Int. J. Man–Machine Studies (1989) **30**, 377–398

Building routine planning systems and explaining their behaviour

B. CHANDRASEKARAN, JOHN JOSEPHSON, ANNE KEUNEKE AND DAVID HERMAN

Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210, USA

(Received 22 September 1987 and in revised form 3 January 1988)

It has become increasingly clear to builders of knowledge-based systems that no single representational formalism or control construct is optimal for encoding the wide variety of types of problem solving that commonly arise and are of practical significance. In this paper we identify a class of problem solving activities which we have labeled *routine* planning. We consider the constructs necessary to represent the problem solving which appropriately characterizes this class, and describe DSPL, a high-level language designed specifically to encompass the required knowledge structures and control methodology for routine planning. Finally, we consider what type of structure is appropriate to represent an agent's understanding of how the plan itself works.

1. Introduction

The task of planning is neither simple nor well-defined. Within AI, attempts to achieve a better understanding of the task have ranged from proposals of general theories (Wilensky, 1983; Schank & Abelson, 1977) to more specific approaches for particular types of planning (Hayes-Roth *et al.*, 1979; Fox *et al.*, 1983; Brown & Chandrasekaran, 1985*a*). The purpose of this paper is not to present a highly sophisticated general theory for complex planning tasks. Rather, we identify an element or type of planning activity as a generic primitive of knowledge-based reasoning, i.e. we identify a type of knowledge structure with an associated inference or control mechanism as a *building block* in the construction of knowledge-based systems. This particular generic structure is especially effective in capturing a type of planning which we call *routine* planning.

1.1. PLANNING: AN APPROACH

Commonly, the problem solving activity in planning has been described as the determination of sequences of actions in order to achieve desired goals in given situations. That is, at the topmost level, planning has the character of a synthesis of operations to be performed on the world. When approached from a formal viewpoint, it is thus noted that *planning* and *design* are similar activities in that both are attempting to produce a "product" which can achieve some goal; they are both

A shorter version of this paper was presented at the Darpa Knowledge-Based Planning Workshop, Austin, Texas (December 1987).

This research is supported by the Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010.

0020-7373/89/040377 + 22$03.00/0

activities of construction or synthesis. Synthesis activities, from a computational viewpoint, can be very complex. Generally the search spaces for plausible solutions are very large and within that space there are only a few solutions that, in fact, meet the goals.

One way to reduce this complexity is to use *domain knowledge*: in particular,

- knowledge about how to decompose the given planning problem into planning problems in a smaller problem space, and
- knowledge about subproblems in the form of pre-compiled plans for solving them, often obtained from earlier attempts to solve a similar subproblem. These pre-compiled plans may be *abstract,* i.e., they may include pointers to other plans for further instantiation.

It is worth pointing out that, in the early days of AI, Miller, Galanter & Pribram (1960) proposed that much of an intelligent agent's knowledge may be in the form of mini-plans of this type. The decomposition process typically leads to hierarchical designs or plans. The dependence on partially compiled abstract plans leads to an agent whose knowledge is rich in a vast array of plans, multiply indexed by relevant goals and each pointing to other plans for possible use in completing the problem solving process.

It is clear that planning is a knowledge-intensive activity. Thus, as in all knowledge-based theories, the relevant questions concern the representation of knowledge (in this case, planning knowledge) and the inference mechanisms for problem solving.

We have long advocated that, in contrast to the current approaches where knowledge and inference are separated, knowledge representation should be tailored to the type of task, i.e. type of usage of knowledge. Knowledge should be directly encoded at the appropriate level by using primitives that naturally describe the domain knowledge for a given task. Problem solving behaviour for the task ought to be controlled by regimes that are appropriate for the task (Chandrasekaran, 1984). If done correctly, this would simultaneously facilitate knowledge representation, problem solving, and explanation. In order to do this effectively, one needs a theory of types of generic tasks in knowledge-based reasoning. In Chandrasekaran (1986) we provide such a theory. In particular, we provide a framework that gives a view of the level of abstraction at which it is productive to look for generic types of problem solving. We argue that the level of rules–frames–logic is at too low a level for this characterization and provide an alternative level at which the tasks become much clearer as knowledge level phenomena.

A theory of planning viewed as a knowledge-based activity thus has to account for the following:

- an ontology of the planning task viewed as a generic activity (i.e., the generic terms in which domain specific planning knowledge is encoded)
- how such knowledge is *structured* and *organized* for effective utilization during the planning process
- what kinds of *control processes* are used to access this knowledge and synthesize a plan.

In order to provide such an account, it is necessary to decide whether planning

constitutes a relatively unitary generic problem solving activity or if it is a label under which a number of rather distinct problem solving processes have been coalesced. The variation in types of planning problems leads one to believe the latter. For example, what is common between planning a day's activities in order to complete a collection of errands (Hayes-Roth, 1980) and planning a therapy (Shortliffe *et al.*, 1981), or between generating a plan to arrange furniture in a room and planning the construction of a house? The errand and furniture planning problems have a significant formal similarity; they both have the character of arranging things in some space following some constraints. On the other hand, the routine versions of the therapy and house construction problems generally share the flavour of hierarchical plan refinement. Each of these really corresponds to different knowledge needs and different control processes.

In our research, we have identified one type of planning or design problem which can be characterized as *routine* planning or design. In these kinds of problems, the structure of the object under design is known, i.e. the planning problem can be hierarchically decomposed and the hierarchy of parts is known, and also knowledge for designing the components is available in the form of a set of abstract, compiled and partial plans for each component. The plans are partial because they typically involve calling upon plans for the subcomponents. Typically knowledge about failures and how to back up is also available. This kind of task corresponds to what experts in a given domain are usually faced with.

In this paper we look at routine planning as a generic task. Using the domain of tactical mission planning, we describe DSPL, a high-level language designed specifically for the problem solving activity of routine planning. The advantages of the explicit use of problem solving knowledge are discussed. Finally we consider what type of structure is appropriate to represent an agent's understanding of a plan and thus produce more perspicuous explanation of the planning process (Chandrasekaran *et al.*, 1987)

2. The mission planning domain

Tactical mission planning in the Air Force essentially involves the assignment of resources to various tasks. The resources are primarily aircraft and their stores located at airbases across the theater of operations. The tasks are specified by an "apportionment" order issued by the Joint Task Force Commander to a Tactical Air Control Center (TACC). This order describes the overall military objectives as determined by the Task Force Commander. The TACC is responsible for assigning aircraft and personnel from specific military units to meet the objectives of the apportionment order. The result of these assignments is an "Air Tasking Order" (ATO) which summarizes the responsibilities of each unit with respect to the day's missions. Each mission planned requires attention to such details as the selection of aircraft type appropriate to the mission, selection of a base from which to fly the mission, and coordination with other missions.

