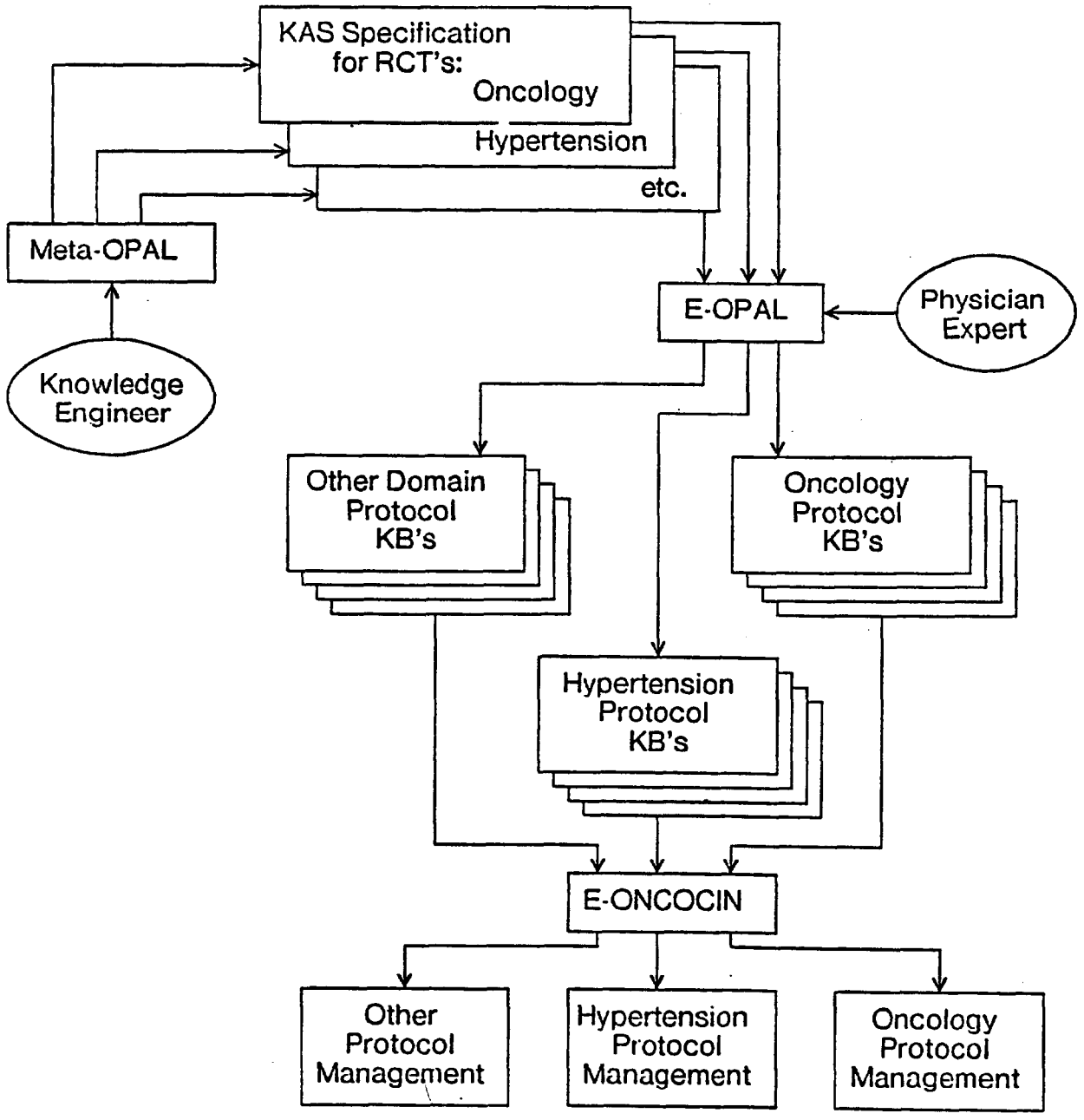

Figure 8: Components of the Meta-OPAL System

reprogramming the whole system and revising the format of the Intermediate Data Structure (IDS). However, if the operation of OPAL itself were completely and explicitly defined in a very high level syntax, it would be far preferable to use Meta-OPAL to revise this representation, automatically creating a new knowledge acquisition system (KAS) that would incorporate (1) the necessary windows and menus, (2) an appropriate IDS format, and (3) specifications for how the IDS should be translated into E-ONCOCIN knowledge bases.

Meta-OPAL and the necessary KAS definition language will allow us maximum flexibility in adopting OPAL for unusual protocols that might be encountered in the future. If the KAS definition language is general enough, it will allow knowledge acquisition for clinical trials outside of the domain of oncology. Because the ONCOCIN inference engine (E-ONCOCIN) makes no specific assumptions about oncology, Meta-OPAL could produce knowledge acquisition systems that would permit physicians to enter new protocols for any kind of clinical trial; E-ONCOCIN could then be used for patient consultations and for data management.

The Domain of Clinical Trials is Well Suited for Meta-OPAL:

Just as the structure of knowledge within clinical protocols is generally easy to anticipate, knowledge *about* clinical protocols is equally predictable. For example, the sequence of interventions to take in any clinical trial should always be representable as a schema. This schema might be similar in syntax to that now used by OPAL to express the order of treatments in cancer protocols. Other representations might be more appropriate for RCT's in different domains. In oncology,

```
CHOP x 6
```

is quite satisfactory. However, stepped care for hypertension might be better expressed using a format such as:

```
STEP 1: Hydrochlorothiazide
STEP 2: add Labetolol
STEP 3: add Captopril
```

Our initial work on Meta-OPAL will include developing a complete and unambiguous syntax for specifying protocol schemas. Part of the KAS definition language will involve declaring formatting options and what entries are permissible when a schema is entered into the resultant KAS.

Knowledge about clinical trials is predictable in other ways. For instance, all protocols list a host of laboratory test results and clinical conditions that must be recorded and that may cause an alteration in the treatment plan. The number of ways in which therapy may be modified within a given class of protocols is finite; these kinds of actions will have to be specified in the KAS definition language.

Knowledge acquisition systems for RCTs also can capitalize on another constraint in their domain: patients with concurrent diseases that might complicate analysis of the study are excluded from participating in protocols. The scope of the knowledge needed for a given expert system can therefore be limited to the one disease under investigation. The task of designing a KAS for a given class of clinical trials is clearly simplified when the scope can be focused in this way.

Although there are many different kinds of clinical trials, knowledge about such studies is always formalized in a protocol document. Examining protocol documents will allow us to generalize about what characteristics are required for knowledge acquisition systems in each of the domains studied and provide the basis for developing Meta-OPAL and its KAS definition language. Our experience in developing and using OPAL will also be essential in guiding our design for Meta-OPAL.

How Meta-OPAL Will Work

The user interface to Meta-OPAL will involve the same menu-driven approach we have adopted in the design of OPAL. Many of the usual difficulties of communicating with the computer will again be obviated by the use of graphics-based editing.

The knowledge engineer, in conjunction with a physician-expert, will use Meta-OPAL to specify the general nature of the clinical trials involved (e.g., sequential therapy, stepped care) and certain study design issues (e.g., cross-over trials, repeated randomizations). The expected modalities of treatment will also be declared. Meta-OPAL can then establish the schema syntax for the protocols to be entered.

The program will then assist the user in structuring the IDS. The names and meaning of each of the IDS entries will then be declared. The relationships among the various IDS components will be specified using graphics.

