B. Chandrasekaran, Todd Johnson, Jack W. Smith, "Task Structure Analysis for Knowledge Modeling," Communications of the ACM, 33-9, Sep, 1992, pp. 124-136.

B. Chandrasekaran, Todd R. Johnson and Jack W. Smith

Task-Structure Analysis for Knowledge Modeling

In recent years there has been increasing interest in describing complicated information processing systems in terms of the knowledge they have, rather than by the details of their implementation. This requires a means of modeling the knowledge in a system. Several different approaches to knowledge modeling have been developed by researchers working in Artificial Intelligence (AI). Most of these approaches share the view that knowledge must be modeled with respect to a goal or task. In this article, we outline our modeling approach in terms of the notion of a *task-structure*, which recursively links a task to alternative methods and to their subtasks. Our emphasis is on the notion of modeling domain knowledge using tasks and methods as mediating concepts. We begin by tracing the development of a number of different knowledge-modeling approaches. These approaches share many features, but their differences make it difficult to compare systems that have been modeled using different approaches. We present these approaches and describe their similarities and differences. We then give a detailed description, based on the task structure, of our knowledge-modeling approach and illustrate it with task structures for diagnosis and design. Finally, we show how the task structure can be used to compare and unify the other approaches.

Knowledge-Based Systems: What are they?

A knowledge-based system (KBS) has explicit representations of knowledge as well as inference processes that operate on these representations to achieve a goal. An inference process consists of a number of inference steps, each step creating additional knowledge. The process of applying inference steps is repeated until the information needed to fulfill the requirements of the problem-solving goal or task is generated. Typically, both domain knowledge and possible inference steps have to be modeled and represented in some form.

In one sense, knowledge is of general utility—the same piece can be utilized in different contexts and problems; so, unlike traditional procedural approaches, knowledge should not be tied to one task or goal. On the other hand, it is difficult to know what knowledge to put in a system without having an idea of the tasks the KBS will confront. In spite of claims of generality, all KBSs are designed with some task or class of tasks in mind. Similarly, they are designed to be operational across some range of domains. Thus, a clear understanding of the relationship between tasks, knowledge and inferences required to perform the task is needed before knowledge in any domain can be modeled.

Background

Tasks

The word "task" has been used in somewhat different senses in the field, contributing to much confusion. For example, Wielinga et al. in [35] describe a task as a "fixed strategy" for achieving a goal, implying that it is a term synonymous with a method or a procedure specification. In our original work on generic tasks (GT) [6], there was a conflation of the goal with the method: the GTs could be thought of as components of a composite method (e.g., the goal of diagnosis is achieved by a method composed of "data abstraction" and "classification"), or they could be thought of as goals as well (the GT called "hypothesis assessment" had the goal to assess hypotheses). In this article, we use the word "task" as synonymous with types of problem-solving goals: for example, we call diagnosis a task, since we want to talk abstractly about the family of problems, all of which are characterized by achieving the goal of generating a causal explanation of observed abnormal behavior. Specifically, we want to separate the task from the method used to achieve goals of this type.

The Knowledge Level

Newell's Knowledge Level (KL) framework [27] is very useful for describing intelligent systems without becoming bogged down in accidental features of implementation languages (see [20] in this issue for additional discussion and examples of the knowledge level). Much of the discussion in the field has been vitiated by too premature a commitment to a symbol-level repre-

sentation (e.g., whether the representation will be rules or frames, and whether backward- or forward-chaining will be employed). Newell proposed that problem-solving agents can be characterized in terms of the knowledge and goals that can be attributed to them, and the Principle of Rationality by means of which intelligent agents can be assumed to use knowledge relevant to their goals. Thus, in discussing a diagnostic system, whether it is implemented as a rule-based system or a connectionist network, we can talk about the task (or goal) as diagnosis of a certain type, and can identify the knowledge content of the system in two ways: as knowledge about the set of malfunctions, and knowledge that aids in mapping from observations to the malfunctions. Hence, at the knowledge-modeling level, we relate the task to the types of knowledge needed to accomplish it. We can then make additional implementation commitments which will, in turn, give us additional constraints on the forms of knowledge.

The knowledge-level view does not include a specific account of how the problem will be solved (i.e., it does not indicate the representations and inference methods utilized to accomplish the task). However, the knowledge-level view can be applied recursively: that is, some commitment to an inference method can be made fairly abstractly: then this too can be represented as knowledge at the knowledge level. This process can be repeated until the knowledge-level description includes some description of the strategies as well. Each commitment to a method requires some commitment to how the problem will be solved, but not as detailed a symbol-level commitment as is normally done when a programming language such as rules or frame languages is employed.

To see how the KL is used, consider how it can be applied to describe MYCIN, a KBS for selecting therapies for bacterial infections of the blood [33]. At the highest knowledge level the goal of MYCIN is to select a therapy. The knowledge required to do this maps signs and symptoms to therapies. At the next level we can reapply the KL and say that MYCIN's therapy goal is accomplished by first identifying the bacterial infection present, then selecting a therapy for that infection. Hence, we break the top goal, therapy, into two subgoals, diagnosis and selection. At this level we can be more specific about the types of knowledge required for the task. Diagnosis requires knowledge-mapping signs and symptoms to an infection. Selection requires knowledge mapping infections and patient data to a therapy. In such a way, we can continue to apply the KL until the system is specified in sufficient detail to allow its implementation. In MYCIN, each of the subtasks is implemented in a backward-chaining rule-based system. The point, however, is that an accurate KL description of MYCIN hides implementation details of the system.

There are many ways to specify an information processing system without describing implementation details. Examples include the knowledge level, abstract algorithm specifications and Marr's information processing level [22]. The selection of an information processing description depends largely on the system being described and the purpose of the description. The KL is primarily designed for use in describing intelligent agents; hence it describes an information processing system as having goals, actions and bodies of knowledge.

Background Work in Knowledge Modeling

Some of the earliest work in knowledge modeling was done as part of the rule-based system approach. In this approach, the agent's knowledge was viewed mainly as directly available recognition knowledge (i.e., knowledge that indicates exactly what to do in a situation). The knowledge modeling scheme was simply to list, as knowledge, the condition-action rules by which a system behaves. Simple condition-action statements, in which the conditions match the current situation and the actions add to or modify that situation, are termed production rules. However, this level of description does not indicate the real control structure of the system at the task level. For example, the fact that R1 [23] performs a linear sequence of subtasks is not explicitly encoded; the system designer "encrypted," so to speak, this control in the pattern-matching of OPS5, the production-rule system in which R1 is implemented.

Another early knowledge-modeling scheme was based on frames. Frames were often proposed to be at the "knowledge level," since they supposedly were used for representing objects in the domain and their relations, a "deeper" level of representation than production rules. The knowledge-level idea behind frames is that they capture stereotypical knowledge; this idea, however, is not sufficient for modeling control knowledge at the task level. The problem is that frames and frame languages do not provide a task-level vocabulary for modeling control knowledge. When frame languages were used, control of a system was often described at a syntactic level: for example, in terms of which links to pursue for inheritance.

The problem was that during the first decade of knowledge-based systems research, the discussion was almost entirely in terms of the symbol level: in the rule-based paradigm all the problems were posed as issues at the rule architecture level. Very little discussion took place at the level of the relation between the task for which the system was being designed and the kinds of knowledge needed. For example, a major research issue for rule-based systems was the development of an appropriate domain- and task-independent *conflict resolution strategy* that would let the system choose which production rule to

fire when multiple rules matched. When the knowledge is viewed at the appropriate level, however, we can often see the existence of organizations of knowledge that bring up only a small, highly relevant body of knowledge without any need for conflict resolution.

