A Declarative Representation of Control Knowledge

PAUL R. COHEN, JEFFERSON DeLISIO, AND DAVID HART

Abstract — **An explicit representation of control called strategy frames is described. The control of several well-known expert systems can be described in terms of strategy frames, although their control is actually encoded in an interpreter. One advantage of strategy frames is that complex control strategies emerge from their interaction, so complex interpreters are not necessary. This idea is illustrated in the context of a process control problem.**

I. Introduction

CONTROL is an important problem in artificial intelligence (AI): knowledge systems are getting very large and difficult to control [1], [2] and, because efficiency is a concern in these systems, control strategies must be flexible enough to balance various costs, especially time [1], [3]–[9]. Independent of efficiency concerns, we are beginning to work on tasks such as design [10]–[12], process control [13], and knowledge-based planning [1], in which "how to" knowledge is important. In these tasks, problem solvers do not use a single fixed strategy but instead change strategies as the situation demands, keeping "trim" to the current situation.

AI researchers are beginning to recognize control knowledge as a kind of expertise and are developing tools to help knowledge engineers acquire it [14]–[19]. Previous work by Thomas Gruber in our laboratory has addressed the problem of acquiring and generalizing strategic "meta-rules" [20], [21] similar to those discussed by Davis [22] and Clancey [23], [24]. Meta-rules give control a "one step at a time" or reactive flavor that, when implemented in a medical expert system [25], [26] failed to capture some aspects of diagnostic expertise [27]. For example, it was difficult to formulate meta-rules that had contingent actions on their right sides—to, for example, "do test *A* and if the answer is positive do test *B* otherwise do test *C*." The current work was initiated to develop representations for these little contingency plans, but it swiftly became an exploration of representations for the range of knowledge one needs to control complex AI systems. Throughout, we have required these representations to be declarative, so that they might be accessible to knowledge engineers, and also to be knowledge acquisition tools.

This paper represents a snapshot of our current work on control. It represents work in progress and so poses problems that it does not solve. We think it is essential that as AI researchers explore more realistic environments, they report where they are, how they got there, and where they are going; even if they still have a long way to go [28], [29]. In this spirit, we have organized the paper into six sections that correspond roughly to aspects of a journey.

We begin with an analysis of the control literature that led us to the idea of strategy frames—declarative structures that represent problem-solving strategies (Section II). Next, we discuss the purpose of the journey, our intent being to empirically test some hypotheses about local and global control via strategy frames (Section III). In the next two sections we describe a process control task and an implementation in terms of strategy frames which together provide the environment for our empirical work (Sections IV and V). Preliminary results are described in Section VI.

II. Strategy Frames: A View of Control

Our view of control is motivated by the following observations, which are based on analyses of the control strategies of Casnet [30], [31], Pip [32], Hearsay-II [2], [33], Mum [25], MDX [34], [35], Neomycin [23], [36], Dominic, and Dominic-II [10], [12].

Observation 1: The control strategies of many knowledge systems can be viewed as the interaction of a small number of simpler strategies. We call these *strategy frames*.

The structure of the strategy frames will be discussed in Section V. For now, one can imagine them controlling inference within and between levels of *hypothesis spaces*. For example, Casnet's hypothesis space had three levels, for data, pathophysiological states, and disease hypotheses, respectively (Fig. 1). Its control strategy can be viewed in terms of the interaction of three strategy frames: bottom-up inference from data to pathophysiological states, lateral inferences along causal pathways between pathophysiological states, and bottom-up inferences between pathophysiological states and disease hypotheses. The tails and heads of the arrows in Fig. 1 are different regions of the hypothe-

Manuscript received April 30, 1988; revised August 15, 1988. This work was supported by ONR contract AFOSR 4331690-01 and DARPA/RADC contract F30602-85-C0014.

The authors are with the Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

IEEE Log Number 8927049.

0018-9472/89/0500-0546$01.00 ©1989 IEEE

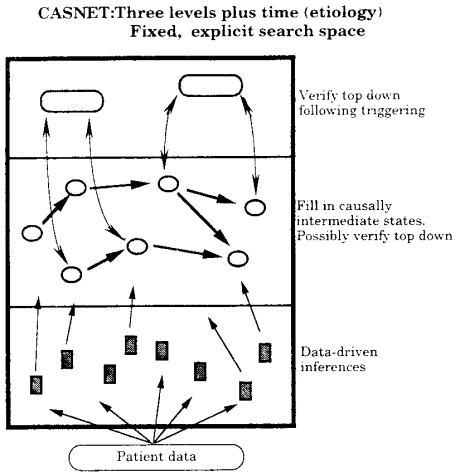


Fig. 1. Schematic representation of control in Casnet, showing levels of hypothesis space and strategy frames represented as vectors.

sis space and are the domain and range, respectively, of strategy frames.

Observation 2: Strategy frames do not relinquish control after every inference, nor do they keep control throughout problem solving.

A strategy frame says, for instance, "I'm in control, and we're going to do some bottom-up processing until it appears that some other kind of processing would be more useful." For example, Mum was controlled by *strategic phases* that changed relatively infrequently. A control cycle in Mum begins by looking at the state of the hypothesis space to determine whether the applicability conditions of the current strategic phase are still in effect. If not, a new strategic phase is invoked. In either case, the next step is to select a focus of attention, and then to select evidence for or against that focus. The evidence is solicited and the state of the hypothesis space is updated. Then the cycle begins again. The system can stay in a strategic phase (i.e., under the control of a single strategy frame) for many cycles. For example, one strategic phase—called "Deal-with-Critical-Possibilities"—was active as long as critical dangerous hypotheses (e.g., heart attack) had some degree of support. Within this phase, Mum selected the hypotheses for its focus of attention in order of their criticality; it first sought low-cost evidence against the focus of attention, but had no prohibition against high-cost evidence, pro or con the focus. Unlike other strategic phases, the Deal-with-Critical-Possibilities strategy gave Mum's problem solving a distinct "this is important, so hang the cost" flavor for the duration of time that critical hypotheses were active.

One is tempted to equate knowledge sources in Hearsay-II with strategy frames. But although the stimulus and response frames of knowledge sources (KS's) in Hearsay-II are analogous to the domain and range of strategy frames, the latter take control of processing for intervals that can involve many inferences whereas KS's generate knowledge-source instantiations (KSI's) for each possible inference and relinquish control to the scheduler after every inference.

In fact, the designers of many systems have found it desirable to give them something like the functionality of strategy frames, even when they were initially designed to have opportunistic control, or at the other extreme, completely fixed control. Hearsay-II was designed to have opportunistic control, but was later modified to have two phases—a bottom-up phase followed by an opportunistic one. Even Mycin, which is commonly thought to be an exhaustive backward-chaining production system, switched to limited forward chaining when it was presented with particular kinds of data [37].

Observation 3: Strategy frames are nested structures in which there are *tactical instantiations* of the components of a strategy.

