and precondition predicates.

Obviation conditions are designed to be an additional relevance check: if any obviation condition evaluates to *true*, the KS is not permitted to execute. Obviation conditions are permitted to examine any part of the blackboard state (any configuration of entries in the simplest case) in an attempt to preserve consistency and to ensure that the execution of the KS will genuinely contribute to the solution of the problem currently under attack.

The division of the KS condition into a trigger and a precondition is designed to separate the event-based triggering mechanism from the state-based matching processes typical of many KS conditions. According to the definition stated above, KSs trigger as the result of changes to the blackboard state. However, there are often circumstances in which the condition should also examine the blackboard state to determine if the KS should execute. KS trigger conditions are intended to respond to events and preconditions to blackboard states: they represent a finessing of Knowledge Source semantics.

Although these two features are not included in the definition of the blackboard architecture, their presence in a system does not preclude that system from being a blackboard system. These additions are implied by the definition given above and in no way contradict the terms of that definition.

The definition given above clearly differentiates the blackboard architecture from other problem solving models. Let us consider the production system architecture for comparison.

Production rules (Newell, 1973) are frequently assumed to be like black boxes for they represent small pieces of information and have internal behaviours (albeit very simple internal behaviours) which are not open to inspection by an observer. Production rules are often explained as generating working memory items represented as attribute/value pairs (OPS5 is a case in point -- (Forgy, 1977)). Working memory serves the production architecture in the same way that the blackboard serves its architecture: it represents a single, global, database through which all inter-production communication should pass (indeed, Rychener (Rychener, 1978) describes IPS's working memory *as* a blackboard). The contents of working memory are generated or updated by the execution (firing) of production rule actions, and firing may only take place when the situation-fluent (condition part) of a production has been satisfied.

These properties are common between the blackboard and production rule architectures: this fact is a result of the structure required for pattern-directed knowledge processing -- both architectures fall into this class, of course. It is also a result of the psychological theories from which they are both derived. In both architectures, there is a procedural representation of Long Term Memory (LTM), and LTM elements operate on a declarative representation of Short Term Memory (STM). In the blackboard architecture, STM is represented by the blackboard; in the production rule architecture, working memory represents STM.

The differences between the architecture really become clear when one considers the structure of the Short Term Memory representation and the model of control each presupposes. In the blackboard architecture, STM is represented as a structured database; in the production rule architecture, STM is a collection of items with no external structure imposed. The blackboard architecture states that STM elements have arbitrary internal structure and that elements are related to each other in various ways. That is, the blackboard architecture imposes different structuring and organisational constraints upon entries. The production rule architecture goes no further than to require working memory to contain intermediate results. When considering control, it is apperent that the blackboard architecture makes many more assumptions than the production rule architecture. It is also clear that blackboard systems can exhibit far greater ranges of behaviour than can production systems. Control in production systems is usually based upon syntactic notions of match and of situation-fluent length: it can also base its notion of control on the concept of counting the number of rule firings or of counting the number of rule action components executed per cycle. The blackboard architecture, on the other hand, permits very much more sophisticated and *semantically* or epistemologically based control regimes. This additional control capability extends the functionality of the blackboard architecture beyond that of the production rule architecture and it allows blackboard systems to make decisions in a far more reasoned and rational manner than can production systems -- for one thing, the scheduling mechanisms of a blackboard system can take a *global* view of solution development.

These points serve, I believe, to distinguish the blackboard from all other architectures. They also indicate, I think, that the blackboard is very much more powerful than other architectures. This is not to suggest that I believe that the blackboard is the *ultimate* answer, for, as I will argue, there are severe problems with it and these problems are not just those cited by Anderson (Anderson, 1983, p.130) which are concerned with the computational and memory loads imposed by blackboard systems.* It is to these wider issues that I now turn.

---

* In any case, hardware costs are dropping so what is computationally expensive today is cheap tomorrow. Anderson does not make his argument too clear in this respect: is he talking about absolute measures or measures relative to the computer processing powe he has available?

3. Opportunism

It is often said that opportunism is the central feature which distingushes the blackboard architecture from all others. The claim is that opportunism is really "what blackboard is all about" for no other architecture accomodates it as readily as the blackboard. In this section, I will argue that the concentration on opportunism is a fundamental misconception of the architecture and that suggestions that the blackboard architecture should be considered for some problems "because they are opportunistic" is based on too superficial an analysis of the assumptions and consequences of the architecture. The main proponent of the 'blackboard = opportunism' movement is Penny Nii, so I will be spending most of my time refuting her arguments.

Why examine opportunism in detail? The reason is that I wish to establish a view of the blackboard architecture which puts opportunism in its correct place -- as one control regime among many.

In order to examine Nii's account of opportunism, I will concentrate on the remarks she makes in the first part of her survey papers (Nii, 1986a; Nii, 1986b). I will present two quotations taken from (Nii, 1986a), both on page 39. The first quotation appears on the top right of the page:

> "In an *opportunistic reasoning model*, pieces of knowledge are applied either backward or forward at the most "opportune" time. Put another way, the central issue of problem solving deals with the question "What pieces of knowledge should be applied when and how?" A problem-solving model provides a conceptual framework for organizing knowledge and a strategy for applying that knowledge. ... The blackboard model of problem solving is a highly structured special case of opportunistic problem solving. In addition to opportunistic reasoning as a knowledge-application strategy, the blackboard model prescribes the organization of the domain knowledge and all the input and intermediate and partial solutions needed to solve the problem."

(Italics and spelling as in original.) This quotation appears at the end of a definition of the term 'problem solving model' which includes descriptions of forward and backward chaining rule-based reasoning models.