The MPA system we have developed currently addresses only a single type of mission, the Offensive Counter-Air (OCA) mission. An OCA mission is an air strike directed specifically against an enemy airbase. Our selection of the OCA mission arose in part because of the availability of KNOBS and its knowledge base

of relevant domain facts. More importantly, however, we are interested in the planning process itself and the ability to explain the reasoning behind this process.

3. Routine design

Our approach to tactical mission planning treats the Air Tasking Order (ATO) as an abstract device to be designed. The planning of the missions of the completed ATO involves a process similar to the process a designer undergoes when faced with a complex device to design. An overview of the design domain will illuminate this analogy. For a more comprehensive description of the mechanical design domain see Brown (1984).

The general domain of design is vast; it involves creativity, various problem-solving techniques, and many kinds of knowledge. Goals are often poorly specified, and may change during the course of problem solving. However, a spectrum of classes of design problems can be identified, varying in complexity from open-ended to routine. The complexity of the design problems in each class depends on what sort of knowledge is available to the problem solver prior to the start of design.

What we have called "Class 3 Design" characterizes a form of routine design activity. We postulate that several types of knowledge are available prior to problem solving. First, complete knowledge of the components of the device to be designed, including their attributes, is available to the problem solver. Thus, the final design consists only of components known in advance, and no novel components need to be synthesized. Second, complete knowledge of design actions in the form of plan fragments is available to the problem solver. The problem solving proceeds by using recognition knowledge to select among the previously known sequences of design actions. While the choices at each point may be simple, this does not imply that the design process itself is simple, nor that the components so designed must be simple. It appears that a significant portion of everyday activity of practicing designers falls into this class. In order to represent this class of design problems we have developed the high-level programming language, DSPL (Design Specialists and Plans Language).

3.1. DSPL: A STRUCTURED SOLUTION

A routine design task can be broken down into a hierarchy of planning tasks. At each level of the hierarchy some design commitments are made. Lower levels of the hierarchy further refine the design. In DSPL, a design problem solver consists of a hierarchy of cooperating, conceptual specialists, with each specialist responsible for a particular portion of the design. Specialists higher up in the hierarchy deal with the more general aspects of the device being designed, while specialists lower in the hierarchy design more specific sub-portions of the device, or other design subtasks. Any specialist may access a design database (mediated by an intelligent database assistant). This organization of the specialists captures the divide and conquer nature of the designer's expertise in the problem domain.

Each specialist in the design hierarchy contains local design knowledge necessary to accomplish its portion of the design. There are several types of knowledge represented in each specialist, three of which determine the overall control strategy of a DSPL system. First, explicit design plans in each specialist encode sequences of

possible actions to successfully complete the specialist's task. Different design plans within a specialist may encode alternative action sequences, but plans within a particular specialist are always aimed at achieving the specific design goals of that specialist. A second type of knowledge encoded within specialists is encoded in design plan sponsors. Each design plan has an associated sponsor to determine the appropriateness of the plan in the run-time context. The third type of planning knowledge in a specialist is encoded in design plan selectors. The function of the selector knowledge is to examine the run-time judgements of the design plan sponsors and determine which of the design plans within the specialist is most appropriate to the current problem context. These three types of knowledge, design plans, plan sponsors and plan selectors, form the core of the control knowledge within a specialist.

Constraints are another type of local, declarative knowledge in a DSPL specialist. Constraints are used to decide the suitability of incoming requirements and data, and on the ultimate success of the specialist itself (i.e., the constraints capture knowledge about those things that must be true of the specialist's design before it can be considered to be successfully completed). Other constraints embedded in the specialist's design plans are used to check the correctness of intermediate design decisions. The use of such constraints in the MPA system easily captures the kinds of knowledge encoded as constraints in KNOBS, but incorporating the constraints into a rich overall control structure further allows the constraint knowledge to be utilized during problem solving in a sharply focused manner.

Control in a DSPL system proceeds from the top-most specialist in the design hierarchy to the lowest. Beginning with the top-most specialist, each specialist selects a design plan appropriate to the requirements of the problem and the current state of the solution. The selected plan is executed by performing the design actions specified by the plan. This may include computing and assigning specific values to attributes of the device, running constraints to check the progress of the design, or invoking sub-specialists to complete another portion of the design. Design plans which refer to a sub-specialist are refined by passing control to that sub-specialist.

The discussion of the control strategies in a DSPL system has thus far only included successful plan execution. However DSPL does include facilities for the handling of various types of plan failures, and for controlling redesign suggested by such failures (Brown, 1984; Brown & Chandrasekaran, 1985*b*).

4. Approaches to mission planning: hierarchical decomposition and template instantiation

4.1. MISSION PLANNING AS ROUTINE DESIGN

We view tactical mission planning as essentially a routine design task. The problem can be decomposed into the design of subcomponents of the mission plan. In the device design domain, the design of a device is decomposed into the design of sub-assemblies and their components, etc, where each sub-assembly or component can be designed in a fairly independent fashion. In the tactical mission planning domain the ATO is decomposed into various missions or groups of missions of known types. Furthermore, each mission or group of missions can be planned

relatively independently of the others, except for resource contention considerations. In both the mission planning and the mechanical design domains, of course, each of the solutions to the subproblems must be appropriately combined into the solution for the problem which they decompose. Due to the well known limitations of human problem solving capacities, it is apparent that a human problem solver can be successful in such a situation only to the extent that he can also decompose the problem into a manageable number of somewhat independent subproblems which can be solved separately and combined into a final solution. Using DSPL as a natural mechanism for representing and structuring the necessary knowledge, the MPA system closely mirrors these ideas. To better illustrate the power of this approach, we have analysed the strategy of the KNOBS system, which takes a less explicit approach toward planning.

4.2. THE KNOBS APPROACH

KNOBS (Engelman *et al.*, 1983; Scarl *et al.*, 1984) is a planning system which had been applied to the domain of tactical mission planning. In fact, the domain knowledge found in the MPA system was mostly derived from this system. KNOBS presents an alternative to the hierarchical decomposition of planning problems suggested by the DSPL system. This section briefly describes this approach and some of its weaknesses.