A list of all knowledge base *parameters* will then be declared and the same rule definition language that we will develop for OPAL will be used to specify how the rules to conclude each of the parameters can be generated from the IDS. Parameters will fall into several categories (e.g., clinical conditions, concluded drug dosages, intermediary parameters used in the reasoning process) and the nature and use of each parameter will have to be specified. For example, the user must specify which parameters correspond to items that must appear on a patient's "flow sheet" when displayed by E-ONCOCIN. Similarly, it will be necessary to indicate those parameters whose values will represent the system's "recommendations" during a consultation.

Finally, Meta-OPAL will prompt the knowledge engineer with the basic information needed for each of the windows that will appear in the completed KAS. The exact window formatting will then be entered by selecting locations on the screen with the mouse and typing in the text that should appear there when the KAS is generated. If dissatisfied with the location of particular blanks, the knowledge engineer will be able to use the mouse to rearrange the formatting. For each blank, the system will ask the knowledge engineer to specify the corresponding menu that will appear when the blank is selected by the KAS user. The knowledge engineer must also indicate where the entry for the blank is to be stored in the IDS and any information needed to check for completeness or consistency.

Once the user has completed entry of information into Meta-OPAL, a new data file will be created that will contain all of the specifications of the KAS. This file will serve as input to the E-OPAL program, which will follow the file's guidance in displaying windows and gathering data during the knowledge acquisition process. The information will be in a format that can be modified by a standard text editor, if necessary, as well as by Meta-OPAL. The knowledge will be encoded using a KAS definition language.

KAS Definition Language:

We will limit the scope of representations expressible in the KAS definition language to the area of knowledge acquisition for clinical trials. This not only makes implementation of Meta-OPAL more feasible, but restricting the scope of the system will also make the finished program easier to use because the necessary input will be more focused. The kinds of knowledge contained in this output from Meta-OPAL should be apparent from our previous discussion of how Meta-OPAL will work. The syntax we will develop must express a number of different concepts:

1. Various definitional items must be specified to the system. For each kind of knowledge acquisition system Meta-OPAL can create, we must have a syntax for declaring the names and the properties of:

a. The *modalities* of treatment; for example, oncology protocols involve chemotherapies and radiation. A protocol for treating esophageal varices might use various surgical or endoscopic procedures as modalities.

b. The *agents* of treatment; for example, the three drugs vincristine, adriamycin, and methotrexate are the agents used in modality VAM in cancer chemotherapy. "Positive reinforcement" and "negative reinforcement" are two agents of the modality "behavior modification" that could be used in psychiatry protocols.

c. *Standard toxicity grades and their text definitions*, representing various measures of adverse effects on organ systems. Each toxicity grade would also be linked to a *parameter* so that E-ONCOCIN would be able to draw conclusions based on the presence or absence of certain adverse conditions.

2. The list of parameters and their associated properties must be indicated, including rule definition language specifications on how to generate the rules that may conclude each parameter's value. The types of parameters include:

   a. Physical examination findings

   b. Laboratory tests and test results

   c. Clinical conditions, such as "no evidence of disease", "complete response", or "progressive disease"

   d. All "conclusions" reached by the system, including final treatment recommendations.

3. We must permit specification of all of the various *actions* one might take to change any component of the treatment plan. Such actions could involve alteration of the protocol at any level. For example, the protocol itself could be terminated or extended. Administration of any of the modalities of treatment might be delayed or canceled. The dosages of any of the therapeutic agents might be changed, or new agents might be substituted.

   Other actions that do not specifically modify therapy need to be declared. For example, based on some set of parameters, one might want to "order a lab test" or "notify the principle investigator" of some problem.

   Each of these actions will appear as potential entries in portions of the IDS and will accordingly be specified in menus in the resulting KAS (i.e., in menus displayed by E-OPAL as it takes its directions from a KAS file that was produced by Meta-OPAL). Such menus will offer steps to take in response to various values of defined parameters.

In addition to the domain knowledge, the KAS definition language will require declaration of important systems information, including:

1. A description of the high-level appearance of the knowledge acquisition system, including the contents and layout for each window and the nature of each blank and its corresponding menu. Meta-OPAL will determine this knowledge from the graphical inputs of the user when defining the KAS.

2. Specification of the necessary IDS to use for the specific E-OPAL

application, including the complete IDS format, the mapping of blanks from the various windows into the IDS, and the control information needed to translate the IDS into the final knowledge base for use by E-ONCOCIN.

E-OPAL:

In order for Meta-OPAL to produce new knowledge acquisition systems, we will first have to develop E-OPAL, a program that will capture the behavior of the present OPAL prototype. However, E-OPAL will acquire all of its formatting specifications for windows and menus from the output data file produced by Meta-OPAL, rather than from structures internal to the program itself. It will use the knowledge encoded in the KAS definition language to produce an IDS, transfer knowledge from display windows to and from that IDS, and use the IDS to produce a knowledge base for the ONCOCIN inference engine. The physician will enter protocol knowledge in E-OPAL in a manner identical to the present OPAL system.

E-ONCOCIN:

The current ONCOCIN system has been written with care to keep the ONCOCIN knowledge base separate from its inference engine. Thus a relatively complete version of E-ONCOCIN already exists, and this separation is being further refined as part of our translation of ONCOCIN to run on the 1108 workstation. However, we anticipate further changes as our understanding of the IDS and Meta-OPAL evolve.

Encoding New Protocols with Meta-OPAL:

We will test the Meta-OPAL system by rewriting OPAL using Meta-OPAL. This will be accomplished by producing a knowledge acquisition description file using Meta-OPAL and showing that E-OPAL, driven by Meta-OPAL's output, produces a knowledge acquisition system with behavior grossly identical to that of OPAL. This will produce a more generalizable version of OPAL that overcomes some of the limitations of the initial prototype.

We will also use the system to encode protocols in at least one (and possibly two) other medical domain. Dr. Peter Rudd, a member of the Division of General Internal Medicine at Stanford, is conducting randomized controlled trials of new antihypertensive medications and has agreed to collaborate on knowledge base development. This domain of hypertension and its treatment will provide a useful environment for testing the definition of new knowledge acquisition systems using Meta-OPAL. In addition, Dr. Gordon Banks from the University of Pittsburgh (a member of the INTERNIST/CADUCEUS project) has approached us about adapting ONCOCIN for us in protocol-directed management of epilepsy patients. This may well provide another pertinent domain for testing the generality of the notions described here.

Strategic Therapy Planning

ONYX is an ONCOCIN-related subproject designed to fill the need for planning in application areas where traditional planning methodology is difficult to apply. While the program is being developed to assist with the planning of cancer therapy, its architecture is intended to be of use whenever goals are ill-specified, plan operators have uncertain effects, or trade-offs and unresolvable conflicts occur between parts of the goal. ONYX combines strategic "rules of thumb" with a mechanistic model of the domain to determine a set of plausible therapy plans. This is accomplished with a three step process: (1) generate a small set of plausible plans based on current data; (2)

simulate those plans to predict their possible consequences; and (3) based on the results of those simulations, rank the plans according to how well each meets the goals for the situation.

Much of the early work in artificial intelligence techniques for planning made simplifying assumptions about the various choices that can be made at each step of the plan, and in representing the effects of each of these planning steps. In medicine, the planning task often cannot be represented in a form useful to a conventional planning program. Often the goals are ill-specified and the operators have uncertain effects. Furthermore, incomplete and unresolvable interactions occur between the parts of the goal, limiting the usefulness of some of the techniques developed least commitment and plan repair techniques. Consequently, medical therapy planning programs such as VM [17], ONCOCIN, and ATTENDING [49] have frequently relied on algorithms or step-by-step protocols to provide explicit guidelines in the construction of plans appropriate to a particular patient's condition.

