March 1989

Blackboard System Generator (BSG): An Alternative Distributed Problem-Solving Paradigm

Barry G. Silverman
University of Pennsylvania, basil@seas.upenn.edu

Joseph S. Chang
IntelliTek, Inc.

Kostas Feggos
George Washington University

Blackboard System Generator (BSG): An Alternative Distributed Problem-Solving Paradigm

Abstract
The classical blackboard model employs a number of relaxations of team decision theory that are commonly organized into three panels of AI heuristics, including: 1) a shared information panel that offers a capability for ensuring agent knowledge sharing, 2) a contract formalism for the agent and event scheduling, coordinating, and control panel, and 3) a blackboard panel for metalevel planning and guidance that offers whole situation recognition, top down reasoning, and adaptive learning. The nature and implications of these relaxations are explained in terms of the blackboard system generator (BSG) and via comparisons to what is done in other blackboard shells. Particular attention is paid to theoretical relaxations inherent in the classical blackboard model and to research opportunities arising as a result. Progress made to date to counteract adverse effects of some of these relaxations is described in terms of a project management/work breakdown paradigm adopted in BSG that: 1) alleviates the knowledge engineering bottlenecks of traditional blackboards and that provides BSG with a semantic rather than just syntactic understanding of blackboard control and scheduling; 2) allows a distributed problem-solving capability for connecting agents at virtual addresses on a logical network and that permits concurrent processing on any machine available on the network; 3) establishes an open architecture that includes techniques for integrating preexisting agent methods (e.g., expert systems, procedures, or data bases) while laying the foundation for assessing the impact of "black boxes" on the global and local objective functions; and 4) utilizes project management techniques for team agents planning as well as an analogical reasoner subsystem for BSG metaplanning and generic controlled learning. This latter item is supported by a connectionist scheme for its associative memory. The techniques of each of the three panels and of the four sets of paradigm-related advances are described along with selected results from classroom teaching experiments and from three applications using BSG to date.

Blackboard System Generator (BSG): An Alternative Distributed Problem-Solving Paradigm

BARRY G. SILVERMAN, SENIOR MEMBER, IEEE, JOSEPH SHIH CHANG, AND KOSTAS FEGGOS

Abstract—**The classical blackboard model employs a number of relaxations of team decision theory that are commonly organized into three panels of AI heuristics, including: 1) a shared information panel that offers a capability for ensuring agent knowledge sharing, 2) a contract formalism for the agent and event scheduling, coordinating, and control panel, and 3) a blackboard panel for metalevel planning and guidance that offers whole situation recognition, top down reasoning, and adaptive learning. The nature and implications of these relaxations are explained in terms of the blackboard system generator (BSG) and via comparisons to what is done in other blackboard shells. Particular attention is paid to theoretical relaxations inherent in the classical blackboard model and to research opportunities arising as a result. Progress made to date to counteract adverse effects of some of these relaxations is described in terms of a project management/work breakdown paradigm adopted in BSG that: 1) alleviates the knowledge engineering bottlenecks of traditional blackboards and that provides BSG with a semantic rather than just syntactic understanding of blackboard control and scheduling; 2) allows a distributed problem-solving capability for connecting agents at virtual addresses on a logical network and that permits concurrent processing on any machine available on the network; 3) establishes an open architecture that includes techniques for integrating preexisting agent methods (e.g., expert systems, procedures, or data bases) while laying the foundation for assessing the impact of "black boxes" on the global and local objective functions; and 4) utilizes project management techniques for team agents planning as well as an analogical reasoner subsystem for BSG metaplanning and generic controlled learning. This latter item is supported by a connectionist scheme for its associative memory. The techniques of each of the three panels and of the four sets of paradigm-related advances are described along with selected results from classroom teaching experiments and from three applications using BSG to date.**

"Give me a fruitful error anytime, full of seeds, bursting with its own corrections, you can keep your sterile truth to yourself."

Comment on Kepler,
Vilfredo Pareto, circa 1900.

Manuscript received June 15, 1987; revised January 28, 1988 and February 9, 1989. This work was supported in part by NASA JPL Small Business Innovative Research (SBIR), NASA GSFC SBIR, and the GSFC code 522 Contracts.

B. G. Silverman is with the Institute for Artificial Intelligence, George Washington University, Washington, DC 20052 and IntelliTek, Inc., Rockville, MD 20852.

J. Chang is with IntelliTek, Inc., Rockville, MD 20852.

K. Feggos is with the Institute for Artificial Intelligence, George Washington University, Washington, DC 20052.

IEEE Log Number 8820137.

I. INTRODUCTION

THE BLACKBOARD "MODEL" for an expert system is illustrated in Fig. 1(a) as a conference table around in which a Chair convenes a meeting of a number of specialists. Each agent is a specialist in a few subjects and no single agent has the breadth and depth of intelligence to solve the shared problem in isolation from the others.[1] Such meetings are common in everyday problem solving, examples of which could be a presidential cabinet meeting, a spacecraft design team meeting, or a family planning session. There are various rules of protocol for guiding the conduct of such meetings that the Chair and agents agree to at the beginning of the meeting. These rules generally exist to permit each agent to share their unique viewpoints about their common problem and/or its possible solution; to be stimulated from the differing viewpoints of the other agents; and to reason spontaneously and opportunistically in attempting to reach a creative solution to a common problem.

In the blackboard model there are generally three principal subsystems: 1) the blackboard that is a shared, global data space that facilitates communication and cooperation; 2) the agents that are knowledge sources capable of reacting to and modifying the blackboard data structures, and 3) the Chair that both plans agendas and controls agent activity. An assumption exists that the agents can pool their insights to build toward the common goal.

A Blackboard can organize information into a hierarchy of "panels" and "levels" within each panel (see Fig. 1(b)). The agents share information in the lowest panel while the two higher panels include levels of abstraction of the shared information and of the overall progress toward the system's goal. This paper is devoted to the description of what happens in each panel (see Sections II–V). By way of overview: the bottom panel contains what has been said and what is believed, the middle panel consists of agent proposals to take action and Chair tactics for controlling

[1]Alternately, an agent may be "cloned" and identical copies of it may be run on idle resources. Similarly, numerous blackboards can access the same agent.

0018-9472/89/0300-0334$01.00

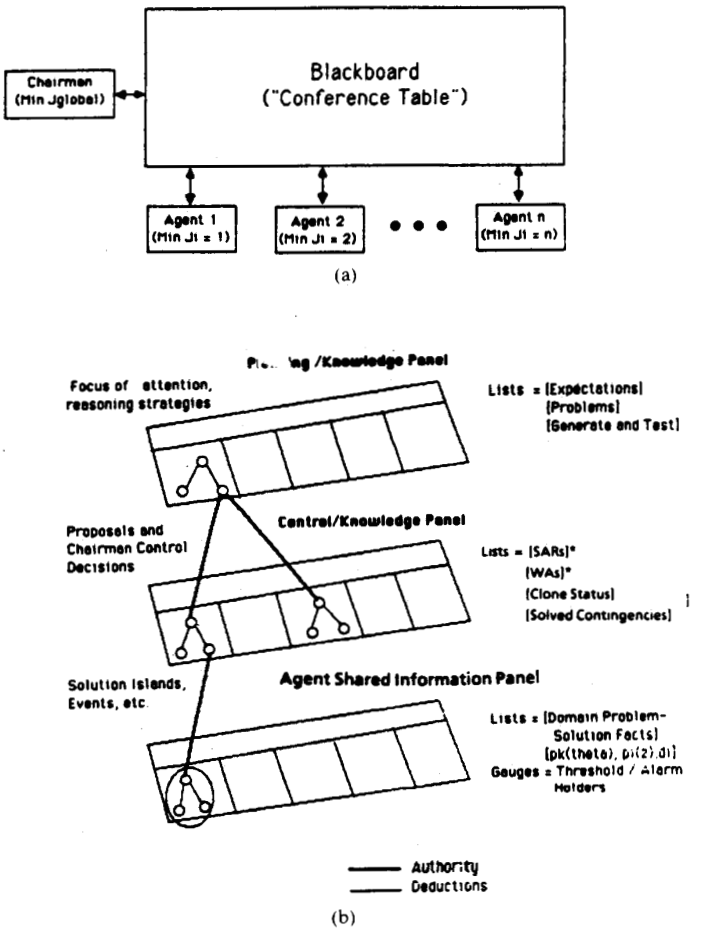


Fig. 1. Overview of blackboard model. (a) Blackboard and chairman support distributed agent cooperation. (b) Blackboard panels and levels. *SAR is specialist activation request and WA is work authorization.

the "meeting" agenda and schedule, while the highest panel focuses on overall meeting agenda planning and learning from past meetings.

Historically, team–agent theory probably draws its roots from dialectical systems theory (originating in Aristotillian times) although the modern origins of blackboard technology truly began in the 1970's with the Hearsay II project (Lesser *et al.*, [2]) and has recently been transformed into the "classical" blackboard model at several AI centers of note: e.g., see [1], [10], [13].

Formally speaking, the classical blackboard model is an attempt to apply heuristics to a mathematically intractible team-decision problem. Specifically, let there be two assumptions:

1) Different but correlated information for each agent concerning some underlying uncertainty, and
2) Need for coordinated actions on the part of all agents to realize system-wide payoff.

Further let,

$\theta(t) = [\theta,(t), \theta_2(t), \cdots, \theta_n(t)]$
= a vector of random variables (RV's) that are the states of nature;

$p(\theta(t))$ = the probability distribution surrounding each state of nature;

$Z = [Z_1(t), Z_2(t), \cdots, Z_N(t)]$ = observations vector on the states by each of N agents;

$Z_n = K_n[0(t)]$ = observation vector (or array) of the nth agent $(n = 1, N)$ where K is a set of observation rules or operators.

$D(t) = [D_1(t), D_2(t), \cdots, D_N(t)$
= decision variables of the N agents (each agent has an array of decision variables).

$D_n(t) = \{d_{ni} = WP_{ni}(Z_n(t)) | WP_{ni} \in G_n\}$
= the decision variable of the nth decision maker given the observations he has made on the states of nature.

$D_n(t)$ takes the value of one of the admissible class of decision functions, G_n, of that agent. Note: d_{ni} can also be written as $d_{ni} = WP_{ni}$ $(K_n(\theta(t)) = F_{ni}(\theta(t)))$.

$p(D(t))$ = the proability distribution surrounding the transition from one state to another.

$$p(D(t)) = p[D_1(t) = d_{1i} D_2(t) = d_{2i}, \cdots, D_N(t) = d_{Ni}]$$
$$= p[\theta(t+1) = j | \theta(t) = i_t, \theta(t-1)$$
$$= i_{t-1}, \cdots, \theta(0) = i_0]$$

Loss $= L[D(t), \theta(t)]$ = the loss or payoff criterion that estimates the discrepancy between the current state and the goal state.

Then the team–agent decision problem becomes:

$$\text{Find } D(t)_{\text{optimum}}, \text{ or } \mathrm{WP}_{n\,\text{opt.}} \in G_n$$
$$\text{for all agents } (n = 1, N) \text{ such that}$$
$$\min J(t) = \text{"expected value" } \theta(t)\big(L[D(t)_{\text{opt.}}, \theta(t)]\big). \tag{1}$$

The blackboard is thus no more than the lists θ, Z, $p(Z)$, etc. Loosely put K_n is the set of forward reasoning operators that an agent has for moving from states toward observations while WP is the agent's library of back reasoning operations for moving from Zs to Zs. D_n is the agents proposals (decision variables) to apply Ks, Fs, and WPs to the data and to attempt to make a state transition. Opportunistic reasoning is facilitated via the observations $Z_n(t)$ and decisions $D_n(t)$ of each agent. The widely discussed "principle of least commitment" is capturable as $p(Z)$, $p(\theta)$. Agent viewpoints are represented as multiple, conflicting $p(D_n(t))$ and $p(Z_n(t))$ stored simultaneously and in their desire to optimize their local objective functions (Min J_n). Cooperation is the global optimization problem (Min J_{global}).

Casting the blackboard model in these terms permits its recognition as a decision theoretic problem of long standing (e.g., [21]). Further, it facilitates the application of a large body of technique from other disciplines such as, but not limited to, operations research, decision science, modern control theory, and probability and statistics, to mention a few.