KNOBS uses template instantiation to control the planning process. When problem solving begins, a template of the desired plan is instantiated. As problem solving progresses, slots in the template are filled with acceptable values. Each slot represents an attribute of a pre-determined component of the final plan. This aspect of the planning process is similar to our view of routine design and the DSPL methodology: all of the components and all of the attributes of the components are known prior to the start of planning. The mechanism used to organize the selection of values, however, is significantly different.

In this system, the order in which the slots are considered is defined in advance by the plan template. Acceptability of slot values is based on constraint satisfaction. The constraints are organized as a list of "buckets", ordered to express priority in constraint satisfaction. Each bucket contains an unordered list of constraints. The testing of slot values is accomplished by traversing and checking constraints in the order specified by the priority buckets.

In order to determine acceptable choices for values of slots, KNOBS associates a generator with each slot to enumerate potential values. The generator produces a subset of all possible values of the slot. The generator is derived by "inverting" constraint knowledge pertinent to the slot.

Given the slot ordering, constraints, and generators, problem solving proceeds as follows: the generator of the first slot is asked for its first candidate, the generator for the second slot is asked for its first candidate, and so on. At each slot filling, all applicable constraints are checked. If any are not satisfied, then the slot generator is asked for another candidate. If another candidate exists, it is tried, and so on until either all slots have accepted values or a generator runs out of candidates. When no candidates are available, the system backs up to the most recently filled slot mentioned in the failing constraint, and generates a new value. Planning is

successful when all slots are filled and all constraints are satisfied. The basic planning strategy can be described as generate and test with dependency-directed backtracking.

While the generate and test approach to planning represents a general solution to a broad range of planning problems, it does exhibit several drawbacks. The methodology will not typically scale up well, since, as the size of problem space increases, the exhaustive depth-first nature of the search makes the technique computationally infeasible. Also, the ability to automatically generate explanations of the planner's behaviour is limited since most of the planning knowledge is implicit (and thus hidden) in the ordering of the slots and the constraint buckets.

These problems arise in part because the system does not have significant amounts of explicit knowledge about planning strategies. Explanation is limited to answers based on constraint knowledge—either a value is bad because it fails a constraint, or it is good because it satisfies a constraint. There is no knowledge of why the system should satisfy a constraint (its functionality) nor why this constraint (*vs* any other) is being considered now (plan strategy). Thus, neither the planning control strategy nor the functional knowledge of the domain can be justified. For a planner to achieve explanation of this type, the planning and functional knowledge must be represented with appropriate explicit structures.

5. The MPA system

The following discussion gives a general description of the planning strategies particular to the MPA system as currently implemented in DSPL. It should be noted that several caveats are in order concerning the domain of the MPA system. The MPA system currently only handles the planning of OCA missions, although we believe other missions could be handled in a similar fashion. Our prototype system does not address several minor bookkeeping aspects of mission planning, which although of no theoretical interest, would be necessary to a fully functional mission planner. Such items as assigning radio frequencies to a flight and designating mission call-signs fall into this category. Finally, although the specific military knowledge in the MPA system is adequate for demonstration purposes, it is by no means meant to reflect complete or even accurate knowledge of aircraft capabilities. We believe that the knowledge represented is representative of the knowledge utilized by a human mission planner, and that the problem solving exhibited by the system fairly represents the human problem solver's activities.

The prototype MPA system contains six specialists. The hierarchy of specialists is shown in Fig. 1. The topmost specialist, *OCA*, accepts the mission requirements and ultimately produces the final mission plan. The OCA specialist divides its work between two subspecialists, *base* and *aircraft*. The base specialist is responsible for selecting an appropriate base, while the aircraft specialist selects an aircraft type. The aircraft specialist has three subspecialists, one for each of three aircraft types known to the MPA system. As needed, one of these specialists will select an appropriate configuration for its aircraft type.

Problem solving begins when the OCA specialist is requested to plan a mission. Currently, the OCA specialist contains only a single design plan which first requests the base specialist to determine a base, and then requests the aircraft specialist to

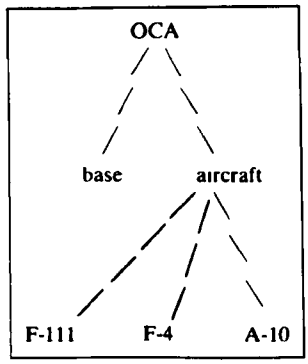

FIG. 1. The MPA hierarchy of specialists.

determine (and configure) an appropriate aircraft for the mission. The current base specialist simply selects a base from a list of candidate bases geographically near the target. The aircraft specialist uses considerations of threat types and weather conditions at the target to select an appropriate aircraft for the mission. The aircraft specialist and its three configuration subspecialists represent the most elaborate aspects of domain knowledge in the MPA system.

In the current version of the MPA system, the aircraft specialist is entered with a tentative selection for the base already specified. The target and required probability of destruction are known from the input requirements of the mission. When the aircraft specialist is entered, each of its plan sponsors are executed by the DSPL

```
(SPONSOR A-10
(BODY
target (KB-FETCH TARGET)
timeOverTarget (KB-FETCH TIMEOVERTARGET)
threat
  (TABLE
    (MATCH (airborne)(AAA)(SAM))
      (IF F F F THEN PERFECT)
    (OTHERWISE UNSUITABLE))
conditions
  (TABLE
    (MATCH (night)(weather))
      (IF F CLEAR THEN PERFECT)
      (IF F PARTIAL THEN SUITABLE)
    (OTHERWISE UNSUITABLE))
REPLY (COMBINE THREAT CONDITIONS)))
```

FIG. 2. DSPL code for a Design Plan Sponsor.

