The Generic Task Toolset: High Level Languages for the Construction of Planning and Problem Solving Systems

B. Chandrasekaran, J. Josephson, and D. Herman
Ohio State University
Columbus, OH 43210

Abstract

We begin by criticising the current generation of languages for the construction of knowledge-based systems as being at too low a level of abstraction, and argue for the need for higher level languages for building problem solving systems. We introduce our notion of generic information processing tasks in knowledge-based problem solving, and describe a toolset which can be used to build expert systems in a way that enhances intelligibility and productivity in knowledge acquisition and system construction. We illustrate the power of these ideas by paying special attention to a high level language called DSPL, and describe how it was used in the construction of a system called MPA, which assists with planning in the domain of offensive counter air missions.

Scope and Intent

This paper is intended to be an introduction to the generic tasks approach to analyzing knowledge systems, descriptions of some of the particular generic tasks that have been identified, and a description of the software tools that have been build as a result. The approach is illustrated by discussing a particular mission-planning system, MPA, which was built using the DSPL language (or tool). The intent of the paper is primarily tutorial, much of the material is summary or repetition of material already presented elsewhere.[1, 2]

Level of Abstraction Problem in Characterizing Knowledge-Based Systems

Much of AI can be said to be a search for the "Holy Grail" of a level of abstraction at which intelligence behavior qua intelligent behavior emerges. Above this level, presumably, are particular pieces of knowledge of the task domain, and below this level are specific details of how the intelligent level is implemented. For example, in a rule-based system such as Mycin, everyone would agree that rules of medicine are not per se matters of AI research. That is, the content of the knowledge base itself is made up of domain particulars. At the same time the language in which the rule system is written, e.g., Lisp, is typically thought of as an implementation-level detail, of no particular interest as AI.

The AI interest of a given expert system is often a matter of the level at which it is viewed. Clearly, taking Mycin as an example, all the following points of view are correct: (i) it suggests therapy for certain kinds of infectious diseases, (ii) it is an embodiment of a diagnostic and therapeutic strategy, and (iii) it is a decision-maker which uses backward chaining to navigate a knowledge base of rules, connecting pieces of evidence to conclusions, in order to arrive at a reliable conclusion from a given set of data. Point of view (i) is of limited interest to AI, and point of view (iii) has been the level at which the system as an AI system has been generally presented; this is the level at which the claim for generality of the underlying approach is made. All the excitement surrounding the "knowledge base/inference engine" paradigm, and the idea that knowledge acquisition and explanation can be keyed to phenomena at the rule level, emphasizes that the level corresponding (iii) has been the level of abstraction at which AI interest has been expressed, and claims of progress have been made.

What of the level corresponding to (ii)? We have for several years at Ohio State worked on the hypothesis that this is indeed an important level of abstraction for AI; and that knowledge representation, control regimes for problems solving, knowledge acquisition, and explanation all can be significantly advanced by looking at phenomena at this level. B. Chandrasekaran has put forth this view in[1, 3, 4]. In this paper we give a critical analysis of an important part of AI, viz., knowledge-based reasoning, and propose that a point of view based on a particular level of abstraction corresponding to generic information processing tasks has a number of advantages both for clarity of analysis and for system design. This is not pure theoretical speculation; researchers in our laboratory have built many systems and tools based on this framework. We wish to point to this level of abstraction as a productive level for concentration, and to indicate the conceptual advantages of it. Knowledge acquisition, system design, control of problem solving and generation of explanation all are facilitated *at the same time*, indicating that there is a naturalness to looking at phenomena at this level.

[1]Research supported by Air Force Office of Scientific Research grant 82 0255, National Science Foundation grant MCS-8305032, and Defense Advanced Research Projects Agency, RADC Contract F30602 85 C 0010. Toolset development is also supported by grants of equipment from Xerox Corporation, IBM, and Texas Instruments.