The first set of insights regarding the analysis of knowledge systems at one level removed from their implementations came from a number of sources. Gomez and Chandrasekaran identified classification as a common element in diagnosis [15], and Mittal and Chandrasekaran added data abstraction as another common element [25]. This work led directly to a knowledge-modeling scheme in the form of generic task (GT) languages [6]. A generic task identifies a task of general utility (such as classification), a method for doing the task and the kinds of knowledge needed by the method. The language is made of primitives that allow the required knowledge to be directly described for any domain in which the task can be performed. Chandrasekaran and his colleagues identified a number of such generic tasks [6]. They also showed by example how complex tasks such as diagnosis could be decomposed into such generic tasks [9].

Hierarchical classification [5, 9] was the first generic task to be identified and will serve as a good example of the task-based approach that we and other researchers have been developing. The task of hierarchical classification is the identification of an object based on a set of features. For example, diagnosis can be viewed as classification in which the input is a set of manifestations and the output is the disorder associated with these manifestations. The name of the generic task, hierarchical classification, alludes to the method identified to solve the task. The method, called *Establish-Refine*, assumes the existence of a classification hierarchy of output categories. In the case of medical diagnosis, the hierarchy contains diseases. More general classes of diseases are located at the top of the hierarchy; more specific diseases are located near the bottom. The method operates by first attempting to establish (i.e., confirm to a certain level of confidence) the topmost category. If this can be established, it is then refined: its successors become category hypotheses for the system to consider next. Categories that cannot be established are not refined; hence the hierarchy below these categories does not need to be explored. The generic task description clarifies the control structure and knowledge of a classification system. Instead of describing the system in terms of rules or frames, we can describe the system in terms of categories, category evaluation and refinement. The knowledge required to use hierarchical classification is made explicit: knowledge must be available to test and refine categories. The control of the system is explicitly described at a task level—categories are evaluated and then (if necessary) refined—rather than at the implementation or symbol level.

Somewhat near this time, Clancey had identified "heuristic classification" [11] as a somewhat abstract pattern of inference implicit in MYCIN (see Figure 1). Heuristic classification itself was presented as independent of the rule language in which MYCIN was written so that this higher-level inference pattern could be seen independent of the rule-level representation. Clancey's approach is similar to the GT approach, having identified a task (classification), a method (heuristic classification) and the kinds of knowledge needed to use the method. In fact, the three inferences in heuristic classification (see Figure 1) can be interpreted as three subtasks. MDX [9], the generic-task diagnosis system, also incorporated the same task decomposition. Data abstraction was done using an intelligent database; heuristic match was done by establishing categories in the classification hierarchy, and refinement was done during classification. The difference between the two approaches lies in the explicit identification by Clancey of this combined inference structure as heuristic classification, whereas Chandrasekaran et al. had broken this structure down into its components.

McDermott and associates started investigating the roles of knowledge in various methods for several tasks [24]. Their goal was to develop programs that could automatically acquire knowledge from a domain expert. To do this they developed *role-limiting methods* for solving general tasks, such as the cover-and-differentiate method for diagnosis and the propose-and-revise method for design. Role-limiting methods are "methods that strongly guide knowledge collection and encoding [24]." They proceeded to specify the roles various types of knowledge play in the operation of each method. The major difference between the role-limiting method approach and most of the other approaches discussed here is the requirement that a role-limiting method be completely specified (i.e., that all tasks and subtasks be prespecified down to the level of primitive operations).

Musen investigated ways to model classes of planning problems and the required domain knowledge [26]. He advocated the development of a task model followed by the use of the model to acquire domain facts. His system, Protegé, provides a language for modeling classes of planning problems based on skeletal plan refinement. Once modeled, Protegé created a knowledge editor domain which experts could interact with to build KBSs that solved problems in the planning class.

Gruber and Cohen investigated task models as mediating representations of knowledge acquisition for a diagnostic task [16]. They constructed MU, a task-specific architecture for building application programs that do prospective diagnosis. A companion system, called

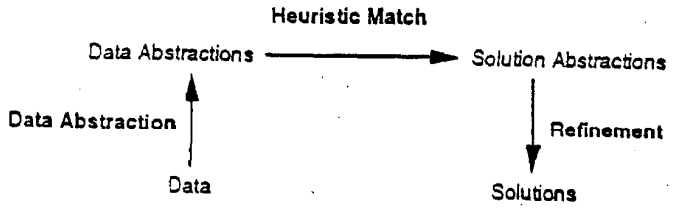


Figure 2. Inference structure for heuristic classification. Adapted from [11].

ASK for "Acquisition of Strategic Knowledge," interacts with an expert to acquire knowledge for MU-based systems. Because ASK is written specifically for MU, it knows about the strategy and types of knowledge needed for the task; hence ASK can interact with an expert at the task level rather than at a lower implementational level.

In Europe, Wielinga and Breuker proposed a set of primitive terms in which to model the tasks that expert systems perform and, in turn, the use of these terms as a modeling language to capture the knowledge in the domain [1]. Their methodology, called KADS, advocates a bottom-up approach to expert knowledge modeling where the knowledge modeler begins with an expert verbal protocol, models this with primitive terms, then builds higher levels of analysis on top of the primitive model. This is quite different from the methodology behind GTs in which the knowledge modeling takes place in a top-down fashion by matching known generic tasks to the task being modeled.

More recently, Steels has proposed work along lines that build on the notion of tasks and task structures [34]. In his formulation, the task structure is intended to specify the task/subtask decomposition of a complex task such as diagnosis. There is a clear recognition that the subtasks of a task depend on the method used for the task. For example, a task such as diagnosis might be done using a classification method. Classification specifies additional subtasks, such as evaluating and refining hypotheses. The task structure notion of Steels does not explicitly represent alternate methods for each of the tasks; instead it is a tree of tasks and subtasks, with the method chosen implicit in the analysis. Thus a given task structure implicitly assumes the choice of some method for a given task. Therefore, it is a good tool for the description of how a particular knowledge system solves the task for which it is intended. The notion of the task structure we will develop later in this article explicitly represents the methods for each task, which then provides a framework for the dynamic selection of methods at run time.

These approaches share two important features. First, they identify tasks at various levels of abstraction above the implementation language level. Second, they identify types of knowledge and strategies closely associated with such tasks. This is the key point in knowledge modeling: once such terms are identified, we have a language in which to model the knowledge in the domain and the strategies to solve the problem. The terms in the vocabulary can be used to encode knowledge, mediate knowledge acquisition [4] and provide suitable explanations [10].

Need for Uniform Framework

In spite of the last decade having seen a clear consensus in favor of task-level analyses and the advantages they offer for knowledge modeling and acquisition, confusion remains regarding the following:

1. Distinctions between tasks and methods and how complex generic tasks and simple generic tasks are related.
2. The great variety of knowledge-modeling terms that have been proposed without any simple way to map between them.
3. Avoiding overdetermination and rigidity in the ways various tasks are performed in the various proposals. That is, we need to show how these various task-level ideas, at various grain sizes, can be combined flexibly.

To overcome these problems, we develop the notions of a task, method, subtask, and the concept of a task structure. The task structure is a uniform task-level analysis framework for describing systems. By viewing the various task-level approaches in terms of the task structure we can begin to compare the approaches and also unravel the current confusions.