The general form for the body of plan sponsors, plan selectors and steps is a list of pairs of *variable expression*, where the value of the expression is assigned to the variable. REPLY is a special variable whose value is returned when the execution of the body is complete.

```
(PLAN A-10
(BODY
  (TASK assign-mission aircraft)
  (DESIGN A-10)
  (TASK determine-mission-pd)
  (TASK assign-squadron)))
```

FIG. 3. DSPL code for a Design Plan.

interpreter. The DSPL code for one such plan sponsor and its associated plan is given in Figs 2 & 3. The three plan sponsors determine the appropriateness of their respective plans. In this case, each of the three plans determine which of the three aircraft types should be used for the mission. Thus the three plan sponsors determine the appropriateness of using, respectively, F-111s, F-4s, or A-10s for the mission. Plan sponsors may access a global database as necessary in their execution. In the MPA system, items such as target characteristics and weather conditions are requested in determining the appropriateness of a particular aircraft type. After the suitability of all plans in the aircraft specialist has been determined, the DSPL interpreter executes the plan selector in the specialist. Figure 6 contains the DSPL code of the plan selector. The plan selector, given the suitabilities of each of the aircraft types, can then determine which aircraft is most appropriate for the mission. The plan selector returns the name of the plan to the specialist, which then executes the plan. If no plan is found to be applicable, the plan selector returns the primitive **NO-PLANS-APPLICABLE** to the specialist.

Suppose the mission requirements call for a night raid. The plan sponsors for both the A-10 and F-4 would rule out the possibility of using these aircraft, since (in our domain model) neither of these aircraft have night flying capability. The F-111 plan sponsor, since it is an all-weather fighter with night capabilities, would not be excluded. The plan sponsor for the F-111, based on this and other considerations (range, ability to carry appropriate ordinance, target characteristics, etc) would find the F-111 suitable for the mission. The plan selector in the aircraft specialist, finding that two design plans have ruled out, would select the "suitable" F-111 design plan, and return this information to the specialist. The specialist executes the F-111 design plan, which includes setting the aircraft type in the mission template to "F-111", and invoking the F-111 configuration specialist which in turn decides an acceptable ordinance load for the F-111 for this mission. Note that the design plan in Fig. 3 indicates an invocation of a subspecialist by the DSPL notation (**DESIGN** *specialist-name*). Once the configuration of the aircraft is known, the single aircraft probability of destruction in the mission context can be computed. Finally, knowing the mission capabilities of each aircraft, the required number of aircraft can be determined in order to achieve the required probability of destruction, and the required aircraft can be reserved from the proper unit. The DSPL code for the *squadron* task and one of the steps in that task is given in Figs 4 & 5, respectively.

The MPA system could be readily extended in several directions. Additional situation knowledge at the OCA level would allow for more robust planning with less backtracking. One example of this is discussed in some detail in the next

```
(TASK assign-squadron
(BODY
  (STEP SetNumberOfAircraft)
  (STEP SelectSquadron)
  (STEP AssignBase)
  (STEP GetRange))
```

FIG. 4. DSPL code for a Task.

```
(STEP SetNumberOfAircraft
(BODY
  (KNOWNS
    configuration (KB-FETCH MISSION CONFIGURATION)
    required-PD (KB-FETCH MISSION REQ-PD)
    targetType (KB-FETCH TARGETTYPE))
  (DECISIONS
    number . . . (compute number of
      aircraft for the
      mission based on
      – configuration
      – required-PD
      – targetType)
  REPLY (KB-STORE MISSION
        AIRCRAFT-NUMBER
        number))))
```

FIG. 5. DSPL code for a Step.

```
(SELECTOR AircraftSelector
(BODY
REPLY (IF (MEMBER A-10 SUITABLE-PLANS)
  THEN A-10
  ELSEIF (MEMBER F-4 SUITABLE-PLANS)
    THEN F-4
    ELSEIF (MEMBER F-111 SUITABLE-PLANS)
      THEN F-111
      ELSE NO-PLANS-APPLICABLE)))
```

FIG. 6. DSPL code for a Design Plan Selector.

section. More complete knowledge of the OCA mission for specifying various aspects of the flight plan, etc., could be added. Also, as previously mentioned, other types of missions could be encoded in hierarchies similar to the OCA hierarchy. The most theoretically interesting addition to the MPA system would be abstractions above the single mission level. Clusters of coordinated missions and even a complete ATO abstraction should be possible within the Routine Design framework. For example, extended range OCA missions requiring coordination with refuelling and escort missions should be able to be planned in a straightforward fashion. The single greatest hindrance to such work is the lack of access to experienced domain experts.

6. Performance of the MPA system

One important advantage of the DSPL architecture is the ability of the planning system to be responsive to the situation at hand *in a computationally efficient manner.* A planning system must be able to adapt its behaviour to the demands of the problem. As an example of this, consider the use of two particular design plans in the OCA specialist, shown in Figs 7 & 8. One plan, the *best-case* plan, prescribes

```
(PLAN best-case
(USED-BY OCA)
(BODY
  (DESIGN aircraft)
  (DESIGN base)
  (REPORT-ON MISSION)))
```

FIG. 7. The *best-case* plan.

that the ordinance and aircraft subproblems of the OCA mission be solved first, followed by the selection of a base and squadron. The second plan, the *limited-fuel-resource* plane, prescribes the reverse order. In this plan, candidate bases are selected first followed by the solution of the ordinance and aircraft subproblems. Let us examine two points: first, the conditions under which each plan would be used and second, the impact each plan has on problem solving.

In mission planning the objective of the planner is to choose an ordinance which is most effective against the target and a delivery vehicle which is most likely to penetrate to the target and return safely. Various combinations of ordinance and

```
(PLAN limited-fuel-resource
(USED-BY OCA)
(BODY
  (DESIGN base)
  (DESIGN aircraft)
  (REPORT-ON MISSION)))
```

FIG. 8. The *limited-fuel-resource* plan.

aircraft type will be appropriate for different situations and targets. Thus the usual strategy of the mission planner in searching for the most effective OCA mission plan is to first select the optimal ordinance and aircraft for the mission and only then turn his attention to the details of choosing a base from which to fly the mission. Under ideal conditions (e.g., there is little contention for the resources necessary for this mission, or negligible enemy interference), the most important job for the mission planner is to match the ordinance/aircraft to the target. The planner first solves the ordinance/aircraft problem, then solves the secondary problem of selecting a base and squadron. Intuitively, to do otherwise may result in considerable backtracking while searching for an acceptable mission plan.

However, the "ideal conditions" assumed for such a strategy may not always be met. Consider the less than ideal situation (others certainly exist) in which the mission planner knows that sufficient refuelling capabilities are unavailable at the time of the mission. Further, suppose that the optimal ordinance/aircraft combination is quite remote from the target and hence the proposed mission requires that the attacking aircraft be refuelled in mid-flight. In this case, the more prudent planning strategy is to first select potential bases from which to fly the mission. This strategy anticipates planning failures due to selecting an aircraft which is only available at a base too distant to fly the mission. If the mission planner constrains the types of ordinance and aircraft considered to those found only at nearby bases, it is more likely that the optimal mission plan will be quickly found. Importantly, needless backtracking is reduced.