Intuitively one thinks that there must be some commonality of reasoning processes that characterize "diagnosis" as a generic activity, even across domains as different as medicine and mechanical systems. There *should* be *control strategies* and ways of using knowledge that are common to diagnostic reasoning as such, or at least typical of diagnostic reasoning. Similarly there should be common types of knowledge structures and control strategies for, say, design as a kind of reasoning activity. Further, we expect that the structures and control regimes for diagnostic reasoning will be generally different from those for design reasoning. However, when one looks at the formalisms (or equivalently the languages or shells), that are commonly used in expert system design, the knowledge representation and control regimes do not typically capture these distinctions. For example, in diagnostic reasoning, one might generically wish to speak in terms of malfunction hierarchies, rule-out strategies, setting up a differential, etc., while for design, the generic terms might be device/component hierarchies, design plans, ordering of design subtasks, etc. Ideally one would like to represent diagnostic knowledge in a domain by using the vocabulary that is *appropriate for* the task. But typically the languages in which the expert systems have been implemented have sought uniformity across tasks, and thus have lost clarity of representation at the task level. The computational universality of representation languages such as Emycin or OPS5 -- i.e., the fact that any computer program can in principle be written in these languages -- often confuses the issue, since after the system is finally built it is often unclear which portions of the system represent domain expertise, and which are programming devices. In addition, the control regimes that these languages come with (such as forward or backward chaining) do not explicitly indicate the real control structure of the system at the task level. For example, the fact that R1 [5] performs a linear sequence of subtasks -- an atypically simple strategy for design problem solving -- is not explicitly encoded; the system designer so to speak "encrypted" this control in the pattern-matching control of OPS5.

These comments need not be restricted to the rule-based framework. One could represent knowledge as sentences in a logical calculus and use logical inference mechanisms to solve problems; or one could represent it in a frame hierarchy with procedural attachments in the slots. (It is a relatively straightforward thing, e.g. to rewrite MYCIN [6] in this manner, see[7].) In the former, the control issues would deal with choice of predicates and clauses, and in the latter, they will be at the level of which links to pursue for inheritance, etc. None of these have any natural connection with the control issues specific to the task.

The other side of the coin, so to speak, regarding control is the following: because of the relatively low level of abstraction relative to the information processing task, there are control issues that are artifacts of the representation, but often in our opinion misinterpreted as issues at the "knowledge-level." E.g., rule-based approaches often concern themselves with conflict resolution strategies. If the knowledge were viewed at the level of abstraction appropriate to the task, often there will be organizational elements which would result in only a small set of highly relevant pieces of knowledge or rules to being brought up for consideration, without any conflict resolution strategies being needed. Of course, these organizational constructs could be "programmed" in the rule language, but because of the status assigned to the rules and their control as knowledge-level phenomena (as opposed to the implementation level phenomena, which they often are), knowledge acquisition is often directed towards (typically syntactic) strategies for conflict resolution, whereas the really operational expert knowledge is at the organizational level.

This level problem with control structures is mirrored in the relative poverty of knowledge-level primitives for representation. For example the epistemology of rule systems is exhausted by data patterns (antecedents or subgoals) and partial decisions (consequents or goals), that of logic is similarly exhausted by predicates, functions, inference rules, and related primitives. If one wishes to talk about types of goals or predicates in such a way that control behavior can be indexed over this topology, such a behavior can often be programmed into these systems, but no explicit rendering of them is possible. E.g., Clancey [8] found in his work using Mycin to teach students that for explanation he needed to attach to each rule in the Mycin knowledge base encodings of types of goals so that explanation of its behavior can be couched in terms of this encoding, rather than only in terms of "Because ... was a subgoal of ...." This is not to argue that rule representations and backward or forward chaining controls are not "natural" for some situations. If all that a problem solver has for knowledge in a domain is in the form of a large collection of unorganized associative patterns, then data-directed or goal-directed associations may be the best that the agent can do. But that is precisely the occasion for weak methods such as hypothesize and match (of which the above associations are variants), and, typically, successful solutions cannot be expected in complex problems without combinatorial searches. Generally, however, expertise can be expected to consists of much more organized collections of knowledge, with control behavior indexed by the kinds of organizations and forms of knowledge in them.

