Machine Learning, 4, 347-375 (1989)
© 1989 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Automated Support for Building and Extending Expert Models

MARK A. MUSEN MUSEN@SUMEX-AIM.STANFORD.EDU
Medical Computer Science Group, Knowledge Systems Laboratory, Stanford University, Stanford, California 94305-5479

Abstract. Building a knowledge-based system is like developing a scientific theory. Although a knowledge base does not constitute a theory of some natural phenomenon, it does represent a theory of how a class of professionals approaches an application task. As when scientists develop a natural theory, builders of expert systems first must formulate a model of the behavior that they wish to understand and then must corroborate and extend that model with the aid of specific examples. Thus there are two interrelated phases of knowledge-base construction: (1) model building and (2) model extension. Computer-based tools can assist developers with both phases of the knowledge-acquisition process. Workers in the area of knowledge acquisition have developed computer-based tools that emphasize either the building of new models or the extension of existing models. The PROTÉGÉ knowledge-acquisition system addresses these two activities individually and facilitates the construction of expert systems when the same general model can be applied to a variety of application tasks.

Key Words. knowledge acquisition, knowledge engineering, human–computer interaction, visual languages, domain modeling

1. Introduction

Knowledge acquisition is the process of eliciting the expertise of authorities in an application area and of formalizing that knowledge within a computer program. From the time of McCarthy's [1968] early proposal for the "Advice Taker" (a theoretical program that could act on the statements about the world that its users typed into it in predicate logic), workers in artificial intelligence (AI) have described tools that could facilitate the knowledge-acquisition process. Knowledge acquistion often is depicted as the cumbersome activity whereby expertise is transferred from the minds of application specialists to those of the computer scientists who build expert systems (knowledge engineers), and thence to the knowledge bases of expert systems. Most builders of knowledge-acquisition tools consequently perceive knowledge acquisition as a problem in *knowledge flow.*

The depiction of knowledge acquisition as the transfer of expertise has caused many researchers to view knowledge engineers as middlemen, whose naiveté in the application area impedes communication and clogs the pipeline during knowledge extraction. Davis' [1976] landmark knowledge-acquisition program, TEIRESIAS, was predicated on the proposition that, if domain experts could enter their knowledge directly into expert systems, the need for knowledge engineers during the refinement of new knowledge bases would be eliminated. Although Davis' suggestion was influential, TEIRESIAS never actually was used by the expert physicians for whom it was intended. During the more than one dozen years that have ensued since the development of TEIRESIAS, a score of computer-based knowledge-acquisition tools have been constructed, most designed to eliminate the need for knowledge engineers [Boose 1989]. Despite this nearly universal goal, not one of these tools has supplanted the humans needed to assist application specialists in the construction and maintenance of production-quality expert systems [Kitto 1989]. Although current knowledge-acquisition tools may greatly facilitate the process, development of most expert systems still requires intermediaries and still is often bottlenecked.

The emphasis on knowledge transfer and the view of the knowledge engineer as an intermediary, however, have hindered the recognition that knowledge acquisition is a creative and inventive activity. When knowledge engineers interview application specialists to develop expert systems, they begin to form mental models of how the experts solve problems; the experts, of course, have mental models of their own that attempt to capture their professional problem-solving behavior. In the course of building the expert system, both the knowledge engineers and the experts continually revise their respective mental models. Although the knowledge engineers and the application specialists may have very different mental models at the outset of their collaboration, the models eventually converge. This convergence is possible (1) because the knowledge-acquisition process forces all parties to commit their mental models to a fixed, publicly examinable form—typically, the emerging knowledge base; and (2) because the frequent consideration of examples and test cases forces the system builders to assess, corroborate, and revise their models. The often-cited difficulties of knowledge acquisition can be ascribed, in general, to creating and agreeing on a shared model of problem solving [Winograd and Flores 1986; Regoczei and Plantinga 1987].

The creation of a knowledge base is much like the creation of a scientific theory. Unlike traditional scientists, however, builders of expert systems are not concerned with the elaboration of theories of natural phenomena; these knowledge engineers instead seek to develop theories of expert behavior. In constructing a knowledge base, system builders first define a general model (or theory) of the application task to be performed. In the case of the MYCIN system [Buchanan and Shortliffe 1984], for example, that general task model was one of diagnosing and treating infectious diseases. Given the initial model, MYCIN's developers validated and revised that model as necessary, attempting to fit the model to specific clinical problems. Once the essential model was worked out, it was then extended to include knowledge of particular kinds of bacteremia and, later, of meningitis. For example, after the basic system had been designed, the developers of MYCIN augmented the program's knowledge base to permit diagnosis and treatment of bacterial, fungal, viral, and tuberculous meningitis by making four separate extensions to the original MYCIN model.

Thus knowledge acquisition can be viewed as comprising two interrelated phases: (1) building a general task model—that is, creating an *intention* of the proposed system's behavior, followed by (2) filling in the specific content knowledge in the domain that is consistent with the general model—that is, creating *extensions* [Addis 1987]. In this paper, I shall discuss the special nature of these two stages of knowledge acquisition, with an emphasis on the kinds of computer-based tools that can facilitate the two phases. Knowledge-acquisition systems such as ROGET [Bennett 1985] are *model-building* tools that are particularly well suited to help knowledge engineers and application specialists to develop theories of expert problem solving. Other systems, such as OPAL [Musen, et al. 1987], are *model-extending* tools that are best used by domain experts working along to define specific applications. Recent work on the PROTÉGÉ knowledge-acquisition system [Musen 1989a, b] demonstrates how a model-building tool can help knowledge engineers to fashion a general task model, such that that model then can be used by a second model-extending tool to permit experts to define specific applications. In particular, PROTÉGÉ allows system builders to create general models of application tasks that can be solved with the method of skeletal-plan refinement [Friedland and Iwasaki 1985]; PROTÉGÉ then generates automatically knowledge-acquisition tools like OPAL that domain experts can use to enter the content knowledge for individual applications.

2. The Problem of Creating Models

Computer-based knowledge-acquisition tools, unlike traditional machine-learning programs, assume that knowledge will be formalized as the consequence of an interaction with a human expert. This interaction, which undeniably constitutes the greatest strength of the knowledge-engineering approach, also is the source of substantial liability. Application specialists cannot simply transfer their expertise to a computer, and knowledge-acquisition programs often cannot accept an expert's entries at face value. Understanding why a direct transfer of expertise is impossible both points to a major distinction between current research in knowledge acquisition and work in machine learning, and motivates important design decisions made in the construction of PROTÉGÉ.