It is the order of these two activities that is important in finding the most direct route to the best solution. The character of the search for the final mission plan completely changes by merely changing the order of these two planning activities.

A constraint-based system using a generate and test algorithm can, with backtracking during problem solving, arrive at solutions similar to the ones above. However, as the size of the planning task increases so also does the size of the problem space, and a reliance on backtracking can quickly make such a search computationally infeasible. We are not suggesting that the use of a plan selection and refinement strategy such as found in DSPL eliminates all backtracking. Certainly, any given problem solving strategy will eventually fail and require backtracking, except, perhaps, in trivial domains. However, the plan selection and refinement architecture can reduce or even eliminate the need for backtracking in many cases since the planning system can recognize the need for particularized strategies and apply them to the problem solving context. These strategies, in the form of design plans within a specialist, are thus more likely to find a computationally efficient solution to the problem.

7. Explanation in the MPA system

Explanation of the behaviour of an expert system is typically accomplished by combining a trace of the behavior of the system with the code that produced the trace. In a rule based system, for example, an explanation of why action *B* was chosen is explained by reference to the known fact *A*, and rule *IF A THEN B*. The rule plays the role of the source code, and the program trace produces the actual value of *A* when the rule was fired. More difficult to explain, however, is the

relevance of that particular piece of knowledge to the current problem solving context. This problem is often addressed by the addition of meta-rules to the production system.

In our system, the additional context of the DSPL control structure provides the springboard for a more comprehensive explanation facility. In addition to the necessary ability to examine particular attributes of a mission plan, the control structure provides the ability to examine the problem solving strategies of the planning system. This kind of explanation is not easily extracted from a system which uses weak methods as its primary mechanisms for problem solving, since the strategies are often hidden in code, i.e. implicitly represented. The identification of routine planning as a generic task not only makes the strategies for problem solving explicit, but also provides a framework of organized knowledge for efficient use.

The explanation of constraints in the MPA system illustrates this point. Analysis of the success or failure of constraints, generated from the trace of the problem solver's execution, yields explanation capabilities similar to that found in KNOBS, but with the additional context provided by the rich DSPL control structure. For example, a constraint may be associated with a particular design plan, and used to monitor the progress of the planning process. Constraints may also serve to mediate backtracking in a focused manner.

The implementation of explanation in DSPL is based on the organizing principle that the agent which makes a decision is responsible for justifying it. The DSPL agents which contribute to the final plan are: Specialists, Design Plans, Design Plan Selectors, Design Plan Sponsors, Tasks, Steps, and Constraints. In the present implementation there are some 200 of these agents, though not all of them contribute to any particular plan. All of these agents perform tasks which are epistemically significant so that explanation of any one agent's problem-solving decisions can be given in terms of the goals of the agent which uses it, and the function of the agents it uses.

The final answer produced by the MPA can be viewed as a list of attribute-value pairs as in KNOBS.

That is, a list of the form:

Target = Berlin
Aircraft Type = F-111
Number Aircraft = 6
. . .

We have decided to concentrate on questions of the form, "How was it decided?" which can be asked of the value of any attribute. For example, selecting F-111 in the above list would initiate a dialog on the question of how MPA decided to use F-111 as the value of **Aircraft Type**. More particularly, an explanation window would appear containing the answer to "How was it decided?" produced by the agent which actually set the value of **Aircraft Type** to F-111. In this window certain other things would be selectable. Selecting any of them will produce another window with a similar explanation for the proper agent. In this way the user will be able to pose follow-up questions by steering through the decision dependencies.

To support "How was it decided?" explanations we determined three basic

questions which all agents must be able to answer:

(1) "Give me the bottom line: what did you do?" This question would be answered with a one-sentence summary of the result of the agent's action.
(2) "What is your purpose?" This question would be posed by sub-agents who want to have knowledge of the context they are operating in, and should be answered by a short description.
(3) "How did you do it?" This question would be answered by displaying a window with a complete explanation of the context of the agent's activation followed by a functional description of its action. The agent may have to ask its sub-agents question 1 and its super-agent question 2.

Then, in general, the explanation for "How was it decided?" is the answer to question 3 above. The answer to question 3 is a combination of the answer to question 2 for the calling agent and question 1 for all sub-agents. So an explanation window contains:

In the context of ⟨answer to question 2 for calling agent⟩
we did the following:
⟨answer to question 1 from sub-agent 1⟩
⟨answer to question 1 from sub-agent 2⟩
⟨answer to question 1 from sub-agent 3⟩
...

Our work to this point is about generating explanation fragments and does not address other issues of explanation such as summarization, user modeling, or human factors.

The following is an example of explanation automatically generated from a run of the MPA system. The explanation is generated by combining a trace of the system with the DSPL source code that was executing when the trace was generated. The example is of a plan sponsor, which is attempting to determine the suitability of its associated plan during problem solving. A sponsor matches characteristics of the plan to information about the problem at hand and produces a measure of how useful the plan will be on a scale of: Ruled-Out, Unsuitable, Don't-Know, Suitable, and Perfect. The code for the *A-10* plan sponsor is given in Fig. 2. It first sets some local variables by looking them up in the database (using the *KB-FETCH* built in function which fetches values from the global database). It then uses the *TABLE* construct, which is essentially a group of rules which all depend on predicates of the same values. For example, the table setting the variable *conditions* contains a single rule which depends on the values returned by the functions *night* and *weather*. The rule requires *night* to return *F* and *weather* to return *CLEAR*. If the predicates are true, then *conditions* will be *PERFECT*. The rules of the table construct are executed in order, and the table is finished when a rule matches. If no rule matches, the *OTHERWISE* clause is used. *REPLY* tells DSPL that what follows is the final result of the sponsor's computation.

The explanation for this sponsor is given in Fig. 9. Values for the local variables are given, those fetched from the database are not justified while those determined by tables are given justification. The final *REPLY* is used to determine the actual decision made by the sponsor.

The context of *selecting an aircraft to consider for the mission* determined that:

- target is **BrandenburgSAM**
- timeOverTarget is **1300**
- threat is **UNSUITABLE** because:
  - SAM is TRUE
- conditions are **PERFECT** because:
  - night is FALSE
  - weather is CLEAR

I determined the value of plan A-10 to be **RULE-OUT** because:

- threat is UNSUITABLE.