Our work with ONCOCIN has revealed an important limitation of medical planning systems which use explicit criteria such as algorithms and protocols. The knowledge in these specifications is a "compiled" version of pathophysiological knowledge of the human body, and of the strategic knowledge of the domain. In ONCOCIN, plan elements are selected strictly according to the characteristics of the current treatment situation without considering the causal mechanisms of the domain or many of the strategies useful in prescribing therapy. Consequently, when a situation arises for which the algorithmic knowledge does not apply, the planning system often recognizes the problem, but cannot plan alternative therapy. The ONYX system is designed to suggest expert quality therapy plans in such difficult cases.

The planning process used by ONYX consists of three steps:

1. *Plan generation.* Using current and past data about the patient, and exploiting the hierarchical nature of possible plan steps, generate a small set of "plausible plans" which are consistent with the patient's current state and the treatment goals for the patient.

2. *Qualitative simulation.* Using causal knowledge of the human physiology, and of this patient's in particular, predict the future states of the patient if each of the plausible plans were in fact executed.

3. *Plan Ranking.* Using knowledge about how patient data satisfy the goals for the patient's progress, rank each of the plausible plans according to the extent that the simulation's predictions for each plan meet the therapy goals.

Cancer treatment strategies are often general statements such as "Try to give a greater quantity of therapy during the early stages of treatment". Restated in a particular context, this might indicate a preference for decreasing a drug dose to 75% rather than just 50% in response to a particular problem. Other strategies may be applied to a wide range of decisions in the plan generation process, from broad therapeutic choices (e.g. whether to give drug therapy or radiation therapy) to specific decisions about individual drug doses. One such strategy is: "If a problem is encountered with a treatment, try to eliminate the part of the treatment that might be causing the problem." In one context, this is interpreted as a suggestion to decrease or eliminate the previously administered drug that is the likely cause of toxicity. In another context, it may also be used to help decide between continued drug therapy and alternative treatments. Currently, such a strategy must be represented in each context in which it applies, rather than as a single more general principle.

The input to the planning process is the database of patient measurements (e.g., the size

of the tumor, white blood count) collected over a number of prior treatment cycles. These input data are processed by the list of treatment strategies. The output of the plan generation phase is a set of possible treatment plans for the current patient visit.

The possible treatment plans are sent to the simulation component to determine the likely ramifications of the treatment. We have designed special software to allow for graphical description of the simulation model. The structure of the domain models is organized hierarchically according to *part-of* relationships. The behavior of a model is determined by the behavior and interconnections of its parts and by three knowledge bases which describe its behavior in response to stimuli. The state of a model is represented by a group of *state variables*, and by the states of its parts. Each model has *ports* through which it communicates with other models using message passing techniques provided by the object oriented system. Such hierarchical models can be built interactively on a Xerox 1108 LISP workstation.

The behavior of each model is described by three rule bases containing production rules. The first rule base dictates how a model will change its state according to the stimuli it receives through its ports from other models. The second rule base contains knowledge about how to make further conclusions about its state based on any recent changes. The third rule base dictates how the new state of the model will be propagated to neighboring models using a simple message passing scheme which acts along connections between models.

Simulation can provide information which the plan evaluation process can use to determine the likelihood that a plan will satisfy the goals for the patient. While the plan ranking phase of ONYX is still under development, early experiments indicate that the rule form used in the plan generation phase will provide some power in the ranking of plans after simulation. In addition, decision analytic techniques can be used to evaluate the decision trees developed by the strategic planning and simulation components.

E-ONYX:

We have thus far challenged and tested this developing system with only a single cancer protocol. However, we believe that the techniques can be expanded to other cancer protocols, and then to other types of clinical trials. We propose to generalize this program, with much of the work involved in representing the various types of plans that may occur among different clinical trial experiments. We expect that the form of the strategies may have to be modified for other medical trials. In addition, we need to verify that the hierarchical nature of the simulation process is sufficient to represent the dynamic processes as the treatment regimen of the clinical trial affects the body of the patient as well as the disease process.

2.2.1.2. Basic Research in AI

Overall Goals and Plans

Our basic AI research projects focus on understanding the roles of knowledge in symbolic problem solving systems -- its representation in software and hardware, its use for inference, and its acquisition. We are continuing to develop new tools for system builders and to improve old ones. The research crosses a number of application domains, as reflected in the subprojects discussed earlier, but the main issues that we are addressing in this research are those fundamental to all aspects of AI. We believe this core research is broadening and deepening the groundwork for the design and construction of even more capable and effective computer programs to aid in reasoning about biomedical problems.

As mentioned above, although our style of research is largely empirical, the questions we are addressing are fundamental. The three major research issues in AI have, since its beginning, been knowledge representation, control of inference (search), and learning. Within these topics, we will be asking the following kinds of questions. As our work progresses, we hope to leave behind several prototype systems that can be developed by others in the medical community.

In particular, we will focus on four areas with immediate coupling to biomedical applications problems and on several others that may have future application:

1. Blackboard Model of Reasoning -- can we design and construct a domain-independent framework for problem solving programs using the blackboard model and can we reason explicitly about control in that framework?

2. Constraint Satisfaction -- given a number of symbolic and numeric constraints defining a satisfactory solution to a problem, how can a problem solver efficiently find a solution?

3. Knowledge Acquisition -- how can knowledge-based programs effectively acquire the large amounts of domain-specific knowledge needed for high performance problem solving?

4. Qualitative Simulation -- how can biological modelling systems be constructed that use domain-specific knowledge to reason approximately about outcomes?

5. Other Research Areas -- architectures appropriate for highly concurrent symbolic computation, a retrospective on the AGE blackboard tool, logic-based systems, self-aware systems, and the SOAR general problem-solving architecture.

These major research themes are discussed in the subsections below and build upon the workstation and advanced computing environment technology also being developed under SUMEX core research.

1. Blackboard Model

GOALS

The long term goal of this part of our research is to improve the usability, the flexibility, and the inferential power of AI software systems for handling problems of hypothesis formation, signal understanding, constraint satisfaction and planning. We proposed to design and implement domain-independent tools for building complex

reasoning systems within the blackboard framework. These include development aids as well as run-time utilities. In other research, we have a coordinated goal of applying the Blackboard framework as an organizing framework for parallel processing.

For the research described below, we have two main objectives. These are:

a. to develop scientific understanding of the "support environment" for Blackboard framework systems and of key tradeoff issues; to design tools for building systems; to implement a domain-independent system incorporating those tools.

b. to implement a substantial reasoning system in the BB1 framework in order to experiment with tradeoffs in the design. Specifically, we will work with the PROTEAN collaborative project to implement and experiment with the program that infers tertiary structure or proteins from NMR data (plus knowledge of primary and secondary structure). This work is described in the research plan for the PROTEAN project.

MOTIVATION

In building knowledge-based systems, we have come to understand the importance of flexibility in its operation. In the KSL, we have experimented with many frameworks for building systems including rule-based, frame-based, and logic-based frameworks. We have also experimented with various methods of inference and control, including goal-directed, data-directed, and opportunistic reasoning. Of the paradigms we know about, the one that seems to offer the most flexibility (at development time and run time) is the blackboard model of reasoning. It has not been as well studied or used as the rule-based or logic-based paradigms have been. Thus we believe a substantial research effort is warranted in order to understand its strengths and limits, and to build a suite of tools that allows us to experiment with it.