Like the construction of other large pieces of software, the engineering of knowledge-based systems requires significant creativity on the part of system builders. Creativity is essential because the application specialists whose professional acumen is to be encoded as a knowledge base often cannot verbalize how they actually go about solving problems. Experts may not be merely inarticulate in explaining their behavior; they frequently are tongue-tied for reasons stemming from the very nature of human intelligence.

2.1. The Paradox of Expertise

Human cognitive skills appear to be acquired in at least three generally distinct stages of learning [Fitts 1964; LaBerge and Samuels 1974; Johnson 1983]. Although different authors have used different terms to describe the three phases, there is concordance regarding the qualitative changes that occur in the way that people seem to retrieve information during problem solving. Initially, there is the *cognitive* stage, during which an individual identifies the actions that are appropriate in particular circumstances, either as a result of direct instruction or from observation of other people. In this stage, the learner often verbally rehearses information needed for execution of the skill. Next comes the *associative* phase of learning, in which the relationships noted during the cognitive stage are practiced and verbal mediation begins to disappear. With repetition and feedback, the person begins to apply the actions accurately in a fluent and efficient manner. Then, in the final *autonomous* stage, the learner *compiles* the relationships from repeated practice to the point where he can perform them without conscious awareness. Suddenly, the person performs the actions appropriately, proficiently, and effortlessly—without thinking. The knowledge has become *tacit* [Fodor 1968].

There is substantial evidence that, as humans become experienced in an application area and repeatedly apply their know-how to specific tasks, their knowledge becomes compiled and thus inaccessible to their consciousness. Experts lose awareness of what they know. The knowledge that experts acquired as novices may be retrievable in a *declarative* form, yet the skills that these professionals actually practice are *procedural* in nature [Anderson 1987]. Although there is no consensus on how such procedural knowledge is stored within the nervous system [Rumelhart and Norman 1983], the inability of experts to verbalize these compiled associations is well accepted [Nisbett and Wilson 1977; Lyons 1986]. The consequence is that the special knowledge that we would most like to incorporate into our expert systems often is that knowledge about which experts are least able to talk. Johnson [1983] has identified this phenomenon as *the paradox of expertise*.

The paradox is confirmed by experimental data, as well as by much acecdotal experience. Johnson [1983], for example, reports that he once enrolled in classes at the University of Minnesota Medical School as part of his investigation of the process of medical diagnosis. At the same time, Johnson had the opportunity to study a medical colleague (one of his teachers) caring for patients on the hospital wards. Johnson compared the physician's observed clinical behavior with the diagnostic methods his colleague was teaching in the classroom. To Johnson's surprise, the medical-school professor's behavior in practice seemed to contradict what the teacher professed. When confronted with these observations, Johnson's subject responded:

> Oh, I know that, but you see I don't know how I actually do diagnosis, and yet I need to teach things to students. I create what I think of as plausible means of doing tasks and hope students will be able to convert them into effective ones. [Johnson 1983, p. 81]

The clinician in this example recognized explicitly that he could not verbalize his compiled expertise in medical diagnosis. The problem for knowledge engineers and for builders of knowledge-acquisition tools, however, is that people rarely know the limits of their tacit knowledge. When asked to report on their compiled expertise, subjects often volunteer plausible answers that may well be incorrect. In experimental situations, subjects have been shown to be frequently (1) unaware of the existence of a stimulus or cue influencing a response, (2) unaware that a response has been affected by a stimulus, and (3) unaware that a cognitive response has even occurred. Instead, subjects give verbal reports of their cognition based on prior causal theories from their nontacit memory [Nisbett and Wilson 1977]. Furthermore, because Western culture mistakenly teaches us that accurate introspection somehow should be possible [Lyons 1986], people freely explain and rationalize their compiled behaviors without recognizing that these explanations frequently are incorrect.

2.2. Authentic and Reconstructed Strategies

When asked questions about tacit processes, experts volunteer plausible answers that may not reflect their true behavior. These believable, although sometimes inaccurate, responses are known as *reconstructed* reasoning methods [Johnson 1983]. Reconstructed methods typically are acknowledged and endorsed by entire problem-solving communities. They form the basis of most major textbooks. The disadvantage of these methods, however, is that they do not always work. Slovic and Lichtenstein [1971], for example, asked stock brokers to weight the importance of various factors that influenced these brokers' investment decisions. A regression analysis of *actual* decisions made by the stock brokers revealed computed weights for these factors that were poorly correlated with the brokers' subjective ratings. More important, there was a *negative* correlation between the accuracy of introspection and the stock brokers' years of experience. More recently, Michalski and Chilausky [1980] found that decision rules elicited from plant pathologists for the diagnosis of soybean diseases performed less accurately than did a rule set that was automatically induced by application of the AQ11 algorithm to a library of test cases. (The experts' actual diagnoses were used as the gold standard against which the two sets of rules were judged.)

Many workers in knowledge acquisition have consequently argued for the elicitation of *authentic* (as opposed to reconstructed) methods of reasoning in hopes of improving expert-system performance [Johnson 1983; Cleaves 1987; Meyer, et al. 1989]. The goal is determination of the behaviors actually used by experts in performing relevant tasks. Acquisition of authentic knowledge, not surprisingly, requires more than just posing direct questions and asking application experts to introspect. Despite intense research to develop non-biasing interviewing techniques [for example, Ericsson and Simon 1984], psychometric methods [for example, Cooke and McDonald 1987], and ethnographic approaches [for example, Belkin et al., 1987], the elicitation of authentic problem-solving strategies remains cumbersome and often is impractical. The translation of authentic reasoning methods (when such methods can be elicited) into current knowledge-system architectures in a manner that avoids artifacts due to the knowledge-representation language itself also is an unsolved problem.

Knowledge engineers, therefore, must apprehend both the authentic and the reconstructed knowledge derived from application specialists and must assess that knowledge objectively. The engineers serve the important function of detecting gaps in the knowledge and of helping the application specialists to fill those gaps by defining plausible sequences of actions that can achieve the necessary goals. Knowledge engineers thus create theories of how the experts tacitly solve problems. The knowledge bases that embody those theories may not achieve the same level of performance as do the procedures actually used by domain experts, but the knowledge bases nevertheless can be observed, extended, and easily disseminated to other people in need of advice. It is incorrect to view a knowledge base as an embodiment of some human's problem-solving expertise. Knowledge bases instead represent only *models* of expert behavior—models that attempt to approximate, but that do not reproduce, the actual problem-solving steps used by humans [Clancey 1986].