The Task Structure

The Task Structure is the tree of tasks, methods and subtasks applied recursively until tasks are reached that are in some sense performed directly using available knowledge. Figure 2 graphically represents part of the task structure for diagnosis. A task (as we defined earlier) is a problem type, such as diagnosis. Tasks are represented graphically using circles. A method is a way of accomplishing a task. These are represented graphically using rectangles. In the figure, *Bayesian Explanation*, *Abductive Assembly* and *Cover-and-differentiate* are identified as methods for diagnosing. All of these methods can be classified as abductive methods,[1] hence they appear as a subtype of *Abduction Methods*. In general, a task can be accomplished using any one of several alternative methods; thus in the task structure we can explicitly identify alternative methods for each task. A method can set up subtasks, which themselves can be ac-

[1] Abduction is the problem of reasoning from effect to cause.

complished by various methods. For example in the *Diagnosis* task structure *Abductive Assembly* has been decomposed into two subtasks: *Generate Plausible Hypotheses* and *Select Hypotheses*.

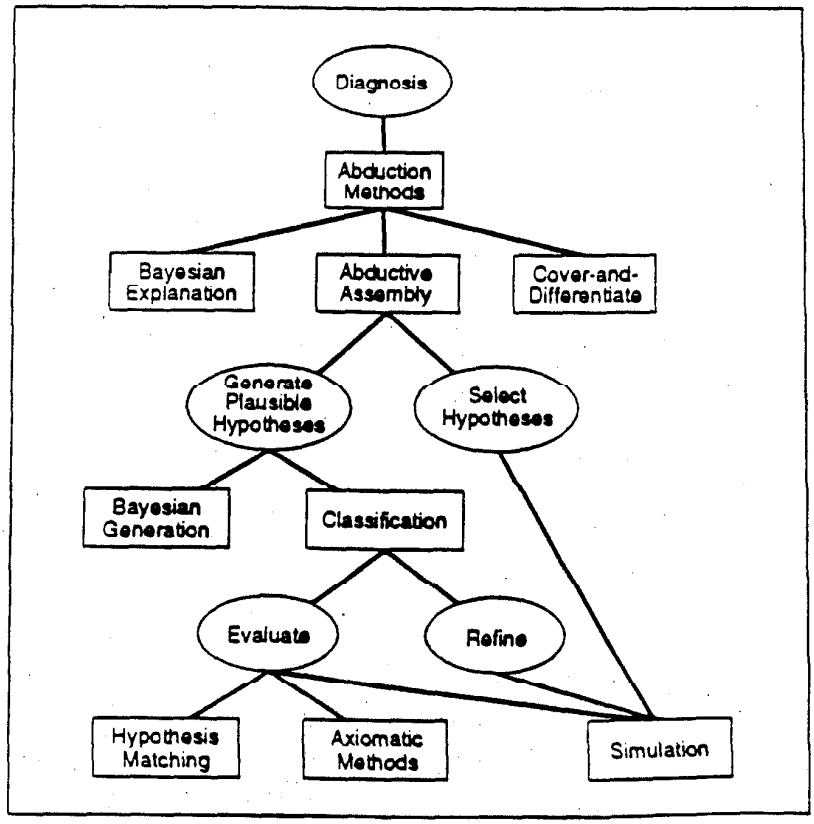
Knowledge in a task structure comes in four forms. First, each task must be accomplished using knowledge that maps the input of the task to the output. Second, knowledge must indicate when an applicable subtask is needed. Third, when a method consists of subtasks, knowledge is needed to sequence the subtasks. Fourth, when a task can be accomplished using two or more methods, knowledge is needed to select a method.

The following subsections describe the task structure in detail. The section "Examples of Task Structure for Design and Diagnosis" discusses the task structures for design and diagnosis in detail.

Tasks

Tasks are specified as transforming an initial problem state with certain features to a goal state with certain additional features. For example, in the diagnostic task the initial state includes malfunction observations and the goal state includes information about the causes of the malfunctions. It is important to distinguish between a task and a task instance. A task instance is a particular problem/goal state pair, such as the diagnosis of a particular patient with specific symptoms. In contrast, a task specifies a family of task instances of a certain type. This family can be defined at various levels of generality. For example, the diagnosis of a patient with specific symptoms is a task instance of the medical diagnosis task, which is itself a subclass of the general diagnosis task.

Part of the task structure for Diagnosis. Circles represent tasks; rectangles represent methods. See section "Examples of Task Structure for Design and Diagnosis" for a discussion of the role of simulation.



Methods and Subtasks

Methods are ways of accomplishing tasks and may be of many types: they may be computational, or "situated," (i.e., involve extracting information from the surrounding physical world). For example, the task of predicting behavior of a device may be solved by a computational method that performs a simulation, or it may be solved by manipulating a physical model of the device and seeing what happens. Within the class of computational methods, a method may be couched as executing a precompiled algorithm, search in a state space, as a connectionist network and so on.

Our uniform framework for describing methods is based on the problem-space computational model[2] [28] and was adopted as a result of work on TIPS [31], an architecture for dynamically integrating generic tasks, and work done integrating generic tasks using problem spaces in the Soar architecture [18]. We define a method to be a set of subtasks that can be used to transform the initial state of a task to the goal state. The method may contain additional information about ordering the subtasks, called *search-control knowledge* [21], or this knowledge can be generated at run time.

While the task structure allows the specification of methods of different types, those that are modeled as problem-space search have a special role for two reasons. For one thing, one way to understand knowledge systems as a distinct type of information technology is to note that the role of explicit knowledge in them is to set up alternatives, evaluate and refine them. In MYCIN, for example, the knowledge in its knowledge base enables it to set up and evaluate various bacterial infection hypotheses. Second, the architecture that integrates the different methods itself can be viewed as operating in a

[2]For an example of a direct application of the problem-space computational model see [20] in this issue.

search space of methods and making selections in it. Third, we will see that the notion of subtasks emerges naturally in the framework of search in problem spaces.

To clarify this, let us consider how to represent the *Establish-Refine* method for hierarchical classification (see earlier subsection "Background Work in Knowledge Modeling") using the framework described. Hierarchical classification is used in many diagnosis systems as a way of quickly focusing on possible malfunctions. The initial state of the classification task is a set of data (e.g., manifestations in a diagnosis task) and an initial high-level hypothesis (e.g., liver disease). The goal state is one containing plausible malfunction hypotheses (i.e., the most detailed hypotheses consistent with the data). The method works by first considering a high-level malfunction category, such as liver disease, to determine if the malfunction appears likely given the data at hand. If it appears likely, then the malfunction is refined to more specific diseases, hepatitis and cancer, for example. The more specific malfunctions are then evaluated against the data and any that appear likely are refined. This process continues until no more malfunctions can be refined.