The second quotation is taken from the definition of the control component of the blackboard model (page 39, bottom right):

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

(Bold face as in original.) One reaction to this is to say 'of course' and to say no more, for it says nothing. Quite clearly, KSs have to respond to entry configurations on the blackboard. Opportunism can be viewed as responding to beneficial items and events in one's environment. What is at issue, really, is the use of the word "respond": it suggests and is implied by control processes of one form or another.

The contrasting view is exemplified by Hayes-Roth (Hayes-Roth, 1983, p. 9):

"The capacity for and disposition toward opportunistic behavior are characteristic of the blackboard architecture."

(Spelling as in original.)

The view espoused by Hayes-Roth and by me is that opportunism is typical of the architecture and the architecture invites opportunistic control: this contrasts strongly with Nii's claim that opportunism is a fundamental property of the architecture. The alternative view must be argued for. This I shall do on two fronts: I will look very carefully at Nii's statements, then I will examine the consequences of opportunism. Along the way, I will be saying a few things about opportunism itself (it appears to be another 'informal construct' in the blackboard theory): in particular, I want to say what sort of thing opportunism is.

The first thing which must be said about Nii's remarks is that they do not add up to much. Her definition of an opportunistic model is one that applies relevant knowledge at the 'most "opportune"' time, yet she gives no criteria upon which to determine what is meant by "opportune". Secondly, she contrasts opportunistic models with other models and gives no clear differences. Purely on the basis of her remarks, it is possible to derive a counter-argument which reduces the first quotation to meaninglessness. In addition, there is no strong definition of what is meant by 'applying knowledge' -- this, too, needs to be tightened. It needs to be tightened because the import of Nii's whole argument is that the basic *control model* employed by the blackboard architecture is fundamentally opportunistic in nature and all other control regimes are, in some sense, additional and *de trop*. The question of control is a vexed one for Nii, for on the same page (page 39), in footnote six, she states quite clearly:

"There is no control component specified in the blackboard model. The model merely specifies a

general problem-solving behavior."

The import of her remarks, when taken together with footnote six, suggest that opportunism is the framework within which control components should be specified. This, however, she does not justify, nor does she make clear the relationship between opportunistic control and other regimes.

Let us now examine the definition of opportunistic control and see where it leads. Let us state, for the purposes of clarity, the definition of knowledge application as follows: in the blackboard model, knowledge is applied whenever the action of a Knowledge Source is executed to cause changes to the blackboard. This is, I assume, the definition which Nii has in mind. With this definition, it is possible to expand the definition of opportunistic problem solving as follows.

> In an opportunistic problem solving model, such as the blackboard is claimed to be, Knowledge Source actions are executed at the most "opportune" time. A Knowledge Source's action may only execute when its condition has been satisfied by the blackboard state (by the definition of KS triggering: this is accepted by Nii). Thus, after triggering a KS on the blackboard, its action may be executed sometime later but at the most opportune time.

This elaboration seems to be reasonable and seems to be in line with what Nii claims. It still contains the use of the word "opportune" and it is precisely that word to which I object.

Now, for contrast, consider the case of a forward chaining production rule interpreter. Production rules have their situation-fluents matched on each cycle of the intepreter and those rules whose situation-fluents have been satisfied enter the conflict set. When a conflict set has been created, a conflict resolution procedure selects one or more productions to fire, causing changes to working memory. In the case of a production rule interpreter, there is usually no choice but to execute one or more productions immediately, for there is not usually a concept of retaining triggered rules between cycles. If triggered KSs are retained in a control database between interpreter cycles and are retained for consideration for execution across a number of blackboard interpreter cycles, all is well and one can see the sense in the distinction Nii is attempting to draw. The very fact that KS instantiations (triggered KSs) are retained allows for the possibility that they may be executed later when certain conditions obtain which do not obtain at the time of triggering.

Even with this distinction, I do not see that Nii has achieved very much. The distinction between the so-called opportunistic blackboard and the non-opportunistic production system seems to rest on the facts that production rule interpreters do not retain instantiated rules in the conflict set between cycles (at the end of a cycle, all those rules not fired are discarded and the conflict set is created afresh) and that production rule interpreters do not normally imply problem solving strategies at the interpreter level.

The distinction becomes very much finer when one considers the application of a problem solving strategy. Consider the case of a problem solving system which applies a strategy to derive solutions in an efficient manner and assume that the problem solver represents its operators as condition-action pairs (neither of these is, I think, a particularly strong assumption). Now, during the application of its strategy, the problem solver examines all those operators whose conditions have been satisfied by the current solution state. From this set of operator instantiations it is permitted to select one or more for application. It selects those instantiations it considers will best further the solution and which are most in keeping with the strategy. If there is a choice to be made between operator instantiations, the problem solver will opt for that operator it considers will have the best chance of achieving the goal state (or sub-goal state, clearly) and apply that to its representation of the problem state.

It might be objected that the deletion of nodes (states) from the search space causes problems. The argument rests only upon an abstract characterisation of the problem solver. In abstract characterisations of search processes, the concept of deletion need not appear because effort is always concentrated on expanding the most promising portions of the state. Since the state only changes at one place at a time, it is now possible for our problem solver to retain operator instances across cycles, and it now becomes possible for the problem solver to apply operator instances at a time very much later than the time at which the condition was matched. We could, in fact, supply the problem solver with an infinite number of operators, one for each possible state in the search space (this is really what adding variables to operators provides), but the argument would remain the same. For example, the problem solver could apply an operator, find it useless, then move to another portion of the search space and apply an operator instance it had created sometime before. Since the action of applying an operator to one portion of the seach space has no global effect, the applicability of other operators in the space extended by the application of an arbitrary operator does

not alter. Even if the assumption about pruning is relaxed (for implementation purposes, say) and pruning is permitted, the assumption of the locality of the effects of an action assures us that previously generated operator instances which refer to other parts of the search space are still consistent with the remainder of the state.