All strategic phases in Mum have the same nested structure:

1) applicability conditions,
2) criteria for selecting focus of attention,
3) criteria for selecting evidence,

but they differ in how the components or slots of the strategies are instantiated. For example, the criterion for selecting focus of attention in the "Discriminate-Strongest-Hypotheses" strategic phase is plausibility; this phase focuses on hypotheses that are likely given the evidence. In contrast, the Deal-with-Critical-Possibilities phase focuses on hypotheses that are dangerous and have some level of support; these hypotheses may actually have a low plausibility.

A similar view is found in Dominic-II, a program for *iterative redesign* of mechanical devices. The program has a five-step basic control cycle:

1) select an aspect of the design to improve,
2) determine how much improvement is desired,
3) select a design variable that, when changed, is expected to improve the design,
4) determine how much to change the design variable,
5) decide whether or not to change the design variable.

For example, Dominic-II may want to improve the *expected life* of the pulley system (step 1), from "short life" to "medium life" (step 2), by changing the diameter of the drive pulley (step 3), from four to five inches (step 4). If this change is predicted to have the desired effect, and it improves the overall evaluation of the design, then it will be adopted. Then the redesign cycle starts again. Roughly, redesign in Dominic-II is hill-climbing because each change to a design improves its overall evaluation. But it is not strict hill-climbing because, depending on the tactical instantiations of the five steps in the redesign cycle, Dominic-II can actually allow a design to get worse before it gets better. For example, one tactical instantiation of Dominic-II's basic design strategy is as follows.

1) *Select an aspect of the design to improve:* Select the aspect that has the *largest negative effect* on the overall evaluation of the design.
2) *Determine how much improvement is desired:* Require an improvement sufficient to ensure that the aspect no longer has a negative effect on the overall evaluation of the design.
3) *Select a design variable that, when changed, is expected to improve the design:* Select any design variable that has not been changed in the last two cycles.
4) *Determine how much to change the design variable:* Change the value of the design variable "a lot."
5) *Decide whether or not to change the design variable:* Even if the overall quality of the design decreases, accept the change if it improves the specific aspect of the design as desired.

Dominic-II monitors the current state of its design, looking for pathological situations. When it finds one, it typically switches from one tactical instantiation of the basic redesign strategy to another, more appropriate one. For example, a common problem in hill-climbing is the *mesa effect*: instead of moving steadily up a hill, a system gets trapped in a relatively flat area, making many small changes but not improving overall performance. When Dominic-II detects this situation, it adopts a tactical instantiation of the redesign strategy that makes very large changes to a design variable even if they reduce the quality of the design. This is like taking large steps instead of little ones to get off the plateau onto a hill—even if one lands on the hill below one's current altitude. In a series of experiments we found that Dominic-II, which could select among tactical instantiations, always outperformed an earlier system, Dominic-I, which could not select [12].

Observation 4: The same strategy frames show up in many knowledge systems, though these systems differ parametrically.

In the last few years there has been a sense that many AI tasks are very similar. This sense was given voice by Clancey's characterization of diagnostic reasoning [38], [39] and by Chandrasekaran's evolving taxonomy of AI tasks [40]–[42]. It has been argued that *task-level* architectures more general than particular knowledge systems but less general than weak methods, such as generate and test, [43], [44] can be designed. What makes AI tasks similar is not the facts and heuristics we use to solve them, for these vary from one domain to another, but rather the general *kinds* of knowledge they require and, most important from our perspective, *how* they are solved. In our analysis of many knowledge systems, we believed we repeatedly saw the same strategies. For example, all the diagnostic systems made some distinction in their control strategies between data that "trigger" hypotheses and those that cannot trigger hypotheses but can support previously triggered ones. Most of these systems also made their control strategies sensitive to data that could categorically rule out hypotheses. The basic control strategy for diagnosis, though slightly different in each of the several systems, was to first use a subset of the data to generate a small set of hypotheses (using all the data would create an unmanageable combination set), then to try to rule out or rule in these hypotheses, typically in an order that reflects the importance of the hypotheses.

The advantage of identifying general strategies is that they become part of the knowledge engineer's toolbox. We imagine providing task-level architectures complete with a variety of declarative strategy frames, each easily parameterized for the particular application, much as knowledge-engineering tools currently provide declarative representations to be filled with domain-specific facts and heuristics. Several steps have already been taken in this direction [14], [16], [18]–[20].

Observation 5: Control strategies can depend intimately on the structure of the hypothesis space.

Casnet's control strategy exploited causal associations among pathophysiological states, and MDX's strategy exploited hierarchical associations in a taxonomy of diseases. It seems that the diversity of control strategies depends on how many types of relations exist among objects in the hypothesis space. For example, if the only possible relation in the hypothesis space is "evidence-for," then a system is limited to blind data-directed or goal-directed control. It cannot focus on hypotheses that are causally related to other likely hypotheses unless causal relations are explicit in the hypothesis space. In Neomycin, Clancey describes many relations that are needed to support a wide range of diagnostic subtasks. These include binary relations (e.g., causal and hierarchical) relations, and also unary relations or properties of the objects in the hypothesis space. A similar approach was taken in Mum and the subsequent MU project. In MU, one defines sets of objects based on their relations with other objects, such as *the set of all tests that potentially confirm any object in the differential and are inexpensive*. Here, *potentially-confirm* is a binary relation and *object in the differential*, *test* and *inexpensive* are unary ones. Sets in MU can function either as foci of attention or, as in this example, as sets of potential evidence.

In Pip, the relationship between control—specifically focus of attention—and the structure of the hypothesis space is extremely tight. The hypothesis space is a network of associated frames, most of which represent diseases. These frames are "activated" or "illuminated" when their associated symptoms are found in the patient. Hypotheses become active (i.e., part of the focus of attention) when activation spreads over relations in the hypothesis space. For example, Pip has triggering relations between data and hypotheses that make hypotheses active if the data are present. A more complex role is played by relations such as *may be caused by*: if two frames are associated by this relation, and one becomes active, then the other becomes

semiactive, which means, roughly, that it will have a greater propensity to become active as more data become available.

It is easy to see the importance of the structure of the hypothesis space when that structure is *explicit*, as it is in Casnet, MDX, Neomycin, and Mum. By explicit we mean that all data, intermediate hypotheses, and conclusions are known in advance before the system is ever run. In contrast, objects in *implicit* hypothesis spaces are generated by search during execution. For example, in the Dominic systems we do not traverse an explicit space of thousands of designs, but rather we generate the space by iteratively modifying each design to produce the next. In such cases, objects in the hypothesis space are not associated by explicit relations, as they are in explicit hypothesis spaces. Consequently, control strategies are designed for the implicit structure of implicit hypothesis spaces. In Dominic this structure was assumed to be a hill, and so design was viewed as hill climbing. In fact, Dominic-II is able to detect the local topology of the hill and, if it is a plateau, modify its control strategy appropriately. In Hearsay-II the implicit structure of the space of interpretations was assumed to contain many constraints—often referred to as redundancy in the speech signal—so that partial interpretations of one part of the signal could help the system interpret other parts. Like Dominic-II, Hearsay-II could detect where it was in the space and could extend relatively certain regions into less-certain areas. This was called island-driving.