Unfortunately, (1) is the simplest form of the team–agent decision problem. It is completely static: only one team decision is made. To correct this it is necessary to introduce new and/or modified constraint functions (operators) such as $D_n(t) = \mathrm{WP}_n(Z_n(t-i))$, $D_n(t) = F_n(\theta(t-i), Z_n(t-i))$, $Z_n(t) = K_n(Z_n(t-i), D_n(t-i))$, where $t-i$ indicates one or more prior cycles of activity.

More importantly, the controls (D_n) of each agent in (1) do not depend on the actions of the other agents.[2] In (1) cooperation occurs: a) statically through the system-wide objective function, and b) dynamically through changes to θ. A more realistic result would be to let $Z_n = K_n(\theta, D_n, Z_{n'})$, $D_n = \mathrm{WP}_n(Z_{n'})$, etc. where n' is an indicator of agents other than agent n. That is, to let an Agent's observations depend on both the states and the decisions/observations of other Agents. This is referred to as IR_n (information required by agent n).

This single set of refinements brings us to an unsolvable (analytically) team decision problem that seems so central to the concept of opportunism characteristic of the blackboard model. That is (the features most often expected by users), a blackboard should allow agents to have opportunities in the sense that they can: 1) inspect each other's conjectures and decisions, IR($\cdot$); 2) alter their own points of view and beliefs as a result of (1) or as problem solving evolves (i.e., maintain $p(Z_{nca})$, $p(D_{nca})$ where c and a are cycle and alternative stamps, respectively); 3) maintain several potentially conflicting hypotheses (explanations) simultaneously; and/or 4) suspend a given line of reasoning K_{nca} or WP_{nca} (possibly for later resumption) in order to pursue what appears to be a more important reasoning pathway at present. Since the competing hypotheses are weighted, the blackboard system is expected to give what it considers its best answer or solution at any point in time: it obviously will give more correct solutions as time proceeds, as more observations are formed, and as more operations upon these operations occur.

This paper investigates an implementation of the "blackboard model" by explaining the progress made to date in the data structures, reasoning schemes, and techniques of a tool called the blackboard system generator (BSG). BSG represents heuristic relaxation of the optimization requirement implicit in (1). BSG replaces the global optimization guarantee ("holy grail") with a set of plausible heuristics expected to arrive at reasonably robust local optima. The ultimate research objective of BSG is to describe a decision theoretic version of the blackboard model that will facilitate application of known decision technology that can help minimize team decisions that are clearly dominated ("valleys") in terms of the global and local agent objective functions (J_{global} and J_n).

II. The Blackboard Shared Information Panel

The shared information panel is segmented into levels that capture and hold distinct sets of input/output information that needs to be shared either between different agents or across cycles by a single agent. These levels are important in the blackboard model in general, however, the model gives little insight into how they must be designed for a given application so as to maximize their likely impact upon the blackboard's efficiency and effectiveness. The rule of thumb to follow when designing levels is that lower levels should hold data that will tend to change more rapidly (or earlier) than that in the upper levels. For example, many applications will tend to place states of nature θ's at the lowest level and the desired global decision, D, at the uppermost level with various combinations of F and Z data at the intermediate levels. This rule of thumb is vague and the precise data to be placed in each level will vary from application to application.

Experience by the lead author with teaching BSG over the past five semesters to graduate students majoring in AI has shown that proper design of levels and objects is a difficult concept to convey. Discussions with other BB developers has revealed a similar obstacle encountered by

[2] Both Barbara Hayes-Roth, whose BB1 exists at over 30 sites, and Jagannathan ("Juggy") at Boeing, whose ERASMUS has been used internally quite heavily, indicated user training and knowledge engineering to be the major bottleneck. (Discussions at the "Workshop on Blackboard Systems", AAAI-87, Seattle, 7-13-87.)

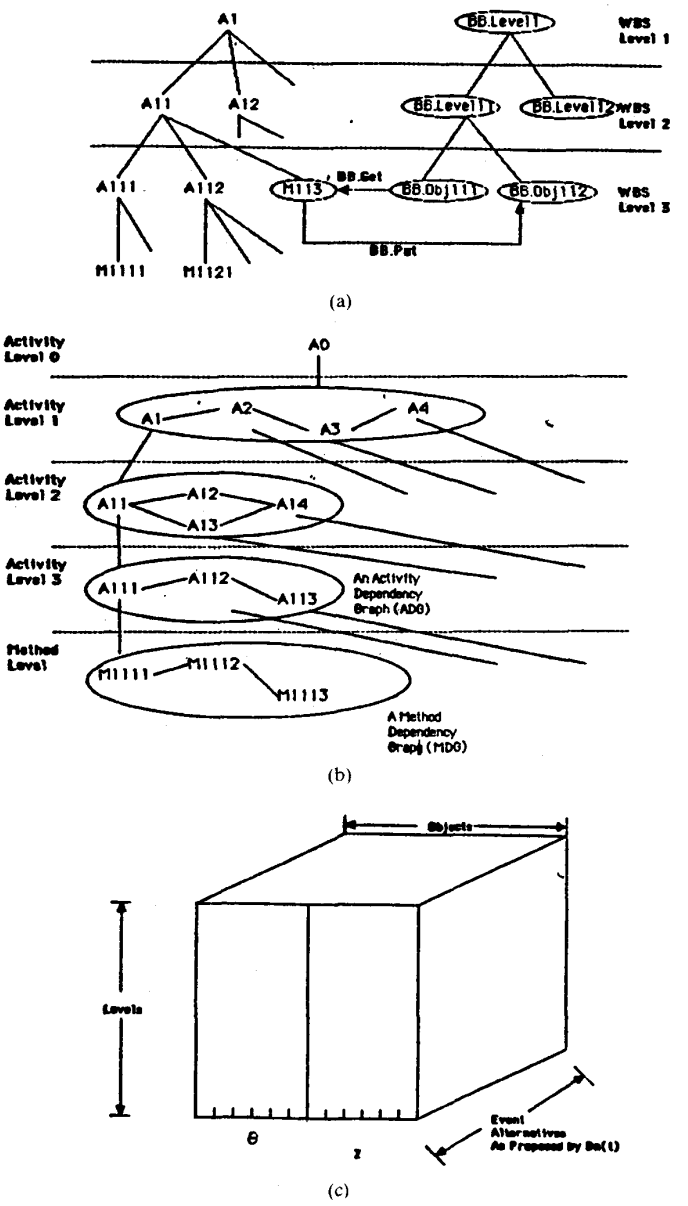


Fig. 2. Activity and event networks for distributed problem solving agents. (a) Activity work breakdown structure (WBS). Defines blackboard levels and state transition events. (b) State transition operators imply activity and method dependencies. (c) Shared information panels becomes blackboard transaction data base. A—activity. M—method.

their users.[2] To overcome this obstacle, BSG is now taught in terms of a workflow metaphor known as project management [22]. The project manager is the Chair whose job is planning and control, while the agents serve as project team submanagers, and agent methods are project team "personnel."

A. Activity Work Breakdown Structures (WBS's)

The BSG knowledge engineer will generally want BSG to replace actual human-performed tasks and activities. To that end the project management techniques of work breakdown and task analysis are utilized for knowledge engineering. That is, the knowledge engineer analyzes the current human tasks and assembles a "work breakdown structure," which is an activity hierarchy mapped onto a goal hierarchy (connected hierarchical graph). The WBS hierarchy starts with the top level goal as its root node while successively lower levels hold work packages (hierarchies of activities) to be performed by BSG agents.

B. Blackboard Levels, Events, and Objects

The shared information panel of BSG corresponds to the WBS structure in that they both must have (by convention) the same levels. Blackboard objects are event-ori-

ented, however, and do not represent the work packages (that is the role of agent methods). When BB Objects are read to or from, or altered by the agent methods, a milestone or event is said to have occurred in the project management sense of these terms (see Fig. 2(a)).

Given both the importance of arriving at an efficient levels schema and the difficulty of predicting what that scheme should be *a priori*, BSG had to be given a number of features that facilitate rapid prototyping and reediting of the levels scheme. These include mouse-buttonable, pop-up menus that permit create, move, delete, and edit type operations on the object that defines a level and its attributes. Any number of sublevels also can be defined, however, each sublevel may have only one direct parent.

C. Activity and Method Dependency Implications

The WBS paradigm requires specification of permissible sequences of work packages at each level of the WBS. In such diagrams the activities are the nodes while events are represented by links, as shown in Fig. 2(b).

Blackboard users mentally construct such diagrams when knowledge engineering a new application. The coding of BB Objects on the blackboard levels combined with the event transition operators of the work packages facilitates the specification of event-activity interplay. This is a second knowledge engineering bottleneck of blackboard applications (specifying WBS's was the other mentioned so far).

D. A Note On Blackboard Object Engineering

BSG is built on top of an object definition language (ODL) and hence uses an object oriented programming paradigm throughout, as illustrated for the shared information panel in Fig. 2 and as will now be further explained. This paradigm was selected to facilitate exploratory design, rapid prototyping, definition by specialization, message passing, and a uniform frame-like representation for shared data. (Although Knowledge Engineering Environment (KEE®) was initially adopted, ultimately an internally developed ODL tool was utilized to facilitate portability and satisfaction of requirements discussed in Section V-B-3.)

BB Objects actually hold the information that is to be shared between agents and/or across cycles. They are created (via mouse button) as child objects of the levels/sublevels. There is no practical limit to how many BB Objects can be attached at each level, although they must themselves be leaf nodes (they cannot have child objects).

E. Pros and Cons of Blackboard Objects

The ultimate advantage of any object oriented representation lies in the ability to imbue the objects with local intelligence so that they can independently process and respond to messages they receive from the agents, the chair, other BB Objects, etc. Several types of local intelligence are embedded in BB Objects as follows:

- *They are design advisors to the knowledge engineer*—BB Objects are smart about which agents can operate upon them. Three operations are possible: BB Put, BB Get, and BB Change. The user (or application builder) can query the BB Objects to discover who will post information and who will react to that posting. In a second regard the blackboard objects provide knowledge engineering support. This lies in their ability to insulate the knowledge engineer from difficult programming tasks. In effect, once the knowledge engineer has entered his levels and BB Objects, his design work is over. In reality, this information must be parsed into the (empty) "possibility" space as in Fig. 2(c). This is done automatically by BSG objects.
- *They are control advisors to the Chair*—There are two types of control (and planning) knowledge a BB Object communicates to the Chair:
  1) work packages sequence dependencies, which the Chair uses in its conflict resolution and control heuristics, are sent to the Chair at runtime (see Section IV). Sequence dependencies are simply the precedence relations established by the agent BB Put, BB Get, etc., operations that in turn are internally recognized as WBS events and which, thus have a schedule and time dimension significance in the Chair's "project control" paradigm; and
  2) competing hypotheses (third dimension of Fig. 2(c) are remembered by the BB Objects and relevant portions of these histories are sent to the Chair for its control needs (or to the Explain function, see Section IV).
- *They will be self-diagnosing error handlers*—A feature not yet implemented but for which the proper foundation now eixsts is for the BB Levels and BB Objects to advise the BSG knowledge engineer of constructional and/or runtime errors and their source. While object oriented interfaces are useful for the aforementioned purposes, they run the risk of inefficiency under operation. Should the number of alternative hypotheses, observations, etc., for a given BB Object grow large, the history slot acts as a sequential "file" that proves somewhat slow to process. Several alternative data structures are currently being investigated, including relational structures (as in GBB, Corkill [11]) and BB Object instantiations. The goal, however, will be to maintain the friendly interface and only parse the "history slot" into a more efficient structure at runtime.

F. Case Study of the Design Elicitation Paradigm

Before implementing the WBS, many of the students' independent study projects tended to adopt distributed expert system rather than true blackboard architectures.

®Kee is a registered trademark for IntelliCorp, Palo Alto, CA.