When attempting to automate knowledge acquisition, we must identify the roles that knowledge engineers—and that computer-based tools—can play in either the creation or the extension of expert models. The PROTÉGÉ system has been developed under the premise that, at present, it is neither possible nor desirable to build tools to automate the entire knowledge-acquisition process. We can find data in support of that proposition by examining how knowledge engineers and experts have tried to use previous knowledge-acquisition tools to develop practical knowledge bases. Some automated tools help system developers to craft a model of the application task to be performed. Other tools assume that a model of the task area already exists. We now consider these two classes of knowledge-acquistion programs in detail.

3. Tools for Creating Task Models

When building an expert system, developers must first perform a requirements analysis and must identify the *task* that the expert system will perform. Then, knowledge engineers and application specialists traditionally must work together to construct a model of the proposed system's behavior. This model generally corresponds to the developers' theory of how the expert actually solves problems. Much of the necessary modeling activity entails what Newell [1982] refers to as *knowledge-level analysis*—determining (1) the goals for an intelligent system, (2) the actions of which the system is capable, and (3) the knowledge that the system can use to select actions that can achieve the goals. The process of knowledge-level analysis makes no assumptions about the set of symbols with which the expert system ultimately will be encoded (that is, about the rules, frames, or other data structures within the knowledge-representation language). The concern at this stage is only the *behaviors* of which the system will be capable.

There is increasing agreement in the literature that system builders should model the behavior of a proposed system at the knowledge level before they begin to implement the system. One modeling approach centers on defining abstract, domain-independent strategies known as *problem-solving methods* that can form the basis of languages that system builders can use to describe specific application tasks [Clancey 1985; McDermott 1988]. For example, Clancey's [1985] model of the method of *heuristic classification* includes abstract notions such as (1) *conclusions* that the problem solver may select from a pre-enumerated set, (2) *solution-refinement hierarchies* that allow the problem solver to narrow down the set of conclusions that it makes, (3) *data-abstraction hierarchies* that allow the problem solver to generalize from specific input data, and (4) *heuristics* that link abstractions of the user's input data to potential solutions.

Clancey derived the heuristic-classification model from a retrospective analysis of the behavior of a number of expert systems. Knowledge engineers, however, can apply such models of problem solving *prospectively* when they create new knowledge bases, structuring and clarifying the models that they create. Given an application task, such as MYCIN's task of identifying potential causes of infectious disease, developers can use the domain-independent concepts in the heuristic-classification model to define the intended behavior of an evolving system without reference to individual data structures that might be required to implement that behavior within the computer. By relating task-specific knowledge (such as attributes of possible infectious deseases) to well-understood problem-solving methods (such as the method of heuristic classification), developers clarify the roles that the knowledge plays in the system's production of recommendations, facilitating both the encoding and the maintenance of that system [McDermott 1988].

Researchers in AI have identified a number of domain-independent problem-solving methods that can assist system builders in the creation of knowledge-level models [Clancey 1985; Chandrasekaran 1986; McDermott 1988]. Considerable work concentrates on the elucidation of still other models of problem solving, particularly methods that might be applied to tasks that cannot be performed using classification. Although there is increasing consensus on the importance of the modeling approach, the knowledge-acquisition literature is fragmented by the use of inconsistent terminology. For example, whereas many researchers use the term *problem-solving method* for these abstract strategies [Clancey 1985; McDermott

1988; Boose 1989; Musen 1989c], workers at Ohio State University advocate the term *generic task* [Chandrasekaran 1986]. Yet most authors use the word *task* (without the "generic" modifier) to refer to an application problem to be solved. Unfortunately, the distinction between a *task* and a *generic task* often confuses both readers and authors. The developers of the KADS system for knowledge acquisition [Breuker and Wielinga 1987] use the expression *interpretation model* to refer to the formalization of a problem-solving method. In this paper, I consistently use the expression *problem-solving method*—or simply *method*—when referring to an abstract solution mechanism. The term *task* denotes the statement of an application problem, without regard to how that problem might be solved.

A source of additional confusion may arise in this paper, however, because there often are two kinds of models under discussion. First, there are models of *methods*, which represent sets of both terms and relationships for describing abstract, domain-independent solution strategies. Second, there are models of *tasks*, which represent terms and relationships for defining application problems to be solved. Frequently, system builders use the terms and relationships of a model of a problem-solving method (for example, heuristic classification) to define the specific terms and relationships that are needed to model an application task (for example, organism identification in MYCIN). If the task can be solved using the method, then the model of the method can provide a structure for the model of the task. Indeed, task models often can be viewed as direct extensions (or instantiations) of models of problem-solving methods [Musen 1989c].

Recently, several workers have developed computer-based knowledge-acquisition tools that expand this notion of relating task-specific knowledge to a predefined model of a problem-solving method [for example, Bennett 1985; Eshelman 1988; Marcus 1988]. Each of these tools presupposes a model of a different problem-solving method. Knowledge engineers use the terms and relationships in these models of problem solving to create new models for the solution of application *tasks* (Figure 1). In this paper, I refer to these method-oriented programs as *model-building* tools, because these tools help their users to devise and refine task models. To create the task models, users *extend* a pre-existing model of some problem-solving method. Each extension defines how the domain-independent method can be used to solve a particular application task.

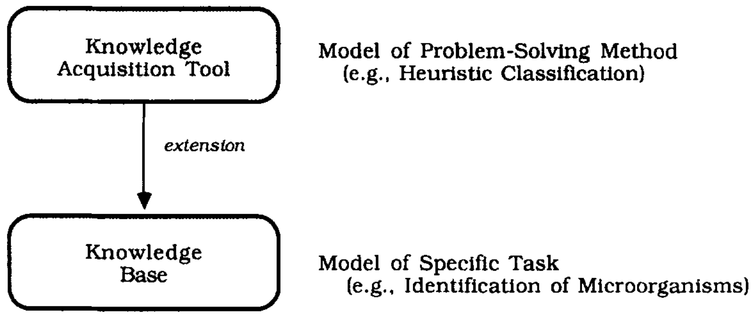

Knowledge Acquisition Tool
Model of Problem-Solving Method (e.g., Heuristic Classification)
extension
Knowledge Base
Model of Specific Task (e.g., Identification of Microorganisms)

Figure 1. Creating a task model. Knowledge-acquisition tools such as ROGET contain models of domain-independent problem-solving methods. Users of such tools extend the problem-solving models to define specific application tasks.