We can specify this method using two subtasks:

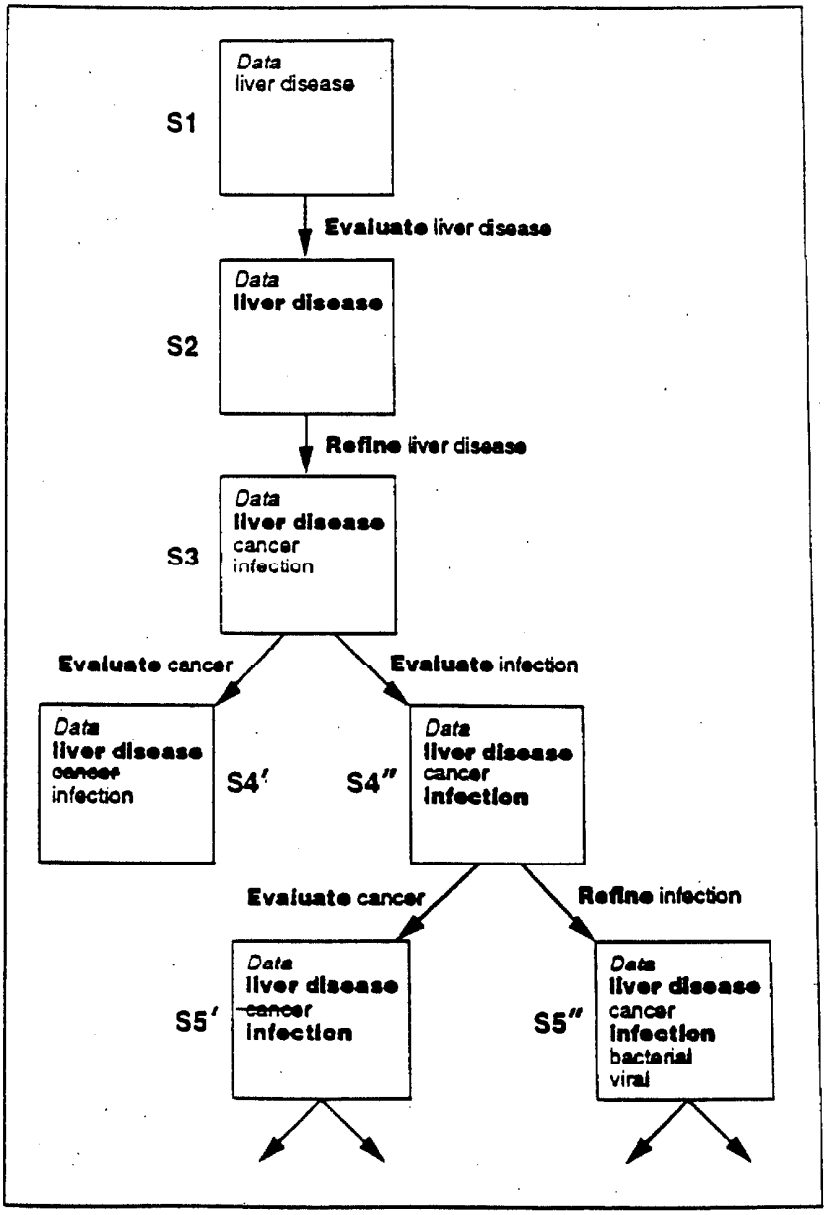
evaluate *hypothesis*
refine *hypothesis*

The first, *evaluate*, takes some hypothesis (such as a malfunction hypothesis) and assigns a likelihood based on the current case data. A precondition for applying *evaluate* to a hypothesis is that the hypothesis must not have already been evaluated. The second subtask, *refine*, takes a hypothesis as input and produces the refinements for that hypothesis. *Refine* has two preconditions: the hypothesis must be likely and must not have already been refined.

We must also specify when an operation should be considered for application to a state. For the hierarchical classification method we are describing, the operations *evaluate* and *refine* should be considered whenever their preconditions are met.

The initial and goal states and the subtasks define a search space or problem space. Figure 3 illustrates the search space that results when the method described is applied to a liver diagnosis problem. The search space is the set of states reachable from the initial state by applying the operators for the method. The figure shows part of the search space of the task, beginning with the initial state, S1, containing manifestations indicative of a viral infection (labeled *Data*) and the high-level hypothesis *liver disease*. The only operator applicable to this state is *evaluate liver disease*. Application of this operation re-

Figure 3. Part of the search space for the hierarchical classification method as applied to liver disease. The data for this example are indicative of a viral infection. Hypotheses in plain text are unevaluated; those in bold are likely; those shown in ~~strike-through~~ are unlikely.



sults in a new state, S2, in which liver disease is rated likely (in the figure this is noted by setting the hypothesis in **bold face**). Only one operator is applicable to S2, *refine liver disease*, resulting in S3 which contains the refinements of *liver disease: cancer* and *infection*. At S3 two operators are applicable: *evaluate cancer* and *evaluate infection*; hence the tree branches to show both possibilities: *evaluate cancer* results in S4' in which *cancer* is determined to be unlikely and *evaluate infection* results in S4" in which *infection* is rated likely.

In the PSCM framework, problems are solved by searching through a problem space for a path from the initial state to the goal state. Problem-space search is done by enumerating subtasks applicable to the current state (which at the start of problem-solving is the initial state of the task instance), selecting from these a single subtask and then applying that operation to the current state. The resulting state then becomes the new current state and the whole process of operation selection and application is repeated until the goal state is reached.

Search-control knowledge guides the search through the problem space. For example, in hierarchical classification the agent might apply a heuristic that it is better to evaluate hypotheses with higher likelihoods than those with low likelihoods, or it might decide that the decision about which evaluate operator to apply is not important, hence either operator can be selected. In the task structure, we specify the minimum amount of search knowledge needed for each method. No search control knowledge is specified for the hierarchical classification method because any such knowledge would unduly constrain the method. For example, if either of the heuristics mentioned previously were included in the search-control knowledge for the method, it would limit the application of the method to those domains and task instances in which the heuristics apply.

The independent specification of search control knowledge and subtasks lead to two of the primary advantages of the problem-space approach to specifying methods:

1. In the task structure we are not forced to specify a particular subtask sequence. We can specify the search-control knowledge that is general to all task instances for a method and defer other decisions about subtask sequencing to system designers or run-time computation. By doing this, we ensure that the method can be applied to as wide a range of task instances as possible. In contrast, early GT work often overconstrained the sequencing of subtasks, limiting the use of each method to a narrow range of problems.
2. Search control knowledge ensures a dynamic or situated system. Each bit of search control knowledge is sensitive to the subtasks and the current state, hence the precise sequence of subtasks is determined dynamically at run time.

Method Selection Knowledge

Functionally, there are four types of knowledge in a task structure. We discussed three of these (search control, subtask application and subtask proposal knowledge) in the previous subsection, since they are related to the description of a method. The remaining type, method selection knowledge, is associated with a task, or a task/method combination. For example, the task structure for Diagnosis includes multiple methods for evaluating a hypothesis. When a system has two or more of these methods, method selection knowledge must be present to determine the best method to take for the task instance being solved.

Direct vs. Derived Knowledge

The four types of knowledge in a task structure can be available in two forms: it can be directly available for the task or it can be computed by another method. Directly available knowledge is in a form that maps the input of the task to the output. For example, directly available knowledge for *refine* is of the form:

> If task is *refine hypothesis* then refinements are *r1*, *r2*, *r3* . . .
> For instance:
> If task is *refine liver disease* then refinements are *infection* and *cancer*.

No complex computation is required to use this knowledge to accomplish the refine task—the knowledge is in a form directly applicable to the task. If knowledge is not directly available, it must be derived from existing knowledge or acquired from the external environment. In either case, a method must be used to acquire knowledge of the desired form. For example, *refine* can be accomplished using a method that knows about different refinement dimensions, such as refinements along etiologic and subpart relations. This method can evaluate various dimensions and then select the dimension appropriate for the task instance. For example, liver disease could be refined using etiology to *infection* and *cancer* or using anatomic structure to *central-area* and *portal area*. The most appropriate dimension to use depends on the task instance (i.e., the kinds of manifestations available).

Whenever any of the four types of knowledge is not directly available, subtasks to acquire the knowledge can be created and set up as new problems. These subtasks are viewed like any other task: they have an initial state and a goal state and can be accomplished by the application of a method consisting of a set of operations. Hence, although a method requires certain kinds of knowledge to be applied to a task, this knowledge does not have to be known before problem-solving can begin, but can be dynamically acquired or derived at run-time.