III. Motivations

Although AI researchers have been building control structures and problem-solving strategies for years, one of the basic questions about control remains unanswered: Under what conditions can you achieve a sequence of actions that look like they were selected by a global strategy, when in fact they were selected by one or more local strategies? Both global and local are vague and, at best, relative terms: one strategy relies on "more global" information than another. Informally, local means "based on a subset of the available information." In terms of performance, because local strategies require less information and less integration of information, they are valuable when the cost of obtaining and processing relatively global information is prohibitive. On the other hand, performance is typically better when informed by global information. For example, imagine picking out a route across a city; a relatively local strategy determines the route given the global goal and the immediate environment, whereas a more global strategy considers the environment further away. The local strategy requires less information and less planning, but may take us away from our goal and into "blind alleys"; the more global strategy can avoid these problems because it has access to more global information, such as a map. In terms of this example, we want to know under what conditions one can traverse a route that *appears* to be guided by a map when it is not.

Our research poses this question not in terms of actions such as traversing a route, but in terms of strategies (implemented by strategy frames) that select actions: Under what conditions can a sequence of strategies appear to be guided by global information when, in fact, it is not? The question is important because AI is working in task domains that seem to require multiple strategies (or, at least, multiple tactical instantiations of strategies). Examples include Dominic-II and Mum (Section II), as well as recent work suggesting that alternative strategies should be selected by resource demands and availability [3], [13], [45]. We want to know the conditions in which a system with multiple strategies needs global information to select them and, conversely, when relatively local information will suffice. Under what conditions does the computational cost of global information outweigh the benefit of having it? Under what conditions does relatively local selection of strategies result in inefficiency, incoherence, or other pitfalls analogous to "blind alleys" in the example above?

Our research is designed to explore these questions. Initially, we thought about reimplementing some of the systems discussed earlier—Casnet, Pip, Mum, and so on—with strategy frames. But on reflection it seemed uninteresting to demonstrate yet another architecture for diagnosis. Instead, we have used strategy frames to control a system that solves a relatively new and uncommon kind of task: Process control tasks require a system to monitor and adjust to variations in an ongoing process, many or all of which are unpredictable.

IV. The McD Problem

McD is a simplified model of a fast-food restaurant. Orders are presented at varying rates over time. McD tries to fill all orders as quickly as possible without building up large surpluses of items. It does this by changing the rates at which it produces items—by shifting employees from one activity to another. For example, if McD has a surplus of hamburgers but a shortage of shakes, an employee may be moved from the grill to the beverage station.

The McD problem has these characteristics.

Dynamic demands:	the problem solver must respond dynamically to changing demands; for example, changes in the rate at which orders are placed.
Resource allocation:	problems are solved by dynamically shifting resources from one activity to another, thereby changing the configuration of the problem solver; for example, by moving employees to different stations.
Tradeoffs:	even if a system has the resources to handle any level of demand, it should not want them to be committed when demand is slack, since they would be largely unproductive. There is a tradeoff be-

tween the costs associated with the failure to meet demand and those associated with excess supply. And this is but one of many tradeoffs that together determine the *efficiency* of a system.

In McD, resources are employees and products are menu items (hamburgers, fries, soda, shakes, apple pies) and service items (cashiers, clean tables, and condiments). The number of employees is variable, but in our experiments is initially seven. Employees work at stations. There are two grills (each with space for two employees), two fryer stations (one employee tends each), one drinks and dessert station (tended by one employee), two busing stations that clean tables and manage condiments and utensils (tended by one employee each), and four cashier stations (each tended by one employee). Obviously, more than seven employees are needed to fully staff all the stations in McD. Successful management of McD involves shifting employees from one station to another in response to demands. McD can also call in additional employees and send surplus employees home.

McD is presented with blocks of orders at varying rates. A block includes the number of people in a group, whether they require tables, and how many of each menu item are desired.

Since cashiers take orders in McD, and the number of cashiers is limited, some orders will not be processed immediately. Moreover, not all menu items are available all the time, so some orders will not be filled immediately. Two components of McD's performance are the average time waited for a cashier and average time waited for food.

McD recognizes three special situations: shortages, surpluses, and breakdowns. A shortage is indicated when the expected demand for an item significantly exceeds the number on hand (minor disparities between demand and supply are ignored). Conversely, a surplus is indicated when the expected demand is significantly less than the number on hand. (Expected demand is just the average demand over the last few time segments.) Breakdowns remove stations from commission until resources are allocated to fix them. For example, one of the two grills may go down and may result in a shortage if demand is high and it is not fixed quickly.

McD has strategies for each of these situations. All involve moving people from one station to another. A shortage of hamburgers, for example, is handled by searching for an employee who can stop what he is doing and move to a grill. Of course, if the capacity of the grills is exceeded (i.e., more than four people are already working them) then McD can do nothing to reduce the shortage. Thus shortages of beverages and desserts cannot be overcome unless these stations are unmanned, because each of these items is produced at a station that has a capacity of one employee. Breakdowns are fixed by moving an employee to the broken station to fix it. Surpluses, conversely, are rectified by removing an employee from a station.

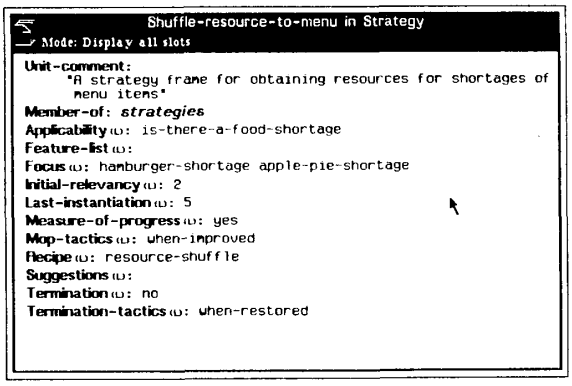


Fig. 2. Strategy frame for moving resources to cover shortages of menu items.

We must digress briefly to explain what is strategic about problem solving in McD. Colloquially, strategies select actions and remain in effect longer than individual actions. By strategy, we mean a configuration of the problem-solving system (McD) that remains in effect over many problem-solving actions. For example, McD may fill many customer orders with just two employees working at a grill, but eventually a shortage may require a change in this configuration. Adding another person to a grill will change McD's behavior: it may now produce a surplus of hamburgers, or experience shortages of other items, or the waiting period for food may decrease, or it may increase due to a bottleneck introduced by removing the employee from his previous station. The strategy frames described below change McD's behavior by changing its configurations. A similar notion of strategy is found in, say, Dominic-II (see preceding), where different strategies produced different hill-climbing behaviors.

In sum, the McD problem is to dynamically monitor and alter configurations, that is, allocations of resources to stations, so that shortages and surpluses are avoided. McD detects shortages, surpluses, and breakdowns, and selects among strategies for rectifying these situations.

V. Strategy Frames

Strategies are represented by strategy frames. Fig. 2 is a strategy frame called "shuffle-resource-to-menu" ("shuffle," for short). The figure shows an instance of shuffle that was invoked somewhere in the middle of a run of McD.