This hypothetical problem solver saves operator instantiations and applies them at a later date. When an operator has been shown to lead to a poorly valued state (say, one with greater differences from the goal state than other states), the problem solver is able to apply another operator instantiation from the store it has created on previous cycles. This, I claim, is identical to Nii's definition of an opportunistic problem solving model. The hypothetical problem solver applies knowledge when it is "opportune": that is, when it is forced to go back on the result of applying an operator instantiation. Since the first operator has failed to generate a useful or better state, the problem solver is forced to apply another operator in an attempt to rectify matters.

If this is all that is meant by Nii's definition, she is not saying anything interesting either about opportunism or about the blackboard model.

Here is a very simple reason why this should be so. The hypothetical problem solver I have described could be behaving in the way it does with respect to operator instantiation caching because of an implementation trick. It is conceivable that problem solving systems will store operator instantiations for future use because of complications with the task at hand or with matching problems or because the state is only ever extended by the application of an operator. Under these circumstances, it might actually be cheaper to employ some indexing mechanism to relate operator instantiations with states in the solution space. Would one actually aggrandise a hack with the portentous title "opportunism"? I doubt it.

This argument destroys, I think, Nii's point by showing that she says nothing of content. Does this mean that the blackboard architecture has *nothing* to do with opportunism? If one accepts a better definition, there is a sense in which the blackboard should adopt opportunistic behaviour, but I am also claiming that opportunism is not the whole story.

Rather than concentrate on asking 'what time is meant by "opportune"?', I will suggest an interpretation of opportunism which is very much closer to that used by the Hayes-Roths in describing the behaviour

of subjects during the experiments which led to the development of OPM (Hayes-Roth, 1979). The suggested interpretation has the benefit that it tends to coincide rather better with what is understood by 'opportune' when it is used in ordinary language.

Opportunism can be construed as making use of beneficial accidents: that is, making the best possible use of what appears in the environment by accident or of that which has been overlooked. In other words, opportunism is concerned with using things which happen to appear rather than appearing by design. This amounts to claiming that it is a species of control strategy. In control terms, one can say that the effect of applying one or more control strategies may produce a state in which there is little or no benefit to be derived from continuing with the strategy or strategies and in which the accumulated effects provide a better foundation for making progress toward a solution than does the continued application of the strategy. The Hayes-Roths state (Hayes-Roth, 1979)*:

> "However, we assume that people's planning activity is largely *opportunistic*. That is, at each point in the process, the planner's current decisions and observations suggest various opportunities for plan development. The planner's subsequent decisions follow up on selected opportunities"

In control terms, one can take this to being the statement that control strategies generate situations in which there are various opportunities for control decisions. One opportunity is to continue with the current strategy (assuming there is only one); another is to examine the results of applying that strategy, or of other previously followed strategies, to see if they present an alternative way to make progress. The alternative definition of opportunism is that it is simply a class of control strategy.

This view of opportunism clearly rests upon the concept of control strategies and their effect upon the solution state. This definition or interpretation improves upon Nii's by removing the ill-defined term to which I objected above. Instead, it relies upon very much better defined terms. The only question is what constitutes an 'opportunity', but that appears to be covered by the statement that the control model may make choices: an opportunity in this case is the exercise of choice at any point in time where decisions of that kind are permitted. The exercise of choice depends upon there being alternatives. By assumption it is

---

* As far as I am aware, this is the first time the term 'opportunism' appears in the literature.

the case that, while following a control strategy, there will be opportunities to make decisions which are not in accordance with the prescription of the strategy -- for one thing, any strategy will exclude certain states or will ignore some alternative decisions.

This alternative approach to opportunism, which I consider to be the *correct* approach, has many benefits over Nii's. Nii's approach* does not explain how opportunism meshes with other control strategies (are they, for example, *specialisations* of opportunism?): this approach does. Nii's approach turns opportunism into a quasi-magical consequence of the architecture: this approach does not and, furthermore, it gives an account of how opportunism enters into problem solving within the blackboard model. This approach tells us much more about the phenomenon as well as putting it into its proper place -- the phenomenon is best regarded as *strategic opportunism* and that tells us, quite clearly, what sort of thing it is. Finally, my approach provides a framework within which to make predictions about the appearance of opportunistic behaviour in a blackboard system: Nii's does not, for it tends to suggest that controlled behaviour is a deviation from an opportunistic norm, a hypothesis which is not supported by evidence (Ernst, 1969; Anderson, 1983; Clancey, 1985).

Taking the last point at face value, it is possible to see the consequence of accepting the position that opportunism is the fundamental behaviour in blackboard systems. Opportunism depends, according to Nii, on finding the right time to do something. This, in the limit, entails either than nothing is done, or that everything is done immediately. The consequence of this is that there is no control. Again, there is more than ample evidence to suggest that intelligent behaviour depends to a great extent upon the presence and exploitation of control mechanisms.

Let us now move on to a prediction about the behaviour of blackboard systems. This prediction takes us along the road to a fuller discussion of the model and the architecture.

From the definition of opportunism introduced above, it is possible to see where and when systems will adopt this control strategy. A blackboard system will become strategically opportunistic when there is no other applicable strategy, or when the prescriptions of all current strategies are inadequate to cope with the current control requirements of the problem under attack or when the current strategies do not produce

* By the argument above, I may no longer consider her remarks as constituting a *definition*.

optimal values for scheduling parameters in the current state of the problem solving process. That is, one way in which opportunism appears is that the current state happens to present alternatives control criteria which are locally better than those provided by all active pre-defined strategies. One way to see this as happening is to consider that many control strategies are based upon the examination of only a few of the possible parameters which might affect problem solving behaviour: indeed, many strategies are heuristic in nature. Viewed this way, it is not surprising that strategies define regions of an abstract control space in which they are at their most effective. One could draw power curves for control strategies to discover their effectiveness, very much in the way that Lenat draws power curves for heuristics (Lenat, 1982).* Where there are multiple strategies, the theory predicts that opportunistic behaviour will appear between the peaks of strategies and the points at which their successors begin to have a real effect -- i.e., filling in the gaps between strategies.