ROGET [Bennett 1985], for example, was a knowledge-acquisition tool that contained a model of diagnosis that was a specialized form of heuristic classification. The program asked its user to identify the *problems* to be diagnosed, the *causes* of those problems, and the *data* that could be used to suggest, to confirm, or to rule out those causes and problems. A user's dialog with ROGET created a knowledge-level specification of the application task, which was then translated into EMYCIN symbols that could form the basis of a working consultation program. The knowledge engineer, however, modeled the application task (for example, the organism-identification task in MYCIN) in terms of the abstract notions of "problems," "causes," and "data." The developer never had to think in terms of the production rules or other data structures that EMYCIN ultimately would require to generate the proper diagnostic behavior.

A number of analogous method-based tools have been described subsequently, including MORE [Kahn, Nowlan, and McDermott 1985], MOLE [Eshelman 1988], and SALT [Marcus 1988]. PROTÉGÉ (which I shall describe in Section 5) is also of this class. Each of these tools provides a language that allows its users to create models of how application tasks can be solved. In each case, that language is one of a particular problem-solving method. Like ROGET, both MORE and MOLE assume that a user's task can be solved using a specialized form of heuristic classification. PROTÉGÉ and SALT, on the other hand, adopt problem-solving methods in which the solution is constructed. The method assumed by PROTÉGÉ is a specialized form of skeletal-plan refinement [Friedland and Iwasaki 1985]. The method built into SALT is a constraint-satisfaction strategy known as *propose and revise*.

Tools such as MORE, MOLE, and SALT allow their users to do much more than to create models of application tasks. Users of these tools also *extend* the task models that they develop with the many domain-specific facts that are necessary to generate complete knowledge bases. Unlike PROTÉGÉ, these other tools do not sharply distinguish between the activities of building models and those of extending them. However, because the process of task-model extension is necessarily preceded by that of task-model creation, it is appropriate to view such method-based tools as knowledge-acquisition aids that assist users in building task models.

In principle, all these model-building tools can be used by domain experts working alone. Indeed, mechanical engineers used SALT to develop an expert system that configures elevators for new buildings [Marcus 1988]. Such method-based tools, however, are used most effectively by knowledge engineers [Musen 1989c]. The terms and relationships of the problem-solving models assumed by the tools (for example, terms in ROGET such as "problems" and "causes") have precise semantics—distinct from these terms' vernacular meanings—that may not be clear to untrained users. A naive user who recognizes such terms as familiar lexical entities, but who may not appreciate the subtleties of the problem-solving model that the terms denote, will be incapable of translating his mental model of a domain task into an effective knowledge base. More important, the tacit nature of human expertise often makes it difficult for application specialists independently to develop robust models of their own behavior. For example, Kitto [1989] reports that when domain experts attempted to use the KNACK knowledge-acquisition tool [Klinker 1988] without the aid of knowledge engineers, the experts' inability to create models of the tasks to be performed constituted a major stumbling block. The entry of instantiating knowledge to extend task models that already had been developed with help from knowledge engineers, however, was much more straightforward for these experts.

4. Tools for Extending Task Models

Regardless of whether a computer-based tool is used to help developers to fashion the task model, after a knowledge engineer and domain expert have created a model of the intended behavior of the expert system, that model must be validated. An important form of validation is to ascertain how well the model applies to closely related application tasks. For example, given a task model that correctly identifies the presence of infections involving one class of micro-organism, system builders will want to confirm that the model can be extended to identify additional classes of potential pathogens. In this phase of knowledge acquisition, the developers test their model by establishing how that model applies to new situations. The system builders' original knowledge-level model is an *intention* of how problem solving occurs; each specific situation for which the model can be shown to apply is an *extension* of that model.[1]

Although creating a knowledge base may be difficult, extending an existing model is less cognitively taxing. Whereas experts may not be able to introspect and to articulate the *process knowledge* that allows them to solve problems [Johnson 1983; Winograd and Flores 1986], these experts certainly are adept at voluntering the *content knowledge* that may be either consistent or inconsistent with a given model. For example, a physician may not be able to provide a coherent description of how he actually diagnoses infectious diseases, but he may be able to describe readily the differences between bacterial and fungal meningitis. Thus, although knowledge engineers typically are needed to help to craft an initial task model, application experts may require little assistance either in extending an existing model or in identifying specific situations in which a given model fails. The frequently raised concern that the experts may not articulate authentic knowledge becomes moot when the specification of only content knowledge is at issue.

The automated knowledge-acquisition tools that are most suited for direct use by domain experts consequently are those that ask their users to extend existing models, rather than to create new ones [Musen 1989c]. Such tools both assume a predefined problem-solving method and incorporate a model of a *class* of application tasks; users extend the general task model to define specific applications (Figure 2). Unlike the detailed task models that knowledge engineers create and extend using tools such as MOLE and SALT, the task models that developers build into this latter set of model-extending knowledge-acquisition tools remain relatively abstract; the models are *intentions*. Rather than describing a particular task to be performed, these models define the characteristics of classes of application tasks that users might want to specify.

An example of such a tool is OPAL [Musen, et al. 1987], which was built by our laboratory to streamline knowledge acquisition for a medical expert system known as ONCOCIN [Tu, et al. 1989]. OPAL contains a model of the general task of administering cancer therapy and asks physicians to extend that model to specific cancer-therapy plans. OPAL's task model presupposes that patients will be treated with groups of drugs called *chemotherapies*. OPAL

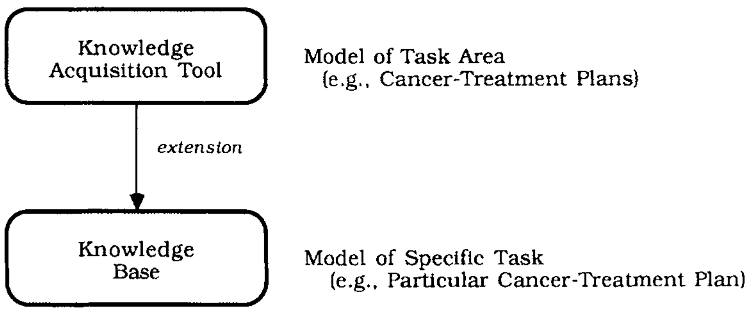
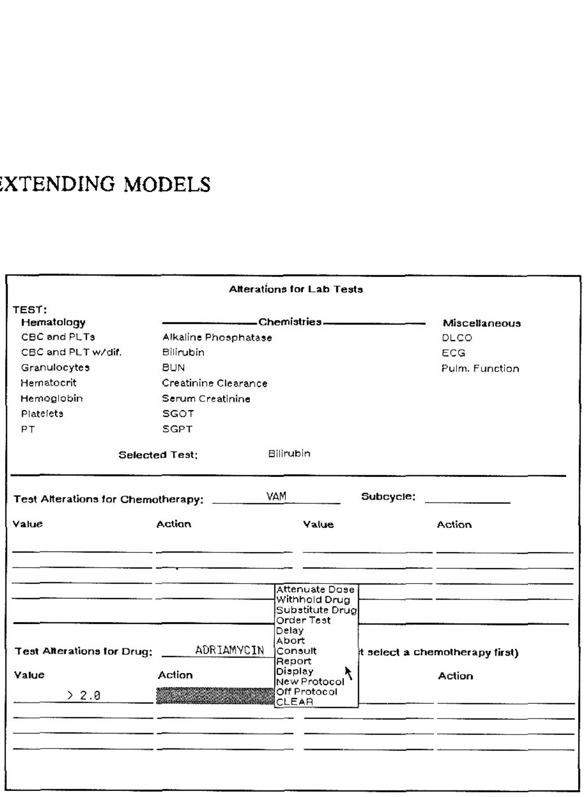