Shuffle is invoked to fix shortages in menu items, which include shortages of hamburgers, fries, soda, shakes, and apple pies. Shuffle has an *applicability* slot that contains a lisp function called "is-there-a-food-shortage," which the interpreter runs to determine whether there is a shortage of any food items. If the applicability condition is met, the interpreter adds shuffle to the list of applicable strategies. Typically, this list contains several strategies, ranked by their *relevance* slots. Relevance can be calculated many ways, that is, the relevance slot can have many tactical

instantiations. But in shuffle and other strategy frames that deal with shortages, only one tactical instantiation is currently used (others are planned but not yet implemented). It ranks strategy frames by the number of known shortages they can potentially fix and by the severity of those shortages. For example, if McD is currently suffering shortages of hamburgers *and* fries, then shuffle can potentially fix two problems, and so is more relevant than, e.g., a strategy for correcting service item shortages if only one kind of service item—e.g., tables—is in short supply.

The *initial-relevance* slot of a strategy frame is used to bias whether McD attends to a situation. Shuffle's initial relevance is zero, which means that McD will not attend to it before other strategy frames with higher initial relevance. We have used this mechanism to configure McD to attend to shortages before surpluses and vice versa. Once a strategy is selected, its *focus* slot ranks the problems it will deal with. The order of elements in shuffle's focus slot dictates that hamburger shortages should be fixed before fries shortages. This order is determined dynamically, based in part on the severity of the shortages.

The *recipe* slot of a strategy frame contains a series of actions—typically a reallocation of resources. For example, shuffle's recipe is a lisp function called resource-shuffle that attempts to obtain a resource from somewhere else in the system to assign to the shortage. It has two tactics for doing this. One favors obtaining employees who are not assigned to any station. The other looks for employees on stations that are producing surpluses. In either case, if shuffle fails to get resources, it returns control to the interpreter.

Sometimes McD's strategy is appropriate but not aggressive enough. For example, a shortage strategy may move one employee to a new station, but two or more employees are needed to correct the shortage in a reasonable time. In such cases, McD may reinvoke that strategy; if shuffle moves one employee but is ineffectual, then McD may reinvoke shuffle to move another employee.

This raises the question of how we keep track of whether a strategy is having the desired effect. Shuffle has a *measure-of-progress* slot that contains a function—currently binary—that tells the strategy whether it is progressing. If the system notices no progress, or if the situation is actually getting worse, the strategy may be reinvoked as described earlier to obtain more resources. McD can measure progress in one of three ways, specified in the *mop-tactics* slot of a strategy frame. Shuffle's mop-tactics are "when-improving," which means that progress is being made so long as the situation (in this case, a shortage) continues to improve. The measure-of-progress slot in shuffle says that progress is indeed being made.

Finally, the strategy has *terminating-conditions* that specify when to quit work. Tactical instantiations of this slot determine exactly when the strategy quits. Three instantiations are

- *when-restored*—stay in effect until the problem is solved (e.g., until there is no more shortage),
- *when-improved*—stay in effect until the situation improves (even if the problem is not solved),
- *time*—stay in effect for N cycles.

Shuffle has a when-restored termination tactic, and its termination slot contains no, which means that the termination criterion has not been achieved yet.

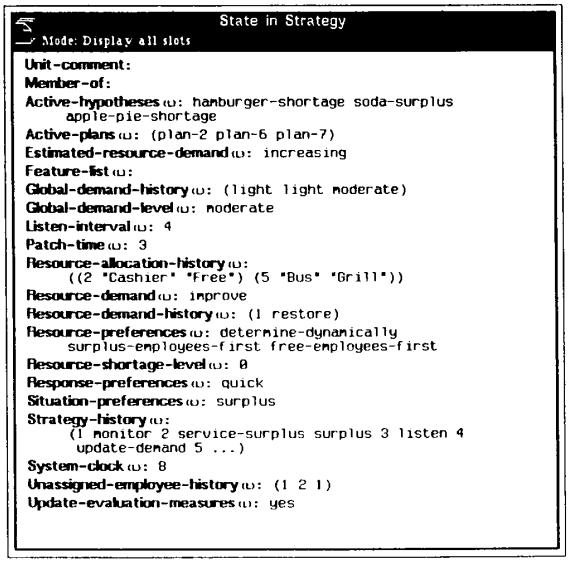


Fig. 3. State frame, which represents McD's global state.

A. State

One special frame, called *State*, maintains a global view of McD and is accessible to all strategies. Fig. 3 shows the state of McD at a particular time during a run (in this case, at time = 8, as seen in the *system-clock* slot).

State contains some global information that strategies may need to select appropriate tactics. Strategy frames do not maintain global views of McD's world. They are "egocentric" in the sense that they attempt to solve problems as well as possible, irrespective of the global ramifications. For example, a strategy might want to allocate several people to a shortage for as long as necessary to reduce it, irrespective of other shortages. But because strategies compete for resources, a global State must tell strategies what they need to know to resolve these conflicts. In particular, strategies consult the *resource-demand* slot (Fig. 3), which contains the overall level of demand for resources, when they decide which tactical instantiations to prefer. This slot contains a recommended termination tactic, which may be restore, improve, or time, as discussed in the preceding. McD determines which termination tactic is appropriate by considering known shortages, surpluses, breakdowns, available resources, and rough estimates of trends in the values of these parameters. If the demand for resources is high, then State recommends conservative termination tactics—strategies run for N time units and then quit (the *patch-time* slot sets N.) Conversely, if the

level is low, then State recommends termination tactics that permit strategies to run until they solve their problems. Intermediate levels result in tactics that terminate strategies after some improvement.

State also maintains a list—called *active-hypotheses*—of problems (i.e., surpluses, shortages, and breakdowns) and a list of *active-plans*, which are the problems McD has responded to in the past. The *strategy-history* is simply a list of strategies in the order they are called, so McD can see when strategies were last called. The list in Fig. 3 shows that in addition to surplus and shortage strategies, McD has strategies to *listen* for incoming orders to *update demand* and other statistics, and to *monitor* measures of progress and termination conditions.

McD maintains a projection of resource demands, called *estimated-resource-demand* that contains a moving average of resource-demand. In Fig. 3, estimated demand is increasing. Obviously, estimated demand can be calculated in many ways; for example, we know that demand for food (and thus resources) increases during breakfast, lunch, and dinner hours, and so might include the time of day in the calculation of estimated resource demands.

The purpose of the other slots in State will be described later, when we present examples of McD.

B. How McD Works

McD has a basic control cycle in which the interpreter first polls strategies to find those that are applicable, then ranks the applicable strategies, and then gives control to the top-ranked strategy. That strategy will attempt to change McD's configuration (i.e., the allocation of resources to stations) and will then return control to the interpreter. If the strategy succeeds—and it may not if resources are unavailable—the new configuration will remain in effect until the strategy terminates it. We now describe these steps in more detail.

C. Polling and Ranking Strategies

At each cycle, the interpreter asks each strategy whether it is applicable and thereby discovers whether McD has surpluses, shortages, or breakdowns at any stations. A frame-like structure is created for each problem situation, and a pointer to it is added to the active-hypotheses list in State. Sometimes, several instances of the same situation arise (e.g., several shortages), and a strategy will be applicable to each. This and other factors are combined by a function in the relevance slot of each strategy. Relevance represents the strategy's own assessment of how much it can contribute to keeping McD running smoothly. In general, the more situations to which a strategy applies (and the greater their severity), the higher its relevance score. The strategy with the highest relevance is selected for execution unless another with equal relevance was invoked less recently. Note that, except for this last clause, the selection of strategies is based on information local to the strategy frames. The interpreter then turns control over to the selected strategy.