This idea is also closely related to the distinction between "deep" and "shallow" knowledge, sometimes

called "deep" and "compiled" knowledge. There is also often another distinction between model-based and rule-based reasoning, models being more general knowledge describing the principles of the domain, while rules refer to relatively *ad hoc* associations between evidence and hypotheses. In [8], we provide an analysis of these terms and develop a notion of "depth" of knowledge that is important for knowledge modeling. We give a brief description of this idea.

Let $K(T,M)$ denote the knowledge needed by method M in performing the task T. If a knowledge system performing T using M has the knowledge $K(T,M)$ directly available in its knowledge base, let us say the knowledge system has the knowledge in a *compiled* form. However, suppose some knowledge element k in $K(T,M)$ is missing in the knowledge base, and the task of generating this knowledge is set up as a subtask. If there exists some other body of knowledge in the knowledge base, say K', so that by additional problem-solving using K' we can generate the knowledge element k, we can say that K' is *deep* relative to k.

In the *refine* example we saw that *anatomic structure* is one of the dimensions along which refinement could be done. So-called model-based reasoning is an approach in which structural descriptions of the device under diagnosis are used to generate refinement hypotheses. From this device model, we can generate a list of malfunctions (e.g., one malfunction category can be assigned to the failure of each of the functions of each component; moreover, malfunction categories can correspond to errors in connections between components). The same structural model can be used to generate knowledge needed for the evaluation subtask in Figure 2. The structural model can be simulated for each malfunction, and information about the relation between malfunctions and observations, which is the type of knowledge needed for the methods of the evaluation subtask, can be generated (see the next section for information on simulation). Thus the structural model is a deep model for the methods of classification and hypothesis evaluation that are generally used in the diagnostic task.

The approach to defining the notion of depth of knowledge in the framework of the task/methods/knowledge triple generalizes the intuitive notion that has equated structural models with deep models. Under our definition depth is a relative notion (i.e., it is relative to a method for a task), and there is no notion of characterizing knowledge as deep or shallow in some absolute way.

Examples of Task Structure for Design and Diagnosis

The specification of a task structure consists of three parts:

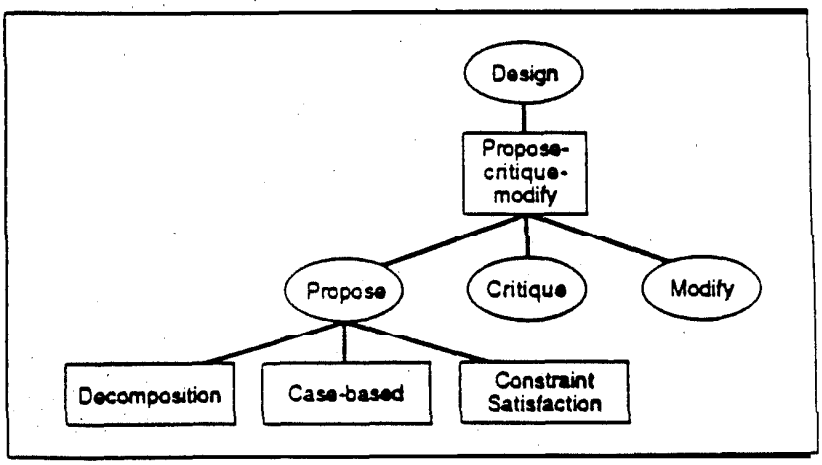
1. an input-output relation that denotes the task;
2. the identification of methods and their subtasks (as in Figure 2); and
3. knowledge to propose subtasks, implement subtasks, sequence subtasks (search-control knowledge) and select methods.

The task-structure diagrams do not list the kinds of knowledge or the input-output relations of the tasks; this is, however, an important part of the specification of the task structure. The following descriptions of design and diagnosis illustrate the main points about specifying the task structure.

Part of the task structure for design is shown in Figure 4. In the task structure diagrams, circles represent tasks and rectangles represent methods. The top task for the design task structure is, of course, design. The design task can be solved using a family of methods called *propose-critique-modify* (PCM) [7]. These methods have the subtasks of proposing partial or complete design solutions, critiquing the proposals by identifying causes of failure, if any, and modifying proposals to satisfy design goals; hence the three subtasks shown for PCM: propose, critique and modify. These subtasks can be combined in fairly complex ways, but the following method is one straightforward way in which a PCM method can organize and combine the subtasks.

Step 1. Given design goal, propose solution. If no proposal, exit with failure.
Step 2. Verify proposal. If verified, exit with success.
Step 3. If unsuccessful, critique proposal to identify sources of failure. If no useful criticism available, exit with failure.
Step 4. Modify proposal; return to 2.

Part of the task structure for design (7). Circles represent tasks; rectangles represent methods.



There can be numerous variants on the way the methods in this class work. For example, a solution can be proposed for only a part of the design problem, a part deemed to be crucial. This solution can then be critiqued and modified. This partial solution can generate additional constraints, leading to further design commitments. Thus, subtasks can be scheduled in a fairly complex way, with subgoals from different methods alternating. One could generate all such variations and identify them all as distinct methods, but both the need for descriptive parsimony and the sheer numerousness of the methods that would result argue against doing that.

Each of the PCM subtasks can be achieved using various methods. Three such families of methods are shown for the proposal task (see Figure 4): decomposition, case-based and constraint satisfaction. In decomposition methods, domain knowledge is used to map subsets of design specifications into a set of smaller design problems. The use of design plans is a special case of the decomposition method. Case-based methods are those retrieving from memory cases with solutions to design problems similar or close to the current problem. Constraint-satisfaction methods use a variety of quantitative and qualitative optimization techniques.

Part of the task structure for diagnosis is shown in Figure 2. The diagnosis task can be viewed as an abductive task, the construction of a best explanation (one or more disorders) to explain a set of data (manifestations). The task structure shows three typical subclasses of abductive methods: Bayesian, abductive assembly [19] and parsimonious covering [30]. Bayesian methods require knowledge of prior probabilities of disorders and conditional probabilities between disorders and manifestations. They use this knowledge to estimate posterior probabilities of disorders. Abductive assembly requires knowledge of disorders and the manifestations they explain. This method works by first generating plausible hypotheses to explain parts of the data and then using these hypotheses to assemble a complete explanation of the data. Parsimonious covering works by stepping through each manifestation, updating the current set of parsimonious explanations as each manifestation is considered. Two subtasks for abductive assembly are shown in the diagram, *generate-plausible-hypotheses* and *select-hypotheses*. These tasks can be done using many kinds of methods. Since Bayesian and classification methods have typically been used to generate plausible hypotheses, these are shown in the task structure.

The task structure for diagnosis also shows that simulation can be used to implement many subtasks. By simulation we mean structure-to-behavior simulation, that is, determining how some device will behave under changes to its structure by simulating its behavior under those conditions. We have earlier discussed the role of simulation for accomplishing subtasks (see previous subsection "Direct vs. Derived Knowledge").

The knowledge required to use abductive assembly consists of control knowledge for sequencing subtasks as well as the knowledge required to accomplish the subtasks. Control knowledge is specific to an application or domain, but the knowledge for accomplishing subtasks can be defined using the input/output specifications of the subtasks. *Generate-plausible-hypotheses* takes as input one or more manifestations and outputs one or more disorders that could be used to explain those manifestations. *Select-hypothesis* takes as input the set of manifestations, the set of disorders currently being used to explain manifestations, and the set of disorders that could be used to explain one or more additional manifestations (i.e., the output of *generate-plausible-hypotheses*). The output of *select-hypothesis* is the disorder it has determined to use to explain the manifestations. Hence, by describing the input output of the subtasks of abductive assembly we also specify the knowledge required to use the method. Simulation can be used to evaluate a hypothesis because the simulation can reveal whether the hypothesis is possible given the data about the device. Causal refinements of a category can be determined by simulating to determine the possible outcomes of a set of inputs to a device. Simulation plays an important role in many task structures because it is a fairly general method for generating knowledge based on the structure of a device. We did not show the simulation method in the design task structure, but there too it can play an important role, especially for critiquing designs.