BACKGROUND

Though the Blackboard framework for problem-solving and hypothesis formation was conceived at Carnegie-Mellon during the DARPA Speech Understanding project in the early 1970's, it has received much of its scientific and practical development by scientists of our laboratory. The first post-CMU/HEARSAY development was in connection with the HASP system for passive sonar signal understanding. Subsequent efforts involved experiments with scientific applications (to x-ray crystallography), intelligence problems (ELINT and COMINT), and planning; as well as the development of the first software tool to assist knowledge engineers in constructing systems using the Blackboard framework (AGE-1).

As the last decade unfolded, the Blackboard framework was seen to be the most flexible and powerful set of software concepts we had encountered for organizing the processing of knowledge-based systems. It allowed arbitrary mixing of data-driven inference steps ("bottom up") with model-driven steps ("top down"). It allowed a hierarchy of levels of abstraction in the ongoing solution formation, from the most abstract (the global situation) to the least abstract (the supporting data or problem conditions). And it allowed multiple sources of knowledge to provide the links between these levels (i.e. supported information fusion).

The growing significance of the Blackboard framework has given importance to entering a second phase of its development: extensions of the basic concepts (e.g. reasoning from uncertain evidence) and extensions of the suite of software tools for building such systems.

BB1 [27] is a domain-independent environment for building AI systems in a "blackboard control architecture" [28]. Like the standard blackboard architecture [16], BB1 solves problems through the actions of independent knowledge sources that record,

modify, and link individual solution elements in a structured database (the blackboard) under the control of a scheduler. It expands upon the standard architecture as follows:

1. It provides an interpretable, modifiable representation for knowledge sources with these attributes: event-based predicates for triggering; pattern-matching functions for identifying multiple triggering contexts; state-based predicates for assessing transient pre-conditions, and rule-based actions that instantiate prototypical blackboard modification templates. BB1 provides support facilities for knowledge source creation, modification, and checking.

2. Its blackboard representation permits dynamic assignment of attributes and values to objects on the blackboard and provides selective, demand-driven inheritance of attributes from linked objects, with local caching of results.

3. It provides explicit reasoning about control--the selection and sequencing of knowledge source actions--with control knowledge sources that construct dynamic control plans out of modular heuristics on a control blackboard. BB1 defines specific levels of abstraction and solution intervals for the control blackboard. It provides a vocabulary and syntax for expressing control heuristics. A simple scheduler decides which domain and control knowledge sources to execute by adapting to whatever control heuristics currently are recorded on the control blackboard.

4. It provides strategic explanation of problem-solving activities.

5. It provides generic learning knowledge sources to acquire new control heuristics automatically.

6. Its run-time user interface provides capabilities for: displaying knowledge sources, pending actions, and objects on the blackboard; graphically displaying partial solutions via a user-specified interface; recommending pending actions for execution; permitting a user to override a recommendation; executing a designated action; operating autonomously until a user-specified criterion is met.

BB1 is an evolving system incorporating the best results of several research activities. It currently is being used as a framework for the PROTEAN system here at Stanford and for several applications by other research and industrial organizations. We propose to continue developing BB1 as a prototype "next-generation" blackboard architecture.

RESEARCH PLAN

Trade-Off Between Knowledge and Control

As the complexity of the applications we attack increases, the tendencies have been to build more complex control structures. This is a natural consequence of a strategy of "divide-and-conquer" -- having broken the problem into manageable subproblems, the question arises as to how and when to bring the sub-problems together. The other factor that contributes to different control schemes is the difference in quality of knowledge that can be brought to bear at different points in the problem solving process. For example, if there is not much situation-specific knowledge to be applied at a particular point, a system can resort to a method of generating all possible solutions and testing them for credibility.

In the study of concurrent problem solving frameworks, control represents a serialization of knowledge applications. A preliminary study indicates that there can be a trade-off between knowledge and control. An almost control-free blackboard system

may or may not converge to problem solutions. To date there is no research that addresses the trade-off possibilities between degrees of control and various kinds and amounts of knowledge. A blackboard architecture provides a very fertile medium in which this research can be conducted, because all information, including control information, is available on the blackboard. This provides an opportunity to vary the amount of utilization of the control information and control knowledge sources at the same time as adding and modifying task-specific knowledge.

Debugging at the Blackboard System Level

We propose to investigate what would constitute an effective suite of debugging aids for blackboard tools. This investigation will be based primarily on our experience in both using and building various blackboard tools.

The blackboard debugging aids that we will investigate include:

1. A blackboard break package. This package would permit, for example, execution-time insertion of conditional break-points for a specific type of modification of the blackboard nodes of a given class or classes, specific knowledge source invocation, and specific rule evaluation or invocation.

2. A blackboard inspector package. This inspector would permit the inspection of blackboard nodes and the relations between them at various levels of abstraction. These levels of abstraction might range from the entire blackboard presented as a graphics display of nodes by class icons with node relations represented by colored links to the detailed attributes and their values for a specific node presented as formatted text.

3. A stepper which would allow the single-step execution of a blackboard program at various levels of resolution, for example, event posting, knowledge source invocation and rule evaluation. This stepper could be turned off or on by the user or by the execution-time insertion of conditional stepper switch points.

4. A static analyzer which would analyze and present the relationships between, for example, event postings, knowledge source preconditions, knowledge source invocations, and possible blackboard node modifications.

We will use the results of this investigation to design and implement a suite of prototype blackboard debugging aids. Although these aids will be implemented in the context of a particular blackboard tool, for example, BB-1 or an AGE derivative, the underlying concepts should be applicable to a variety of blackboard tools. In particular, we plan to investigate how these debugging concepts could be extended to blackboard tools running on parallel computational systems.

Control Blackboards

In attempting to solve a domain problem, an AI system performs a series of problem-solving actions. Each action is triggered by data or previously generated solution elements, applies some knowledge source from the problem domain, and generates or modifies a solution element. At each point in the problem-solving process, several such actions may be possible. The control problem is: which of its potential actions should an AI system perform at each point in the problem-solving process?

Our approach to intelligent control problem-solving entails empowering AI systems to achieve the following behavioral goals:

- Make explicit control decisions to determine which problem-solving actions to perform at each point in the problem-solving process.
- Decide what actions to perform by reconciling independent decisions about actions that should be performed and actions that can be performed.
- Adopt variable grain-size control heuristics, including global strategies (e.g., first anchor all pieces of secondary structure in partial solutions; then refine the most credible partial solutions), local objectives (e.g., fill in gap *g* in the current solution), and general scheduling policies (e.g., exploit the most reliable knowledge sources).
- Adopt control heuristics that focus on whatever action attributes are useful in the current problem-solving situation, including attributes of their knowledge sources, triggering information, and solution contexts.
- Adopt, retain, and discard individual control heuristics in response to dynamic problem-solving situations.
- Decide how to integrate multiple control heuristics of varying importance.
- Dynamically plan, interrupt, resume, and terminate strategic sequences of actions.
- Reason about the relative priorities of domain and control actions.

In sum, systems following the proposed approach would forgo efforts to predetermine "complete" or "correct" control procedures that anticipate all important problem-solving situations. Instead, they would develop control plans incrementally while solving particular domain problems, adapting their behavior to a wide range of unanticipated problem-solving situations. (See [28] for more discussion.)