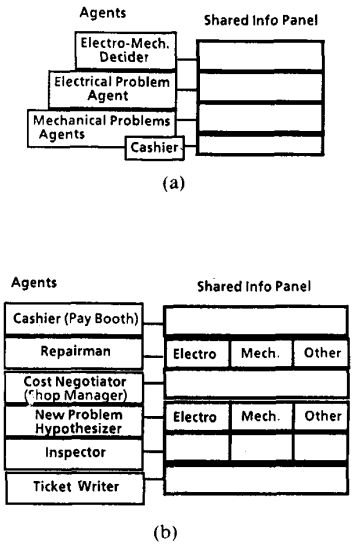


Fig. 3. Overcoming common design pitfalls with BSG elicitation paradigm. (a) Car mechanic problem as distributed expert system (before WBS paradigm). (b) Car mechanic problem recast as interdependent problem solving heirarchy (after WBS paradigm).

That is, each level of the blackboard shared information panel was devoted to a separate expert system with little to no problem solving interdependence between levels.

For example, one student team [3] knowledge engineered BSG for a car mechanic's shop in which they initially created three levels corresponding to three specialist agents: electrical, mechanical, and electro-mechanical decider: the cashier greets the car owner and accepts the job, hence, he was placed at the bottom level even though he also collects payment when the repair is finished. The top level agent decides whether a new problem is electrical or mechanical and one of the two lower level agents then completes the diagnosis and repair. In this case, most of the problem solving "power" of BSG has been wasted: BSG could be replaced by an expert system shell with a single if–then control rule since none of BSG's functionality for processing intermediate results and hypothesis are needed.

An alternative design was then elicited utilizing the WBS paradigm. The improvements include: 1) Problem solving sequences are now better structured; the cashier's begin and end functions are separated as they should be and the cost negotiations prior to actual repairs are now evident (and integrated across electrical mechanical problems), 2) important events previously hidden in the agents (and repeated from agent to agent) have been made explicit on, the BSG while agents now hold only methods (activities); and 3 the Chair now has enough domain levels and interdependencies to be able to intelligently sequence activities and level interplay/interaction. It is no longer possible to use a single expert system shell on this problem (due to the cycles, negotiations, and hypotheses that must be ventured and retracted). Fig. 3(b) is a viable application of BSG while Fig. 3(a) was not.

III. Distributing Agent Deductions Across Machines and Packages: Processing versus Control

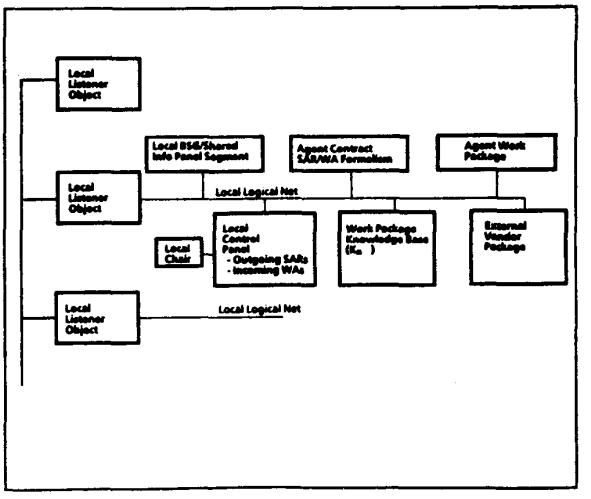
Two important distributed problem-solving considerations include distributed processing and distributed control. Distribution of processing allows parallel computation of concurrent activity for various levels of granularity. Large granularity might correspond to multiple copies of a BSG application distributed over several machines; medium granularity could involve distribution of only selected agents, WP methods, or knowledge bases; and small granularity distributed processing might involve parallel firing of individual rules or rule subparts in the knowledge base's AND/OR graph. Numerous schemes exist in the general, nonblackboard literature for each of these levels of granularity: e.g., see [11], [14]–[19]. The choice of granularity level is often influenced by machine constraints, however, most blackboards to date fall under the large granularity case. Reference [16] appears to be a proposal for a medium granularity blackboard system.

BSG has been designed in a modular fashion so as to ultimately offer most of the possibilities just mentioned for distributed processing and distributed control. At present only the large and medium granularity processing options and low and medium control distribution cases have been attempted. The BSG features that facilitate distribution of processing and control will now be described.

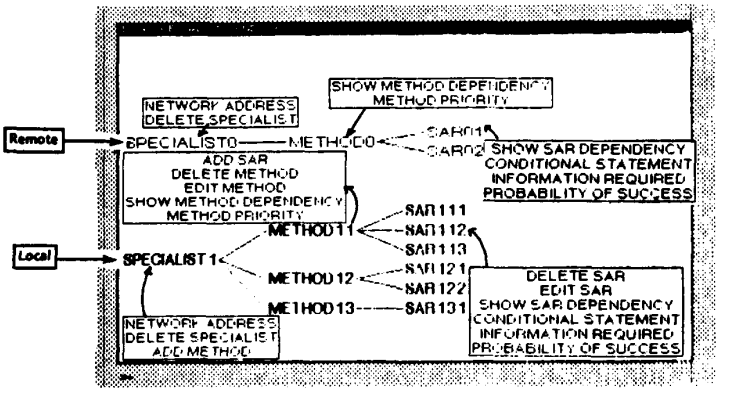
A. Logical Nets, Virtual Addresses, and Modular Agent Objects

BSG uses a logical network with virtual addresses denoting the various agent and shared information panel element locations. The logical network insulates BSG operations from actual hardware peculiarities. For large granularity, centralized control applications run on a single processor and the logical network may be turned off to save processing time. Virtual addressing permits agent modules and panel segments to be implemented in almost any degree of granularity and distribution desired. It also facilitates: 1) instantiating or cloning of agents for use on idle resources, 2) multiple blackboard applications utilization of the same agent, and/or 3) integration of agents that run almost any shell (e.g., Kee, Art, OPS-5), package (e.g., Lotus, DB III, Connectionism/Neural Net), or procedure (e.g., *in situ* model, simulator, or trainer). Discussion of instantiating or cloning is postponed to Section III-D.

The BSG Logical Net is decomposed into multiple instances of the local logical net that, in turn, consists of multiple instances of seven types of objects as listed in Fig. 4 and as now described. Local Listener Objects exists on each branch of the logical net corresponding to either a physical process or a physical machine. They route messages to on/off of both the local panel segments. They also route suspend, kill, and restart process messages relevant to their local net. Each local net may have two categories of bulletin boards: the remote BSG/SIP seg-



(a)



(b)

Fig. 4. Formalisms for distributed agent addressing, layering, and editing. (a) Alternatives exist for distributing agent control and/or processing. (b) Specialist lattice and pop-up menus.

ments and the local task queue. The latter is simply a buffer for outgoing or incoming files.

The shared information panel segment is what the specialists or agents generally have read/write access to across the logical net. Only agents, generally, are allowed to modify information on the blackboard (user as an agent). Each agent has an objective function or goal that causes it to read lists of Z's, θ's and D's from permissible blackboard panel levels and to formulate Z's, D's, θ's, etc., that it contributes to the objective or goal, Min J_n.

Central to this interplay are the agent contract formalisms and work package (WP) method modules that can be attached to the local logical net for one or more agents. The agent contract formalism is addressed in Section III-B. While some WP methods will need to stand alone, the more common case is where WP Methods are designed as high level λ functions that integrate and coordinate the queries, asserts, firings, etc., of the "lower layer" tools and packages. In general, knowledge engineers will utilize the WP methods to integrate external shells and preexisting tools. In such methods, the external shells provide an inference engine that can be used to propagate Z's and F's and to deduce new Z's, F's and new θ's. These external tools are also convenient for organizing and holding the extensive knowledge bases of rules, procedures, facts, etc: the knowledge engineer can program in whatever shell he is comfortable with.

The knowledge engineer can add specialists locally (boldprint in Fig. 4(b)) or remotely. Remote specialists appear in normal print and are read-only from the host. When the knowledge engineer adds a specialist or agent, n, to the blackboard he enters a lattice of r WP methods (WP_{nr}) into the BSG's object definition language as shown in Fig. 4(b). That is, a set of WP methods ($r = 1, R$) are created for: 1) monitoring and updating lists on the shared knowledge panel, 2) firing off expert systems and other packages attached as additional methods to the lower layers of the agent architecture, and 3) performing any other procedure desired. WP Methods are created as lisp objects that inherit several slots from the generic WP

method embedded in the BSG that facilitates the above three purposes. The knowledge engineer's task is to edit the value slot (a WP function partially filled in) and to specialize it so it will perform one of the three purposes for the application of interest.

WP methods thus can owe their origin to one of three sources: 1) inheritance in total or in part from BSG—the principal methods that exist totally within BSG and are noneditable are described in Section 3-D; 2) off-the-shelf vendor packages; and/or 3) user-edited portions of the WP functions. In all three cases, however, a WP object is created within the WP agent method lattice that has the following slots:

[WP method name (WP_{nr} name),
Parent WP methods (list),
Children WP methods (list),
Address (value),
Information required: (list of variables, θ, $IR(\cdot)$),
Expected output (types of variables generated, $P(z)$),
K_{ng} partitions referred to (list),
Triggering features (list of types of problems that can be solved),
WP method value (actual code of WP functions and/or pointer to code file)].

Knowledge bases and external packages are similarly connected to the logical net.

D. The Contract Formalism

For most purposes the agent cannot utilize its WP methods without first sending a proposal or specialist activation request (SAR) to the Chairman (see Section IV on the control panel). SAR's are a contract formalism for proposing a useful task.[3] If the Chair accepts the SAR, a return message called a work authorization (WA) is issued and the Specialist's WP method will then be automatically initiated.

SAR's are the common language by which the agents can ensure participation. They are also the set of decision variables $D_n(t)$ available to the agent (a decision is made when a SAR is committed to and sent). SAR's are a packeted message (object) consisting of a unique header identifying the specialist as well as the point at which they were created. In addition they include seven fields or slots as indicated in the pop-up menu on the bottom right hand side of Fig. 4(b).

While the user enters the slot values in English-like syntax via responses to each items on the SAR editor, the true effect of entering this information is to establish the following elements of the distributed objective function list.

SAR	$D_{ni}(t) = [\mathrm{WP}(Z),\ t,\ B,\ p(Z),\ K_{rq}(\theta),\ \mathrm{IR}_i(p(\theta), p(Z_j), d_j)]$. The SAR is a proposal (decision) from the ith agent instantiated on the kth cycle to attempt state transition activity.
WP(Z)	planned (activity) process of Min J_n.
t	duration of the planned process or activity.
B	priority of this process.
u	uncertainty of achieving the expected outcome.
$p(Z)$	expected output (distribution of outcome if outcome is achieved).
$K_{rq}(Z)$	rule set or procedure to invoke (i.e., this is part of the planned process) if SAR is accepted.
$\mathrm{IR}_j(\cdot)$	information required to complete the planned process: i.e., information required from the blackboard, $p(\theta)$, as well as from other agents j.

Several SAR's may be created for each WP_{nr}; there will generally be $s = 1, S_r$ of them created for each WP_{nr} for a total of $\Sigma_{r=1}^{R} S_r$ SAR's per agent. Since SAR's inherit a number of their slot values from their WP method parents, creating SARs does not require as much time as might be expected. Also, since parent WP methods may themselves be combinations of several lower level child methods (as discussed earlier) a given SAR may actually be a proposal for more than one method. This in turn creates a possibility for a given WP_{nr} to give rise to yet other WPs in other cycles. A related point worth noting is that WP_{nr} and K_{nq} are essentially a fixed or unchanging possibility set from cycle to cycle. The decision process, d_i, on the other hand provides the precise trace of reasoning taken through all cycles, n.

The significance of the values entered into each slot is further elaborated in the discussion of (1) as contained in Sections III-C, IV, and V. Since each SAR object spawns many instances of itself during execution, a record is automatically generated of decision processing that can be used to "explain" its choices of WP's, K's, SAR's, etc.

1) Local agent object function: Moderately distributed control: In operational terms each specialist strives through a series of processing states to utilize its various layers to achieve its local objective function or goal. These states are summarized in Fig. 5, which depicts the specialist as an ongoing process constantly trying to solve (1) by watching the blackboard for triggering events, WA's, or goals. Upon detecting a change or stimulus, the left-hand loop of the diagram is pursued by the agent's methods assess whether any action is needed, and if so they then formulate and issue a SAR. The process cycles to the top of the diagram and if a WA is received the left-hand loop determines whether it is still an appropriate task and, if so, the right-hand loop is then pursued in which the task is performed and the results appropriated propagated. In

[3]The Chairman, in effect, convenes a Proposal Evaluation Board to rate and select the best proposal(s) on each cycle. A useful discussion of contract formalism may be found in Smith [15], although, that describes a fully distributed design wherein each agent is equivalent to the BSG Chair in functionality.