Thus, there is a need for understanding the generic information processing tasks that underlie knowledge-based reasoning. Knowledge ought to be directly encoded at the appropriate level by using primitives that naturally describe the domain knowledge for a given generic task. Problem solving behavior for the task ought to be controlled by regimes that are appropriate for the task. If done correctly, this would simultaneously facilitate knowledge representation, problem solving, and explanation.

At this point it will be useful to make further distinctions. Typically many tasks that we intuitively think of as generic tasks are really **complex generic tasks**. I. e., they are further decomposable into components which are more elementary in the sense that each of them has a more homogeneous control regime and knowledge structures. For example, what one thinks of as the diagnostic task, while it may be generic in the sense that the task may be quite similar across domains, it is not a unitary task structure. Diagnosis may involve classificatory reasoning at a certain point, reasoning from one datum to another datum at another point, and abductive assembly of multiple diagnostic hypotheses at another point. Hierarchical classification has a different form of knowledge and control behavior from those for data-to-data reasoning, which in turn is dissimilar in these dimensions from assembling hypotheses. Our research focuses on tasks at

both these levels, but the latter are viewed as somewhat "atomic" tasks into which more complex, but still generic, tasks such as diagnosis and design can often be decomposed.

To summarize the view presented so far: There is a need for understanding the generic information processing tasks that underlie knowledge-based reasoning. Knowledge ought to be directly encoded at the appropriate level by using primitives that naturally describe the domain knowledge for a given generic task. Problem solving behavior for the task ought to be controlled by regimes that are appropriate for the task. If done correctly, this would simultaneously facilitate knowledge representation, problem solving, and explanation.

Over the years, we have identified, and built systems using, six such generic tasks. Our work on MDX[9, 10], e.g., identified **hierarchical classification**, **knowledge-directed information passing**, and **hypothesis matching** as three generic tasks, and showed how certain classes of diagnostic problems can be implemented as an integration of these generic tasks. Since then we have identified several others: **object synthesis by plan selection and refinement**[11], **state abstraction**[4], and **abductive assembly of hypotheses**[12]. There is no claim that these six are exhaustive; in fact, our ongoing research objective is to identify other useful generic tasks and understand their knowledge representations and strategies for control of problem solving.

Some Generic Tasks

Characterization of Generic Tasks

Each generic task is characterized by: a **task specification** in the form of generic types of **input** and **output** information; specific **forms of knowledge** needed for the task, and specific **organization of knowledge** particular to the task; a family of **control regimes** that are appropriate for the task.

A task-specific control regime comes with certain characteristic types of **strategic goals**. These goal types will play a role in providing explanations of its problem solving behavior.

When a complex task is decomposed into a set of generic tasks, it will in general be necessary to provide for communication between the different structures specializing in these different types of problem solving. Also there is not necessarily a unique decomposition. Depending upon the availability of particular pieces of knowledge, different architectures of generic tasks will typically be possible for a given complex task.

We will now give brief characterizations of the generic tasks that we have identified.

Taxonomic Classification

Task specification: Classify a (possibly complex) description of a situation as an element, as specific as possible, in a *classification hierarchy*. E.g. classify a medical case description as an element of a disease hierarchy.

Forms of knowledge: one main form is <partial situation description> ---> evidence/belief about confirmation or disconfirmation of classificatory hypotheses. E.g., in medicine, a piece of classificatory knowledge may be: certain pattern in X-ray & bilirubin in blood —> high evidence for cholestasis.