This account of the appearance of opportunism in blackboard systems with explicit control strategies turns opportunism into an alternative strategy which comes into operation when other strategies are no longer optimal. It also appears because the control mechanism may be unable to detect the sub-optimal performance of the current strategy, and so is unable to adopt an alternative. This makes opportunism into a sort of filling-in process: it applies when other strategies are not performing or cannot apply. Given the fact that opportunism is a class of strategy, it is possible to define regions of a problem's control space which are to be treated opportunistically (HEARSAY-II is a good example of this). In addition, therefore, to allowing opportunism as a kind of default behaviour, it can be exploited as a genuine strategy class in its own right and explicitly included in the control plan for a given problem.

Is opportunism an essential property of the blackboard model? I think, on balance, that my answer is in the negative. What I believe is that opportunism has naturally been exhibited in blackboard systems because they deal with realistically complex control strategies. Why should it be that opportunism and blackboard are so closely related? The answer is, I think, a result of the fact that control mechanisms in blackboard systems have been provided with memories to record past situations. In HEARSAY-II, BB1

* Hans Berliner has, I remember, also described heuristics in terms of graphs. Specifically, he talks about the effectiveness of heuristic in terms of n-dimensional surfaces. This allows him to consider the power and utility of heuristics in terms of discontinuities in the surface. Unfortunately, I cannot remember any references to this work, nor have I been able simply to find references.

and NBB, control is mediated by an agenda of KSARs: KSARs record triggering information and encapsulate the blackboard state at triggering time. In HASP/SIAP, the control database is implemented as a set of event lists in which are recorded KS instantiations and the events which caused their creation. The fact that KS instantiations are recorded means that part of the developing solution on the blackboard is available for inspection by the control mechanism (in the blackboard control model (Hayes-Roth, 1985), the scheduled KSs are recorded, so a greater portion of the state is available), which entails that it is able to keep track of possibly beneficial deviations from the current strategy. Although not actually required by the model, this implementation of the control component has gained favour because it provides the control mechanism with a database encapsulating the state: the controller does not have to search the blackboard to make decisions.*** The very fact that the blackboard state is retained makes it easier to determine the appropriateness of deviations from any current strategy.

The last paragraph suggests what I wanted to say from the beginning. Here, then, is a summary of this section. Strategic opportunism is uniquely in evidence in the blackboard architecture not because of any architectural dictates, not because of anything in the theory. It is a side-effect of the way in which the original metaphor has been construed for implementation. Strategic opportunism is clearly the correct interpretation of the opportunism phenomenon because of its explanatory power and because it does not impute magical properties to the architecture.

Because opportunism is a class of strategy, just as top-down is a class, it is possible explicitly to include opportunistic control in blackboard systems and that inclusion is eased by the presence of the control database so frequently used in implementations. Opportunism also appears because of gaps in the applicability of other strategies, so it can also seem to be a species of default behaviour and it is this I think Nii has latched on to. Opportunism has become an issue because of implementations and not as a necessary theoretical entailment of the architecture. Opportunism is, I would argue, an important ingredient in control (and this seems to be reflected in studies of higher cognitive processes in humans -- e.g., (Hayes-Roth, 1979)): people sometimes behave opportunistically, so it is reasonable to make the option of following rigid strategies (sub-goaling, for example), or of behaving opportunistically. One of the arguments for

*** This idea of the agenda *encapsulating state* is the implementation recommendation I promised above.

using the blackboard architecture is that it is very flexible: opportunism is one way in which flexibility is introduced.

To conclude: although opportunism is not a necessary component of a blackboard system, it is often found to be of utility because of the flexibility it engenders.

4. Modularity

One of the claims made for the blackboard architecture is that it is highly modular. The architecture is usually claimed to be modular on the basis of the independence of Knowledge Sources. Other claims could be made about the relative independence of the abstraction levels which typically form the vertical structure of the blackboard database: relative independence because the blackboard architecture is commonly taken to be a representative of models which exploit near-decomposability.

The conventional method for building blackboard systems emphasises the relative independence of abstraction levels. Knowledge Sources are constructed for a particular level of the blackboard at a time. The Knowledge Sources are tested on entries at the chosen abstraction level: the results of executing the KSs are just placed in a nameless space during the earliest stages of development. When one level's KSs have been shown to operate properly, another level (frequently one that is adjacent) is then developed. The process iterates until all levels have been covered. This method relies upon an assumption that it is possible to decouple a particular abstraction level from the overall structure and operate upon it. It is also a way of managing complexity during development which relies upon the modular structure of the architecture. The method uses the fact that the effects of KSs can be separated out. Depending upon task, KSs which trigger on the chosen level and then those which modify that level are first introduced into the system; next, KSs which trigger on the level and modify other regions of the blackboard are introduced and tested. The effects of this second kind of Knowledge Source are collected in an effectively un-named space which represents 'the rest of the blackboard'.

The reason for introducing an aspect of the development methodology is to show that the blackboard architecture is considered to be highly modular by application builders and to show that it can be considered to be so, to an approximation. The caveat is important for I wish to show that the architecture is not

quite as modular as one might think.