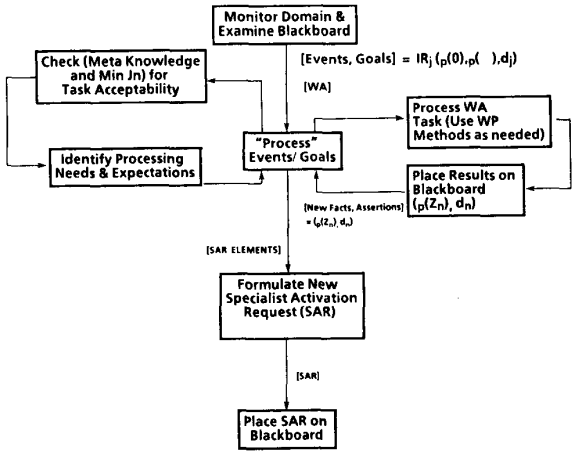


Fig. 5. Processing states associated with agent objective function.

either case the left-hand loop is then reinitialized and a new SAR is formed.

At this point it is appropriate to rewrite (1) to reflect the local conflict resolution activity of each agent plus other refinements such as agent decision sharing and dynamic, multicycle behavior. Specifically, on each cycle i the agent must

$$\left.\begin{aligned}&\text{Find } D(t)_{\text{opt}}, \text{or } \mathrm{WP}_{nri} \in \mathrm{SAR}_{nsri} \text{ in order to}\\ &\text{Min } J_n = \theta(t)\left[L_n(d_{ni}(t) = \mathrm{WP}_{ni}(K_{ni}(\theta_i(t), \theta_i', \mathrm{IR}_j(\cdot)\right]\end{aligned}\right\}. \tag{2}$$

Thus each agent has a time continuing objective function J_n discretized over $i \leqslant I$ intervals (the cycle counter). In each cycle, the agent is attempting to select the appropriate method plus knowledge baselet combination from those available in the list of SAR's. Appropriateness is measured by the loss function L_n, a BSG embedded WP method that evaluates and selects a current cycle. L_n is sufficiently similar (although simpler than) the Chairman's L_{global} such that its internals will be addressed in Section IV.

C. Pros and Cons of Distributed Problem Solving in BSG

1) Distributed Processing Objects: BSG offers the foundations of a virtual operating system that permits the knowledge engineer to combine diverse machines, vendor packages, and previously built agent modules into a cooperating entity. For applications that do not need to be distributed the logical net can be eliminated. For moderate and higher granularity concerns, however, the BSG can recognize distributed virtual addresses on the logical net.

At present, logical networks and virtual addresses are available and the knowledge engineer can experiment to find the optimum allocation of agent modules to machines. Future research would be useful for a dynamic optimizer that alleviates knowledge engineers of having to find the optimum distribution and automatically re-allocates processes to idle machines during the course of operation. References [14] and [16] are two schemes for dynamic allocation of tasks to machines that illustrate the need for research on this subject: using opposing schemes with complementary advantages and disadvantages. Research into efficient, dynamic distribution schemes is needed that combines the advantages of various existing schemes.

2) Design Guidance: BSG provides the apparatus for writing and programming a distributed objective function. Via the generic portions of the WP methods and SAR's and via the embedded processing cycle, agent's WP methods are guaranteed to be part of the distributed algorithm. Once again, the local intelligence advantages of the object orientation are exploited. In the following ways the WP method and SAR objects provide intelligent assistance beyond the simple agent purposes:

- *They are design advisors to the builder*—WP Method Objects are smart about which BB Objects they react to and post to. The user can query the Method in the pop-up menu of Figure 4b and a dependencies window will show the Method in the center with links to nodes representing the BB Object to/from that Method. SAR objects have the same design insight.
- *They are control advisors to the chair*—Dependency knowledge is sent to the Chair at runtime that the Chair compiles into its scheduler techniques (see Section V).

3) Distributed Control Concerns: With all this capability comes a lot of power and room for knowledge engineer error. To this date, the settings of the local objective function are entirely determined by the user via the WP function (WP method) extensions and via the values of the SAR slots. The present BSG design has no theorem proving techniques for assessing the validity (or effectiveness) of user defined objective function elements.[4] The advantage of specifying the blackboard model in decision theoretic terms, however, is to pave the way for future theorem proving developments.

It is felt that low and medium distributed control levels are consistent with the control models of this paper (e.g., (2)). Fully distributed control, however, offers no possibility of implementing theorem proving or suboptimality checking at a later date. For that reason, the fully, distributed control model has been avoided in BSG. In addition, while distributing the processing across various vendor packages and facilitating the incorporation of the third-party expert systems are advantages of BSG from the usability perspective, it is deleterious to the goals of distributed objective function verifiability and suboptimality avoidance. Use of a third party vendor shell introduces the

[4]While not an excuse, this is the prevailing mode in inference engine/expert system shells as well. Most shells offer no help to the knowledge engineer on the robustness of his or her KB designs (i.e., validity of the decision trees).

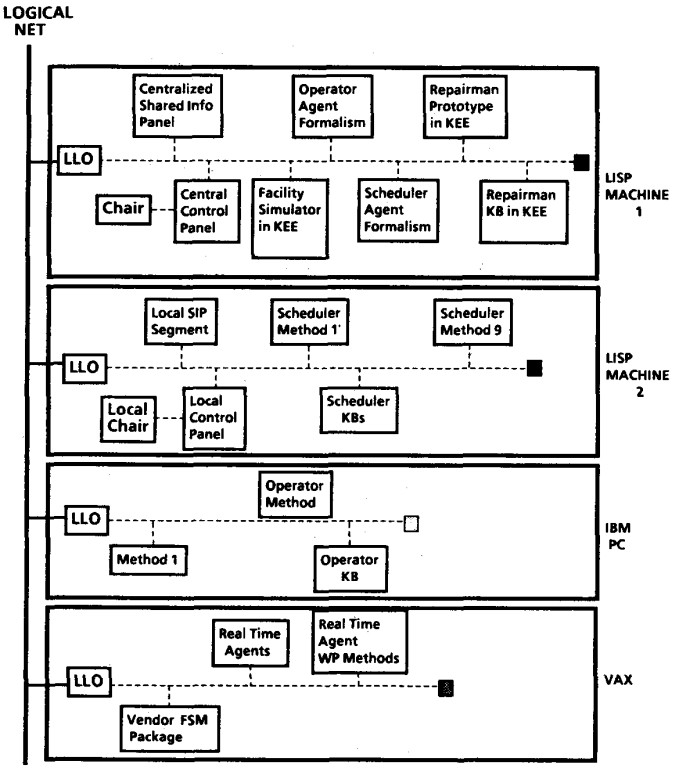


Fig. 6. Overview of BSG usage in NASA's faculty advisor (distributed problem-solving testbed).

prospect of a "black box" for some pieces of the distributed objective function. Research is needed for objective function proving techniques that can transcend and/or interpret the value of the missing pieces. Some of the work to date on competitive, rather than just cooperative, problem-solving systems is expected to become relevant to this line of investigation, e.g., [4], [20], [21].

D. Case Study: Facility Advisor DPS Tested

Many of the distributed agent design features just described have been attempted in a NASA testbed called the facility advisor. Facility advisor is a testbed for ground system autonomy techniques capable of replacing or better supporting supervisory staff at ground-based spacecraft control centers. A control center (or facility) typically requires four categories of human staff positions that facility advisor attempts to emulate: facility manager, telecommunications job scheduler, who schedules user requests to send to (receive from) their spacecraft, facility operator, and repairman.

Real facilities may have multiple individuals staffing each of these four categories of positions (and variants of them). Such staff are generally supervisory controllers who interact with the facility via a CRT or console to detect, isolate and correct anomalous behavior in the portion of the facility under their purview. They are idle if anomalies (or service requests) do not occur, but in busy periods each staff member may find himself confronted by as many as several dozen messages requiring action at any given moment.

To keep up with this workload, while simultaneously maximizing quality and quantity of service objectives, is the requirement. The problem-solving requirements of each position include: the fast pace, magnitude of jobs, simplicity of procedure needed to solve most of the anomalies (jobs), and deep complexity of a few of the reoccurring anomalies. No single calculus has yet proven pertinent to the satisfaction of requirements [4]. This is an application for which several expert system failures had already occurred, and hence, the facility advisor was conceived as a testbed of distributed problem-solving techniques applicable to an autonomous control center operation. Several thousand expert heuristics (rules) were elicited from interviews with a number of supervisory controllers. A cognitive model of supervisory protocols was also constructed that mapped the heuristics onto nine cognitive function emulators built on top of a real time situational calculus and numerous off-line calculi depending on problem type and depth of processing needed (see [4], [5]).

The testbed integrated three types of machines: Lisp machines for offline problem solving, and the VAX and IBM PC for real-time facility troubleshooting (see Fig. 6). Several off-the-shelf vendor packages were incorporated including 1) a BBN finite state machine language available on the VAX for hierarchical factory control and adapted for real time control center operations, 2) a back-chaining shell called M.1[c] on the IBM PC, and 3) the knowledge

engineering environment (Kee®) on Lisp Machine #1. Using BSG, the blackboard was designed, the agents were implemented and the various hardware and software packages were integrated.

The facility advisor is still in its relative infancy as testbed but some of the results already achieved include: 1) distributed processing at the medium granularity level; 2) distributed control at the medium case level for off-line problem solving and fully distributed for real time problem solving (a necessary "evil" due to the pace requirement); 3) parallel processing on Lisp Machine #2 of schedule contingencies to support maximizing quantity of service objectives in the face of link outage predictions generated by the repairman on Lisp Machine #1; and 4) quality of service (i.e., depth of processing) available as a function of variation in guaranteed response time required (see Section V-A for further discussion of this result).

In addition, the need has been demonstrated for establishing a long term activity devoted to the collection, testing, and refinement of distributed problem solving techniques relevant to the control center domain. Reuse of effective, validated DPS components is essential to avoid "reinventing the wheel" from mission to mission.

The facility advisor has already provided a rich domain for further researching and development of numerous BSG features including, but not limited to: 1) efficient techniques for dependency directed backtracking in a distributed processing milieu; 2) how to recognize the occurrence of a situation for which a previously solved contingency already exists (e.g., how much of a solved schedule file needs to remain in or be loaded into virtual memory in order to recognize its applicability); 3) what are the optimal clone and distribution management techniques under varying degrees of logical network traffic loading; and 4) are there design heuristics guiding the distribution of processing modules between local and remote branches of the logical net. These are topics of investigation in the computer sciences literature at large and for which there are no unique answers.

Research in these and related areas is continuing in a second phase of activity intended to build a DPS system of immediate use in the Space Telescope Operations Control Center. This will also hopefully serve as a reusable DPS module for the Space Station era advanced autonomy software.

IV. The Chair's Agent Control and Agenda Management Panel

The agent specialists are independently capable of intelligent decisions and of parallel, opportunistic reasoning. Nevertheless, it is important to ensure that all agents receive equal attention as times proceeds and that the overall project objectives are continuously being addressed. The Chair is introduced to facilitate these concerns and to manage the project agenda and work authorization schedule. Like a project manager, however, the bulk of the Chair's knowledge lies in the control protocols and procedures. It knows a minimal amount about the domain and is instead adept at initiating blackboard cycles, collecting and rating SAR's, and issuing work authorization (WA's) for specialists.

A. Four Types of SAR's

As shown in Fig. 7, the Chair consists of a schedule controller who issues WA's in response to inputs from four other Chair components. In essence the schedule controller provides a conflict resolution function (i.e., find the best SAR's to fire in this cycle) while the other components nominate four types of SAR's for consideration onto the All SAR's Agenda.