Knowledge Acquisition Tool
Model of Task Area (e.g., Cancer-Treatment Plans)
extension
Knowledge Base
Model of Specific Task (e.g., Particular Cancer-Treatment Plan)

Figure 2. Extending a task model. Knowledge acquisition tools such as OPAL contain models of application-task areas. Users of tools such as OPAL extend the general task models to define specific applications (for example, particular cancer-treatment plans).

does not require its user to stipulate how chemotherapies are administered; such a model was developed by the knowledge engineers who built OPAL. Rather, the program asks its physician-user only to identify the sequence of chemotherapies in a particular treatment plan, to enter the doses of the relevant drugs, and to indicate how the administration of chemotherapy must be modified in response to changes in a patient's condition. Although the individual treatment plans are complex, the pre-existing task model reduces the process of defining new cancer treatments to simply filling in the blanks of graphical forms from menus (Figure 3), or to piecing together sequences of icons using a graphical flowchart language [Figure 4; Musen, et al. 1988]. OPAL thus solicits from the user an extension to its predefined task model that specifies a new treatment plan; the program then automatically generates from that extension a knowledge base that can be interpreted by the ONCOCIN system to carry out that plan.

The task model in OPAL makes assumptions regarding everything from the nature of chemotherapy to the kinds of conditions that can mandate modifications to a physician's treatment plan. Such assumptions define a *closed world*. There is no way to add new concepts to the model. OPAL allows physicians to create novel instantiations of existing concepts (for example, a user can readily define a previously unknown drug or chemotherapy), but the general classes of concepts in the model are predetermined. The task model tends to be sufficient, however, because of the highly stylized nature of cancer therapy. Because the terms in the model have precise, intuitive meanings that match the physicians' common usage of these terms, it is relatively simple for application specialists to fill in the blanks and to connect the flowchart icons in proper sequence to define new therapies. In 1986 alone, physicians used OPAL to enter 36 cancer-treatment plans. Each plan could be entered in a few hours or days. Previously, knowledge engineers and cancer specialists had typically required several weeks of work to encode each such plan using traditional, manual techniques.

System builders construct tools such as OPAL with the assumption that they will create multiple extensions to a given task model (for example, that they will create multiple chemotherapy knowledge bases). It would not be practical to incur the expense of programming such a tool if the system were not to be used repeatedly. There are a number of application areas where knowledge engineers have built tools to facilitate the construction of multiple,



Alterations for Lab Tests

TEST:

Hematology	Chemistries	Miscellaneous
CBC and PLTs	Alkaline Phosphatase	DLCO
CBC and PLT w/dif.	Bilirubin	ECG
Granulocytes	BUN	Pulm. Function
Hematocrit	Creatinine Clearance	
Hemoglobin	Serum Creatinine	
Platelets	SGOT	
PT	SGPT	

Selected Test: Bilirubin

Test Alterations for Chemotherapy: VAM Subcycle:

Value	Action	Value	Action

Test Alterations for Drug: ADRIAMYCIN ...t select a chemotherapy first)

Value	Action		Action
> 2.0			

Figure 3. OPAL form for actions related to laboratory-test results. In this form, the physician is specifying how therapy should be modified if the level of bilirubin in a patient's blood is elevated to more than 2.0 mg/dl.

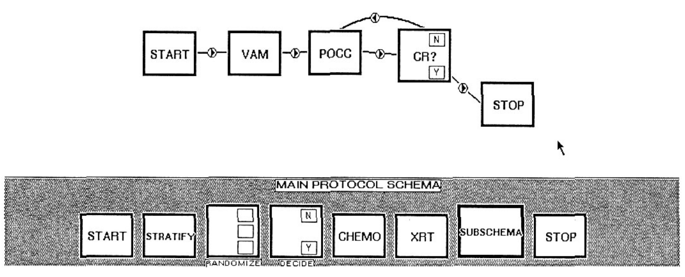


Figure 4. OPAL flowchart language. OPAL allows physicians to create visual programs corresponding to the procedural specification of chemotherapies (CHEMO) and X-ray therapies (XRT) in a given cancer-treatment plan. Below the region where the flowchart is entered is a palette of reference icons, used to add new nodes to the graph. The specification that has been entered in this figure calls for a single course of VAM chemotherapy to be given, followed by administration of POCC chemotherapy until the parameter CR (complete response) becomes *true*.

related knowledge bases. For example, Freiling and Alexander [1984] developed INKA to aid knowledge acquisition for an expert system that troubleshoots electronic instruments; each knowledge base created with INKA specifies fault-detection strategies for diagnosing a particular device. Similarly, Gale [1987] built a program called Student to aid knowledge acquisition for an expert system that advises researchers on the use of data-analysis programs; each knowledge base produced with Student specifies the use of a different statistical routine. In diverse domains such as medical therapy, event scheduling, and process control, system builders would benefit from tools such as OPAL that allow application experts to work alone, extending pre-existing task models to specify the knowledge that defines new task instances.

Although model-extending tools such as OPAL can be powerful in allowing domain specialists to author large knowledge bases without the concurrent need for knowledge engineers, each such tool is necessarily tied to a specific task model. For example, if someone is not interested in constructing a knowledge base for cancer chemotherapy, OPAL is useless to him. Tools such as OPAL can play a significiant role in the life cycle of expert systems when developers require multiple knowledge bases for sets of related domain tasks. The challenge for tool builders is to recognize appropriate application areas and to generate such domain-specific programs rapidly and efficiently.