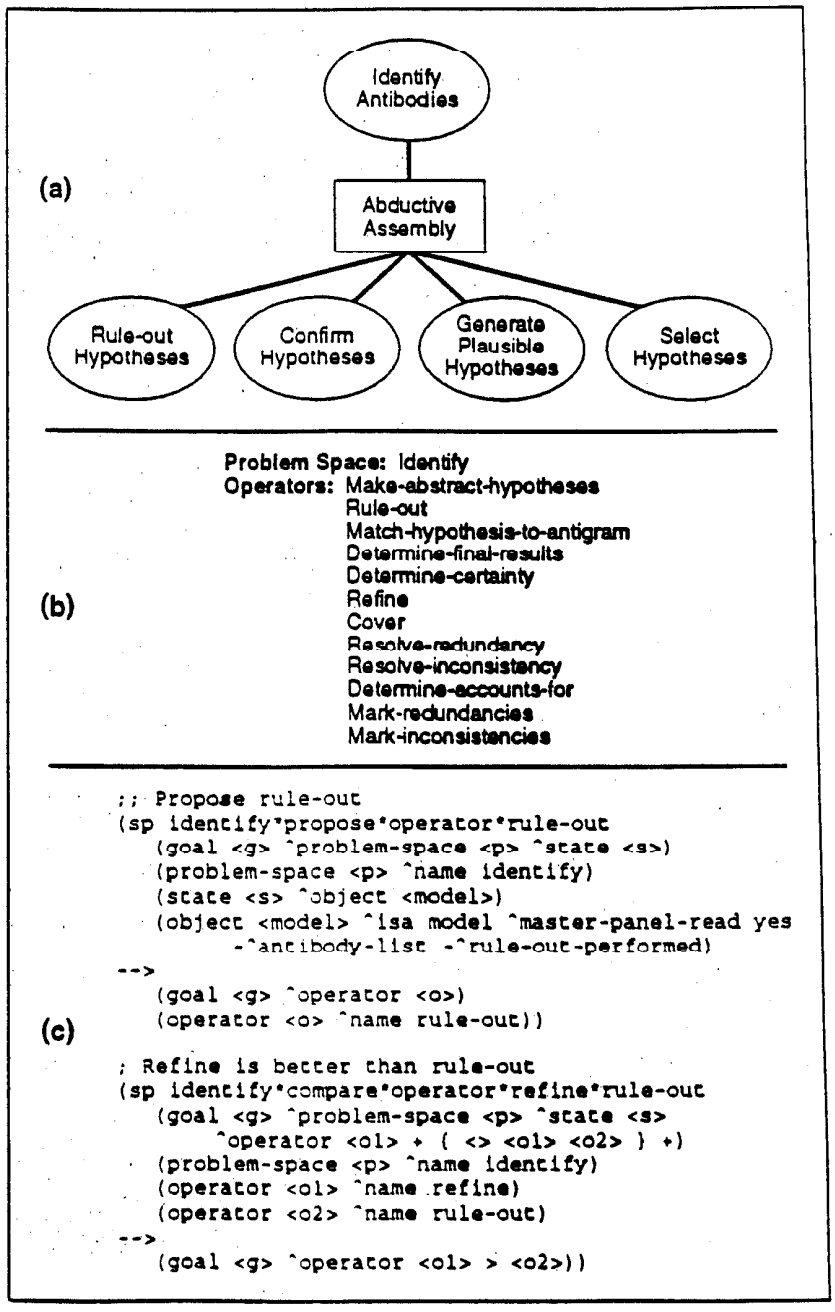
These task structures are based on the methods and subtasks implicit in many expert systems that perform the tasks. Neither of the task structures is meant to be complete; both, however, capture a wide range of the methods useful for achieving the respective tasks. As we discover additional methods, these can be added to the structure. Some methods (such as depth-first search) are so general they can be used to solve any problem. These methods are not listed in the task structure since they would appear everywhere, cluttering the diagram.

The task structure is meant to be an analytical tool. We do not mean to imply that the implementation of a system must have a one-to-one correspondence to the task structure, but that a system that performs diagnosis or design can be *viewed* as using some of the methods and subtasks. In particular, the task structure does not fix the order of subtasks or dictate that a single method must be used to achieve each task. It is also not meant to correspond to a procedure-call hierarchy, although that is one way to directly implement a task structure. The task structure simply provides a vocabulary to use in describing how systems work. The systems

being described might be based on neural networks, production rules, frames, or a task-specific language; this is unimportant for the use and construction of the task structure.

To further emphasize the importance and role of the task structure for describing systems, let us consider three descriptions of RedSoar, a complex abductive system that interprets immunohematologic tests in order to identify antibodies present in a patient's blood [17]. In describing a complex knowledge system such as RedSoar, we can use three levels: 1) the task structure; 2) a computational level (such as problem spaces); and 3) a symbol level. Figure 5 shows each of these levels for RedSoar. RedSoar uses abductive assembly of antibody hypotheses to construct a best explanation of the test data. In the task, the test data are the manifestations; antibodies are the "disorders" or explanations. RedSoar is directly implemented in Soar's production-rule language and can be described by listing all the rules in the knowledge base, such as those in Figure 5c. About 1,000 of these rules constitute the symbol-level view of RedSoar. However, this description fails to capture the task-level control and knowledge in the system. To do this, RedSoar can be described at a computational level by listing the problem spaces defined by the Soar production rules, as in Figure 5b. That is, we can abstract away from the symbol-level production rules to focus on the problem spaces, their initial and desired states and their operators. This level of description is much closer to the task level, but would still contain too many details present as artifacts of the implementation (e.g., extra operators that must be used for low-level manipulation of representations). At the task-structure level (see Figure 5a), we can simply describe the system as using abductive assembly and then point out how it generates and selects hypotheses: i.e., the methods and knowledge that it uses. RedSoar uses conditional and *a priori* probabilities to generate plausible hypotheses and a scoring function based on explanatory coverage and plausibility ratings to select a hypothesis. As shown in the figure, RedSoar also uses two additional subtasks, *rule-out* and *confirm hypotheses*. These are domain-specific subtasks. The first allows the system to quickly rule out clearly absent antibodies. The second lets the system focus on antibodies that are likely present. By describing RedSoar at this level, a comparison can be made between it and other abductive assembly systems by comparing the methods and knowledge used to generate and select hypotheses.

RedSoar described at three levels: a) the task structure; b) the problem-space level (a computational level); and c) production rules (the symbol level).



Knowledge Modeling and the Task Structure

The task structure described in the previous section facilitates knowledge modeling in several ways. First, it associates tasks with methods that accomplish them and the knowledge required to use the methods. The multiple levels of the task structure show how knowledge can be decomposed into bodies of knowledge that are associated with specific tasks. The task structure also highlights the generality and specificity of the knowledge needed for a problem-solving method. That is, it allows methods to be compared based on the required knowledge. Hence, we can see how some methods require little domain knowledge (such as depth-first search, which only requires knowledge to recognize a goal state), while others require considerable domain knowledge (such as hierarchical classification, which needs a domain-specific hierarchy of categories).