1) *Agent SAR's*: The agent/event schedule manager utilizes an agent scheduling paradigm to presort the agent SAR's received in the ith cycle.
2) *Event SAR's*: The agent/event schedule manager also utilizes an event scheduling paradigm to presort those SAR's precipitated by Blackboard events, i.e., preset guage thresholds, clock/calendar alarms, etc. That is, a threshold or alarm on a blackboard data structure can store SAR's that get sent to the Chair when cautionary or emergency conditions are reached.
3) *Clone SAR's*: A feature useful primarily in parallel processing environments is that any agent, method, etc., can be cloned and activated to work on a parallel or distributed problem. Clone scheduling becomes particularly important when events and/or plans require agent operations from already busy agents.
4) *Plan SAR's*: The meta level of intelligence should monitor the whole situation that the agents are involved in and constantly offer SAR's for events and/or agent activity it feels would be appropriate for testing of alternatives, for redirecting overall progress, and for evaluating results. Meta knowledge and plans is still an experimental component of BSG that is addressed more fully in Section V when the planning panel is introduced. For now it is only important to view it as a fourth source of SAR's for the schedule controller to manage.

B. Agenda Control and Scheduling

As indicated earlier the Chair schedules activity suggested by the various SAR's with the help of control insight it obtains from the BB Objects and Method Objects themselves. Two principal types of control information the Chair relies on are summarized in Fig. 8(a) and (b) as the work breakdown structure and method dependency graph, respectively. The work breakdown structure (WBS) is helpful as a quick sort heuristic that the Chair uses to organize SAR's into their respective work package. SAR's that support work packages the Chair has already committed to (or postponed), and can be immediately detected and placed on the CANDIDATES. SAR'S.AGENDA (or left on the

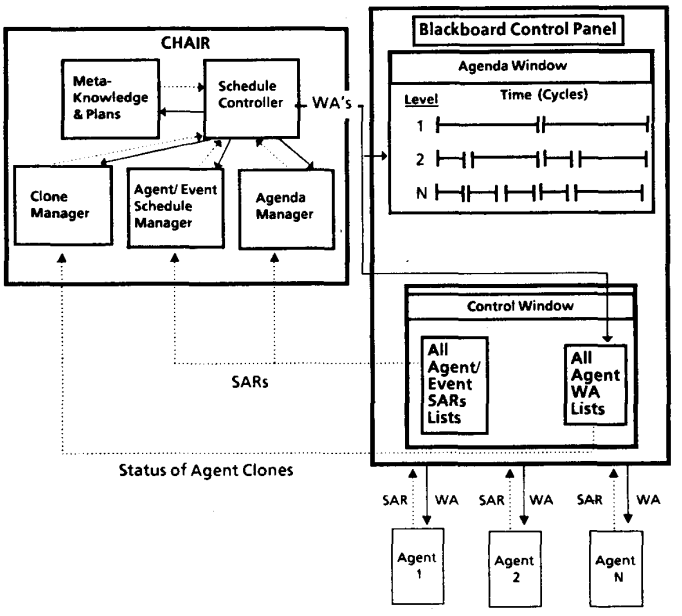


Fig. 7. Chair's control architecture. → is control and ⋯› is reporting.

ALL.SAR'S.AGENDA, or sent to the REJECTED.SAR.TRACE). The method dependency graph in turn is used by the Chair to help sort the CANDIDATE.SAR'S.AGENDA into a permissible sequence of activities (methods) to be fired and to generate the READY.SAR'S.AGENDA of those SAR's whose action does not depend on other SAR's.

The Chair next uses two sets of heuristics KB's to inspect SAR.Parmeters such as priority, probability, task duration, etc. so as to organize the READY.SAR'S.AGENDA into a PRIORITIZED.READY.SAR'S.AGENDA (using other parameters to break priority ties). The latter is the agenda of SAR's to be authorized ("WAed") in the current cycle and as quickly as the available computer resources will permit. Finally, a COMPLETED.SAR.TRACE and FIRED.WA.TRACE are maintained for backtracking and explanation purpose.

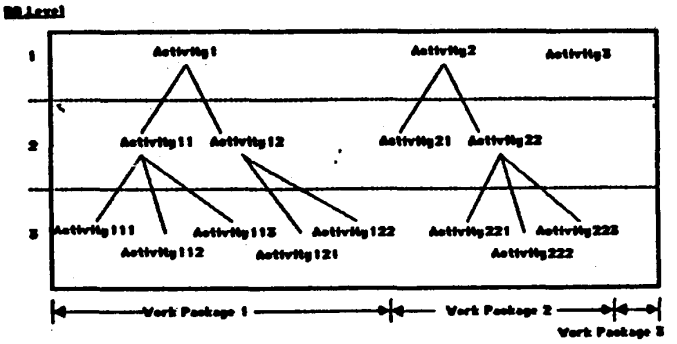
The agenda control and activity scheduling heuristics just described permit opportunistic behavior similar to the classical blackboards (e.g., Hearsay II, AGE, or BB1). By way of example, Fig. 8(c) depicts a permissible agenda in which activities of a work package are plotted by level in the vertical dimension and time is plotted in the horizontal (this is an actual BSG control panel window). This agenda shows adherence to sequences, system dependencies, and WBS levels/packages. The reoccurrence of activity 111 four separate times indicates opportunism. That is, the Chair selected the agent method that performs activity 111 to be run four separate times: 1) the first run initiated activity 1 and in part made activity 112 possible; 2) the second and third runs appears to interrupt activity 112 and corresponds to a second and third hypothesis being tested prior to the completion of 112; and 3) the final run reinitializes all of work package #1, possibly as a result of something learned in work package #2 (or due to a new state of nature).

In addition to encompassing the behavior of the classical blackboard model, the agenda of Fig. 8(c) reveals several added features. The agenda reveals activity concurrency, not just sequences of activities. Due to the virtual addressing of agents on the logical net, BSG will allocate activities to available resources. Activities stacked vertically on Fig. 5(c) exist simultaneously on the run time stack. They will be parallel processed automatically if resources are available to do so. Further, runs 2 and 3 of activity 111 may be automatically relegated to separate clones for parallel processing.

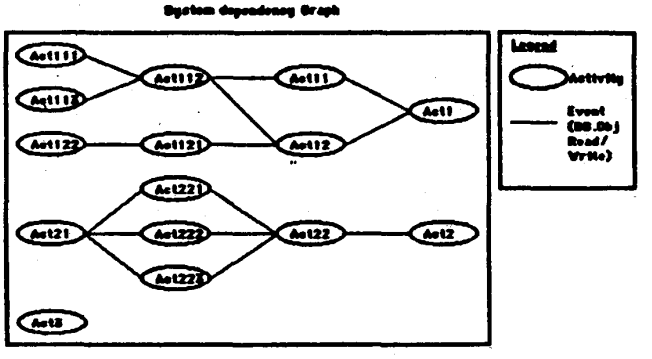
The two important features just mentioned—concurrency and opportunism—are desired in varying quantities from application to application. Some applications tend to be highly parallel while others are serial in nature. Similarly, opportunism may be rampant or virtually nonexistent in various applications. For these reasons BSG includes two toggle switches that are user-set to tailor BSG to their application: 1) the parallel/serial switch that turns the logical network on or off, and 2) the opportunism lever (low, medium, high) that shortens the number of SAR's that may be fired in the current cycle's execution tree from all (low opportunism) to only one (highly opportunistic).

The choice of settings primarily influences the speed and efficiency (number of retracted cycles/total number of cycles) with which BSG reaches its conclusions (see Section IV-E). A completely serial application has no use for a logical network routing scheme. A highly opportunistic application requires close Chair scrutiny of each new SAR proposed (i.e., after each SAR is fired, the agenda should be entirely reconstructed).

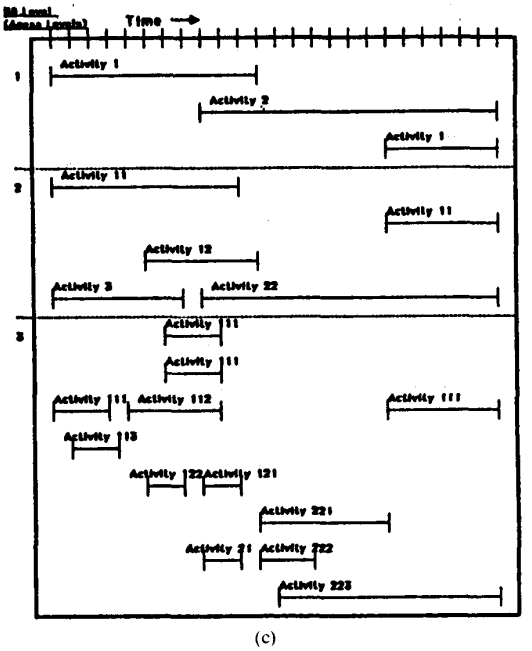
The toggle switches also permit emulation of certain control strategies described in the blackboard literature. For example, Barbara Hayes-Roth's BB1 [12] is serial and



(a)



(b)



(c)

Fig. 8. Project control paradigm of BSG's Chair. (a) WBS insight sent to Chair by BBOs and agents method objects. (b) Sequence dependency insight sent to Chair by BB's and agent method objects. (c) Typical agenda permitted by Chair.

uses one SAR per cycle, the high opportunism setting. Victor Lessor's distributed vehicle monitoring testbed (DVMT), on the other hand, would be emulated with the parallel toggle setting combined with the low-opportunism toggle setting [11].[5] The latter toggle permits execution trees containing multiple SAR's to be constructed and processed (as a plan) since a new SAR watcher is invoked that determines whether the execution tree must be interrupted due to a valid opportunity. If the interruptions are infrequent, the solution time will be significantly faster under the low opportunism setting.

C. Chair Conflict Resolution Algorithm

The Chair's conflict resolution algorithm is a restatement of (2), the major difference being that instead of evaluating only one agent's SAR's, the Chair is evaluating across four sets of SAR's: all agent, events, clones, and meta plans. The algorithm is:

1) Collect the four sets of SAR's for this cycle, i, to generate the All SAR's Agenda of SAR_{ni}. (Note: each manager prescreens his own list.)
2) For SAR's in the current work package (CANDIDATE.SAR's.AGENDA) create an execution preference tree and ensure no SAR is executed ahead of another SAR whose output it depends on.
   Thus,
   IF: $IR_{ni} = p(Z_{ji}(t), d_{ji}$ for any $N \neq j \in N$
   THEN: Schedule SAR_{ji} ahead of $SAR_{ni} \forall j$ on the candidate SAR's agenda.
3) Put those SAR's whose input is independent from other SAR's on the ready SAR's agenda.
4) Sort the ready SAR's agenda by priority value, B_{ni} into the priortized ready SAR's agenda.

Break priority ties by executing shorter, more certain processes first. Longer, less certain processes are candidates for cloning and for metaplanning.

IF: $B_{ni} = B_{ji}$, for any $n \neq j$ in SAR_{ni}
THEN: Break priority ties with process uncertainty and duration values as
reverse sort: $[U_{ni}(p_n(Z)), \text{sort}(t_{ni})]$ and construct ordered SAR's agenda.

5) Check the control panel's contingency list to ensure that no SAR is executed in the current cycle, i, that was anticipated and run several cycles earlier in i'.
   Thus,
   IF: $SAR_{ni}[WP_{nr}(Z), p(Z_n(t)), K_{nq}(\theta_i(t))] = SAR_{ji'}$ for any j on the contingency list
   THEN: Cancel SAR_{ni} (place it on the rejected SAR trace) and place results from $SAR_{ji'}$ on the blackboard shared information panel.
6) Executes as many SAR's from the ordered SAR's agenda as permitted by the toggle switch settings, update the agenda as soon as these SAR's are completed, and return to Step 1.

The Chair's resolution algorithm is executed in BSG as a knowledge base of heuristics that is editable by the knowledge engineer: the algorithm is thus only an embedded WP_{nr} method plus an editable K_{nq} knowledge base partition. The heuristics of this conflict resolution algorithm are only a starting position. For example, in some BSG applications it might be desired to sort rather than reverse sort in Step 5 of the algorithm. That is, if a distributed system tends to become communications bound, a more effective strategy is to always perform the longest tasks first. This tends to decrease communications and increase agent computation. Other similar changes are equally possible, and in its present implementation the interface permits a nonprogrammer to make changes to the conflict resolution algorithm via a scheduler heuristics edit window (which also includes the two toggle switches alluded to earlier).