Building the models that form the basis of systems such as OPAL and Student is itself a problem in knowledge acquisition. Constructing such task-specific tools thus constitutes another kind of bottleneck. OPAL, for example, required 3.5 person-years to develop before any knowledge bases could be encoded. Building OPAL was cumbersome because, whenever developers altered their model of cancer therapy, OPAL had to be reprogrammed. More important, because that task model was not represented explicitly within OPAL, refining the model required kowledge engineers to modify LISP expressions throughout the system's program code; there was no knowledge-level representation of the model. These obstacles to maintaining OPAL, and the desire to transfer the methodology to application areas other than cancer therapy, prompted the development of PROTÉGÉ.

5. Generation of Tools that Extend Task Models

A tool for *building* task models (such as ROGET), which presupposes a particular problem-solving method, is best used by knowledge engineers to create knowlede-level models of the tasks that expert systems will perform. On the other hand, a tool for *extending* task models (such as OPAL), which presupposes a particular set of application tasks, can be used by application experts independently to define specific task instances. The two classes of tools are each suited for distinct phases of the expert-system life cycle. Because model building is invariably followed by model extension—and because the process of model extension often uncovers deficiencies in the original model that need to be repaired—an important goal is to make the use of these two types of tools as integrated as possible. An example of the necessary integration has been achieved with the research system called PROTÉGÉ [Musen 1989a, b].

5.1. The PROTÉGÉ System

PROTÉGÉ is a knowledge-acquisition tool that, like ROGET, assumes a particular problem-solving method—namely, a variant of skeletal-plan refinement [Friedland and Iwasaki 1985]. In performing skeletal planning, a problem solver decomposes a problem's abstract (skeletal) solution into one or more constituent plans that are each worked out in more detail than is the abstract plan. These constituent plans, however, may themselves be skeletal in nature and may require further distillation into subcomponents that are more fleshed out. The refinement process continues until a concrete solution to the problem is achieved.

The expert systems that PROTÉGÉ ultimately constructs produce as their output fully instantiated plans for their users to follow. In the cancer-chemotherapy domain, for example, such plans provide the details of the treatment that physicians should prescribe for an individual patient at specific stages of therapy. The method of skeletal-plan refinement has been applied to practical tasks not only in the ONCOCIN system [Tu, et al. 1989], but also in Friedland's [1979] MOLGEN program and in various versions of the Digitalis Therapy Advisor [Silverman 1975; Swartout 1981]. The method is well suited for applications that require construction of solutions for which the problem solver's reasoning does not need to concentrate on the details of selecting and ordering individual plan operators. In tasks that can be solved by skeletal-plan refinement, the availability of substantial domain knowledge makes it possible for the nuances of operator selection and of constraint satisfaction to be precompiled into the skeletal plans themselves. The problem-solving method consequently avoids search in favor of the instantiation of predefined partial plans [Friedland and Iwasaki 1985].

PROTÉGÉ allows a system builder to create an explicit model of a set of application tasks that can be solved by skeletal-plan refinement. PROTÉGÉ then *generates automatically* a knowledge-acquisition tool like OPAL that is custom-tailored for the set of application tasks that was modeled (Figure 5). PROTÉGÉ recently has been used to construct *p-OPAL*, a knowledge-acquisition tool for the cancer-therapy domain that reproduces the functionality of OPAL. A second program created using PROTÉGÉ, called *HTN*, allows physicians to enter treatment plans for the management of patients with hypertension [Musen 1989a]. Unlike OPAL, which required many months to program by hand, both p-OPAL and HTN were generated with PROTÉGÉ after only a few days of work.

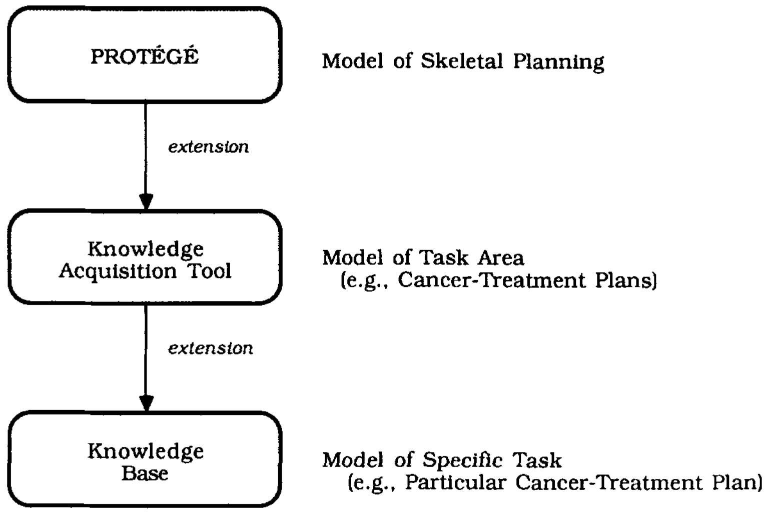

PROTÉGÉ
Model of Skeletal Planning
extension
Knowledge Acquisition Tool
Model of Task Area (e.g., Cancer-Treatment Plans)
extension
Knowledge Base
Model of Specific Task (e.g., Particular Cancer-Treatment Plan)

Figure 5. Creating and extending task models with PROTÉGÉ. Knowledge engineers extend the model of skeletal-plan refinement in PROTÉGÉ to create general task models; the application-specific tools that PROTÉGÉ generates then allow domain experts to extend those task models to define individual applications.

With PROTÉGÉ, a knowledge engineer defines a task model by filling out a series of graphical forms in a manner similar to the way in which oncologists fill out the forms in OPAL. As in OPAL, the PROTÉGÉ forms cluster together related information for presentation to the user and allow data to be examined and edited using direct-manipulation techniques. Although both PROTÉGÉ and OPAL acquire knowledge using hypermedia interfaces [Conklin 1987] and share common styles of human–computer interaction, the nature of the knowledge that users enter into the two systems is quite different. Whereas OPAL acquires knowledge of specific application tasks, PROTÉGÉ acquires knowledge of general task areas. Users enter into OPAL knowledge that is expressed in terms of that program's predefined model of cancer-therapy administration. The knowledge that users enter into PROTÉGÉ, on the other hand, is couched in terms of a predefined model of skeletal-plan refinement.

When PROTÉGÉ is first activated, the system's main menu appears on the workstation screen (Figure 6). This form allows access to other PROTÉGÉ forms that are available at the next organizational level. Via the main menu, the user can cause forms to be displayed that allow him to enter and edit the terms and relationships in a task model.