FIG. 9. Explanation for a Design Plan Sponsor.

8. How a plan works: understanding a mission plan

In expert system problem solving there exists a close relationship between structures of understanding and explanation capabilities. For a system to "understand itself", it should be able to examine its own knowledge structures, its problem solving strategy, and its problem solving behaviour particular to a specific case. The previous sections focused on the representations necessary to achieve these goals for the task of planning. In this section we concentrate on the structure needed to provide a system with the capability to justify its knowledge.

We have shown how, during planning, the knowledge base accesses plan fragments which are instantiated and assembled into longer plans. On occasion, the problem solver may be called upon to explain the rationale behind the plan fragment, i.e. why is this particular fragment appropriate or relevant. Knowledge of this type can be obtained in a number of ways: appeals to authority (books, manuals, etc.), statistical generalizations, or justifying the fragments by showing how they are derived from a deeper understanding of the domain. Here, we will focus on the latter: the use of deep models.

To illustrate the variability of explanation capabilities arising from knowledge structures within a system, consider the task of OCA mission planning. The routine planning task accomplished by the MPA system involves specification for a set of pre-established components. That is, the planner knows the mission needs a certain type of component—its job is to make a concrete commitment as to which specific component of that type would be best. The planner requires only a limited knowledge of these components in order to make such decisions. Its understanding of the resultant mission plan is thus restricted.

More specifically, suppose a user of the mission planner asks the question, "Why was an F-15 used?" Depending on the intentions of the inquirer, the question could be answered in different ways. For a *particular* mission, the question might be addressed directly by the mission planner. Here, the inquiry is interpreted as, "Why

did you use an F-15 instead of any other aircraft for this mission?" Explanation would indicate what makes the F-15 appropriate (speed, weather compatible, etc.). Since this is the specific information the system used in making its decision, the planner should be able to explain it.

In the above interpretation, the question was, "Why choose an F-15?". An alternate interpretation could be, "Why is the F-15 used in the mission plan?". A good response here might be, "The F-15 is an aircraft. Aircraft are used in OCA's because they have the ability to fly and to deliver the ordinance. These functions are used to get to the target location and to destroy the target—the primary goal of an OCA mission." This explanation requires a deeper understanding of the domain than the planner has readily available within its compiled planning knowledge. Here we need a structure to represent distinctly *how the plan works.*

To represent this understanding, we propose use of a knowledge structure based upon the *Functional Representation of Devices* as designed by Sembugamoorthy & Chandrasekaran (1986). A device is any structure (concrete or abstract) which serves a purpose. Thus, a plan can be viewed as an abstract device in that it has components which fit together in such a way to achieve a desired goal. As such, we will show how a plan can be represented functionally and how use of this representation can provide a richer understanding of the planning process.

9. The functional representation: an overview

The first concept is that an agent's understanding of how a device works is organized as a representation that shows how an intended function is accomplished as a series of behavioural states of the device. The device itself is represented in various levels. The topmost level describes the functioning of the device by identifying which components and more detailed behaviours are responsible for bringing about the various state transitions. If a transition is achieved using a function of a component, the next level describes the functioning of this component in terms or the roles of its subcomponents, and so on. Ultimately all the functions of a device can be related to its structure and the functionality of the components within this structure.

At each level of a device's representation there may be five significant aspects of an agent's knowledge of the functioning of the device:

STRUCTURE: specifies the components of a device and the relations between them.

FUNCTION: specifies *what* is the result or goal of an activity of a device or component.

BEHAVIOUR: specifies *how*, given a stimulus, the result is accomplished.

GENERIC KNOWLEDGE: pointers to general knowledge that shows how key states occur.

ASSUMPTIONS: under which a behaviour is accomplished.

The *functional specification* of the "abstract device" OCAMission is illustrated below by describing the main function of an OCA—to destroy a target.

FUNCTION: DestroyTarget
ToMake: (Destroyed Target)
If: (Functional Target)
Trigger: (DesignatedLoadTime OCAMission CTime)
Provided: (Operational Flight)
By: OCAplan

The description indicates that the plan, OCAMission, has a function called DestroyTarget. This function is appropriate **if** a target is operational (functional) and is **triggered** when the designated load time for the specific plane is the current time. When this function is used, the target will be destroyed **by** a behaviour called OCAplan. This behaviour should succeed in accomplishing the goal of target destruction **provided** the flight is operational throughout the behaviour.

The *behavioural specification* of a device describes the manner in which a function is accomplished by using the functions of components, generic knowledge, and sub-behaviours (further details of state transitions). The behaviour for OCAplan of the **DestroyTarget** function is described by a chain of events brought about by the specified actions as seen in Fig. 10.

The structure is meant to represent the sequence (from top to bottom) of states which occur as a result of actions taken. "OCAplan" indicates that the plan begins when a Target is in a Functional state. At the designated load time, the function PrepareFlight of the component AirBase is used to make the Flight Prepared. Upon achieving this state, the plan uses the component Flight since it has the functionality, i.e. OffensiveAir, to Destroy the Target. Finally, the Flight follows the flightplan back to the homebase.

The structure of the OCAMission is defined by its components and relations as shown in Fig. 11.

Links in the chain indicate subcomponents in the sense that the first component uses the next in order to achieve its goals. The OCAMission *uses* the component AirBase (more specifically, AirBases's subcomponent of GroundCrew) to prepare the aircraft and the component Flight to get to the target and destroy it.

One additional specification is needed to fill out the "FUNCTION" construct. This involves primitives used to identify functions as primary or secondary. In the analysis of functions we have observed that some functions have a *secondary* nature. These are functions which are present *in support of* another *main* function. Specification of such functions is necessary to represent understanding of *why* the function is present in the device and for determining equivalency when considering replacement of components. Three types of secondariness have been delineated:

(1) Subfunctions:
—functions a device possesses simply as a means to establish preconditions for a primary function. (e.g. "takeoff" for "fly" of aircraft)
—functions a device possesses to support a provided clause (assumptions) on behaviours for a primary function. (e.g. a car can be used for transportation *provided* the driver can see the road. The functionality of windshield-wipers allows for this behaviour in inclement weather. Similarly Electronic Counter Measures (ECM) are used on OCA missions to *provide* protection so that the target can be reached and destroyed.)