Second, since methods are characterized by the knowledge they require, domains can be modeled by tools appropriate for the knowledge that is available in the domain. High-level tools based on this concept, such as CSRL [5], DSPL [2], MUM [16], and MOLE [13], illustrate how this approach facilitates knowledge modeling, knowledge acquisition, explanation, and learning.

Third, the task structure view should be contrasted with what one might call a "uniform normative algorithm" view of how to solve complex problems such as diagnosis or design. For example, there have been proposals for a general algorithm for diagnosis: "diagnosis from first principles" [32], and Bayesian networks [29] are two examples. The general algorithms, while guaranteeing an optimal solution within their respective frameworks, are typically intractable. In these cases the engineering of systems to solve the tasks is done by various forms of heuristic approximations, which of course no longer have the normative properties associated with the original algorithm. The general algorithms also do not always make contact with the form in which knowledge is actually available in various real-world domains. Thus, the Bayesian framework may be fine for a domain in which the needed prior and conditional probabilities (or good approximations to them) are available, but in other domains in which the domain knowledge takes other forms, there is often a need for translating from these forms to the probabilistic forms in which knowledge is needed.

The task structure view, on the other hand, views the solution of complex problems as arising from the interaction of many local methods for local tasks. In any domain having a record of successful human problem-solving, the knowledge in the domain helps to decompose the task into manageable chunks, so each of the problems can be solved to the degree of precision and accuracy needed for the domain. It then becomes the task of the AI theorist to develop vocabularies of generic tasks, methods and knowledge. Thus the attention is shifted from the search for uniform algorithms to modeling knowledge and methods by which tasks are decomposed and subtasks are accomplished.

We can also see how such task structures evolve in real-world domains. If classification is a generally effective method for the generate-hypotheses subtask of diagnosis, then over time, the problem-solving community develops the knowledge needed to apply it. Thus the medical community has devoted hundreds of years to the development of disease taxonomies, which is the form in which the classification method needs knowledge. The knowledge compilation techniques (see subsection "Deep vs. Derived Knowledge") are also a means by which knowledge in a less direct form is converted into knowledge in a form that is more directly usable by a computationally attractive method. Thus we see that in domains and tasks of importance, the domain knowledge tends to *evolve* over time so that methods with good computational properties can be supported.

The fact that we do not start with a uniform normative algorithm does not mean we cannot be precise about the behavior of systems built in the task-structure framework. Bylander [3] and Goel [14] are examples of analyses in which the role of specific types of knowledge in producing good computational properties can be studied within the general framework of the task-structure view. For example, Goel et al. show why classification is an attractive method, if knowledge in the form of classification hierarchies is available, and Bylander et al. show how knowledge about the existence of certain types of causal links (and nonexistence of other types) makes the abductive assembly method tractable.

Fourth, the task structure emphasizes that different kinds of methods can be combined: quantitative and qualitative knowledge, heuristic and algorithmic knowledge can be appropriately combined for the accomplishment of a task. For example, if a subtask can be achieved using a known technique, for instance by solving a set of differential equations, that method can be used instead of more traditional AI methods. Since the method that set up this subtask is concerned with the solution, rather than how it was determined, the original task can be implemented using a different kind of method or even a different computational architecture.

Fifth, the generation of new knowledge can itself be viewed as a reasoning task. Hence during knowledge modeling appropriate questions can identify sources of deep knowledge for various methods in the task structure.

Sixth, the task structure outlined can be used to understand the different task-level knowledge-modeling schemes that have been pro-

posed, which were reviewed earlier (see subsection "Background Work in Knowledge Modeling"). KADS, as well as Clancey's heuristic classification have identified extremely general terms or tasks. These tasks can be used to describe almost any method. Hence they can be considered a set of primitive knowledge-modeling terms. Generic task terms are at a higher level of abstraction in the task structure. They are not general enough to be used to describe all methods, but can be used to describe how higher-level tasks such as diagnosis and design can be performed. Generic tasks can, in turn, be described using more primitive terms. This is in fact what has been done throughout the years of research on generic tasks—the large-grained tasks have been repeatedly decomposed into finer-grained tasks.

Finally, the task structure clears up the confusions discussed earlier (see section "Need for Uniform Framework"). These can be addressed as follows:

1. The relation between complex and more primitive generic tasks is, in one sense, relative to the system being described. For example, if a task is implemented by a method that spawns subtasks, we say the subtasks are more primitive, while the higher task is more complex. The concepts are relative because any task, even those lower in the task structure, can potentially be implemented using a complex set of subtasks. In another sense, we can identify tasks appearing as subtasks in a large number of methods as being more primitive or general than tasks appearing in fewer methods. Hence we can say that diagnosis is a less general and more complex task than data abstraction.

2. The variety of knowledge-modeling terms that have been proposed is due to researchers looking at different parts of task structures and having different goals in mind. Some have looked for extremely primitive terms (e.g., Clancey and KADS), while others have tried to identify higher-level terms (e.g., Steels and Generic Tasks). The task structure shows how these terms can be related through tasks, methods and subtasks. There is still a problem with mapping between terms at the same level; however, Clancey's system model-construction perspective (i.e., the view that what KBSs really do is construct models of systems they are reasoning about) provides a scheme to compare these terms by representing each term in a uniform set/graph/operator language [12].

3. Overdetermination and rigidity in methods are avoided by using the task structure because a complete method does not need to be specified (only the subtasks are given and not all of these have to be used to accomplish a task). Furthermore, multiple methods can be used to model domains that do not warrant the selection of a single method for accomplishing a task. Overdetermination and rigidity of implementation can be avoided by dynamically determining methods and subtask sequencing at runtime. Details of how this can be done in the context of generic tasks are given in [18]; but the basic idea is to dynamically determine what to do at each problem-solving step. That is, after each operation is performed the situation is reassessed to determine what can be done next. Knowledge is then brought to bear to select one of these operations.

Knowledge Modeling Is a Task-Specific Enterprise

There have been some attempts to develop a small number of basic terms in which to formulate all the knowledge to be represented in KBSs. We think it premature to discuss representing knowledge in general at this point in our understanding. We can, however, for various types of tasks, develop detailed theories of the methods and knowledge required to implement them and the terms in which such knowledge can be represented. Thus the knowledge-modeling methodology is a cumulative enterprise by researchers around the world: as research in some task (say diagnosis or design) is carried on in various domains around the world, different methods are identified, their knowledge requirements understood, generalizations and commonalities recognized, performance characteristics of the methods are analytically understood, and a task structure which incorporates this collective product of research emerges. Knowledge-modeling for that particular task is then facilitated by this task structure: we know what kinds of knowledge and strategies are needed for the methods and can use the terms of analysis to model the knowledge in the domain.

In this sense, over the last several years significant knowledge has been accumulated for the following tasks: diagnosis, hierarchical design, configuration tasks and some classes of device simulation tasks. We have outlined the task structure for some of these tasks, and shown how the modeling of knowledge is facilitated by this framework.

The usefulness of the task analysis in the form proposed in this article (and the resulting task structure) is not limited to automation of problem-solving. The analysis itself is merely a description of how the task might be decomposed and what kinds of knowledge are needed. It is possible that for one reason or another some of the methods may not be automatable: the needed knowledge may not be available in a computer-processable form, or the method might itself not be sufficiently operationalized. The task analysis still provides the tool for decomposing a task and identifying which subparts of an overall task can be automated. The subtasks for which automatable methods do not exist at a given stage of AI theory-making can be simply directed to a human problem solver with expertise in the subtask. Thus the task structure provides the framework for natural human-machine cooperation. As we understand how to operational-

ize a previously unautomated method and we acquire the knowledge needed for it, that subtask can be given over to the machine. The task structure thus provides a mobile boundary between human and machine in problem-solving.