D. Pros and Cons of Chair Control Techniques

One contribution of BSG to the general blackboard model lies in its ability to reduce the knowledge engineering requirement for explicit control knowledge. In many of the traditional blackboards, the Chair's control heuristics must be specified by the knowledge engineer (user) in a large number of SAR's he must write out long hand. The control of the agenda is thus entirely in the hands of the user via a trial and error approach to control. For example, in BB1 the user must decide if each method is strategy, tactic, or focus (roughly the same as three WBS levels) and label it as such. SAR's must be given weights of importance (the WBS does this implicitly) in addition to priorities, and the user must ensure all method and SAR dependencies are correctly worked out. In short, there is an overtly flexible control model offered to the user in BB1 and other blackboards.

This has several adverse effects, however, including: 1) anyone less learned than the original developers find it difficult to understand, let alone write the control SAR's; 2) since the control heuristics tend to be distributed across numerous SAR's, only an experienced blackboard programmer can program them to avoid local optima; 3) theorem proving techniques for verifying the control heuristics (and for driving towards more globally robust optima) can have little chance of success; and 4) control heuristics must be re-specified for each new application.

The incorporation of a control paradigm (model plus designer's aids) directly into BGS's chair is an attempt to overcome these beginner bottlenecks. A conscious shift to implicit control knowledge has been made. The user needs only to specify the agents, methods, BB Levels, and BB Objects and toggle settings. WBS and MDG knowledge are implicit in these structures and the BSG's chair is intelligent enough to utilize that control insight directly.

[5] Lesser *et al.* have performed numerous and extensive testbed studies of a far greater range of possible levels of opportunism and distributed architecture alternatives than are considered here. The reference here refers to only one of their configurations in which 4 BB's are given a limited planning capability.

E. Chair Toggle Switch Results Assessment

Benchmarks have been run to try and determine if optimal breakpoints can be identified for shifting the two toggle switches from setting to setting as will now be reviewed. The benchmark problem used in this paper is based on a hypothetical working configuration that consists of ten specialists operating on ten blackboard levels. Each specialist has a single method and each method has only one SAR. During each problem-solving cycle, six SAR's (specialist SAR instantiations) will be created on average, but only either one, three, or all six SAR's may be selected as candidates corresponding to the high, medium, or low opportunism toggle settings, respectively. Methods have an adjustable execution duration, which is controlled through a global variable. The entire problem needs 30 SAR's to be solved no matter what opportunism toggle setting is used. In order to examine the effectiveness of parallel processing, the ten specialists can be organized in two different setups: one is completely sequential (all methods are linearly dependent in a line) and the other is completely parallel (all methods are independent).

All benchmarks were run on Xerox Lisp machines using InterLisp (Koto release). That particular Lisp machine permits multiprocesses rather than parallel processing. There is also no way to control the CPU resource allocations between processes and no communication buffer to store incoming messages. Consequently a fix was implemented to permit the distributed version of BSG to run in that environment. In particular, three machines were used in which one machine (called the local machine) holds the Chair plus four specialists while the two remote machines hold three specialists each. Specialists in the Chair's machine cannot be activated concurrently with other specialists.

Table I shows statistics of basic operation and overhead in the current implementation of BSG. The three columns across the top of Table I correspond to the local machine time, remote machine time (including network overhead), and display time. The rows of Table I correspond to major operations associated with the blackboard and contract formalisms. The step completion times in the body of the table correspond to the average time to complete one instance of that operation.

The longer step completion times of Column 2 (remote machines) may be attributed to network overhead. The network overhead in the agent's SAR evaluation is more than twice that in BB Object operations. This is because an agent's SAR evaluation requires two communication link connections while a BB Object operation only needs one connection. The huge amount of overhead in Table I means the benchmark result in the following sections will be slower than other implementations of the distributed BSG relative to the sequential BSG would be. That is, most networks impose less overhead.

1) Degree of opportunism: The benchmark was run three separate times with the opportunism switch set to low (six SAR's in one cycle), medium (three SAR's per cycle), and high (only one SAR fired per cycle), respectively, using the sequential processing benchmark setup. These settings correspond to the three columns of Tables II(a) and (b). Execution time statistics were collected separately for: 1) the agent SAR evaluation steps (i.e., those of Section II-B-1 and Fig. 5); 2) the Chair's SAR processing steps (Steps 2–5 of Section IV-C—Step 1 is part of agent SAR evaluation); and 3) Chair/agent WA processing (i.e., Fig. 5 for agent WA processing plus Step 6, Section IV-C for Chair WA processing). These correspond to the three groups of rows along the sides of Tables II (a) and (b).

TABLE I
TIME REQUIRED TO PERFORM VARIOUS BSG FUNCTIONS (μs)

	Step Completion Times		
	On Local Machine	On Remote (Network oh)	Display Overhead
BB Object Operation			
bb get	31	864	
bb put	41	886	
Agent's SAR Evaluation			
SAR not created	248	2399	
SAR created	808	4805	
Agent's Method Invocation	$2+C^b$	$830+C$	
Basic Control Steps			
Agent's SAR evaluation			6695
Chair's SAR processing			1217
Chair's executive control			N^c567

[a] Display option switch may be turned on or off.
[b] Method duration.
[c] Number of candidate SAR's.

The results of Tables II(a) and (b) indicate that as the number of SAR's fired per cycle increase, the time required to solve the entire problem will decrease roughly following the curve:

T $A+(V/DO)$, where
T total problem solving time
A agent problem solving time (e.g., 30 C from Table II)
DO degree of opportunism (# of SAR's fired per Chair cycle)
V a constant.

This implies that a low opportunism problem being solved in a high opportunism toggle setting will result in an unnecessary slowdown, especially if the opportunism setting is close to 1, the highest setting (see Fig. 9 for the Table II(a) data).

Without considering the network overhead, both Tables II(a) and II(b) are very similar except the percentage of time spent in an agent's SAR evaluations in the multiple machine case is less than that in the single machine case. That is due to parallel SAR evaluations in multiple machines.

2) Degree of Distributed Problem Solving: The differences between distributed and sequential BSG are quite simply that the former: 1) routes all agent and chair

TABLE II
TIME REQUIRED BY VARIOUS BSG COMPONENTS (μs) WITH AND WITHOUT DISTRIBUTED PROCESSING AND AT VARYING DEGREES OF OPPORTUNISM

	Time to Compute 30 SAR/WA Sets					
	High Opp. (1 SAR)		Medium Opp. (3 SAR's)		Low Opp. (All 6 SAR's)	
	Time	Percent	Time	Percent	Time	Percent[a]
A. SINGLE MACHINE-SEQUENTIAL PROCESSING						
Agent's SAR Evaluation						
Total	112844	76	36025	55	17210	38
Cycle average	3761		3602		3442	
Chair's SAR Processing						
Total	16368	11	10190	15	9020	20
Cycle average	545		1019		1804	
Chair and Agent WA Processing						
Total	$20176+30C$[b]	14	$19675+30C$	30	$19079+30C$	42
Cycle average	$672+C$		$1967+3C$		$3815+6C$	
Grand Total						
Entire problem	$149388+30C$		$65890+30C$		$45309+30C$	
Cycle average	$4978+C$		$6588+3C$		$9061+6C$	
B. MULTIPLE MACHINES, SEQUENTIAL PROCESSING						
Agent's SAR Evaluation						
Total	1143030	57	350980	35	199000	24
Cycle average	38101		35098		39800	
Chair's SAR Processing						
Total	187230	9	71520	7	43195	5
Cycle average	6241		7152		8639	
Chair and Agent WA Processing						
Total	$663420+30C$	33	$572160+30C$	58	$570790+30C$	70
Cycle average	$22114+C$		$57216+3C$		$114158+6C$	
Grand Total						
Entire problem	$193680+30C$		$994660+30C$		$812985+30C$	
Cycle average	$66456+C$		$99466+3C$		$162597+6C$	

[a] Percent is computed assuming method completion time = 0.
[b] C is a variable corresponding to the method completion time.

communications over the logical net; 2) implements each agent (and user-specified agent methods) as distinct physical processes; and 3) permits concurrent parallel processing on distributed machines available over the logical net (three machines for the benchmark). The parallel processing power speeds up agent's SAR/WA processing, but the effectiveness is bounded by the number of machines available, machine load (agent distribution), and agent's method execution duration. Inorder to better understand these bounds let,

$$\text{degree of parallelism} = \frac{\#\text{ of running WA's}}{\#\text{ of running WA's} + \#\text{ of SAR's in ready SAR's agenda}}$$

Degree of parallelism equals 1 means all executable SAR's are running (an obviously ideal situation corresponding to one machine per WA). If the degree of parallelism is less than 1, say $\frac{1}{3}$, there are only $\frac{1}{3}$ of the executable SAR's running and the rest of the executable SAR's are waiting for free machines (WA is pending). Delay may be a result of either too few machines in the system or the problem of load imbalance. Table III presents the rows corresponding to two, four, and all six executable SAR's while varying degrees of parallelism are the columns. Our experiment only permits two machines

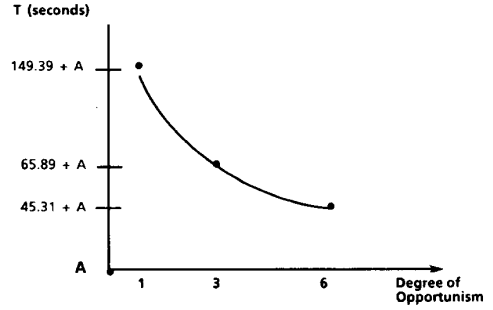


Fig. 9. Opportunism setting versus completion time trade-off curve. Note: A is $30C$ for benchmark, T is WP method completion time, and V is constant. $T = A + (V/DO)$.

working in parallel (Chair is on the third machine), hence, the theoretical maximum speed up rate for any of these cases is 50 percent.

The second column of Table III shows that for each degree of parallelism the results were repeated for each of three levels of WP method delay: no method delay, 10 seconds, and 100 seconds. Finally, there are only two degree of parallelism results per case since the cases can only be run on either one or two machines. For example, two READY.AGENDA.SAR's may be run on two machines (degree of parallelism = 1) or on one machine (degree of

TABLE III
SPEEDUP RESULTS DUE TO VARYING DEGREES OF PARALLEL PROCESSING (MULTIPLE MACHINE, PARALLEL PROBLEM)

	Agent Method Delay (s)	With Parallel Processing		Without Parallel Processing		Speedup Percent
		Time (s)	Degree of Parallelism	Time (s)	Degree of Parallelsim	
Two SAR's	0	39.1	$\frac{1}{2}$	41.2	1	−5
	10	59.1	$\frac{1}{2}$	54.9	1	7
	100	239.1	$\frac{1}{2}$	147.1	1	38
Four SAR's	0	72.1	$\frac{1}{4}$	80.0	$\frac{1}{2}$	−11
	10	112.1	$\frac{1}{4}$	106.6	$\frac{1}{2}$	5
	100	472.1	$\frac{1}{4}$	299.1	$\frac{1}{2}$	37
Six SAR's	0	114.2	$\frac{1}{6}$	194.1	$\frac{1}{3}$	−70
	10	174.2	$\frac{1}{6}$	234.3	$\frac{1}{3}$	−35
	100	714.2	$\frac{1}{6}$	508.1	$\frac{1}{3}$	29

parallelism $= \frac{1}{2}$). The percent speed up corresponds to the speed up of two machines rather than just one machine for the same number of ready SAR's.

The results shown in Table III approach the theoretical maximum of 50 percent speed up for each cluster of cases attempted as method duration lengthens. The negative percent speed up may be explained by the overhead of multiprocesses handling that becomes apparent when the method delay decreases.

As mentioned earlier, the effectiveness of parallelism is bounded by two factors: the number of machines available and machine load balance. For example, in Table III, the case of two SAR's with degree of parallelism 1/2 represents a load imbalance, since only one of the two machines is being utilized. The same degree of parallelism in the case of four SAR's is not an imbalance since all machines are busy.