D. Executing a Strategy

When a strategy gains control, it instantiates itself with *tactics*. The first thing a strategy such as shuffle (Fig. 2) does is to determine its *focus*, which in this case is a shortage of hamburgers and fries. Currently, shuffle can select only a single problem (e.g., hamburgers), but eventually it will be able to address multiple problems from State's active hypotheses list—if the problems are on its focus list and if State permits it the resources. After selecting a problem, shuffle selects the most appropriate measure of progress and also the most appropriate terminating conditions, given the resource-demand slot of State.

Shuffle then tries to run its *recipe*. Since it is an instance of a "menu-item-shortage" strategy, it will try to find additional resources—in this case, an employee to move to a hamburger grill. This is also a tactical issue. As mentioned earlier, the two tactics for finding employees are to favor unassigned people and to favor those on stations producing surpluses. State tells strategies which of these tactics to select; the *resource-preferences* slot in Fig. 3 says to take free employees before those on surplus stations.

E. When Strategies Fail

If a strategy like shuffle cannot find an employee, it will return control to the interpreter without taking any action, and record its failure in the *resource-shortage-level* slot of State. Another kind of failure happens when a shortage plan successfully finds an additional employee but cannot use him because the station that is producing the shortage is already staffed to capacity. Lastly, a plan can fail if it attempts to take an employee off a station that is producing a surplus, but nobody is working at the station (i.e., the surplus is residual).

F. Termination of Strategies

Assuming shuffle succeeds, it will move an employee to a grill and mark that employee as busy. This means that no other plan can grab that employee until shuffle's termination condition is satisfied, at which time the employee is marked as free. Recall that when a strategy is executed, it looks at State to determine one of three terminating conditions: the strategy is allowed to work until the problem is fixed, or until progress has been made, or for N time units. One tactical issue is whether the terminating conditions of a strategy should be set once or whether they should be updated on every cycle. The argument for the latter is that if resource levels were very tight initially, but relax over time, then the strategy ought to be allowed additional resources; and, conversely, if resources were plentiful immediately but now are tight, the strategy ought to be allowed fewer resources. Clearly, there are many tactical possibilities for terminating strategies, but currently McD allows just two:

- *static-termination-conditions*—termination conditions are set once when the strategy is created and not changed,

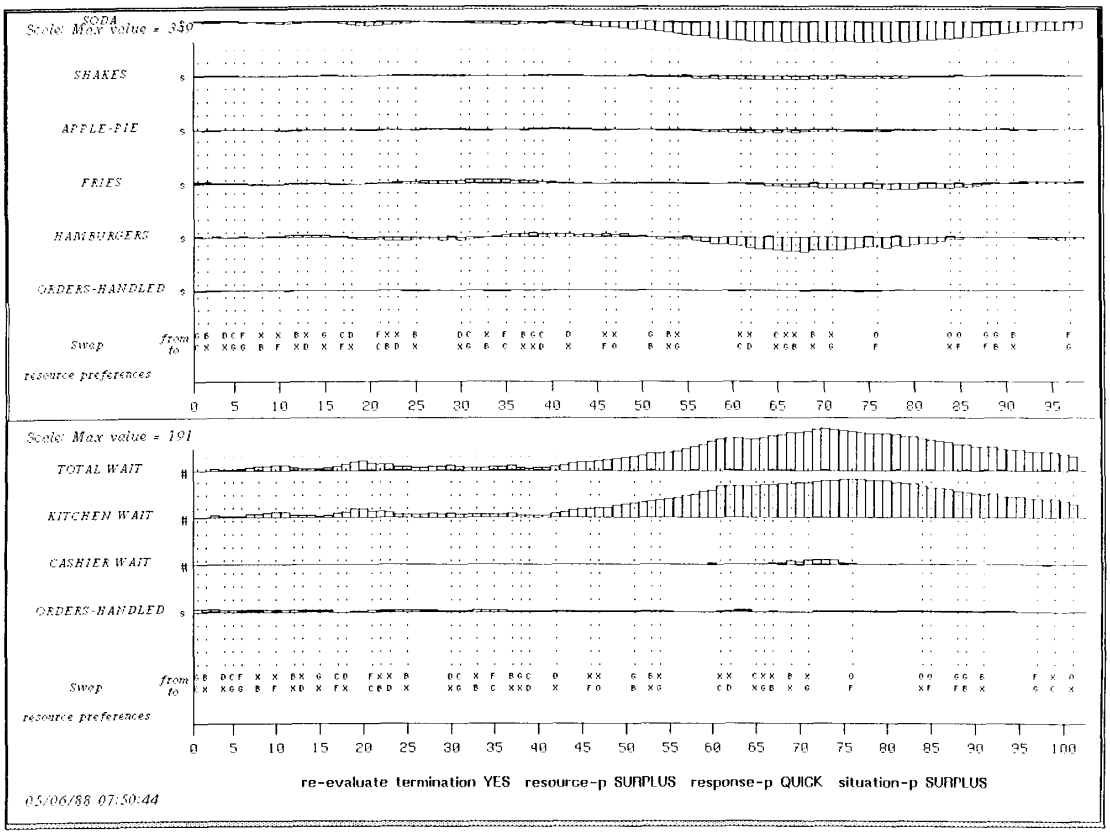


Fig. 4. Graph of initial performance of McD.

- *dynamic-termination-conditions*—termination conditions are updated dynamically on every cycle.

These tactics are selected by the *update-evaluation-measures* slot of State.

In special cases, McD needs resources more rapidly than they are provided by the mechanism just described. It then invokes a strategy called the "terminator." This strategy is always applicable, but its relevance is related to the resource-shortage-level slot of State, so that it is selected for execution only when this level is high. This, recall, is determined by the number of shortage strategies that fail. Once invoked, the terminator will mark resources as free even if their termination conditions have not been met. Three tactics determine how aggressively it does this.

Benign:	Find a resource on a station that has just a small problem (i.e., a small surplus or shortage) and that has already made some progress toward solving the problem.
Edgy:	Like benign, except the problem can be moderate.
Aggressive:	First look for resources on stations where there has been progress, and mark a resource at the station that has the least serious problem; otherwise mark a resource at the station with the least serious problem; but if all stations have equally serious problems, mark the resource that has been assigned longest.

These tactics are currently selected by the resource-demand slot of State. It takes one of three values—restore, improve, and time—depending on the degree of resource demand (restore is the least demand). Resource demand is calculated from historical, current, and anticipated aspects of State.

VI. Examples

In the following examples, we will show how we tuned the McD system by adjusting the values in slots of strategy frames so it would maintain a fairly "trim" configuration. Although this will not demonstrate the necessity of the slots in question, it will demonstrate their sufficiency. We found that we could quickly improve McD's performance to a point using relatively local information. However, once the system achieved a certain level of performance it became impossible to predict whether further changes would improve or detract from performance, which raises questions about the utility of global information.