It is usual to cite Knowledge Sources as the basic modular structure in blackboard systems. Knowledge Sources are independent knowledge structures which cause changes to the solution state recorded on the blackboard. They are not permitted to interact with each other directly, but must communicate only indirectly by posting entries on the blackboard. One KS may not share variables or other data with another KS unless that data is represented as an entry and has been posted on the blackboard. Because of this modularity, it is possible to include or remove KSs from a system without causing configuration problems. By adding or removing a KS, only the quality of the emerging solution can be effected. The emerging solution is represented as a configuration of entries on the blackboard, so the addition of new Knowledge Sources will either allow new entries to appear in the solution, will allow new relationships to come into being between entries, or will alter the range of modifications to existing entries. The removal of KSs is directly analogous, but operates in the reverse direction (e.g., the removal of a KS will reduce the number of possible entries which can be posted on the blackboard). Since there is no direct interaction between KSs, the addition or removal of a KS cannot impact upon the internal workings of any other KS.

Since Knowledge Sources are encapsulated modules, any alterations to their internal structure or to their content, equally, cannot cause changes to anything other than the quality of the emerging solution. By making a particular KS more specific, for example, the entries it can generate will also be more specific -- this is observable externally only as a property of the solution.

The case for KS modularity is, I think, quite sound. Knowledge Sources, however, communicate by posting and inspecting entries on the blackboard. As has been pointed out, a KS may inspect one region of the blackboard and record an entry in quite another region. This prompts the question of how the various regions are related to each other. The simplest case of this (and the one with the most general theoretical importance) is the relationship between the constituents of the blackboard structure's vertical dimension: abstraction levels, that is.

The hierarchy of abstraction levels on the blackboard is one of the most commonly cited structural details in descriptions of blackboard systems. Feigenbaum (Feigenbaum, 1978) describes the hierarchy as defining a prototypical control plan. The hierarchy represents views of the problem at varying levels of

abstraction. The structure and interpretation of the blackboard may be considered as being analogous to the abstractions in hierarchical planning (Sacerdoti, 1974; Sacerdoti, 1977): a problem is solved by considering its properties at a variety of levels of abstraction, the highest of which should provide the most concise description of the problem. This view of blackboard structure actually ignores other structural information as I will now argue.

At any abstraction level of a blackboard, there is usually one Knowledge Source which triggers on the entries there. It is usual for a KS only to trigger on one abstraction level because entries are allowed only to reside on one level. Triggering events are caused by the addition or modification of entries and so are each related to only one particular abstraction level. When a triggered KS is executed, it causes modifications to the blackboard at the same abstraction level as it was triggered or to another level.

The generic Blackboard Control Model KS, Implement-Strategy, for example, is triggered on the Strategy level of the control blackboard and makes modifications to that level *as well as* to the Focus level (the abstraction level immediately below it in the control blackboard hierarchy). Implement-Strategy changes the status of the current Focus entry (similar to Focus1, above) to "Inoperative" and proposes a new focus entry. A consequence of this action is to introduce a new problem solving strategy by posting a new entry on the Strategy level: the new appearance of the new entry on the Strategy level of the control blackboard will cause other KSs to be triggered.

This example clearly shows that the relationships between abstraction levels on the blackboard is rather more complex than is apparent from the definition given above. The example shows, in particular, that entries may, in theory, be recorded on a particular abstraction level by any Knowledge Source in a system. The interface between abstraction levels depends upon the Knowledge Sources which trigger on one level and cause modifications to another. The inter-level interface is hidden in the actions of all Knowledge Sources in a system and is, what is more, *distributed* across KSs. In other words, there is no central interface between abstraction levels. The interface between blackboard events and KSs is described in systems like BB1 and NBB as a causal connection which is locally defined for each KS. The definition is the trigger condition associated with each Knowledge Source. The interface describing the more global relationship between Knowledge Sources and the locus of their effects (the levels at which they alter the black-

board state) is not made explicit, but remains implicitly defined by individual KS actions.

The structure of an abstraction level may be considered to consist of a set of entries and a set of Knowledge Sources which trigger there. The KSs themselves may be thought of as being relations between abstraction levels. Two levels are related if at least one Knowledge Source triggers on one level and alters the contents of the second. This makes clear that the information held in entries at any particular level of the abstraction hierarchy depends upon information held at other levels, not just adjacent levels. It also makes clear the idea that KSs may post or modify entries on levels which are not adjacent in the abstraction hierarchy.

The second of these points is the important point: abstraction levels are related other than simply by the abstraction hierarchy which defines the blackboard. Entries are created as a result of executing KS actions; KS actions are permitted to execute as a result of KS triggering. The entry which causes a KS to trigger determines some, if not all, of the information the action uses to cause changes to the blackboard. Thus, there is a relationship defined by information flow across the blackboard. This relationship is defined in terms of the relationships represented by KSs and is additional to the generator relationship for the abstraction hierarchy. Quite obviously, the information flow across the blackboard is important, yet it seldom recieves attention: it has never, to my knowledge, been expressed in these terms before, but has always been submerged in descriptions of the abstraction hierarchy.

These considerations show that the blackboard database is not an encapsulated (or modular) construct. The interfaces presented by the abstraction levels in the abstraction hierarchy are related by means other than the abstraction relationship. The ways in which levels are related are hidden in KS actions, so there is no explicit statement of how an arbitrary pair of levels are related. What is more, there is no explicit statement of how information on any particular level is, in general, produced -- to obtain this information, one must examine the individual entries to determine which KSs have created and modified them, then examine the KSs mentioned by the entries to see where they trigger. The second step of this process may involve going through historical information to find entries on the KS's triggering level.

The modularity of the blackboard architecture rests *entirely* with Knowledge Sources. KSs have well-defined interfaces with regard to their areas of competence. The levels at which a KS triggers and at

which it executes its action are well-defined inside the KS. The larger issue of interfaces *across* the blackboard is far less clean and clear. The argument above has shown that information from a variety of sources may contribute to form entries at any given abstraction level. One of the justifications for the abstraction hierarchy is that it separates the structure of the blackboard into distinct and distinguishable regions. This justification appears to be defeated in the face of the evidence that information on a particular level may be composed of information from practically everywhere else without any statement (other than in each individual KS) of how it got there.