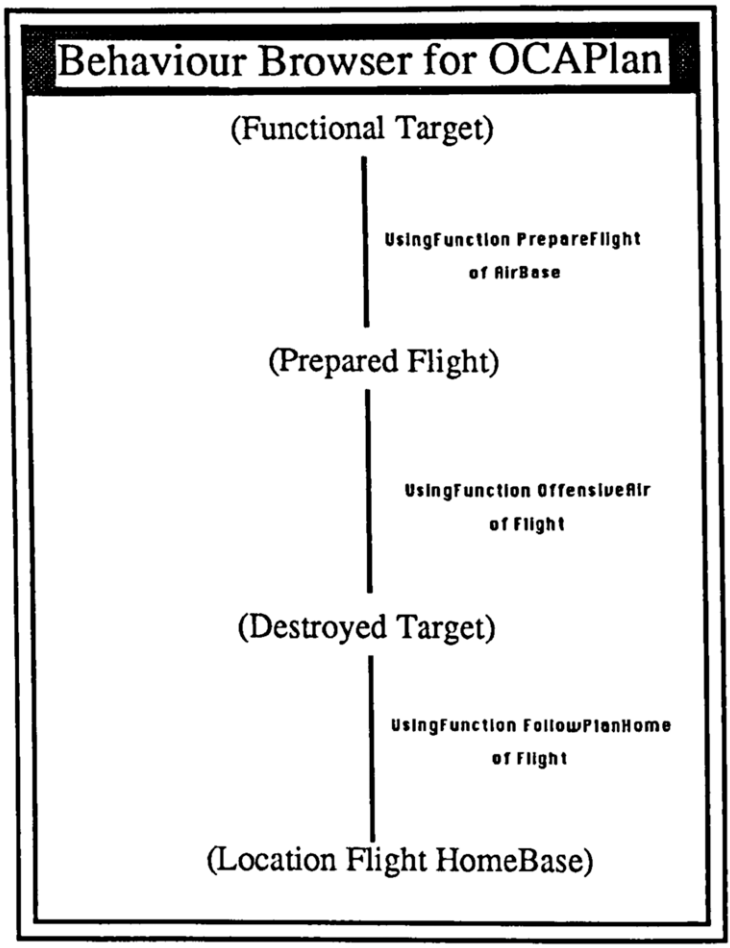

FIG. 10. Behaviour OCAplan for DestroyTarget.

(2) Secondary functions:
With respect to the desired use of the given device, these are extraneous functions. Consider a kerosene lamp. One hundred years ago, its functionality was to give light. Today its use is often purely decorative. When purchased for this purpose, the functionality of producing light is rarely

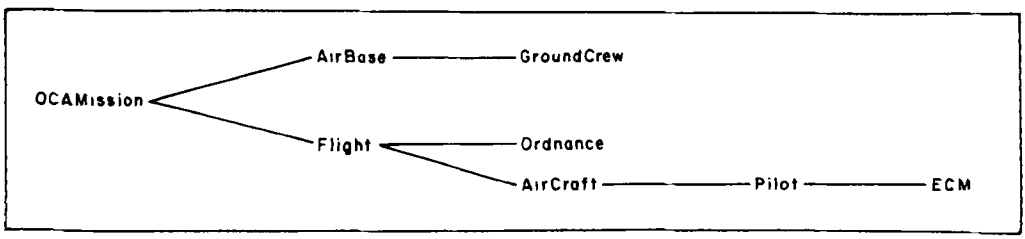

FIG. 11. Components of plan OCAMission.

used. Notice that secondary functions could be primary functions depending on the device designer's and/or the user's purposes.

(3) Other design considerations:

—goals which arise out of the situation or context in which the device is used. (e.g. The component Flight of device OCAMission has the function FollowPlanHome. The main goal of an OCA is to destroy a target. With respect to the device OCAMission, explanation of why FollowPlanHome is needed involves "external" considerations. It is present because in the process of destroying the target we also have the goal of protecting people and conserving resources.)

An example of a function which is secondary in an OCA mission is the Protection function of ECM.

Its functional description is as follows:
FUNCTION: Protection
ToMake: (NOT (Threatened AirCraft))
If: (Threatened AirCraft)
Trigger: (OR (UnderAttack AirCraft)
(ECMSpecified CurrentLeg))
Provided: (Loaded AirCraft ECM)
By: ECMUsage
SubFunctionOf: (DestroyTarget)
ExternalConsideration: YES
(Reason (to protect the aircraft and crew))

9.1 POTENTIAL APPLICABILITY

Understanding of plans (in this case, an OCA mission) can be illustrated through the explanation capabilities inherent in its functional representation. The representation is capable of providing answers about mission devices, functions, and behaviours because of the knowledge encoded using the primitives. The following are some of the questions which can be answered directly from the representation:

Why is this device needed?
What subcomponents does this device require?
What are the secondary functions and their roles?

Why is this function needed?
What does this function accomplish?
How is this function achieved?
Where is this function used?

Consider, for example, an inquiry about the device ECM. If asked why the component is needed, the system responds with, "The device ECM is present because it has the functional capability to protect the aircraft and crew." A query about the need for this function yields the response, "The function Protection is needed for its capabilities to protect the aircraft and crew, to ensure that the aircraft is not threatened, and to support conditions for the function DestroyTarget."† Notice the sources of this information in the function's specification.

† Further inquiry shows that DestroyTarget has a **Provided** clause of the flight being operational.

The capabilities of the functional representation language to express the structure of understanding are not limited to explanations of devices, functions, and behaviours. A diagnostic compiler which takes as input the functional representation of a device, and outputs an expert system for diagnosis of problems of the device has already been implemented in our laboratory at Ohio State. If one views debugging a plan as trouble-shooting of an abstract device, such a diagnostic system is useful for reasoning about why a plan will or will not work.

Similarly, the functional representation should be useful for qualitative simulation of plans. The planner deciding on the use of specific devices may wish to use this potential to check the feasibility of his planning decisions.

Other areas of research concern include the creation of plans and making adaptations to existing plans. Some important characteristics which make this representation useful for the design and repair of devices/plans include:

(1) A component is specified independent of the representation of the device which contains it. More specifically, the specification of a component does not refer to the role of the component in the composite and thus is not limited to use solely within the given device.
(2) Only the names of the functions are carried over to a higher level and not the behaviour specifications of components. This property is important since an agent may desire to replace a malfunctioning component by a functionally equivalent but behaviourally different one. In such situations, *what a* function achieves is more important than *how* it is achieved.