Organization of knowledge: knowledge of the form above *distributed* among concepts in a classificatory concept hierarchy. Each conceptual "specialist" ideally contains knowledge that helps it determine whether it (the concept it stands for) can be *established* or *rejected*.

Control Regime: Problem solving is top down, each concept when called upon tries to establish itself. If it succeeds, it lists the reasons for its success, and calls its successors, which repeat the process. If a specialist fails in its attempt to establish itself, it rejects itself, and all its successors are also automatically rejected. This control strategy can be called *Establish-Refine*, and results in a specific classification of the case. (The account is a simplified one. The reader is referred to[10] for details and elaborations.)

Goal types: E.g., Establish <concept>, Refine (subclassify) <concept>

Example Use: Medical diagnosis can often be viewed as a classification problem. In planning, it is often useful to classify a situation as of a certain type, which then might suggest an appropriate plan.

Object Synthesis by Plan Selection and Refinement

Task Specification: Design an object satisfying specifications (object in an abstract sense: they can be plans, programs, etc.).

Forms of knowledge: Object structure is known at some level of abstraction, and pre-compiled plans are available which can make choices of components, and have lists of concepts to call upon for *refining* the design at that level of abstraction.

Organization of Knowledge: Concepts corresponding to components organized in a hierarchy mirroring the object structure. Each concept has plans which can be used to make commitments for various "dimensions" of the component.

Control Regime: Top down in general. The following is done recursively until a complete design is worked out: A specialist corresponding to a component of the object is called, the specialist chooses a plan based on the specifications and problem state, instantiates and executes the plan which suggests further specialists to call to set details of the subcomponents. Plan failures are passed up until appropriate changes are made by higher level specialists.

Goal Types: E.g., Choose plan, execute plan element <>, refine <plan>, redesign (modify) <partial design> to respond to failure of <subplan>, select alternative plan, etc.

Example: Expert design tasks, routine synthesis of plans of action.

We will characterize the other generic tasks more succinctly. The reader is referred to[1] for more details.

Knowledge-Directed Information Passing

Task: It is desired to obtain attributes of some datum. by deriving from some conceptually related datum. Some forms of knowledge are: <attribute> of <datum> is inherited from <attribute>ψof parent of <datum>, <attribute> of <datum> is related as <relation> to <attribute> of <concept>. Organization: concepts are organized as a frame hierarchy, with IS-A and PART-OF links. Each frame is a specialist in knowledge-directed data inference for the concept. This is basically a hierarchical information-passing control regime.

Example uses: knowledge-based data retrieval tasks in wide variety of situations, as an intelligent data base in support of problem solvers of other types.

Abductive Assembly of Explanatory Hypotheses

Task Specification: Given a situation (described by a set of data items) to be explained by the best explanatory account, and given a number of hypotheses, each associated with a degree of belief, and each of which offers to explain a portion of the data (possibly overlapping with data to be accounted for by other hypotheses), construct the best composite explanatory hypothesis. Some forms of knowledge are: causal or other relations (e.g. special case of, incompatibility, suggestiveness) between the hypotheses, relative significance of data items, and ways to group data items to be explained. Organization: one main, or a hierarchical community of active abducers, each specializing in explaining a certain portion of the data by composing and criticizing hypotheses. *Control Regime:* (See [13] for a fuller discussion.) A specialized means-ends regime is in control, driven by the goals of explaining all the significant findings, with an economical hypothesis, which is consistent, and has been criticized for certain strengths and weaknesses. Some goal types are: account-for <datum>; check-superfluousness-of <hypothesis>, choose the most significant unexplained finding. The Internist system[14] and the Dendral system[15] perform abductive assembly as part of their problem solving.