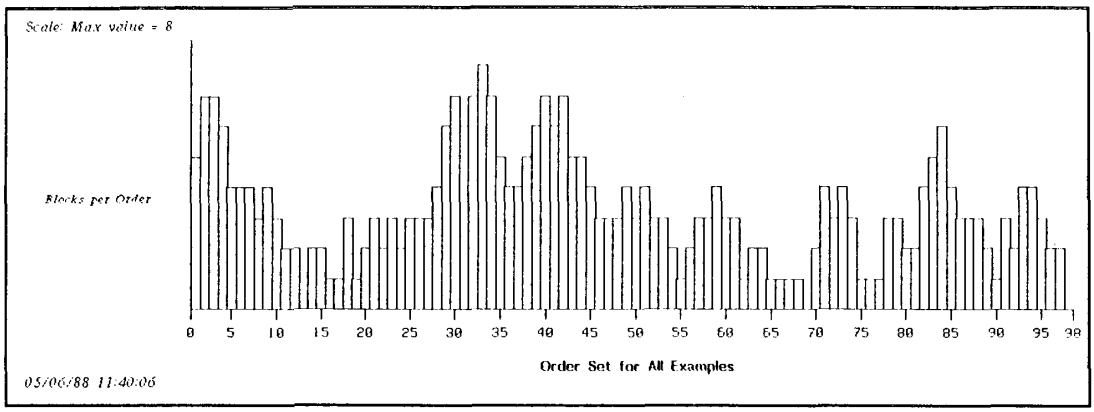


Fig. 5. Order set used for all examples.

We evaluate McD's performance by several metrics. One is a graph, over time, of the supply and demand for various items. Fig. 4 shows a graph in which McD's performance is poor. Although it does not have large shortages or surpluses of food items initially, by time = 20 it develops a shortage of hamburgers, then a little later a surplus of fries. Around time = 40, McD begins to develop a massive shortage of sodas, and around time = 50 a slightly smaller shortage of hamburgers, and then a shortage of fries. As these shortages develop, one can see a corresponding increase in the length of time that customers must wait. This is due almost entirely to waiting for food from the kitchen, although around time = 65 customers must wait a short time for cashiers.

Each movement of an employee is represented in the graph by a vertical dotted line, grounded at a pair of characters that represent the initial and final location of the employee. For example, shortly before time = 40, McD takes an employee from a cashier station (C) and moves it to the drinks station (D), probably to correct the slight shortage of sodas. In fact, the shortage disappears around time = 40, and shortly thereafter McD moves the employee from the drinks station to the list of unassigned employees (X). This, it turns out, was a bad move, since the drinks shortage immediately becomes very serious.

These moves are proposed by strategy frames based on relatively local information. A frame sees a shortage, surplus, or breakdown, and proposes itself to the interpreter. The choice by the interpreter among strategy frames is also based on relatively local information: it selects the strategy frame that is most relevant, that is, the one that potentially fixes the most problems of greatest severity. It does not predict the effects of the resource allocations suggested by each relevant strategy, and in particular does not predict the interactions between these resource allocations and the ones that are already in effect.

McD's performance on each run is summarized with a table of statistics (e.g., Fig. 6 presents the statistics that correspond with Fig. 4). We record the mean surplus, shortage, and total supply of each food item, the standard deviation, and the number of each situation. In Fig. 6 there are 33 situations in which we had a surplus of hamburgers, in which the mean surplus was 30.18 with a standard deviation of 17.98. More significant, of course, is the 65 shortage situations, in which the mean shortage was a distressing −86.31! Overall, the mean level of supply for all 98 cases (including shortage and surpluses) was −47.08. We have a similar pattern of results for the other food items. A summary of these figures is found in the row called "Totals," which contains the weighted means of surpluses, shortages, and overall supply for all the food items. The weighting reflects the rate at which food items can be produced and are typically consumed. Thus big shortages of hamburgers "count the same" as smaller shortages of apple pies, because the latter are produced more slowly. Fig. 6 also includes the average wait for a cashier, and for food once an order has been placed; these figures are 0.27 and 12.20, respectively, indicating that a customer can expect to order in about one-quarter of a time unit, but wait 12 time units for food! Lastly, Fig. 6 includes information about the efficiency with which McD uses its employees. We record the total number of units worked by all employees and the number of those units in which they were productive (not idle). The ratio, or utility mean, is just the ratio of these statistics. In this case, the employees were not particularly efficient (71 percent).

McD's performance in Figs. 4 and 6 is fairly poor. Now we will show how, by modifying the slot values of McD's strategy frames, we improved performance. The previous example and all the subsequent ones use the order set shown in Fig. 5. It contains 100 blocks of orders. Each block contains between zero and eight orders (distributed around a mean of four orders), and each order includes zero to seven orders of pies, burgers, fries, sodas, and shakes.

The performance illustrated in Figs. 4 and 6 is poor for several reasons. First, the mean wait for food is over 12 units—much too long. This is due in part to the enormous mean shortage (−53.9): McD is out of everything most of the time, so customers have to wait. Indeed, there are 308

	Surplus	Shortage	Total
HAMBURGERS			
mean	30.18	-86.31	-47.08
st. dev	17.98	75.92	83.44
number	33.00	65.00	98.00
FRIES			
mean	25.94	-41.66	-8.82
st. dev	17.28	30.73	41.90
number	47.00	50.00	98.00
APPLE-PIE			
mean	9.17	-19.63	-5.91
st. dev	4.95	14.99	18.26
number	46.00	51.00	98.00
SHAKES			
mean	7.80	-27.55	-11.63
st. dev	4.71	20.78	23.31
number	41.00	53.00	98.00
SODA			
mean	19.11	-154.92	-138.94
st. dev	9.96	120.10	125.04
number	9.00	89.00	98.00
TOTALS (weighted)			
mean	17.11	-53.91	-27.74
st. dev	11.94	52.92	54.53
number	176.00	308.00	490.00

Avg. Wait all Customers:		St. Deviation
Cashier wait	0.27	1.37
Kitchen wait	12.20	17.95

Employee Stats:	
Total Productive time units	478.00
Total On Duty time units	678.00
Utility mean	0.71
Utility st. dev	0.19

Fig. 6. Statistics for production and utilization from initial run.

shortage situations and 176 surplus situations. Our immediate goals were to reduce these shortages and reduce the waiting time for food.

By changing the values of slots in strategy frames, we were able to affect the following aspects of McD's behavior.

- *Preference among situations.* McD can react more quickly to surpluses, or more quickly to shortages, or equally to both, as determined by the *situation-preferences* slot of State.
- *Where to get resources.* McD can either favor taking resources from stations that are producing a surplus, or it can favor taking the resources from a list of free resources. In the first case, if no stations are producing surpluses, McD will look to the free list; in the second case, conversely, if there are no free employees, McD will look at stations producing surpluses. This choice is dictated by the *resource-preferences* slot of State.
- *Speed of response.* McD can react quickly or slowly to situations, as dictated by the *response-preferences* slot of State.
- *Whether to re-evaluate termination conditions.* McD can allow strategy frames to re-evaluate their terminating conditions dynamically, or it can set terminating conditions once for each strategy as it is invoked. This is determined by the *update-evaluation-measures* slot of State.