The two different causes of low degree of parallelisms can be distinguished by another measurement, machine utilization. This commonly is defined as

$$\frac{\text{Total machine busy time}}{\text{Total operations time}} \leqslant 1.$$

If degree of parallelism is low, low utilization may be a good indication of load imbalance. On the other hand, low parallelism with high utilization indicates more machines may be needed. If the degree of parallelism is high, the performance cannot be upgraded by adding more machines, unless a different level of processing granularity is also attempted.

In conclusion, the two toggle switch settings in BSG provide a means for users to adjust their applications to an optimal working point. Without sacrificing the needed opportunism to guide problem solving, increasing the number of fired SAR's per cycle significantly speeds up the performance. The best setting may be input from users or automatically adjusted depending on the problem solving state. The degree of parallelism initially depends on the parallelism toggle setting and the agent distribution across machines. The performance can be monitored through the two parameters, degree of parallelism and machine utilization during problem solving. The logical network and cloning mechanism can be used easily to reallocate machines and re-distribute agents, in order to maximize degree of parallelism. A performance monitoring system that facilitates these features will be implemented in a later stage of BSG development.

V. BSG PLANNER PANEL

A generic planner is not an essential component of the general blackboard model and the BSG components described to this point are sufficient to implement a relatively comprehensive blackboard application with. Indeed, except for [11], many of the blackboards built to date have no planner as such and their builders see no reason to add one, e.g., [1], [2], [10], [12]. Instead, the knowledge engineer is expected to explicitly code planning knowledge into the individual SAR's by altering parameter levels as a function of the current situation. That is, SAR parameters such as priority or probability are dynamically altered during runtime to alter the relative importance of a given activity, situation-by-situation (a parameter lookup table is often used).

The research objectives for BSG, however, include trying to isolate an underlying planning model that can help the Chair assure that more global optima are being located and that help alleviate the knowledge engineering bottleneck of asking users to isolate planning parameter levels on a case by case basis for their particular domain.[6] The purpose of a planner is thus to provide a semantic level of insight to the control knowledge that the BSG control subsystem already sees. The BSG Chair's control technique, and most other blackboards for that matter, may be viewed as a syntactic interpreter of plan inputs. It is adept at control decisions but has no higher level semantic understanding of their implications.

[6]Inroads into this topic have also been made by Hayes-Roth in her BB* work [23].

TABLE IV
RESOURCE ALLOCATION PLANNER'S SEMANTIC UNDERSTANDING AND RELATIVE IMPORTANCE OF SAR PARAMETERS

	Resources To Be Allocated						
Situation	Time to Achieve Event/Milestone	Critical Path (CP) Selections	Clone SAR Requests	CPU Allocation	Agent Method Calculus	Solution Accuracy and Quality	Analogical Reasoner
Real time (guaranteed solution time) problem solving sessions	favor shorter CP activity duration	favor shorter CP plans	favor shorter CP activity method	favor chair allocation EOS's	favor situational calculus and/or shallow KB methods	accept local optima	favor lessons learned lookup SAR's
Learning and investigative sessions	favor longer task durations	favor longer CP plans	favor hyothesis testing method Clons	favor agent allocation EOS's	favor agent nondeterministic and/or deeper KB methods	strive for more global optima	favor disanalogy elimination SAR's

Without a planner, the Chair may be susceptible to knowledge engineering errors (in SAR parameter settings). For example, the Chair in Section IV-A was able to accommodate an opportunistic request to run activity 111 four separate times (see Fig. 8). However, the Chair had no internal insight as to whether that was a good idea. It simply parsed parameter settings and syntactically reacted as it was programmed to do. It had no basis for determining whether four hypothesis tests might be inappropriate, for example, in light of a guaranteed response time objective.

This is but one example of a number of resource allocation semantics the planner could enlighten Chair control about. Note, however, that a planner imposes a serial overhead cost. At present the planner panel of BSG is experimenting with several major categories of resource allocation semantics as summarized in Table IV and as further described in the two following sections in terms of: 1) a resource allocation planner, and 2) an analogical model based planner.

A. Resource Allocation Planner

The resource allocation planner has been one of the subjects of the ongoing facility advisor experiments aimed at isolating generic planning elements relevant to the resource allocation concerns of Table IV: see [4] and [5] and Section III-D. In those experiments a testbed has been constructed, called facility advisor, for implementing and benchmarking distributed AI techniques appropriate to spacecraft control centers in particular, and to any supervisory control facility in general. In fact, the resource allocation planner techniques being developed under facility advisor are felt to be entirely generic and will be presented as such in what follows.

First it is important to note that the WBS and dependency graph inputs to the Chair (i.e., earlier Figs. 8(a) and 8(b)) are not plans: they are planning inputs. For example, the dependency graph plus individual activity duration estimates (a SAR parameter) can be used by the planner to construct a plan. They are not yet a plan, and to transform them into one, yet another project planning technique is useful—that of PERT/CPM [22]. PERT/CPM plots activity completion times into the dependency graph (Fig. 8(b)), and isolates those activities that comprise the longest route through the activity network that corresponds to the shortest time in which the project (problem) can be completed (solved). Activities that lie on the critical path (CP) route must be completed on time or else the problem will take longer to solve (other activities have slack).

If a guaranteed solution time requirement were imposed, the planner would use its PERT/CPM methods to seek out CP activities with shorter durations (i.e., lower quality, locally optimal, and/or shallower solutions) or shorter CP plans achieved by efficient machine time allocations and CP activity method cloning. The planner would also have to monitor actual progress being made toward the solution time headline and would need to generate new plans in response to deviations.

In a similar vein, when the situation calls for learning or investigative study rather than real-time solutions, plans need to be generated that favor longer search times, more hypotheses being tested, nondeterminisms being more heavily considered, and time spent by the analogical reasoner to study how past solutions might support the current situation. This latter topic is addressed more fully in Section V-B. It is worth noting that under time constraint the analogy planning subsystem will tend to be bypassed for a simple lessons-learned-lookup operation whereas in the absence of a time constraint disanalogies between the past and present problem–solution pairs can be more fully investigated as will now be elaborated.

B. Analogical Model-Based Planner

As shown in Fig. 10, the heart of this capability is a controlled generic leaning based analogical reasoner that takes a snapshot of the problem–solution pairs currently unfolding on the shared information panel (and on the AGENDA.TRACE.WINDOW) and compares them to past or analogous problem–solution pairs that are already fully defined and that are stored in the analog KB or AKB. If

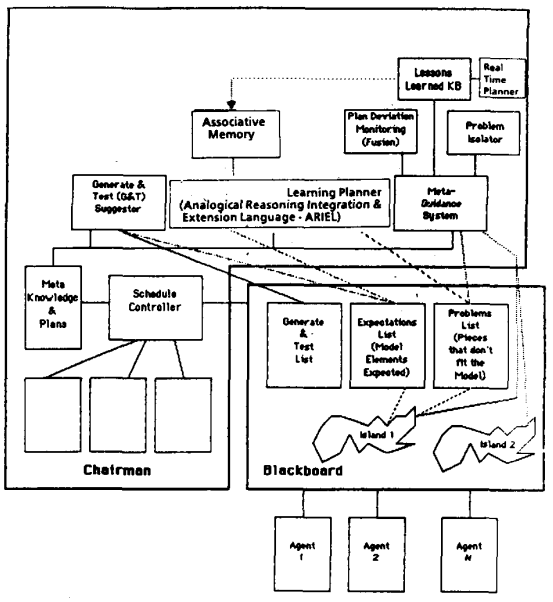


Fig. 10. Details of Chairman planning. → is control and ⋯→ is reporting.

current problems (or problem elements) are seen to match[7] past, analogous ones within a given tolerance, the past problem–solution pair is used as a model of what's expected to happen in the current situation. This analog or model contains a great deal of detail about past problems, $d_n(t-l)$, and solutions that worked (i.e., $\mathrm{WP}_{nr}, \mathrm{K}_{nq}$, $d_n(t)$, $p_n(Z(t))$, etc) and has been implemented as a connectionism concept (modified neural net). That information is placed on the BSG expectations list as a statement of what is likely to unfold in the upcoming cycles. Also, the agents may tap that knowledge to facilitate their own search and problem-solving behavior. As future cycles unfold, the shreds of knowledge (WP, K, d, p, etc.) that begin to deviate from the model are detected by the metaguidance system and are placed on the problems list (see Fig. 10). The problem isolator hypothesizes likely sources of deviation as does the learning based planner, both of which stimulate the generate & test suggestor to create a set of SAR's for alternatives to be investigated by the attached agents. These are the plan-generated SAR's that are placed on the generate and test list on the BSG Planning Panel. These plan-generated SAR's are one of the four types, mentioned earlier, that must be scheduled by the Chairman's schedule controller. The lessons learned *KB* is used to store results of what did and did not work when a new problem is being solved. After that problem is solved the lessons learned *KB* file is transferred to the *AKB*.

[7]Matching here uses a combination of Tverskian attribute weighting, semantic net link distances and net sibling relations.

1) Status of the metaplanning components: The designs, algorithms, and code for Ariel are more fully as described in [6], [7] and example metaguidance system algorithms are described in [4]. A prototype implementation of the entire design was accomplished in about 11 000 lines of code on a Lisp machine in the first half of 1986 as documented in [7] with a follow-up refinement and beta-test effort scheduled for completion in late 1988 as described in [8].

2) Overview of the metaplanning algorithm: The simplest possible algorithmic overview of what was described previously is as follows.

1) Monitor $[\theta_M \rightarrow D_N]$ pairs unfolding at all levels of the shared knowledge panel.
2) Use Ariel to:
   a) identify and genetically merge analogous $[\theta_M \rightarrow D_n]_i^A$ pair(s) making particular use of the semantic information contained in the current problem: $D_n[\mathrm{WP}_{nr}$ (triggering features slot), K_{nq} (typology symptoms slot)].
   b) fill expectation list with detailed analogical knowledge (obtained from associative memory) of the $[\theta_M \rightarrow D_N]^A$ pair such as

$$(\mathrm{WP}_{nr}, \mathrm{K}_{nq})^A \text{ pairs for all } i$$

$$(\theta_m, p_n(Z))^A \text{ pairs for all } i$$

$$(d_n)^A \text{ for all } n, i.$$

   NOTE: Expectation list becomes a Min J_{global} hypothesis.
3) Use the meta guidance system to monitor Min J_{global} = expectation list as suggested by
   IF: $\left|[\theta_M \rightarrow D_N]_i^A - [\theta_M \rightarrow D_N]_i\right| >$ threshold
   THEN: Set uncertainty alarm, and isolate offending agent/deviating elements.
4) Use the metaguidance system to isolate sources of deviations as suggested by:
   IF: emergency > uncertainty alarm > threshold
   THEN: set B_{ni} to caution level, AND
   send G and T the source of the deviation, $(\mathrm{WP}_{nr}, \mathrm{K}_{nq})$
   ELSEIF: uncertainty; alarm $\geqslant$ emergency
   Set B_{ni} to $\gg 1.0$
   and set G and T the source of the deviation, $(\mathrm{WP}_{nr}, \mathrm{K}_{nq})$.
5) Use G and T suggestor to generate SAR's for the relevant agents to investigate whether they can (or should) try to achieve the expected output. This is a request to justify the deviation.
6) If a correction cannot be tendered or if an acceptable justification is offered, the expectation list must be modified and this algorithm repeats.

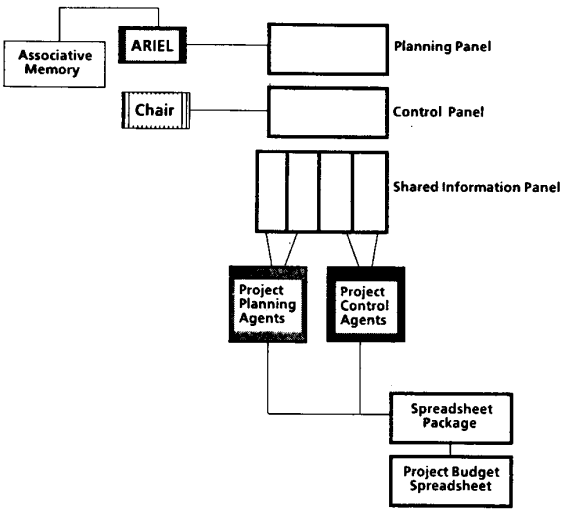


Fig. 11. Architecture overview of expert project management system (EPMS).