To realize these system behaviors, we are investigating a blackboard model of control in which control knowledge sources operate concurrently with domain knowledge sources to construct, modify, and execute explicit control plans out of modular control heuristics on a structured control blackboard. The control blackboard has the levels of abstraction defined and illustrated in Figure 9. Its solution intervals represent problem-solving time intervals.

```
Problem          Problem the system has decided to solve
                 "Elucidate the structure of LAC-Repressor Headpiece"
Strategy         General sequential plan for solving the problem
                 "Anchor all secondary structures; then refine all partial
                  solutions that anchor at least one Secondary element"
Focus            Local (temporary) problem-solving heuristics
                 "Anchor all secondary structures"
Policy           Global (permanent) problem-solving heuristics
                 "Perform actions that generate control heuristics"
o-Do-Set         Pending problem-solving activities
                 "Anchor-Helix helix 1 to Secondary-Anchor4
                  Anchor-Helix helix 1 to Secondary-Anchor5
                  Refine-Partial-Solution anchored by Secondary-Anchor1"
Chosen-Action    Problem-solving activities scheduled to execute
                 "Anchor-Helix helix 1 to Secondary-Anchor5"
```

Figure 9: Levels of BB1's Control Blackboard with Examples from PROTEAN

We also have developed a vocabulary and syntax for expressing heuristics, as illustrated in Figure 10. A simple scheduler, which selects both domain and control knowledge sources for execution, has no control knowledge of its own. Instead, it adapts its scheduling behavior to the control plan currently recorded on the control blackboard.

```
Name            Focus1
Goal            (Eq KS-Type 'Anchor)
Criterion       (for Each-Anchor in ($Find-All 'Solid '((Role 'Anchor)))
                 always ($Object 'Copies Each-Anchor))
Weight          8
Rationale       "Incorporate a copy of each Anchor into at least one
                 partial solution before deciding which partial solutions
                 to refine"
Creator         Chosen-Action 5
Source          Strategy1
Type            Strategic
Status          Operative
First-Cycle     6
Last-Cycle      20
```

Figure 10: An Example PROTEAN Heuristic at the Focus Level

In previous research [28], we developed the blackboard model of control and demonstrated its applicability to the control knowledge used in HEARSAY-II [16], HASP [55], and OPM [30]. We have implemented the control blackboard and several control knowledge sources arising from that research in the BB1 system. We are now using the model to organize control knowledge for the PROTEAN system. We propose to continue refining the model by assessing its applicability in different problem domains and by developing control knowledge sources that are useful for particular problem classes.

Explanation Systems For Control Blackboard Systems

During efforts to solve a domain problem, an AI system should explain its problem-solving behavior. It should justify actions in terms of the situations that trigger them, the knowledge they use, and the solution elements they generate. It should also show how actions fit into an overall line of reasoning, what specific control heuristics they satisfy, and what alternative actions were considered. These explanation capabilities are defining characteristics of intelligent problem-solving. They are also pragmatically desirable as debugging aids for system builders and as credibility checks for domain experts.

We propose to investigate explanation in the context of the blackboard control model and its explicit representation of a dynamic control plan:

- The current scheduling rule for choosing among feasible actions (e.g., "Schedule the highest priority action");
- The current integration rule for combining an action's ratings against multiple control heuristics to calculate its priority (e.g., "Compute each action's sum of weighted ratings against operative heuristics");
- The operative control heuristics (e.g., "First anchor all pieces of secondary structure in partial solutions; Then refine the most credible partial solutions." "Exploit the most reliable knowledge sources.");
- Each action's ratings (0-100) against operative heuristics.
- Each action's priority, computed by applying the integration rule to its ratings.

A preliminary explanation mechanism, implemented in the BB1 system, constructs stylized explanations such as the one shown in Figure 11.

We propose to continue this line of research to develop explanation mechanisms

```
I recommend KSAR 6
Should I Display/Explain/Go/Charge-Ahead/Override: E

KSAR 6: Knowledge Source: Anchor-Helix
        Trigger Event: (Add Solid2)
        Context: ((Anchor Solid1) (Anchoree Solid2))
Control Plan:
        Scheduling Rule: Highest Priority KSAR
        Integration Rule: Sum of Weighted Ratings
        Strategy1: Anchor-Then-Refine
        Rationale: Incorporate a copy of each Anchor
                   into at least one partial solution
                   before deciding which partial solutions
                   to refine
          Focus1: (Eq KS-Type 'Anchor)  Weight 8  Rating 100
        Policy2: (Eq To-BB 'Control)  Weight 10  Rating 0
Priority: 800
KSARs with the same Priority: KSAR 7, KSAR 8, KSAR 9
```

Figure 11: Example of Preliminary BB1 Explanation

appropriate for potentially much more complex control plans and to tailor information selection and presentation to the different interests of system builders and domain experts.

2. Constraint Satisfaction

GOALS

The long-term goal of this part of our research is to produce tools for constructing symbolic constraint satisfaction (SCS) programs, and to analyze and experiment with them to determine their strengths and limits.

The near-term objectives are to implement and experiment with an SCS program in resource management and to generalize it into a prototype SCS framework. We have selected resource management as a test-bed for this research because it involves constraints of different levels of detail and different degrees of "firmness," it involves using the same constraints in the context of somewhat different tasks, it involves time-dependent constraints (e.g., a previously committed resource may become available again in the future), and it involves a large amount of symbolic information that we, as resource managers, know intimately. This work intersects the research on using the blackboard model for constraint satisfaction problems, discussed in the previous section.

MOTIVATION

Reasoning about constraints is a ubiquitous problem with many facets. It occurs in many important problem-solving activities in which a solution is constructed from primitive elements but there are constraints on how those elements are put together. In DENDRAL [43], for example, there were *a priori* theoretical constraints on the *meaningful* constructs and *a posteriori* experimental constraints inferred from the data gathered for a specific problem. Both sets guided the hypothesis generator toward plausible solutions (and away from implausible ones). More recently, the R1 (or XCON) [47] program developed at CMU uses constraints of both types to put together a near-optimal configuration of computer components (including racks and wires). The *a priori* constraints constitute the "rules of the game" -- the components that may and may not be used together, for instance. The problem-specific constraints come from the description of the computer buyer's requirements, such as space available, memory required, and so forth.

Constraint satisfaction problems have not been as well-studied in AI as troubleshooting and diagnostic problems. There have not been, for example, successful generic frameworks developed in which constraint satisfaction systems can be built easily. For troubleshooting systems, on the other hand, several frameworks have been developed and successfully transferred to military and industrial installations. We believe that academic laboratories must intensify research on constraint satisfaction.

In a very large space of possible solutions, each constraint may be taken as a specification of a subset of solutions. In the abstract, then, successive constraints narrow the solution space to just those solutions that lie in the intersection of subspaces specified by all the constraints. This is a first-order model of constraint satisfaction that can, in principal, be applied with constraints of all forms.[1] However, the first-order model must be modified to accommodate several complexities:

- The languages in which constraints and solutions are expressed are not necessarily the same. Some reasoning process must translate from one to the other.

[1]Mathematical methods for constraint satisfaction, while appropriate for many problems, depend on constraints being expressed numerically with some precision. We are concerned here with problems for which mathematical methods are not appropriate.

- Qualitatively different kinds of constraints may apply to a single problem. The problem-solver must integrate them.
- The available constraints may be incomplete. The problem-solver must either characterize the "family" of solutions consistent with the available constraints or choose an arbitrary member of that family.
- The available constraints may be incompatible. The problem-solver must either decide to compromise some of the constraints or identify a dynamic solution that vacillates (in time or space) between states satisfying incompatible constraints.
- There is a potential combinatorial explosion of hypothesized solutions. The problem-solver must restrict search.
- The computational cost of applying individual constraints may be high. The problem-solver must manage these costs.
- Resources available for carrying out planned actions in the real world are constrained over time -- e.g., previously committed resources become available again after a time.