The version of McD discussed earlier was configured to respond to surpluses before shortages, to get employees from surplus stations before free employees, to respond quickly, and to re-evaluate its terminating conditions. Since this version generates huge shortages, it makes little sense to attend to surpluses before shortages. Thus we ran the system again with it attending to shortages first. The results were much better: The mean surplus was 24.7, up from 17.1; but the mean shortage was -32.2, down considerably from -53.9; and the total mean supply was -4.9, down from -27.4. Moreover, the cashier wait dropped from 0.27 to 0.08; and more importantly, the kitchen wait dropped from 12.2 to 6.5. There was no improvement in the efficiency with which McD used its resources. Clearly, attending to shortages before surpluses improves McD's performance.

Still, it is awkward to wait 6.5 time units for one's food. We also noted that cashier wait was almost nonexistent. If cashier wait was increased slightly, then it might delay some orders being placed and reduce the shortages, and thus the kitchen wait time. In our next run we changed the applicability conditions for cashier shortages and surpluses, to register a shortage less quickly and to register a surplus more quickly. (Although these functions are not shown in any of the figures in this paper, they too are explicit and easy to modify.) The net result, we hoped, would be to free up cashiers and make them easier to reassign. At the same time, we changed the resource-preferences slot in State so McD would look for resources on the free list before surplus stations. The net result was to improve performance slightly. The cashier wait increased slightly (from 0.08 to 0.42) and the kitchen wait decreased slightly (from 6.5 to 5.8). At the same time the mean surplus decreased a little and the mean shortage improved a little.

Beyond this, we had little success improving McD's performance. This is a story that has repeated itself many times. We can get performance to some reasonable level by a small number of configuration changes, all of which rely on relatively local information. After that, improvements in performance seem to require relatively global information that is difficult and sometimes impossible to acquire. For example, in the aforementioned we tried changing the applicability conditions for cashier shortages in the hope that, by making it harder to get to a cashier, fewer orders would be placed, and so the wait for food would be decreased. This argument, based on nonlocal interactions of assignments of resources, suggests that relatively global information could improve McD's performance; but as noted above, this strategy had only a small effect. In fact, most of our attempts to predict the effects of interactions of resources, and so improve overall performance, failed. Beyond the effects of the really big changes, which can be selected based on local information, the other effects are too susceptible to the global interactions of many vari-

ables, and levels of shortages and surpluses, and other dynamic factors, for us to predict them.

VII. Conclusion

Starting from the position that complex control strategies can be viewed as the interaction of smaller strategic units, we developed a representation for these units and showed how their tactical instantiations could be adjusted to tune control strategies. We showed that particular slots of strategy frames are sufficient to control McD in a dynamic environment, but we did not show their necessity. Nor were we able to predict the effects of changes to all slots. Clearly then, research with strategy frames is nascent.

Future research with strategy frames will emphasize the subtle interactions between how strategy frames are instantiated with tactics and the dynamic environment in which they are used. That is, we want better predictability. This is essential if we are to study the tradeoffs between the costs and benefits of relatively local and global information. Currently, we have no reliable methods to predict the global interactions of strategies' allocations of resources, so we cannot guess at the utility of this knowledge. This is our priority for work in the future.

Strategy frames were motivated by the need for declarative representations of strategy, and by the observation that, in many systems, devices with similar functionality had been implemented in an *ad hoc* way. Ideally, strategy frames will become part of the AI programmer's toolbox, complete with tactical instantiations for various tasks and interpreters capable of dynamically selecting the most appropriate instantiations. Before that, there is much work to be done. Strategy frames imply "emergent" control, that is, control strategies that arise from interactions of individual control decisions. In McD, and in general, these decisions allocate computational and other resources. The basic question is whether the global interactions between these decisions must be examined before they are taken, or whether acceptable control can be based on relatively local information. Our preliminary experiments suggest that local information is sufficient to replace grossly inefficient configurations of McD with more efficient ones, but beyond that, performance depends on currently unpredictable global interactions among resource allocations. Further progress depends on making these interactions more predictable.

Acknowledgment

Special thanks to Ken Ward for help with programming, and to an anonymous reviewer for insightful comments.

References

[1] "Knowledge-based planning workshop," DARPA, Austin, TX, July 1987.

[2] L. Erman and V. R. Lesser, "A multi-level organization for problem solving using many diverse, cooperating sources of knowledge," in *Proc. Int. Joint Conf. on Artificial Intelligence*, 1975.

[3] V. R. Lesser, E. H. Durfee, and J. Pavlin, "Approximate processing in real-time problem solving," *AI Mag.*, pp. 49–61, Spring 1988.

[4] T. Dean, "Large-scale temporal data bases for planning in complex domains," in *Proc. Tenth Int. Joint Conf. on Artificial Intelligence*, Milan, Italy, 1987.

[5] ____, "Planning, execution and control," in *Proc. DARPA Knowledge-Based Planning Workshop*, Austin, TX, Dec. 1987.

[6] S. Hanks, "Temporal reasoning about uncertain worlds," in *Third Workshop on Uncertainty in Artificial Intelligence*, American Assn. Artificial Intelligence, 1987, pp. 114–122.

[7] M. Herman and J. S. Albus, "Real-time hierarchical planning for multiple mobile robots," in *Proc. DARPA Knowledge-Based Planning Workshop*, Austin, TX, Dec. 1987.

[8] R. Korf, "Real-time heuristic search: First results," in *Proc. Sixth National Conf. on Artificial Intelligence*, Seattle, WA, 1987, pp. 133–138.

[9] D. McDermott, "A temporal logic for reasoning about processes and plans," *Cognitive Sci.*, vol. 6, pp. 101–155, 1982.

[10] A. Howe, J. R. Dixon, P. R. Cohen, and M. K. Simmons, "Dominic: A domain-independent program for mechanical engineering design," *Int. J. Artificial Intell. Eng.*, vol. 1, no. 1; pp. 23–29, July 1986.

[11] M. F. Orelup, "Meta-control in domain-independent design by iterative redesign," master's thesis, Mechanical Eng. Dept., Univ. Massachusetts, Sept. 1987.

[12] M. F. Orelup, J. R. Dixon, P. R. Cohen, and Melvin K. Simmons, "Dominic II: Meta-level control in iterative redesign," in *Proc. Seventh Nat. Conf. on Artificial Intelligence*, St. Paul, MN, Aug. 1988, pp. 25–30.

[13] W. J. Pardee and B. Hayes-Roth, "Intelligent real time control of material processing," Technical Rep., Rockwell International, Science Center, Palo Alto Laboratory, Feb. 1987.

[14] S. Marcus, J. McDermott, and T. Wang, "Knowledge acquisition for constructive systems," in *Proc. Ninth Int. Joint Conf. on Artificial Intelligence*, Los Angeles, CA, Aug. 1985, pp. 637–639.

[15] S. Marcus, "Taking backtracking with a grain of SALT," *Int. J. Man–Machine Studies*, vol. 26, no. 4, pp. 383–398, 1987.

[16] E. R. Bareiss, B. W. Porter, and C. C. Wier, "Protos: An exemplar-based learning apprentice," in *Proc. Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, Oct. 1987.