VI. Application of Analogical Meta Planning to the NASA Project Management Domain

A prototype called the expert project management system (EPMS) generator was designed using BSG plus Ariel to support the analogical planning plus project control needs of NASA project managers (PM's). EPMS is intended itself to be a shell for the project management domain that is organized as indicated in Fig. 11. EPMS is too large to be fully described here. Basically, it has a set of project planning agents and project control agents that help plan and control the project budget (which is stored in a spreadsheet system) and that communicate via the shared information panel. The planing agents are supported by Ariel whom they depend on to retrieve analogous situations (problem solution pairs).

When presented with a few high level symptoms or requirements of a new spacecraft, EPMS locates analogous spacecraft and presents project budgets (cost and staff) by milestone, subsystem, and other dimensions. The PM user (with EPMS's help) selects which elements of the past plans (problem–solution pairs) seem most appropriate and enters these (or modifications of them) as his plan (expectation list). EPMS agents monitor actual project progress, cause the screen to blink as deviations from the plan occur, and shows the user PM the project submanager(s) who was responsible for the deviation as well as what the deviation was. This sequence may be repeated for replanning if the PM user so desires.

The accuracy and usefulness of the analogical reasoning process of EPMS (and Ariel) improves as more domain analogs are saved away in associative memory. The archival associative memory requires an unusual knowledge representation scheme to facilitate rapid search and retrieval. In particular, each analog is stored as three semantic nets; one for the problem, a second for the solution to that problem, and a third net consisting of predicate mappings (holograph pointers) between various problem and solution objects (nodes). The associative memory in turn stores these triplets in yet another net, a context net, that organizes the domain into classes and types of situations.

These knowledge representation requirements have been combined with the BB Object representation requirements, dynamic hypothesis management concerns, and random access object needs (Section IV). Earlier versions of BSG included several alternative representation techniques for these varying requirements. At the time of this writing a unified, modular representation technique called the semantic network language (SNL) is being implemented that will satisfy the peculiar set of requirements imposed by all of these requirements. In addition to features already addressed, SNL is being designed to also support improved default logic and nonmonotonic reasoning, as well as simplifies frame representation (without inheritance).

VII. Conclusion

This paper has reviewed the status of a generic blackboard-based distributed problem-solving environment in which multiple agent cooperation can be effected. This environment is organized into a shared information panel, a chairman control panel, and a metaplanning panel. Each panel, in turn, contains a number of embedded AI techniques that facilitate its operation and that provide heuristics for solving the underlying team-agent decision problem. The status of these panels and heuristics has been described along with a number of robustness considerations.

Blackboard Object Orientation: Unlike other blackboards, BSG stores hypotheses on the blackboard as object slots. This facilitates exploratory design and message passing. Equally important, the use of objects permits BB Objects to be intelligent in several ways: they advise knowledge engineers on how to create them; they recommend agent sequences to the Chair; and they provide a foundation for implementing error diagnostics (not finished yet). One concern about objects is their potential slowness relative to relational data structures such as in GBB [10]. For this reason a random access, dynamic hypothesis management capability is currently being incorporated into the BSG object paradigm.

Distributed Agent Implementation: BSG permits the use of a logical network with virtual addresses denoting the various agent locations. Virtual addressing permits agents (or agent subprocesses) to exist on any machine addressable on the file server. BSG's design also facilitates agent instantiating or cloning to use idle resources, multiple BSG applications accessing the same agent, and/or connection to (embedding behind) traditional off-the-shell packages (e.g., OPS-5, Lotus, DBIII$^+$, etc.). More importantly the virtual addressing/logical networking functionality is the keystone to facilitating low, medium, and high granularity

parallel processing as well as the gamut of distributed agent control options from none to complete. At present these features have been achieved for low and medium categories of distributed processing/control. A future goal is to extend the capability to the highest granularity users for a wider set of distributed processing categories. However, it is argued that fully distributed agent control may encourage suboptimal solution selection.

Generic Control Paradigm: In the area of chair control an effort has been made to research and develop an underlying control paradigm. The designer's mental model has been elicited and partially embedded in BSG to offer intelligent design assistance. The control paradigm embedded in BSG allows it to guide domain knowledge engineers in specifying the activity-event relationships and dependencies implicit in agent methods and BB Objects. Several semesters of graduate student application projects and three BSG applications to date have served to verify the value of the control technique elicitation and paradigm.

Control technique flexibility on the other hand has been maintained by providing knowledge engineers with a control heuristics editor window in which they can reset BSG default heuristics; reset toggle switches for degree of opportunism and degree of distribution permitted; and edit the control rules BSG elicited from them.

Future research in the blackboard field as a whole is needed into still further ways to facilitate blackboard knowledge engineering; to design and test additional generic control heuristics; and to develop theorem proving techniques that can be applied by the Chair to verify correctness and optimality of control decisions taken.

BSG Planning: BSG, and most other blackboards existing today operate with rather simplistic, syntactic planning techniques. Research and development is currently underway to imbue BSG with a semantic understanding of the plans it is controlling and with more intelligent planning capabilities. In particular, two planning paradigms are currently under active development for BSG: guaranteed response time planning and analogical reasoning via a learning and associative memory capability. These paradigms are intended to support two major categories of applications: supervisory process control and reusable solution domains, respectively. Research is needed in generic blackboard planning techniques for yet additional types of applications as well.

Several applications of the blackboard system generator (BSG) to automobile failure diagnosis, to real-time satellite command and control, and to project management have been cited along with some of their results. The need for a number of research, development, and algorithm proving investigations have been delineated. Even without these investigations, blackboards (and BSG) are encountering more widespread usage. A number of BSG projects are underway in case-based reasoning, machine learning by discovery, template-driven knowledge acquisition aids, and in several more domain-relevant applications, that will be utilizing BSG quite heavily [4]–[9] and that will be reporting further results as they occur.

Acknowledgment

The authors gratefully acknowledge a recently awarded U.S. Army SBIR that will support continuation of future research.

Finally, since beginning this article in the Fall of 1986 and finishing the first draft (Winter 1988), many of the ideas expressed here indeed appear to reflect the opening quote. The underlying metaphor (project management) has blossomed into a full-fledged theory and formalism of distributed problem-solving and cognitive modeling. Looking back at this paper from the Winter of 1988 there is no simple way to update it to reflect how far we have come. It should be read as an interim set of fruitful ideas (seedlings) that lead our group to a healthier result in a short span of time.

References

[1] P. Nii, "Blackboard systems," *AI Magazine*, pp. 1–13, Spring 1986.
[2] V. R. Lesser, R. D. Fennell, L. D. Erman, and D. R. Reddy, "Organization of the Hearsay-II speech understanding system," *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-23, pp. 11–23, 1975.
[3] T. Quigley, D. McKenna, and B. Yoon, Term project report, available from the GWU Institute for Artificial Intelligence, Dec. 1986.
[4] B. G. Silverman, "Distributed inference and fusion algorithms for real time control of satellite ground systems," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-17, no. 2, pp. 230–239, Mar. 1987.
[5] ——, "Facility advisor: A distributed expert system testbed for spacecraft ground facilities," *Expert Systems in Government Symp. Proc.*, IEEE-CS Order No. 738, Oct. 1986, pp. 23–32.
[6] B. G. Silverman and V. Moustakis, "INNOVATOR: Representations and heuristics," ch. 17, *Expert Systems for Business*, B. G. Silverman, Ed. Reading, MA: Addison-Wesley, 1987.
[7] B. G. Silverman, A. Murray, C. Diakite, and K. Feggos, "Analogical reasoning in the expert project management system (EPMS)," in *1987 Goddard Conf. On Space Applications of Expert Systems*, NASA/GSFC/Code 514, Greenbelt, May 1987.
[8] D. W. Garrett, "NASA selects small business proposals," in *NASA News*, Dec. 9, 1986.
[9] J. Simkol, G. Wenig, and B. G. Silverman, "JAMS: A computer aided electronic warfare vulnerability assessment (CA-EWVA) technique," *Symp. Aerospace Applications of AI*, AFCEA, May 1987.
[10] D. D. Corkill, K. O. Gallagher, and K. E. Murray, "GBB: A generic blackboard development system," in *Proc. AAAI-86*, pp. 1008–1014, 1986.
[11] V. Lesser and D. Corkhill, "The distributed vehicle monitoring testbed," *AI Mag.*, vol. 4, no. 3, pp. 15–33, 1983.
[12] B. Hayes-Roth, "Blackboard architecture for control," *Artificial Intell. J.*, vol. 26, pp. 251–321, 1985.
[13] R. S. Englemore, "Overview of blackboard tools," in *Blackboard Workshop Proc.*, Seattle, WA, June 1987.
[14] R. Smith, *The Contract Net Formalism*. Ann Arbor, MI: Univ. Mich. Press, 1983.
[15] L. Erman, M. Fehling, S. Forrest, and J. Lark, "ABE: Architectural overview," presented at *Workshop of Distributed Artificial Intelligence*, Palo Alto, CA, Teknowledge, Inc., 1986.
[16] V. Singh and M. R. Genesveth, "Variable supply model for distributing deductions," *IJCAI-85 Proc.*, Palo Alto, CA, pp. 39–45.
[17] R. A. Finkel and J. P. Fishburn, "Parallelism in alpha beta search," *Artificial Intell. J.*, vol. 19, no. 1, pp. 89–106, Sept 1982.
[18] S. J. Stolfo, "Five parallel algorithms for production system execution on the DADO machine," *AAAI-84 Proc.*, 1984, pp. 300–307.
[19] R. D. Fennel and V. Lesser, "Parallelism in artificial intelligence problem solving: A case of Hearsay II," *IEEE Trans. Computers*, vol. C-26, no. 2, pp. 98–111, Feb. 1977.

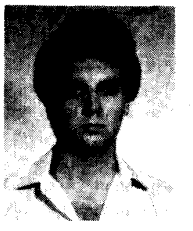
[20] T.Azarewicz *et al.*,"Multi-agent plan recognition in an adversarial domain," 1987, *Expert Syst. in Gov. Conf. Proc.*, J. Benoit, J. Antoinesse, B. G. Silverman, Eds. Silver Spring, IEEE Computer Society, Oct. 1987.
[21] Y.-C. Ho, "Team decision theory and information structures," *Proc. IEEE*, June 1980, vol. 68, no. 6, pp. 644–654.
[22] R. D. Archibald, *Managing High Technology Programs and Projects*. New York, Wiley-InterScience, 1976.
[23] B. Hayes-Roth, "A layered environment for reasoning about action," Stanford University HPP Tech. Report, Stanford, CA, 1986.

Joseph S. Chang received the B.S. and M.S. degrees in computer science, in 1983 and 1986, respectively, from the University of Maryland at College Park. He is a senior research programmer at IntelliTek, Inc., Rockville, MD, where he is the lead product developer for the Blackboard System Generator (BSG). He has developed a control architecture that incorporates a metarule system for multiple control strategies, goal-directed and data-directed reasoning for focus of attention, and an execution monitoring system for feedback control and dynamic planning. His current research includes techniques for multi-agent distributed problem solving.

Mr. Chang is a member of AAAI.

Barry G. Silverman (M'78–SM'83) was born in Boston, MA, in 1952. He received the B.S., M.S., and Ph.D degrees in systems theory in 1975, 1976, and 1977, respectively, from the University of Pennsylvania, Philadelphia.

He is Director of the Institute for Artificial Intelligence, and Professor of Engineering Administration at The George Washington University, Washington, D.C. Since 1980 he has specialized in research on algorithms for analogical reasoning and distributed problem solving and has produced 100 technical reports, over 50 journal articles, and several books on these and related topics. He is also directing several teams of researchers currently building a set of software tools to implement these algorithms for teaching, research, and development purposes under a variety of governmental and industrial sponsorships.



Kostas Feggos received his B.S. in civil engineering in 1983 from the University of Patras, Patras, Greece and his M.S. in engineering administration in 1986 from the George Washington University, Washington, D.C.

He has been working as a Research Engineer with the Institute for Artificial Intelligence of the George Washington University since 1985, where he is currently pursuing a Ph.D. degree in engineering administration.