BACKGROUND

The management of resources is a critical part of most decision-making operations. There are often constraint satisfaction problems in which symbolic and numeric constraints interact at many stages in the decision- making process. Sometimes the constraints are expressed in terms of (a) the goal to be achieved, (b) intermediate goals or states, (c) resources available, or (d) the process itself.

A clear instantiation of this class of problems is the management of financial resources. Financial management encompasses the planning and initiation of new projects and the administration of awarded funding for on-going projects and operations. In most institutional settings, the accounting tools for collecting, recording, and reporting information about actual financial transactions in the performance of work (e.g., salary, procurement, and reimbursement expenditures) are well developed. Typically such systems are able to report monthly and cumulative expenses against a project budget; attempt to capture transactions in progress (completeness and accuracy depending on where a given transaction is in the bureaucratic pipeline when the monthly accounting is run); and help with report abstractions, trend projections, and the mechanics of plan calculations. Increasingly, the resulting information can be available to users in electronic form.

However, the tools for the more judgmental aspects of resource management, planning, and subsequent resource allocation, are much more primitive. The integration of the conceptual planning for work to be done with the financial planning, expenditure initiation, and control processes needed to actually carry out the work is mostly handled in the heads of individual project managers and administrators. It is these human experts who cumulate the working knowledge and experience of how to allocate financial resources to achieve work goals while satisfying the constraints imposed by funding terms and conditions and governing policies and procedures of the funding agency and parent institution. In a research laboratory, considerable specialized expertise develops for managing particular types of work under particular funding arrangements. For example, there are experts at managing contracts, or computing equipment purchases, or electronic assembly subcontracting, or hazardous material procurement, or a myriad of other activities confronting the performance of work objectives. Unfortunately, such expertise is almost never taught and it is acquired

through experience involving trial and error and communication of lore from friends who have already had similar experiences. Frequently, there are wide variations in the ability of individual managers to properly administer these matters because of differing levels of experience and even degrees of caring about such managerial details in the face of the primary professional goals of the group.

Many project groups develop local administrative systems, many of them manual or adaptations of spread sheet software packages (e.g., VisiCalc), to facilitate management tasks. But these help only with the mechanical numerical aspects of management and do not assist in the judgmental matters involving optimal use of resources for work goals or satisfaction of policy and procedural constraints. These systems give little help in selecting and filling out appropriate forms for personnel, procurement, or other transactions. They do not provide intelligent interactive planning help that automatically relates, for example, personnel assignments in budgets with supporting expenditures like salaries, supplies, travel, telephone, and publication costs. appropriate to the work group involved. They do not provide catalog information for budgeting purchases of computing equipment, instrumentation, parts, or other discipline-related items. They do not advise on proper cost allocation and documentation relative to funding terms to assure that costs will be allowed. They do not help with planning expenditures among overlapping funding support so as to effectively achieve work goals within funding constraints. They do not help with the integration of institutional financial performance data with on-going plans, locating errors and reconciling the interface between locally recorded commitments and actual expenditures. And they do not provide the required flexible modes of information presentation such as tables and graphs, monthly details and plan exceptions, subproject detail or aggregation, or cross-project distributions.

Now clearly the above functions combine knowledge from many sources -- some factual and some experiential; constraints from many sources -- some numerical and many symbolic; and frequently no unique solution exists for a given planning problem. Spread sheet programs provide a useful interactive mode of calculating and displaying information but they only do part of the task of assisting with the managerial judgements involving symbolic knowledge and constraints. We have, under separate funding, begun work on a prototype system to utilize some of the techniques developed over the recent past for knowledge-based system design to further facilitate computer assistance in the task of budget planning and resource management.

In the longer term, this is one example of a broader class of complex constraint satisfaction problems. Other examples include space allocation, hospital scheduling and triage, interpreting Nuclear Magnetic Resonance data with other information to determine protein conformations, and system design. In studying the financial resource planning problem, we hope to gain more experience with this class of problems in the hope of developing more general problem formulation and problem solving tools for dealing with them.

RESEARCH PLAN

We propose to build a constraint satisfaction program that is (a) general across several types of problems and (b) useful within one or more specific management problems. The shortcomings of spreadsheet software packages mentioned above will be addressed in the context of the prototype object-oriented system already implemented.

The first system uses strictly numerical constraints to aid in constructing a research budget. It is able to access data stored offline about default values for budget items, such as salaries for individuals, cost of specific equipment, and the university overhead rate. It uses windows to display information rather than the more restrictive spreadsheet. Subsequent improvements will focus mostly on incorporating symbolic constraints in extensions that allow:

- defining forms
- filling out forms consistently
- integrating information from forms with budget information
- producing projections under different perspectives
- managing the flow of expenses over the life of a project

We will target our experimental systems for workstations with bitmapped displays to take advantage of powerful graphics tools which we believe will be necessary for an effective human interface. We will use the existing computing resources of the KSL for this work, including Xerox D-machines, Symbolics 3670's, or possibly Texas Instruments Explorers, while keeping a view for software portability to other workstations that will undoubtedly become available.

We expect to evolve the AI portion of the design carefully, based on requirements. Our view is that the system will start out by taking on some of the onerous manual tasks of financial plan development, with better interactive capabilities and being database driven. It will then become increasingly effective as an advisor for planning, leading ultimately to a more active role in plan formulation and review.

3. Knowledge Acquisition

GOALS

The long-term goal of this research is to develop robust machine learning programs that can be integrated with a variety of intelligent systems, and to develop a set of criteria under which machine learning techniques can be successfully applied to different problem-solving architectures.

In the near-term, we propose to design, implement, and experiment with learning methods in different problem-solving environments. In particular, we propose to: (a) extend the work on induction with rule-based systems in the BB1 and HERACLES architectures; (b) develop methods for learning control heuristics in the blackboard architecture; (c) develop programs for learning by chunking (as already implemented in the subgoaling architecture of SOAR) for the classification architecture of HERACLES and the blackboard architecture of BB1. We also propose to extend our analysis of issues in building machine learning systems, specifically the role of noise, the role of examples, and the role of knowledge representation in machine learning.

MOTIVATION

Over the last decade, many machine learning programs have been implemented for special-purpose acquisition of new knowledge. They have been constructed with an eye to generality but with the generality lying mostly in the descriptions of ideas, not in the details of the method and certainly not in the code. The details need to be analyzed so that the strengths and limits of different methods can be assessed in different contexts.

Domain-independent methods are limited by their lack of semantics underlying the names of features being manipulated. Statistical methods, for example, are generally applicable (for data described with numerical features) but lack the ability to use specialized knowledge of a domain that could increase their power. The tradeoffs between generality and specificity in machine learning systems need to be analyzed in order to build powerful learning methods that apply to more than single tasks. Meta-DENDRAL [43], for example, was completed in our laboratory about 1979, but was not developed outside its original task area until 1985 [19].

In the future, it is imperative that methods for machine learning be well enough understood that "off-the-shelf" packages can be constructed and made available for the different classes of intelligent systems we now know how to build. For example, diagnosis and troubleshooting problems are modestly well understood. There are framework systems, like EMYCIN and its commercial cousins, that aid in the construction of a new expert system, e.g., a diagnostic problem solver for a specific task.[1] But there are no pre-packaged learning programs that can be added to the resulting expert system to give it the ability to learn. Since learning takes many forms, there is not just one single package that will serve all purposes. If there is a large library of cases, then learning by induction may be a good way to begin building, or to refine, a knowledge base. If a problem solver is in routine use, then it may be more appropriate to couple it to a learning program that will refine the knowledge base by interacting with specialists using the system, or by watching -- and forming a model of -- what they do.