Acknowledgments

We thank the members of the LAIR and the Division of Medical Informatics for their comments and discussion on this article. B. Chandrasekaran's work is currently supported by DARPA under AFOSR contract F-49620-89-C-0110. Todd R. Johnson and Jack W. Smith's research is supported by National Heart Lung and Blood Institute grant HL-38776 and National Library of Medicine grant LM-04298.

References

1. Breuker, J. and Wielinga, B. Models of expertise in knowledge acquisition. In *Topics in Expert System Design*, G. Guida, C. Tasso, Eds. Elsevier Science B. V., North-Holland, 1989, pp. 265–295.
2. Brown, D.C. and Chandrasekaran, B. *Design Problem Solving: Knowledge Structures and Control Strategies*. Morgan Kaufmann, San Mateo, Calif., 1989.
3. Bylander, T., Allemang, D., Tanner, M.C. and Josephson, J.R. The computational complexity of abduction. *Artif. Intell. 49*, (1991), 25–60.
4. Bylander, T. and Chandrasekaran, B. Generic Tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition. *Int. J. Man-Machine Studies 26*, (1987), 231–243.
5. Bylander, T. and Mittal, S. CSRL: A language for classificatory problem solving. *AI VII*, 3 (1986), 66–77.
6. Chandrasekaran, B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert 1*, 3 (1986), 23–30.
7. Chandrasekaran, B. Design problem solving: A task analysis. *AI Magazine 11*, 4 (1990), 59–71.
8. Chandrasekaran, B. Models versus rules, deep versus compiled content versus form: Some distinctions in knowledge systems research. *IEEE Expert* (Apr. 1991), 75–79.
9. Chandrasekaran, B. and Mittal, S. Conceptual representation of medical knowledge for diagnosis by computer: MDX and related systems. In *Advances in Computers*, M. Yovits, Ed., Academic Press, 1983, 217–293.
10. Chandrasekaran, B., Tanner, M. and Josephson, J. Explaining control strategies in problem solving. *IEEE Expert* (1989), 9–24.
11. Clancey, W.J. Heuristic classification. *Artif. Intell. 27*, 3 (1985), 289–350.
12. Clancey, W.J. Model Construction Operators. *Artif. Intell. 53* (1992), 1–115.
13. Eshelman, L. MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, Ed., Kluwer Academic, 1988, pp. 37–80.
14. Goel, A., Soundararajan, N. and Chandrasekaran, B. Complexity in classificatory reasoning. In *Proceedings of AAAI* (Seattle, Washington, *July 13–18, 1987*) pp. 421–425.
15. Gomez, F. and Chandrasekaran, B. Knowledge organization and distribution for medical diagnosis. *IEEE Trans. Syst., Man and Cybernetics 11*, 1 (1981), 34–42.
16. Gruber, T. and Cohen, P. Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. *Int. J. Man-Machine Studies 26*, 2 (1987), 143–159.
17. Johnson, K.A., Johnson, T.R., Smith, J.W., Jr., DeJongh, M., Fischer, O., Amra, N.K. and Bayazitoglu, A. *RedSoar—A system for red blood cell antibody identification*. In *Proceedings of SCAMC 91*, McGraw Hill, Washington D.C., 1991, 664–668.
18. Johnson, T.R. Generic tasks in the problem-space paradigm: Building flexible knowledge systems while using task-level constraints. Ph.D. dissertation, Ohio State University, 1991.
19. Josephson, J., Chandrasekaran, B., Smith, J. and Tanner, M. A mechanism for forming composite explanatory hypotheses. *IEEE Trans. Syst., Man, and Cybernetics 17*, 3 (1987), 445–454.
20. Krishnan, R., Li, X. and Steier, D. Development of a knowledge-based mathematical model formulation system. *Commun. ACM* (Sept. 1992).
21. Laird, J.E., Newell, A. and Rosenbloom, P.S. SOAR: An architecture for general intelligence. *Artif. Intell. 33* (1987), 1–64.
22. Marr, D. *Vision*. W.H. Freeman, New York, N.Y., 1982.
23. McDermott, J. R1: A rule-based configurer of computer systems. *Artif. Intell. 19*, 1 (1982), 39–88.
24. McDermott, J. Preliminary steps toward a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, Ed., Kluwer Academic 1988, pp. 225–256.
25. Mittal, S. and Chandrasekaran, B. Patrec: A knowledge-directed database for a diagnostic expert system. *Computer 17*, 9 (1984), 51–58.
26. Musen, M.A. *Automated Generation of Model-Based Knowledge-Acquisition Tools*. Morgan Kaufmann, Inc., San Mateo, Calif., 1989.
27. Newell, A. The Knowledge Level. *AI* (Summer 1981), 1–19.
28. Newell, A., Yost, G., Laird, J.E., Rosenbloom, P.S. and Altmann, E. Formulating the problem space computational model. In *Carnegie-Mellon Computer Science: A 25-Year Commemorative*, R.F., Rashid, Ed., ACM Press: Addison-Wesley, Reading, Mass., 1991.
29. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.
30. Peng, Y. and Reggia, J.A. *Abductive Inference Models for Diagnostic Problem-Solving* Springer-Verlag, New York, N.Y., 1990.
31. Punch, W.F. *A Diagnosis System Using a Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning*, Ph.D. dissertation, The Ohio State University, 1989.
32. Reiter, R.A. A theory of diagnosis from first principles. *Artif. Intell. 32* (1987), 57–95.
33. Shortliffe, E.H. *Computer-Based Medical Consultations: MYCIN* Elsevier, New York, N.Y. 1976.
34. Steels, L. Components of expertise. *AI 11*, 2 (1990), 28–49.
35. Wielinga, B.J., Schreiber, A.T. and Breuker, J.A. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition 4* (1992), 5–53.

CR Categories and Subject Descriptors: D.2.1 [**Software**]: Software Engineering—*requirements/specifications*; D.2.10 [**Software**]: Software Engineering—*design*; I.6.0 [**Computing Methodologies**]: Simulation and Modeling—*general*; I.6.3 [**Computing Methodologies**]: Simulation and Modeling—

pplications; K.6.3 [**Computing Milieux**]: Management of Computing and Information Systems—*software management;* K.6.4 [**Computing Milieux**]: Management of Computing and Information Systems—*system management*

General Terms: Design, Methodology

Additional Key Words and Phrases: Analysis, Modeling

About the Authors:

B. CHANDRASEKARAN is professor of computer and information science and director of the Laboratory for AI Research at The Ohio State University. Current research interests include cognitive architectures, knowledge-based reasoning in diagnosis and design, device understanding, and visual reasoning. **Author's Present Address:** Laboratory for AI Research, Department of Computer and Information Science, Ohio State University, Columbus, OH 43210; email: chandra@cis.ohio-state.edu.

TODD R. JOHNSON is an assistant professor with the Department of Pathology, Laboratory for Knowledge-Based Medical Systems at The Ohio State University. Current research interests include flexible problem-solving, modeling the acquisition of expertise, and reasoning with external information.

JACK W. SMITH is associate professor of pathology and computer and information science and director of the Division of Medical Informatics at The Ohio State University. Current research interests include task-specific and cognitive architectures in knowledge intensive domains, abductive problem-solving, and decision support systems in medicine. **Authors' Present Address:** Laboratory for Knowledge-Based Medical Systems, 376 W. 10th Room 571, Columbus, Oh. 43210; email: tj@cis.ohio-state.edu; smith.30@osu.edu