State Abstraction

Task Specification: Given a change in some state of a system, provide an account of the changes that can be expected in the functioning of the system. (Useful for reasoning about consequences of changes on complex systems.) One knowledge form is: change in state of subsystem> ---> change in functionality of subsystem = change in state of the immediately larger system. The knowledge is organized into conceptual specialists corresponding to systems and subsystems connected in a way mirroring the way the systems and subsystems are put together. The control is basically bottom up, following the architecture of the system/subsystem relationship. The changes in states are followed through, interpreted as changes in functionalities of subsystems, until the changes in the functionalities at the level of abstraction desired are obtained. This form of reasoning is useful for answering questions like: "What system functionalities will be compromised if this valve fails closed?".

Hypothesis Matching

Given a concept and a set of data that describe the problem state, decide if the concept matches the situation. The idea here is to encode the routine knowledge for verification and refutation that the concept applies to the situation. One way this can be done is by using a hierarchical representation of evidence abstractions, where the top node determines the overall degree of matching of the hypothesis to the data, and lower-level nodes represent components or features of evidence for the evidence abstraction at higher levels. Form of knowledge are such as to enable evaluation of strength for each evidence abstraction, and to support mapping degrees of belief in each of these evidence abstractions, to degree of belief in the higher abstractions. Strength for an evidence abstraction can be determined Each evidence abstraction can be determined by matching against prototypical patterns which have evidential significance. Samuel's *signature tables* can be thought of as performing this task.

How Existing Expert Systems can be Analyzed in This Framework

Separating the implementation language and the intrinsic nature of the tasks has been argued as being salutary for a number of reasons. Let us look at some of the better known expert systems from the perspective of the framework developed so far in this paper.

MYCIN's task is to (i) *classify* a number of observations describing a patient's infection as due to one or another organism, and (ii) once that is done, to instantiate a plan with parameters appropriate for the particular patient situation. We have shown in[16] how the diagnostic portion of MYCIN can be recast as a classification problem solver, with a more direct encoding of domain knowledge and a control structure that is directly appropriate to this form of problem solving.

Prospector[17] *classifies* a geological description as one of a pre-enumerated set of formations. Internist[14] generates candidate hypotheses by a form of enumeration (plausibility scoring and keeping only the top few) and uses a form of *abductive assembly* to build a composite hypothesis that accounts for all of the data. Assembly and hypothesis enumeration alternate. Dendral[15] generates candidate hypotheses by a form of *hypothesis matching* and uses a form of *abductive assembly* which puts together the best molecular hypothesis from the fragments produced by the matching process.

Note that in these analyses we have not mentioned rules (Mycin), networks (Prospector), graphs (Dendral), etc., which are the *means* of encoding and carrying out the tasks. This separation is an essential aspect of what we mean by the "right" level of abstraction in analysis.

Encoding Knowledge at the Level of the Task: The Generic Task Toolset

For each generic task, the form and organization of the knowledge directly suggests the appropriate representation in terms of which domain knowledge for that task can be encoded. Since there is a control regime associated with each task, the problem solver can be implicit in the representation language. That is, a shell for each generic task can be constructed such that, as soon as knowledge is represented in the shell, a problem solver which uses the control regime on the knowledge can be created by the interpreter. This is similar to what representation systems such as EMYCIN do, but note that we are

deliberately trading generality at a lower level to gain specificity, clarity, and richness of ontology and control at a higher level.

We have designed and implemented representation languages for versions of each of the six generic tasks we described. Here is a list of the generic task tools, each with a brief description of the task for which it is designed.

- **HYPER** for matching concept to situation to determine confidence or appropriateness.
- **CSRL** for taxonomic classification, typically a major component of diagnostic reasoning.[18]
- **DSPL** for object synthesis by plan selection and refinement, captures knowledge for certain routine design and planning tasks.[19]
- **IDABLE** for knowledge-directed information passing for intelligent data retrieval.
- **PEIRCE** for assembly and criticism of composite explanatory hypotheses, a form of abduction or best-explanation reasoning.[20]
- **WWHI** (What Would Happen If) for prediction by abstracting state changes.