BACKGROUND

[1]The classes of problem solving systems, themselves, need to be better characterized so that framework systems like EMYCIN can be reliably matched to proposed problem areas. Some work along those lines has been undertaken, on which we will build [10, 5].

Recent work indicates the feasibility of building domain-independent learning programs that use knowledge supplied from the outside to guide the learning. Several overview articles written by members of the KSL and others summarize and analyze the state of machine learning and knowledge acquisition systems. Among the most influential of these on our own work was the "Models of Learning Systems" paper in which learning was viewed as a problem-solving activity with distinct components. It is shown in Figure 12 below.

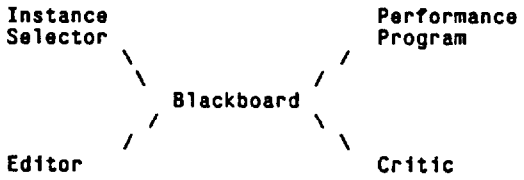

Figure 12: The components of a Learning System.

The problem-solving vocabulary, assumptions, and procedures are defined for all of the components of the system within a world model. One component, the instance selector, chooses training instances to present to the performance program. Performance is critiqued by the critic, whose advice is implemented by the learning element. These steps are not always separate or all automatic.

In the last several years, we have undertaken several experiments in machine learning. Most of these are implemented programs either completed or near completion. Most of these have been done on SUMEX using a biomedically relevant task area as a test domain. They are briefly described in this section with some of the conclusions that are emerging from preliminary analyses.

- *INDUCTION -- LEARNING FROM EXAMPLES*
  - Meta-DENDRAL -- a model-driven induction program that learned new inference rules for the DENDRAL program. It demonstrated the power of heuristic search as an induction method, the power of a "half-order theory" for constraining the search, and the power of a two-tiered search strategy with approximate search followed by detailed search. Its primary mode of learning was generalization from examples, with specialization added in a separate, final step.
  - Version Spaces -- a bidirectional search program that also learned new inference rules for DENDRAL. It demonstrated the power of using generalization and specialization together to refine a subspace of allowable rules (or concept definitions).
  - PRE -- a program that uses a partially formed theory to interpret data in the context of learning refinements and extensions of the partial theory. This "theory-driven data interpretation" program uses constraint propagation methods to keep track of interrelationships in the emerging theory.
  - JAUNDICE -- an inductive learning program that learns new rules for performance programs written in EMYCIN by generalizing and specializing from cases in a data base. It demonstrates the power of bidirectional search, the power of reducing the number of features and filtering out noise.
  - PIXIE -- a program that learns a model of a student's behavior in a

tutoring context from a record of correctly and incorrectly solved problems. It shows the power of starting with a model that "should" produce correct I/O pairs and systematically perturbing the model until the predicted I/O matches the observed data.

- *KNOWLEDGE ENGINEERING -- LEARNING FROM EXPERTS*
  - DENDRAL -- the activity of knowledge engineering was first described (but not named) in 1971 [7] in the context of DENDRAL. It was recognized there as a bottleneck in building knowledge-based programs using experts as sources of knowledge.
  - MYCIN -- several of the now-classical difficulties of knowledge engineering -- such as the problems of welding consensus from incompatible knowledge sources and maintaining a consistent KB -- were first described in the context of our work on MYCIN.
  - TEIRESIAS -- a program that used meta-knowledge in interactively debugging and maintaining a KB (specifically MYCIN's KB). This work demonstrated the value of explanations for understanding the contents of a KB and the value of meta-level knowledge for helping edit a KB efficiently and consistently.
  - EMYCIN -- a generalized framework for building MYCIN-like consultation systems. It incorporated an abbreviated rule language (ARL) that allows an expert on knowledge engineering to write new rules in a stylized form that is easier than LISP (but more telegraphic than English).
  - ROGET -- an experimental expert system whose domain of expertise is knowledge engineering. Although never used outside our laboratory, it showed the extent to which our own knowledge about knowledge engineering could be codified.
  - MOLGEN -- within the UNITS package, MOLGEN included a KB editor that experts, not knowledge engineers, use to maintain a large, complex KB. It demonstrated that experts can and will learn a powerful, but syntactically simple, KB editor when the benefits outweigh the costs.
  - BLUEBOX -- an EMYCIN system with considerable expertise gleaned from the literature by students. It showed that an expert system can be built without tying up an expert if the domain is well structured and well agreed-upon.
  - OPAL -- an interactive KB editor still under construction. It shows the power of building knowledge structures on top of a well designed language. In this case, the language is one of procedures, with temporal predicates.
- *LEARNING BY WATCHING*
- ODYSSEUS -- a program nearing completion that learns by mapping what it infers an expert knows (by watching what an expert does) onto a KB for an expert system. It demonstrates the power of using a modelling system (originally constructed for modelling a student in an intelligent tutoring system, GUIDON) to determine the rules an expert probably uses, without asking the expert directly.

- *LEARNING BY ANALOGY*

- NLAG -- a program that *uses* an analogy, stated as a simple hint, "b is like a", in order to construct new rules in domain B from a KB already built for domain A. It demonstrates the power of an abstraction hierarchy for relating concepts in similar domains and for mapping from one set to another.

- *LEARNING FROM THE LITERATURE*

    - REFEREE -- a prototype EMYCIN program that reasons about the contents of journal articles in order to find new rules in those articles. (Note that answers to questions are supplied by a student who reads the articles, not by a program, or an expert, who reads the articles.) Preliminary results indicate that some journal articles are written clearly enough that a program with only general knowledge of the domain can guide a novice to the new knowledge contained in them.

    - BLUEBOX -- (see above). One lesson is that the literature of a well structured domain can be interpreted correctly by novices to build the KB for an expert system.

- *LEARNING FROM EXPERIENCE*

    - DENDRAL -- a dictionary of previously solved subproblems increased the efficiency of DENDRAL's heuristic search. It illustrated the power of rote learning but also pointed out clearly the tradeoffs between storing and recomputing answers.

    - AM/EURISKO -- programs that use previously computed material to aid in the discovery of new knowledge. These programs illustrate the power of combining existing elements in a KB in various interesting ways in order to construct new elements that are interesting and useful.

    - SOAR -- a general problem-solving system under construction that incorporates a methodology for "chunking", i.e., rote learning with generalization. Preliminary results point to chunking as an effective method for learning from experience in a broad class of problem solving systems.

- *STATISTICAL METHODS*

    - RADIX -- a program that finds statistical correlations in a very large data base, and then discovers whether or not the empirical association is semantically interesting.

RESEARCH PLAN

A) Induction

We propose analyzing the strengths and limits of the generalization and specialization methods in the JAUNDICE program [19], mentioned above, and to implement the same methods in the HERACLES and BB1 architectures. As developed, those methods can be used to learn rules of an EMYCIN syntax from case libraries. The primary techniques are successive specialization guided by general knowledge of the domain, and successive generalization guided by positive and negative examples in the case library. The specialization and generalization operators, as written, are closely tied to the rule-

based formalism, but will be recast to work with the slot-attribute representation used in BB1 and HERACLES.

As described in [19], inductive learning can be considered as either or both of top-down specialization of a general concept or bottom-up generalization of the descriptions of specific instances. The rules used in the JAUNDICE system, which we propose to implement in other systems are summarized in the table below.

Rules of generalization:	1. Dropping conditions 2. Climbing up the value hierarchy tree 3. Creating new symbols 4. Taking minimum or taking maximum 5. Allowing disjunction
Rules of specialization:	1. Adding conditions 2. Climbing down the value hierarchy tree 3. Closing interval

Figure 13: Summary of Rules of Generalization and Specialization by Fu [19]

The search proceeds stepwise using the heuristic rules summarized above as plausible "move generators" in the space of rules, and checking alternative formulations against the data in the case library, as in Meta-DENDRAL [43].