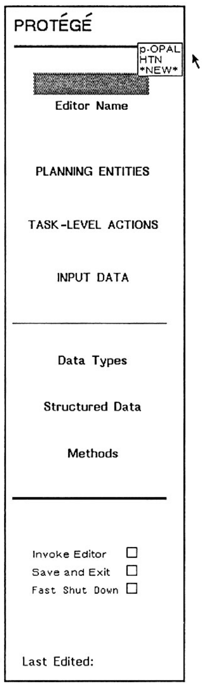


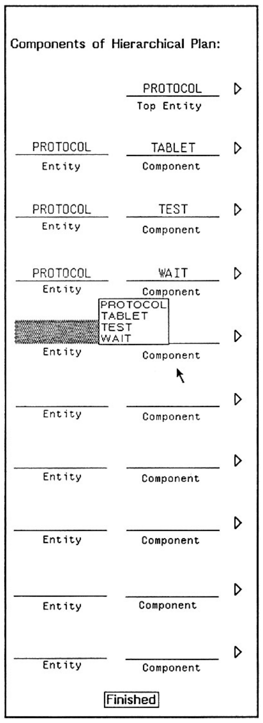
Figure 6. PROTÉGÉ main-menu form. This form asks the user for the name of the knowledge-editing system for which specifications are to be entered or edited using PROTÉGÉ. Once the name has been entered via the pop-up menu, the user can access forms for various topics by selecting the blanks in the menu. The top three items (PLANNING ENTITIES, TASK-LEVEL ACTIONS, and INPUT DATA) correspond to the three principal components of PROTÉGÉ's model of skeletal planning.

Task models created using PROTÉGÉ have the same three general components as does the cancer-therapy task model that was hand-coded into OPAL: (1) the *planning entities* in the domain from which the target expert system will refine its skeletal plans (for example, concepts such as the administration *chemotherapies* and *drugs* in oncology), (2) *actions* that can modify the application of one of the planning entities (for example, concepts such as *attenuating* the dose of a drug or *delaying* treatment), and (3) *input data* that will be entered by the user of the target expert system, the values of which may determine whether any of the *actions* should be applied (for example, concepts such as *laboratory-test results*).

The challenge for PROTÉGÉ users is to create a model of the task area under consideration that can be represented using the terms and relationships of the predefined skeletal-planning model. The knowledge engineer and domain specialist thus must examine the application area and discern the kinds of abstract plans that experts may construct. The developers then map the components of those plans into a hierarchy of PROTÉGÉ *planning entities* and establish the attributes of those plan components that are relevant during problem solving. The users also must determine how experts may modify the standard plans in the task area on the basis of external conditions, modeling such potential plan alterations as a set of PROTÉGÉ task *actions*. Finally, the knowledge engineer and application specialist must consider those external features that may bear on the system's recommendations. These features are modeled as *input data* in PROTÉGÉ's terminology. The PROTÉGÉ interface assists the developers by providing an explicit structure and a convenient notation for recording the components of the task model. Nevertheless, knowledge engineers and application specialists still must collaborate using traditional techniques to elucidate that model in the first place.

The mechanics of entering a task model in PROTÉGÉ are straightforward. For example, selecting PLANNING ENTITIES from the main menu in Figure 6 causes PROTÉGÉ to display the corresponding form for defining the components of skeletal plans in the relevant application domain. Figure 7 shows this planning-entities form filled out for the hypertension-therapy task, as was done to produce the HTN knowledge-editing tool. In the figure, the knowledge engineer has specified that the most general component of a plan is called a *protocol*, and that a problem solver may refine hypertension protocols into more detailed plans that entail the prescription of *tablets*, the ordering of *tests*, and the passage of *wait* periods. The specifications for these components say nothing about the particular kinds of tablets that might be prescribed or the precise tests that might be ordered during the administration of a particular treatment protocol for high blood pressure; the specifications form only an *intention* of the application tasks that are possible in the hypertension domain. Once PROTÉGÉ generates a knowledge-editing tool based on this task model, then application specialists can enter the *extensions* to the model that define individual treatment plans. The problems of building a task model and of extending that model are therefore separated.

In addition to the form for PLANNING ENTITIES in Figure 7, PROTÉGÉ contains eleven other forms that knowledge engineers fill in to describe various aspects of a task model [Musen 1989a]. Each form acquires information related to a particular topic (attributes of planning entities, properties of attributes of planning entities, actions, attributes of actions, and so on). All the forms contain blanks for making entries, as well as icons that allow transfer from one form to the next. When the user selects with the mouse pointing device a triangular-arrow icon in one of these forms, PROTÉGÉ displays a new form for entry


Components of Hierarchical Plan:

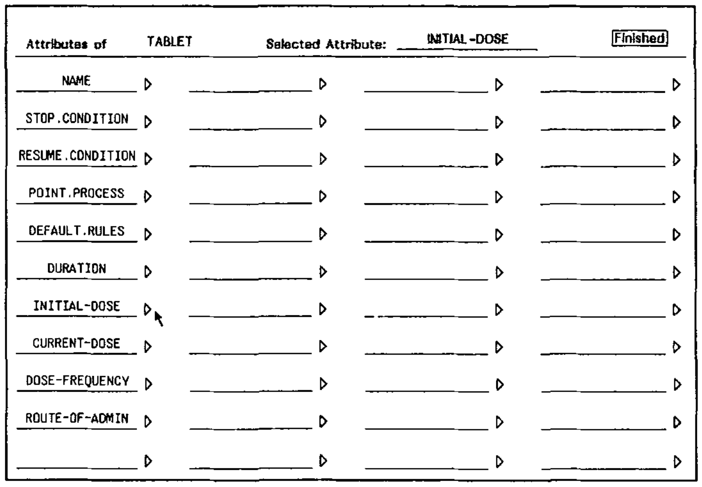
	PROTOCOL Top Entity	▷
PROTOCOL Entity	TABLET Component	▷
PROTOCOL Entity	TEST Component	▷
PROTOCOL Entity	WAIT Component	▷
▨ Entity	PROTOCOL TABLET TEST WAIT Component	▷
Entity	Component	▷
Entity	Component	▷
Entity	Component	▷
Entity	Component	▷
Entity	Component	▷

Finished

Figure 7. PROTÉGÉ form for planning entities. This form is used to enter the planning entities in an application area and to specify their compositional hierarchy. The knowledge engineer types in the names of the entities using the right column. Before the engineer can type in the name of a new entity, however, he must first identify the "parent" entity of which the new entity is a component. In the hypertension-therapy domain, PROTOCOLS comprise the administration of TABLETs, TESTs, and WAIT periods. Selecting the arrow next to the blank filled in with the word TABLET would open the PROTÉGÉ form in Figure 8.

of information at the next lower level of detail. For example, if the knowledge engineer selects the arrow next to the blank for the TABLET planning entity in Figure 7, PROTÉGÉ will open up a form for editing the *attributes* of TABLETs (Figure 8). PROTÉGÉ uses just one form to solicit the attributes of all planning entities that the knowledge engineer may define. Because different entities necessarily have different attributes, however, the way that the knowledge engineer *fills out* the form will depend on the particular entity the attributes of which are to be entered. When the knowledge engineer selects an arrow next to one of the attributes listed in Figure 8, another PROTÉGÉ form appears for editing the *properties* of the indicated attribute. Thus, PROTÉGÉ uses a hierarchy of graphical forms that acquire knowledge at increasingly fine levels of granularity. All forms in the system permit the user to return to the more general form from which the current form was invoked by selecting an icon labeled *finished*.