We have described how this approach directly helps in providing intelligible explanations of problem solving in expert systems.[21] The approach has a number of other implications. E.g., *uncertainty handling* in problem solving is usefully viewed as consisting of different types for each kind of problem solving, rather than as a uniform general method.[22]

In principle the tools can be used together to build composite problem solvers that integrate the different types of reasoning associated with the generic tasks. Systems have been built integrating more than one type of reasoning (the Red[23] system for example relies on four of the types) but these systems predate the availability of the tools. At present the actual toolset consists of separately implemented tools in a variety of languages; each tool having an incarnation in Interlisp. LAIR has under development an integrated version of the toolset in Common Lisp.

The computational architecture of a problem-solving system (or system component) built with any of the tools is based on functionally distinct, highly modular elements, tightly organized. A generic task problem solver is a community of agents, where each agent is of a specific type, each has its own embedded knowledge. The agents are organized so that they have specific lines of communication with each other; and, depending on the generic task, they pass control around in a well-defined way in order to cooperate and solve the problem. A HYPER-built system is made up of knowledge groups, hierarchically organized; a CSRL-built system consists of a hierarchical community of classification specialist, each specializing in a single classificatory concept. This sort of system architecture, besides making implementation in an object-oriented programming framework fairly easy, makes for systems that are distributable and have predictable concurrencies. The high degree of modularity--modules having clear functions, and clear interactions with other modules--makes for good software engineering at the knowledge level. "structured programming of knowledge bases" if you will.

DSPL for building a Mission Planning Assistant

We will describe the use of DSPL (Design Specialists and Plans Language) for the design and implementation of an expert system in the domain of tactical air mission planning. After investigating KNOBS system from MITRE Corporation, an existing mission planning system, we developed the Mission Planning Assistant (MPA) using DSPL and our generic task approach to building expert systems. KNOBS was the primary source of domain knowledge for the MPA system. Our project had two main goals. First, we wanted to explore the use of DSPL for routine planning tasks. Initially, DSPL was developed as a result of studying a routine mechanical design task.[11] It seemed to us, however, that routine planning shares many of the characteristics of routine design, suggesting that they might require some of the same kinds of problem-solving activities. Secondly, we wanted to investigate the explanation facilities that are necessary in planning systems. We wanted to demonstrate that our generic task architectures provide very natural and comprehensive frameworks for explanation.

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 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's airbase. Our selection of the OCA mission arose primarily because of the availability of the KNOBS system and its knowledge base of relevant domain facts. 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 mechanical device to design. A view of design problem solving should illuminate this idea. For a more comprehensive description of design see[11].

Routine Design and DSPL

The general domain of design is vast; it involves creativity, many different problem-solving techniques, and many kinds of knowledge. Goals are often poorly specified, and may change during the course of problem solving. A spectrum of classes of design problems can be discerned, varying in complexity from those problems requiring significant amounts of "creativity", to the most routine design problems requiring no creativity at all. The complexity of a design problem will depend on what

pieces of knowledge are available to the problem solver prior to the start of design, that is, the right pieces of knowledge can remove the need for creativity and turn a complex design task into a routine one.

What we have called "Class 3 Design" characterizes a form of routine design activity which postulates that several distinct types of knowledge are available prior to problem solving. First we assume that complete knowledge of the component breakdown of the to-be-designed device is available to the problem solver, including knowledge of what component attributes need to be specified in order to specify a design. The final design will consist only of components known in advance, and no novel components need to be synthesized. Secondly, we assume that knowledge is available in the form of plan fragments describing the actions required to design each component. A plan for designing a component will typically include the designing of subcomponents as steps in the plan. Thirdly, we assume that recognition knowledge is available that will allow the problem solver to select between the alternative plans for designing a component, depending on the design requirements and the state of the problem solving. The problem solving proceeds by following a top-down process of plan selection and refinement, with localized back up and selection of alternative plans upon failure of a design plan at any level. While the choices at each point may be simple, the design process overall may be quite complex, and objects of significant complexity can be designed. It appears that a significant portion of the everyday activity of practicing designers can be analyzed as class 3 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 specific subportions of the device, or address other design subtasks. Any specialist may access a design data-base (mediated by an intelligent data-base assistant). The organization of the specialists and the specific content of each one is intended to precisely capture the human designer's expertise in the problem domain. Each specialist in the design hierarchy contains locally the design knowledge necessary to accomplish that portion of the design for which it is responsible. There are several types of knowledge represented in each specialist, three of which are described here. 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 all of the 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 judgments of the design plan sponsors and determine which of the design plans within the specialist is most appropriate to the current problem context.