If abstraction levels are genuinely independent in the way that it seems they should, it becomes important to be able to define their relationships with other levels, just so that one can give a definition of what it means for an entry to appear at any given abstraction level. One way to do this is to define cleaner interfaces between abstraction levels. This permits an abstract description of the boundaries between levels. It also allows constraints to be imposed upon information flows across the blackboard. One interpretation of this is that it defines the kinds of information which can legally be used to construct an entry at a given level. At present, any Knowledge Source at any level may, in theory, post information on any other level without constraint -- in the limit, this leads, of course, to chaos.

Fodor (Fodor, 1983), makes the distinction between vertical and horizontal organisations in cognitive systems. Vertical organisations are domain-specific and encapsulated; horizontal organisations, on the other hand, are non-specific and non-encapsulated. Perceptual and linguistic processes are examples of vertical processes; central cognitive processes are cited by Fodor as examples of horizontal processes. In the blackboard architecture, vertical structure is represented by Knowledge Sources. There appears to be a tacit claim that because the blackboard is organised into abstraction levels, there is an encapsulation mechanism here, too. The claim rests upon the usual interpretation of 'abstraction' which implies that objects at different levels of abstraction do not strongly interact. From the way the blackboard is treated and described in implemented systems, it would appear that it has a horizontal structure, but its definition suggests a combination of vertical and horizontal structure. The very fact that the blackboard is organised into abstraction levels implies that each level is specialised and has special-purpose processes operating upon it: this is precisely equivalent to the definition of *vertical process* given by Fodor. The blackboard *as*

a whole may be regarded as a general-purpose process (a horizontal process, that is), but its components are specialised. This indicates that there is scope for re-examining the blackboard architecture from the viewpoint of modularity to see if it is possible to derive a model with clearer structure.

5. Generality

Finally, we must examine the claim that the blackboard architecture is a *general* model of problem solving: that is, that it provides a general framework within which to express and develop theories of problem solving. Nii (Nii, 1986a) states that

> "HASP, as the second example of a blackboard system, not only added credibility to the claim that a blackboard approach to problem solving was general, but it also demonstrated that it could be abstracted into a general model of problem solving."

Hayes-Roth (Hayes-Roth, 1983) specifically asks if the architecture is general. After examining the computational advantages and psychological plausibility of the model, she concludes (p. 23):

> "In summary, I would answer the question posed in the title of this paper: "The Blackboard Architecture: A General Framework for Problem Solving?" in a single word: Perhaps."

Although it seems inuitively plausible that the blackboard architecture *is* a general model of problem solving, this has not been conclusively shown.

Anderson (Anderson, 1983) argues that it cannot be an adequate model because it makes great demands of memory and requires considerable computational power. For Anderson, production rule architectures are the most plausible general model because they make fewer assumptions about computational requirements. Although I take the point, I am disinclined to accept that this is valid for it is not easily possible to express all that one can express within the blackboard architecture within the confines of productions, complex control being an obvious case.*

---

* It has to be admitted that I feel somewhat uneasy about Anderson's ACT* work. In particular, he has presented tasks which appear to support his theory of top-down goal-directed control. Although Means End Analysis and its relatives are attractive, I wonder if they capture the entire story: that is, work on MYCIN, for example, indicates that there are alternatives to sub-goaling strategies. Perhaps my true reaction is that Anderson's account appears to solve too many problems and leaves so little to do -- it all seems too neat.

One of the major generality claims for the blackboard architecture is that it is general because it is so flexible and so powerful. The blackboard has been used, it should be remebered, to tackle high complexity problems. It has been argued that the blackboard is the *only* architecture capable of operating in these domains (this argument is implicit in the selection of the architecture for the task). This seems to suggest that, were the task not so complex, the blackboard would not have been the chosen architecture. We have to be a little careful, here, because there are, effectively, two claims to generality: one is a *psychological* claim, the other rather more pragmatic. The argument that the architecture was chosen because it can handle the complexities of a given domain is a pragmatic one and depends only upon the flexibility of the architecture and upon the architecture's native power. One can see, practically immediately, that the blackboard architecture is not suited to some tasks: resolution theorem proving is one example -- one would not use the full power of the blackboard to do what Prolog does, although one might want to include a lush resolution theorem prover in a KS. The counter to this is that the blackboard provides a methodology within which it is possible to make problem solving and control knowledge more explicit.* The simplest task attempted, to my knowledge, is the reading task (McClelland, 1981; Rumelhart, 1982): but even this task is far from trivial (a property shared, it seems, by most perception-oriented tasks). Since the majority of blackboard systems have not been aimed at Cognitive Modelling, I will examine the generality claims without reference to psychological arguments.

The claim for generality for the blackboard architecture is based on the claims that

(i) it produces a general model of problem solving; and,

(ii) it is powerful and flexible enough to be employed as the architecture for any problem solving system.

The second point is wholly empirical and depends upon a sufficient number of implemented systems, operating in different domains, performing to satisfactory levels. The first claim is more difficult to assess, but there are pointers. The fact that there have been a number of completed and successful blackboard systems indicates that point (ii) has some truth: indeed, as a result of the Strategic Computing Initiative, the level of interest in the blackboard architecture has increased and it is being applied to many, different,

* I have heard a rumour that MYCIN and some of its derivatives are to be re-implemented as blackboard systems. The rationale is precisely the one which I have given.

problems.

If the architecture can be applied to different problems and, what is more, produces a *natural* solution (i.e., one which is aesthetically, theoretically and computationally satisfying), one would be inclined to assert that the architecture can be generally applied. This is said despite the fact that it is very difficult to apply the architecture to some domains and that systems which look highly counter-intuitive can result.