[17] J. H. Boose and J. M. Bradshaw, "Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for expert systems," *Int. J. of Man–Machine Studies*, vol. 26, no. 1, pp. 21–25, Jan. 1987.

[18] L. Eshelman, "MOLE: A knowledge acquisition tool that buries certainty factors," in *Proc. Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, Oct. 1987.

[19] M. A. Musen, L. M. Fagan, D. M. Combs, and E. H. Shortliffe, "Using a domain model to drive an interactive knowledge editing tool," *Int. J. Man–Machine Studies*, vol. 26, no. 1, p. 105, 1987.

[20] T. R. Gruber, "Acquiring strategic knowledge from experts," *Int. J. Man–Machine Studies*, 1988.

[21] T. R. Gruber, "The acquisition of strategic knowledge" Ph.D. thesis, Univ. of Massachusetts, 1988; also to be published in book form by Academic Press.

[22] R. Davis, "Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases," Ph.D. thesis, Computer Science Dept. Stanford University, 1976; reprinted in *Knowledge-Based Systems in Artificial Intelligence*, R. Davis and D. B. Lenat, Eds. New York: McGraw-Hill, 1982.

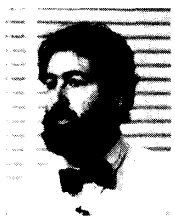
[23] W. J. Clancey, "Acquiring, representing, and evaluating a competence model of diagnosis," KSL Memo 84-2, Stanford University, Feb. 1984.

[24] W. J. Clancey, "Representing control knowledge as abstract tasks and metarules," in *Computer Expert Systems*, M. Coombs and L. Bolc, Eds. Springer-Verlag, also KSL Memo 85-16, Stanford University.

[25] P. R. Cohen, D. S. Day, J. Delisio, M. Greenberg, R. Kjeldsen, D. Suthers, and P. Berman, "Management of uncertainty in medicine," *Int. J. Approximate Reasoning*, vol. 1, no. 1, pp. 103–116, 1987.

[26] P. R. Cohen, M. Greenberg, and J. Delisio, "MU: A development environment for prospective reasoning systems," in *Proc. Sixth National Conf. on Artificial Intelligence*, Seattle, WA, July 1987, pp. 783–788.

[27] P. R. Cohen and D. S. Day, "The centrality of autonomous agents in theories of action under uncertainty," EKSL Technical Rep. 88-14, Univ. Massachusetts, Jan. 1988.
[28] P. R. Cohen and A. E. Howe, "How evaluation guides AI research," AI Magazine, vol. 9, no. 4, pp. 35–43, 1988.
[29] P. R. Cohen and A. E. Howe, "Toward AI research methodology: three case studies in evaluation," *IEEE Trans. Syst. Man Cybern.*, this issue, pp. 634–646.
[30] S. Weiss, C. Kulikowski, and A. Safir, "A model-based consultation system for the long-term management of glaucoma," in *Proc. IJCAI 5*, 1977, pp. 826–832.
[31] S. M. Weiss, C. A. Kulikowski, S. Amarel, and A. Safir, "A model-based method for computer-aided medical decision-making," *Artificial Intelligence*, vol. 11, nos. 1, 2, pp. 145–172, 1978.
[32] S. G. Pauker, A. Gorry, J. P. Kassirer, and W. B. Schwartz, "Towards the simulation of clinical cognition: Taking a present illness by computer," *Amer. J. Medicine*, vol. 60, pp. 981–996, 1976.
[33] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty," *Computing Surveys*, vol. 12, pp. 213–253, 1980.
[34] B. Chandrasekaran, S. Mittal, and J. W. Smith, "Reasoning with uncertain knowledge: the MDX approach," in *Proc. Congress of American Medical Informatics Assn.*, San Francisco, 1982, pp. 335–339.
[35] B. Chandrasekaran and S. Mittal, "Conceptual representation of medical knowledge for diagnosis by computer: MDX and related systems," in *Advances in Computers*, M. Yovits, Ed. New York: Academic Press, 1983, pp. 217–293.
[36] W. J. Clancey, "From guidon to NEOMYCIN and HERACLES in twenty short lessons," *AI Mag.*, vol. 7, no. 3, pp. 40–60, 1986.
[37] E. Shortliffe, *Computer-Based Medical Consultations: MYCIN*. New York: American Elsevier, 1976.
[38] W. J. Clancey, "Classification problem solving," in *Proc. Fourth National Conf. on Artificial Intelligence*, 1984, p. 49.
[39] ____, "Heuristic classification," *Artificial Intelligence*, vol. 27, pp. 289–350, 1985.
[40] T. Bylander and B. Chandrasekaran, "Generic tasks in knowledge-based reasoning: The 'right' level of abstraction for knowledge acquisition," *Int. J. Man-Machine Studies*, vol. 26, no. 2, pp. 231–244, 1987.
[41] B. Chandrasekaran, "Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design," *IEEE Expert*, vol. 1, no. 3, pp. 23–30, Fall 1986.
[42] ____, "Towards a taxonomy of problem solving types," *AI Mag.*, vol. 4, no. 1, pp. 9–17, 1983.
[43] T. R. Gruber and P. R. Cohen, "Knowledge engineering tools at the architecture level," in *Proc. Tenth Int. Joint Conf. on Artificial Intelligence*, Milan, Italy, 1987, pp. 100–103.
[44] ____, "Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition," *Int. J. Man-Machine Studies*, vol. 26, no. 2, pp. 143–159, 1987.
[45] A. Garvey, C. Cornelius, and B. Hayes-Roth, "Computational costs versus benefits of control reasoning," in *Proc. AAAI-87: Sixth National Conf. on Artificial Intelligence*, American Association for Artificial Intelligence, 1987.



Paul Cohen received the Ph.D. degree from Stanford University in 1983.

His research interest is planning and reasoning under uncertainty in knowledge-based systems. He is an Assistant Professor of computer and information science at the University of Massachusetts, Amherst, where he directs the Experimental Knowledge Systems Laboratory.

Dr. Cohen is an editor of *The Handbook of Artificial Intelligence*.

Jefferson DeLisio was born in Washington, D.C. He did his undergraduate study at Marlboro College in Vermont and the University of Massachusetts, Amherst. He received the M.S. degree in computer and information science in 1988 from the University of Massachusetts.

He is co-Developer of a system architecture for interpretation and diagnostic tasks, MU, and has worked in the domain of medical diagnosis in collaboration with Dr. Paul Berman of the University of Massachusetts Health Center. His research interests include real-time problem solving, task-specific architectures, case-based reasoning, and knowledge acquisition.

Mr. DeLisio is a member of the AAAI and the Cognitive Science Society.

David M. Hart was born in 1954 in Natchez, MS. He received the B.A. degree in English from Hampshire College, Amherst, MA.

He is currently pursuing the Ph.D. degree at the University of Massachusetts, Amherst. He worked for four years at the University Computing Center at the University of Massachusetts, Amherst, where he was a member of the software group that developed Lisp/VE for Control Data Corporation. His research interests include knowledge-based systems, process control, real-time dynamic planning, and operations research.

Mr. Hart is a member of the AAAI and ACM.