Control in a DSPL system proceeds downwards from the top-most specialist in the design hierarchy. 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, checking constraints to test the progress of the design, or invoking sub-specialists to accomplish sub-portions of the design. Thus a design plan which refers to a sub-specialist is refined by passing *control* to that sub-specialist in its turn. DSPL also includes facilities for the handling of various types of plan failures, and for controlling redesign suggested by such failures.[24]

Mission Planning as Routine Design

We view tactical mission planning as predominantly a routine design task. The problem can be decomposed into the design of subcomponents of the mission plan, where each component can be designed in a fairly independent fashion. The Air Tasking Order is decomposed into various missions or groups of missions of known types. Each mission or group of missions can be planned relatively independently of the others, modulo resource contention considerations. In both the mission planning and the mechanical design domains, 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 she can decompose the problem into a manageable number of somewhat independent sub-problems, which can be solved separately and combined into a final solution. The MPA system uses DSPL as a natural mechanism for representing the necessary knowledge.

The MPA System

The MPA system contains six specialists. 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 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 type and number for the mission. The aircraft specialist and its three configuration subspecialists represent the most elaborated domain knowledge in the MPA system.

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. 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 necessary number of aircraft can be reserved from the proper unit.

Summary

We have argued the need for high-level task-specific tools for constructing knowledge-based problem-solving systems. We described our approach, based on the notion of generic information processing tasks, and described a toolset which can be used to efficiently build expert systems. Expert systems built according to this methodology have all of the advantages of control strategies and knowledge representations that are especially suited to their information processing tasks. Advantages include knowing what kinds of knowledge to collect, and where to put it away for use in the problem solver, efficient processing at run time, and explanations of system behavior in terms of strategies and problem solving goals keyed to the type of reasoning. We have illustrated the power of these ideas by paying special attention to a high level language called DSPL, and have shown how it was used in the construction of a system called MPA, for assisting with mission planning in the domain of offensive counter air missions.

References

1. B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design". *Invited paper appearing in IEEE Expert*

2. David Herman, John Josephson and Ron Hartung. "Use of DSPL for the Design of a Mission Planning Assistant", *The Proceedings of the IEEE Expert Systems in Government Symposium*, IEEE Computer Society, October 1986, pp. (??).

3. Chandrasekaran, B., "Expert Systems: Matching Techniques to Tasks". Paper presented at NYU symposium on Applications of AI in Business. Appears in *Artificial Intelligence Applications for Business*, edited by W. Reitman, Ablex Corp., publishers, 1984.

4. Chandrasekaran, B., "Towards a Taxonomy of Problem-Solving Types", *AI Magazine*,Vol. 4, No. 1. Winter/Spring 1983, pp. 9-17.

5. McDermott, J., "R1: A Rule-Based Configurer of Computer Systems", *Artificial Intelligence*,Vol. 19. Issue 1,, 1982, pp. 39-88.

6. Shortliffe, E.H., *Computer-based Medical Consultations: MYCIN*, Elsevier/North-Holland Inc., 1976.

7. Szolovits, P./ Pauker, S. G., "Categorical and Probabilistic Reasoning in Medical Diagnosis", *Artificial Intelligence*, , 1978, pp. 115-144.