These properties make the task of replacement of components less complex since they allow for the determination of allowable substitutions by simply comparing functional capabilities of current components with alternatives. Since much of planning involves adaptations of already established plans, these traits which allow for such changes in components and behaviours are valuable, i.e. they make it possible for an automated planner to *create* and *modify* plans based upon the goals and the functional specifications of available components.

A "deep" model, such as the one outlined in this paper, is particularly persuasive if it can support more than one type of problem solving activity. We have demonstrated the usefulness of the functional representation in representing plan understanding, and indicated its support of diagnostic reasoning.

10. Concluding Remarks

Research in planning at our laboratory has been based on the following notions:

(1) Design and planning are very similar cognitive activities in the sense that they share types of knowledge and control mechanisms, and both involve construction of a sequence of steps or operations to achieve a set of goals.
(2) Given a problem solving task, the system should be designed such that the problem solving mechanism matches the appropriate task. That is, the structure of the knowledge and control regime for the system should correlate to an understanding of the task involved.
(3) There is a type of problem solving called *routine design* or *routine planning* in

which complete knowledge of the components (or partial plans) and design plans is available prior to the problem solving activity. Problem solving proceeds by using recognition knowledge to select among previously known sequences of design (or planning) actions to fit the current situation.

We have described the use of DSPL as a high-level language for the construction of a routine planning system. The MPA system demonstrates that the plan selection and refinement strategy used by DSPL is a useful technique for building routine planning systems; the problem solving strategy focusing on the planning process much more sharply than more general problem solving methods. The MPA system also shows how the structure enforced through DSPL clearly and concisely represents planning knowledge.

In general, information processing systems which are analysed and implemented using a framework of generic tasks with associated types of knowledge and a family of control regimes benefit through the following advantages: (i) Since typically the generic tasks are at a much higher level of abstraction than those associated with first generation expert system languages, knowledge can be represented directly at the level appropriate to the information processing task. (ii) Since each of the generic tasks has an appropriate control regime, problem solving behaviour may be more perspicuously encoded. (iii) Because of a richer generic vocabulary in terms of which knowledge and control are represented, explanation of problem solving is also more perspicuous (Chandrasekaran, 1985).

In broader terms, this research is a part of our on-going effort to uncover the multitude of generic structures and processes involved in knowledge-based problem solving. Our continued work includes further identification of problem solving tasks to enrich our collection of generic knowledge structures and mechanisms, integration of these tasks for more complex systems and integration of explanation types within systems to provide for richer system understanding.

References

BROWN, D. C. (1984). *Expert systems for design problem-solving using design renement with plan selection and redesign.* Unpublished PhD dissertation. Ohio State University, Columbus, OH.

BROWN, D. C. & Chandrasekaran, B. (1985*a*). Plan selection in design problem-solving. *Proceedings of The AISB 85 Conference.* Warwick: The Society for AI and the Simulation of Behavior.

BROWN, D. C. & CHANDRASEKARAN, B. (1985*b*). Expert systems for a class of mechanical design activity. In J. Gero, (Ed.) *Knowledge Engineering in Computer-Aided Design.* Amsterdam: Elsevier.

CHANDRASEKARAN, B. (1984). Expert systems: Matching techniques to tasks. In W. REITMAN, Ed. *Artificial Intelligence Applications for Business.* Ablex Corp. Paper presented at NYU symposium on Applications of AI in Business.

CHANDRASEKARAN, B. (1985). Generic tasks in knowledge-based reasoning: characterizing and design expert systems at the right level of abstraction. *Proceedings of The IEEE Second International Conference on Artificial Intelligence Applications.* Miami, Florida: The IEEE Computer Society.

CHANDRASEKARAN, B. (1986). Generic tasks in knowledge-based reasoning: high level building blocks for expert system design. *IEEE Expert,* 23–30.

CHANDRASEKARAN, B., TANNER, M. & JOSEPHSON, J. (1987). Explanation: The role of

control strategies and deep models. In J. Hendler, Ed. *Expert Systems: The User Interface,* Norwood, NJ: Ablex Publishing.

CHANDRASEKARAN, B., JOSEPHSON, J. & KEUNEKE, A. (1986). Functional representations as a basis for generating explanations. *Proceedings of IEEE International Conference on Systems, Man and Cybernetics.* Atlanta, Georgia: IEEE.

ENGELMAN, C., MILLEN J. K., SCARL, E. A. (1983). *KNOBS: An Integrated AI Interactive Planning Architecture DSR* 83–162. Bedford, MA: The MITRE Corporation.

FOX, M. S., ALLEN, B. P., SMITH, S. F. & STROHM, G. A. (1983). ISIS: A constraint-directed reasoning approach to job shop scheduling. *Proceedings of IEEE Conference on Trends and Applications* 83. Gaithersburg, Maryland: IEEE.

HAYES-ROTH, B. (1980). *Human Planning Processes* (Technical Report). Santa Monica, California: The Rand Corporation,

HAYES-ROTH, B., HAYES-ROTH, F., ROSENCHEIN, S. J. & CAMMARATA, S. (1979). *Modeling Planning as an Incremental Opportunistic Process* (Technical Report). Santa Monica, CA: The Rand Corporation,

MILLER, G. A., GALANTER, E. and PRIBRAM, K. H. (1960). *Plans and the Structure of Behavior.* New York: Holt.

SCARL, E. A., ENGELMAN, C., PAZZANI, M. J., MILLEN, J. K. (1984). *The KNOBS System* (Technical Report). Bedford, MA: The MITRE Corporation,

SCHANK, R. C. & ABELSON, R. P. (1977). *Scripts, Plans, Goals, and Understanding.* Hillsdale, NJ: Erlbaum.

SEMBUGAMOORTHY, V. & CHANDRASEKARAN, B. (1986). Functional representation of devices and compilation of diagnostic problem solving systems. In J. L. KOLODNER & C. K. RIESBECK, Ed. *Experience, Memory, and Reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

SHORTLIFFE, E. H., SCOTT, A. C., BISCHOFF, M., CAMPBELL, A. B., VANMELLE, W., & JACOBS, C. (1981). ONCOCIN: An expert system for oncology protocal management. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* Vancouver, British Columbia: IJCAI.

WILENSKY, R. (1983). *Planning and Understanding.* Reading, MA: Addison-Wesley.