The methods developed by Fu & Buchanan in the context of EMYCIN systems, will be generalized so that the dependence on a rule-based representation of knowledge will be removed. This requires clean separation of the credit assignment methods and the editing methods, as discussed in [8]. The credit assignment programs need to determine generally what is wrong and what to fix (when predictions are false), and then communicate this information to the editor in a high-level, representation-independent, language which the editor translates into specific changes for the knowledge structures being used. In a rule-based representation, for example, inferential links are represented exclusively as premise-action pairs of conditional rules. In a frame-based system the inheritance links carry some of the same kind of inferential information. Thus the editor needs to know the semantics, as well as the syntax, of slots and attributes in order to change the appropriate constructs.

B) Learning Control Heuristics by Experience

Articulating and coding domain knowledge is time-consuming for both the domain expert and the knowledge engineer. Acquiring control knowledge poses additional problems [22], [25]. Control knowledge appears to be more difficult for experts to retrieve than domain knowledge and they have difficulty distinguishing domain and control knowledge. Experts produce general heuristics during questioning, but use more specific heuristics during problem-solving. Stimulating experts' retrieval of a comprehensive set of heuristics may require analysis of many example problems that produce no new domain knowledge. At the same time, powerful control knowledge is essential for the solution of many problems.

We propose to study automatic learning of control knowledge in the context of BB1. As discussed above, all cognitive activities in BB1 systems are performed by knowledge sources that are triggered by changes to objects on the blackboard and, when executed, produce new changes to objects on the blackboard. These include domain knowledge sources that construct solutions to the domain problem on the domain blackboard and control knowledge sources that construct control plans for the problem-solving process on the control blackboard [28]. Similarly, knowledge sources for learning will introduce new control heuristics into the current control plan and they will construct new control knowledge sources to generate the new heuristics in the future.

We envision a range of potential learning knowledge sources, including some that learn new control heuristics, some that learn more general or more specific forms of known heuristics, and some that expand or restrict the applicability of known heuristics. Within each category, some learning knowledge sources simply replace the knowledge engineer and interact directly with domain experts. For example, the knowledge source Understand-Preference, a prototype version of which we have already implemented [29], is triggered when a domain expert overrides BB1's scheduling recommendation. Its action interacts with the expert to determine the reason for the override and encodes a corresponding new heuristic. Other learning knowledge sources could operate autonomously. For example, the knowledge source Attribute-Results might be triggered by dramatic improvement (or deterioration) in the current solution to the domain problem. Its action would attribute the change in solution rating to preceding actions and encode a heuristic favoring such actions. Evaluate-Heuristic, another autonomous knowledge source, might be triggered when a new control knowledge source is executed. Its action would evaluate subsequent changes in solution rating and adjust the posted heuristic's assumed importance (Weight) accordingly.

The proposed work will develop specialized mechanisms for these different kinds of learning. For example, Understand-Preference compares attributes of the action recommended by the scheduler to corresponding attributes of the action preferred by the expert and, with the expert's assistance, diagnoses the key differences. By contrast, the knowledge source Evaluate-Heuristic requires a mechanism for measuring and evaluating changes in the quality of a solution and for distributing "credit" for those changes among simultaneously active control heuristics.

BB1 provides a rich and well-structured foundation for learning in its explicit, structured representations of all blackboard objects, knowledge sources, and potential actions. The structure and semantics of BB1's control blackboard entail a prototypical form for all control heuristics used by the scheduler:

```
Goal:      (Function <KSAR:Attributes> <Other-Arguments>) = {0-100}
Weight:    {1-10}
Criterion: (Predicate) = T/F.
```

A heuristic's Goal is a function that, when evaluated for a potential action, produces a rating 0-100. Its Weight is a number 0-10 that signifies the importance of an action's rating on the Goal function. Its Criterion is a predicate specifying an expiration condition that, when met, signifies that the Goal is no longer desirable. All learning knowledge sources will attempt to construct (or modify) control heuristics in this prototypical form. They also will attempt to construct control knowledge sources whose triggering conditions describe appropriate situations in which to adopt new heuristics and whose actions post the new heuristics on the control blackboard.

The proposed work will supplement the BB1 foundation with additional knowledge of canonical forms for semantic classes of control heuristics. For example, control heuristics that rate actions on attributes with numerical values might incorporate Goals in the canonical form:

```
(Translate-Value-To-Scale KSAR:Attribute Maximum-Value),
```

in which observed values on the target attribute are translated into corresponding values on the required 0-100 scale. Alternatively, they might incorporate Goals in the canonical form:

```
(Compare-To-Threshold KSAR:Attribute Threshold),
```

in which observed values on the target attribute are rated 100 if they are above some threshold, and 0 otherwise. Obviously, there are many alternative canonical forms that are potentially appropriate for attributes with different data types (e.g., numerical,

literal, list). Learning knowledge sources must determine which form is appropriate for each new heuristic.

Although we will develop and evaluate learning knowledge sources in the context of the PROTEAN system for protein structure analysis, the knowledge sources themselves will embody generic learning mechanisms applicable to a wide range of problem domains. We will incorporate these learning knowledge sources in the BB1 environment.

C) Knowledge Engineering

We propose to build interactive aids for knowledge engineers in the context of the BB1 and HERACLES frameworks. Many of the aids in EMYCIN, although developed nearly a decade ago, have never been duplicated, or have only been partially duplicated, in other contexts.

These ideas include:

1. meta-level constructs to guide the acquisition and checking of new knowledge;
2. interactive debugging aids for tracking down the source of an error in the context of an incorrect conclusion;
3. explanation facilities.

HERACLES is a tool for building expert systems that we have generalized from our experience with NEOMYCIN, a program designed to clarify the knowledge structures and reasoning processes of MYCIN. HERACLES solves problems by classifying them in terms of a set of pre-enumerated solutions, a method we call *heuristic classification.* For example, a generic form of heuristic classification, commonly used for solving diagnostic problems, is *causal process classification.* We have been studying how causal processes are classified in medical diagnosis, and have recently applied our model to the problem of diagnosing surface flaws in cast iron.

In causal process classification, data are generally observed malfunctions of some device or process, and solutions, pre-enumerated in the program, are abnormal processes causing the observed symptoms. We say that the inferred model of the device, the diagnosis, *explains* the symptoms. Only the simplest devices and processes, can be adequately described in terms of function/structure models, enabling a principled comparison of faulty behavior to intended design. Instead, it is necessary to construct a *causal network* that relates normal and abnormal states to observed behavior and ultimate fault etiologies.

While causal networks of this sort have been incorporated in medical diagnostic programs, for example, for more than a decade, the principles by which they should be constructed is still an area of research. In our own work, we have been investigating heuristics for constructing such networks in knowledge acquisition dialogues. We have discovered that an expert's terminology and explanations of causal processes must be carefully analyzed for the resulting network to be coherent and applicable to many problems. For example, an expert may say, "a brain-tumor causes a brain-mass-lesion." But a network simply linking these two terms will be meaningless: a brain-tumor is *a kind of* brain-mass which *causes* a brain-lesion (cut). The two terms cannot be linked simply by either cause or subtype because the term "brain-mass-lesion" bundles together a location, a cause, and an effect.

In our ongoing research, we propose to continue this kind of analysis to develop a program that can help a knowledge engineer construct a principled causal network. We