8. Clancey, William J., "The Epistemology of a Rule-Based Expert System--a Framework for Explanation", *Artificial Intelligence*,Vol. 20, No. 3. May 1983, pp. 215-251.

9. Chandrasekaran, B. / Mittal, S. / Gomez, F. / Smith M.D., J., "An Approach to Medical Diagnosis Based on Conceptual Structures", *Proceedings of the 6th International Joint Conference on Artificial Intelligence*,August 1979, pp. 134-142, IJCAI 1979

10. Chandrasekaran, B. / Mittal, S., "Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems", in *Advances in Computers*, M. Yovits, ed., Academic Press, Vol. 22, 1983, pp. 217-293.

11. David C. Brown and B. Chandrasekaran. "Knowledge and Control for A Mechanical Design Expert System", *Computer*,Vol. 19, No. 7. July, 1986, pp. 92-100.

12. Josephson, J. R.; Chandrasekaran, B.; Smith, J.W., "Assembling the Best Explanation", *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, IEEE Computer Society, Denver, Colorado, December 3-4 1984, pp. 185-190, A revised version by the same title is now available as a technical report.

13. Josephson J. R., Chandrasekaran B., Smith J., Tanner M., *Abduction by Classification and Assembly*, Philosophy of Science Association, East Lansing, Michigan, 1986, pp. 458-470, ch. VII.

14. Pople, H. W., "Heuristic Methods for Imposing Structure on Ill-Structured Problems", in *Artificial Intelligence in Medicine*, P. Szolovits, ed., Westview Press, 1982 , pp. 119-190.

15. Buchanan, B. Sutherland, G. Feigenbaum, E.A., "Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry", in *Machine Intelligence 4*, American Elsevier, New York, 1969.

16. Sticklen, Jon / Chandrasekaran, B. Smith, J.W. Svirbely, John, "MDX-MYCIN: The MDX Paradigm Applied To The MYCIN Domain", *International Jour. Computers And Mathematics with Applications*,Vol. 11, No. 5, 1985, pp. 527-539.

17. Duda, Richard O./ Gaschnig, John G./ Hart, Peter E., "Model Design in the Prospector Concultant System for Mineral Exploration", in *Expert Systems in the Microelectronic Age*, Michie, D., ed., Edinburgh University Press, 1980, pp. 153-167.

18. T. Bylander and S. Mittal, "CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling", *AI Magazine*,Vol. 7, No. 3, August 1986, pp. 66-77.

19. D. C. Brown, "Expert Systems for Design Problem-Solving Using Design Refinement with Plan Selection and Redesign". PhD dissertation

20. W. F. Punch III, M. C. Tanner and J. R. Josephson, "Design Considerations for PIERCE, a High-Level Language for Hypothesis Assembly", *Expert Systems in Government Symposium*,October 1986, pp. 279-281.

21. Chandrasekaran, B., "Generic Tasks in Expert System Design and Their Role in Explanation of Problem Solving". Invited paper presented at the National Academy of Sciences/ Office of Naval Research Workshop on Distributed Problem Solving, May 16-17, 1985, Washington. D.C., appears in the *Proc. of the Workshop* to be published by the National Academy of Sciences.

22. Chandrasekaran, B. / Mittal, S. / Smith, J. W., "Reasoning with Uncertain Knowledge: The MDX Approach", *Proc. 1st Ann. Joint Conf. of the American Medical Informatics Association*,May 1982.

23. Smith, Jack W.; Svirbely, John R.; Evans, Charles A.; Strohm, Pat; Josephson, John R.; Tanner, Mike, "RED: A Red-Cell Antibody Identification Expert Module", *Journal of Medical Systems*,Vol. 9, No. 3, 1985, pp. 121-138.

24. D. C. Brown, *Failure Handling in a Design Expert System*, Butterworth & Co.. London, 1985.