In particular, consider some of the tasks Anderson chooses for his work on ACT* -- elementary geometry and LISP programming. The reason ACT* seems to be such a plausible account of these domains is that there is a good fit between his architecture and the complexity of the problem. As a paper exercise some time ago, I took the geometry experiment and attempted to re-cast it as a blackboard system. The result was unspectacular. One consequence of adopting the blackboard architecture was that the concept of abstraction levels had to be taken on board and the result of that was that abstractions had to be found where there were none in the experimental data. The geometry production set shown on page 222 of (Anderson, 1983), reveals, on close examination, three abstraction levels: Goal, Sub-Goal and Operation. The solution is retrieved from the Operation level. This analysis appears to be less attractive than the one suggested by Anderson that there are goals and results which are distinguished, basically, by the role each plays in the problem solving process. The control model one would have to adopt to cater for Anderson's results is simply a variation on recursive sub-goaling: each rule in the set on page 222 is sensitive to the presence of a goal and merely expands that goal. This suggests that most of the power of the blackboard architecture is not used, and it also suggests that a blackboard implementation of these productions would be computationally wasteful.

It could be argued that it is comforting to know that the blackboard architecture could be applied to simple problems. This argument continues with the suggestion that the full power of the blackboard is not needed for such a simple problem, but it is there in case it is needed. My point is, though, that the power of the blackboard is always there and that it is being wasted in solving problems of this level of complexity.

A second argument concerns abstraction levels. For many tasks, it is extremely difficult to discover a natural abstraction hierarchy -- oft times, the hierarchy looks highly artificial and contrived (the controlled airspace monitoring problem is such a problem, as is the interpretation of visual images): the Hayes-Roths

found this a difficulty with parts of OPM and admit as much in (Hayes-Roth, 1979). One could say that the system builders simply have not done their analysis correctly, but this will not do, for this problem is encountered by experienced and careful application builders.

Many of the more successful applications of the blackboard architecture have been in domains in which there is a fairly obvious decomposition which can serve as the prototype for the blackboard: the speech processing domain is an example. Even when it is possible to find abstractions (it is usually possible to invent abstractions, at least in the domains I have considered), it is not always possible to form them into a coherent hierarchy. There is a further problem: abstraction hierarchies tend, very often, to be problem-specific. That is, it is not usually the case that an abstraction hierarchy can be used for another problem, let alone another domain. Can one really believe that an architecture is general if it posits structures which cannot be readily generalised? Unfortunately, I believe that it cannot.** This point could be expressed by saying that the abstraction hierarchies are problem-specific (otherwise put, they represent part of a faculty for that task -- this can be shown to be unreasonable). A partial counter-example is afforded by the BB* environment (Hayes-Roth, 1986). A number of assembly tasks are being implemented using the BB1 blackboard interpreter: the tasks use a more-or-less similar abstraction hierarchy. Even if hierarchies can be found for entire domains (and I am doubtful, though willing to be convinced), the faculty argument is pushed only one stage back, and that is to a stage at which it is hard to see how anything but very general organisations could be used: this would make, I suppose, the hierarchy fairly useless for specific problems and would, consequently, reduce the power of a system using these general hierarchies. Basically, an abstraction hierarchy is a knowledge-organising device and is domain- and/or problem-specific. The *capacity* of the blackboard architecture to represent abstractions is not in question: that is a matter of fact. What I am really questioning is the methodological assumption that everything must exhibit hierarchy in this way. Even in her paper on the generality of the architecture, Barbara Hayes-Roth equivocates on the status of abstractions (Hayes-Roth, 1983, section 3.2.7, p. 18).

In summary, these are my points. Firstly, the blackboard architecture is powerful and flexible, but

** I find this genuinely unfortunate because I like the architecture and find it generally quite satisfying. Whether it is satisfactory is, though, another matter.

when applied to simple problems much of the power and flexibility is just not required. Second, abstraction levels are in many ways problematic and only apply within particular domains or within particular parts of a domain. The blackboard methodology requires one to find abstractions in order to build working models, even when this might not appear necessary or advisable. It may be the case that non-hierarchical organisations are more appropriate, in which case, the blackboard becomes a set of panels with little or no internal structure. This last point sounds to me very much like a hack or other trick to force the architecture to cater for every case.

Methodologically, there are problems with the blackboard architecture. Specifically, its methodology is under-specified. It is under-specified in the following sense. The methodology basically states that problem solving systems should be constructed from modular Knowledge Sources which operate on a hierarchically organised database called the blackboard. Insofar as it is suggested that Knowledge Sources trigger on entries at particular abstraction levels of the blackboard, it specifies the relationship between KSs and abstraction levels. The methodology says a little about control, but not much, for the architecture does not really specify its control mechanisms. The methodology for building blackboard systems emphasises the fact that parts of the system can be considered as black boxes until it is fully integrated. A major assumption of the methodology is that the entire power of the architecture is always needed.

If a problem solving architecture is to be a candidate for the title *general problem solving architecture*, it must do a number of things. Firstly, it must be sufficiently flexible that any instance of problem solving activity from the simplest to the most complex can be covered. Second, it must not project unwanted constructs on descriptions of problem solving activity. Third, it must supply a general methodology which can be applied to every case of problem solving: when the architecture or its methodology fall down, a revision of the architecture must be revised. In some of these respects, the blackboard architecture is inadequate and cannot qualify for the generality title.

6. Conclusions

This paper has presented a fairly strict defintion of the blackboard architecture. The definition is quite close to that given by Hayes-Roth, (Hayes-Roth, 1983). The definition provides a framework within which to compare implementations of the blackboard architecture and acts as a standard against which to rate

systems claiming to be 'blackboard' systems.

The paper gives a detailed argument about the status and role of opportunism. Opportunism is cited by some (notably, Nii) as a central characteristic of blackboard systems -- so central, in fact, that systems without opportunistic control may not, by her definition, be properly called blackboard systems. I have argued that Nii's conception of opportunism is fundamentally incorrect and have argued that opportunism is just one of many control strategies which may be employed by blackboard systems. The reason for the identification of opportunistic problem solving and blackboard systems is that the latter are usually implemented using control mechanisms with memory: memory allows opportunism.

Finally, the modularity and generality of the blackboard architecture were considered. It was argued that there are modularity problems: in fact, that the architecture is not as modular as is commonly thought. One breakdown of modularity is in the very structure of the blackboard database itself. Since modularity is frequently stated as one of the desiderata of general problem solving architectures, the generality of the blackboard was briefly considered. There are grounds, based on arguments concerning modularity, to doubt that the blackboard architecture is a truly general model of problem solving.

References

(Anderson, 1983) Anderson, J.R., The Architecture of Cognition. Harvard University Press, London, 1983

(Balzer, 1980) Balzer, R., Erman, L., London, P. and Williams, C. HEARSAY-III: A Domain Independent Framework for Expert Systems. Proc. First Annual Conference on Artificial Intelligence, pp. 108 - 110, 1980

(Clacey, 1985) Clancey, W.J. Heuristic Classification. AIJ, Vol. 27, pp. 289-350, 1985

(Craig, 1986) Craig, I.D. The Ariadne-1 Blackboard System. Computer Journal, Vol. 29, No. 3, pp. 235 - 240, 1986

(Craig, 1987) Craig, I.D. BB-SR. Technical Report No. 94, Department of Computer Science, University of Warwick, February, 1987

(Erman, 1975) Erman, L.D. and Lesser, V.R., A Multi-level Organisation for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge. Proc. IJCAI 4, Vol. 2, pp. 483 - 490, 1975

(Erman, 1981) Erman, L.D., London, P. and Fickas, S. The Design and an Example Use of HEARSAY-III. Proc. IJCAI 7, Vol. 1, pp. 409 - 415, 1981

(Ernst, 1969) Ernst, G. and Newell, A. GPS: A Case Study in Generality and Problem Solving, Academic Press, New York, 1969

(Feigenbaum, 1978) Feigenbaum, E. and Nii, H.P., Rule-based Understanding of Signals. In (Waterman, 1978).

(Feigenbaum, 1982) Feigenbaum, E., Nii, H.P., Anton, J.J. and Rockmore, A.J. Signal-to-signal transformation: HASP/SIAP case study. AI Magazine, Vol. 3, pp 23 - 35, 1982

(Fodor, 1983) Fodor, J.A. The Modularity of Mind. Bradford Books, MIT Press, 1983.

(Forgy, 1977) Forgy, C.L. and McDermott, J. OPS, A Domain-independent Production System Language. Proc. IJCAI 5, pp 933 - 939

(Hayes-Roth, 1979) Hayes-Roth, B. and Hayes-Roth, F., A Cognitive Model of Planning. Cognitive Science, Vol. 3., pp. 275 - 310, 1979

(Hayes-Roth, 1983) Hayes-Roth, B. The Blackboard Architecture: A General Framework for Problem Solving? Report No. HPP-83-30, Heuristic Programming Project, Computer Science Dept., Stanford University, Palo Alto, CA, May 1983

(Hayes-Roth, 1984) Hayes-Roth, B. BB-1: An Architecture for blackboard systems that control, explain, and learn about their own behavior. Technical Report HPP-84-16, Stanford University, 1984

(Hayes-Roth, 1985a) Hayes-Roth, B and Hewett, M. Learning Control Heuristics in BB-1. Technical Report HPP-85-2, Stanford University, 1985

(Hayes-Roth, 1985b) Hayes-Roth, B. A Blackboard Model for Control. Artificial Intelligence, Vol. 26, pp. 251 - 322, 1985

(Hayes-Roth, 1986) Hayes-Roth, B., Garvey, A., Johnson, M.V. and Hewett, M. A Layered Environment for Reasoning about Action. Technical Report No. KSL 86-38, Knowledge Systems Laboratory, Stanford University, 1986

(Lenat, 1982) Lenat, D.B. The Nature of Heuristics. AIJ, Vol. 19, pp. 189-249, 1982.

(McClelland, 1981) McClelland, J.L. and Rumelhart, D.E. An interactive activation model of context effects in letter perception: Part 1. An account of basic findings. Psychological Review, Vol. 88, pp. 375-401, 1981

(Newell, 1973) Newell, A. Production systems: models of control structures. In W.G. Chase (ed.), Visual Information Processing, Academic Press, New York, 1973

(Nii, 1986a) Nii, H.P. The Blackboard Model of Problem Solving. Artificial Intelligence Magazine, Vol. 7, No. 2, pp. 38 - 53, 1986

(Nii, 1986b) Nii, H.P. Blackboard Systems Part Two: Blackboard Application Systems. Artificial Intelligence Magazine, Vol. 7, No. 3, pp. 82 - 106, 1986

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

(Rychener, 1978) Rychener, M.D. and Newell, A. An intrstructable production system: basic design issues. In (Waterman, 1978).

(Sacerdoti, 1974) Sacerdoti, E.D. Planning in a Hierarchy of Abstraction Spaces. AIJ, Vol. 5, pp. 115 -135, 1974

(Sacerdoti, 1977) Sacerdoti, E.D. A Structure For Plans and Behavior. Elsevier, New York, 1977

(Terry, 1983) Terry, A. The CHRYSALIS Project: Hierarchical Control of Production Systems. Memo HPP-83-19, Heuristic Programming Project, Computer Science Dept., Stanford University, Palo Alto, CA., May, 1983

(Waterman, 1978) Waterman, D.A. and Hayes-Roth, F. (eds) Pattern-directed inference systems. Academic Press, New York, 1978