Whenever possible, PROTÉGÉ allows the user to fill in the necessary blanks by making selections from pop-up menus that the system generates dynamically. This approach not only minimizes the amount of typing that is necessary, but also helps to ensure that the



Attributes of TABLET Selected Attribute: INITIAL-DOSE Finished

NAME ▷	▷	▷	▷
STOP.CONDITION ▷	▷	▷	▷
RESUME.CONDITION ▷	▷	▷	▷
POINT.PROCESS ▷	▷	▷	▷
DEFAULT.RULES ▷	▷	▷	▷
DURATION ▷	▷	▷	▷
INITIAL-DOSE ▷	▷	▷	▷
CURRENT-DOSE ▷	▷	▷	▷
DOSE-FREQUENCY ▷	▷	▷	▷
ROUTE-OF-ADMIN ▷	▷	▷	▷
▷	▷	▷	▷

Figure 8. PROTÉGÉ form for attributes of planning entities. This form lists the attributes of the selected class of planning entity—in this case, *tablet*. PROTÉGÉ enters the first six attributes automatically, as these are common to all classes. The knowledge engineer types in the remainder of the attributes. Selecting one of the arrows causes PROTÉGÉ to display another form that describes the properties of the corresponding attribute.

knowledge engineer's entries are consistent with information that has been stipulated previously. The specifications that the user enters into PROTÉGÉ are stored as *n*-tuples in a relational database. When the user selects *invoke editor* from the PROTÉGÉ main menu (see Figure 6), the system queries the database and constructs a knowledge-acquisition tool based on those data that is tailored for the intended application area.

The semantics both of the knowledge engineer's entries into PROTÉGÉ and of the relational-database schema are grounded in the system's predefined model of skeletal-plan refinement. Thus, when a user indicates that hypertension protocols comprise the administration of tablets, tests, and wait periods (as in Figure 7), the intention of these compositional relationships is established by the meaning ascribed to relationships among plan components in the model of skeletal planning. In interacting with PROTÉGÉ, a knowledge engineer consequently uses his understanding of the terms and relationships in the skeletal-planning model to define task-specific concepts in a domain-independent manner.

For each attribute of each task-specific entity that the knowledge engineer describes for PROTÉGÉ (see Figure 8), the engineer must determine how the attribute is associated with a particular distinguishing value and what the data type of that value is. For each such attribute, the knowledge engineer must indicate whether the corresponding value is constant for all instances of that entity. If the value is indeed fixed, then the knowledge engineer simply enters that value into PROTÉGÉ. (For example, the ROUTE-OF-ADMINISTRATION attribute of all instances of antihypertensive tablets has the value *oral*.) If the value varies depending on circumstances that can be determined only at the time that the skeletal plan is refined, then the knowledge engineer indicates to PROTÉGÉ how the target expert system can ascertain that value at run time. (For example, the CURRENT-DOSE attribute of all tablets has an integer value that the target expert system computes via rules that are invoked at the time of each patient consultation.) Alternatively, the value of an attribute may be independent of consultation-related conditions, but contingent on the particular *instance* of the planning entity. (For example, the value of the INITIAL-DOSE attribute of antihypertensive drugs may vary from tablet to tablet, but still may be a constant for any individual tablet instance.) These instance-specific values represent elements of domain knowledge that can be precompiled into the skeletal plans that the target expert system ultimately will refine. The knowledge-acquisition tools that PROTÉGÉ generates allow users to define such instance-specific values for particular application tasks. In entering these values, users of the PROTÉGÉ-generated tools extend the general task model that the knowledge engineer created using PROTÉGÉ, describing individual applications within the task area.

5.2. Custom-Tailored Model-Extending Tools

The knowledge-acquisition tools that PROTÉGÉ creates produce as their output usable knowledge bases. These knowledge bases allow an expert-system shell extracted from the ONCOCIN program (called *e-ONCOCIN*) to solve application tasks via the method of skeletal planning. Users of the PROTÉGÉ-generated knowledge-acquisition tools, however, are not required to think in terms of either the structure of these knowledge bases or the skeletal-planning method. Instead, the users view their interactions in terms of the task model developed using PROTÉGÉ. Like OPAL, the tools generated by PROTÉGÉ help their users to create new knowledge bases by facilitating the extension of task models, and thus are intended for use directly by application specialists [Musen 1989c].

The hypertension-therapy model discussed previously has been used by PROTÉGÉ to create a knowledge editor, *HTN*, that allows physicians to construct knowledge bases for hypertension management [Musen 1989a]. The description of planning entities entered into the PROTÉGÉ form in Figure 7, for example, provides the basis for a graphical environment in HTN in which users depict the procedures for carrying out individual hypertension protocols (Figure 9). The model of skeletal-plan refinement built into PROTÉGÉ assumes that effecting any given plan component necessarily entails carrying out a sequence of operations involving instances of plan components at the next level of granularity. Because the task model entered into PROTÉGÉ states that hypertension protocols comprise the administration of tablets, tests, and wait periods (see Figure 7), the HTN user automatically is presented with a flowchart language for indicating how individual hypertension protocols are composed of a sequence of instances of precisely such elements. The domain-independent icons with which the user represents the flow of control (namely, START, STOP, RANDOMIZE, and DECIDE) and the SUBSCHEMA icon with which he creates graphical subroutines are built into the graphical language; however, the domain-specific icons (namely, TABLET, TEST, and WAIT) are derived from the task model defined at the PROTÉGÉ level. The flowchart shown in Figure 9 describes a typical experimental protocol in which researchers first administer a placebo tablet for three visits, while monitoring the patients' baseline blood pressure. The physicians then prescribe an active antihypertensive drug for several visits, then withhold all medication and observe the patients for any withdrawal effects. Concurrent with this procedure, a number of laboratory investigations are performed at designated intervals.

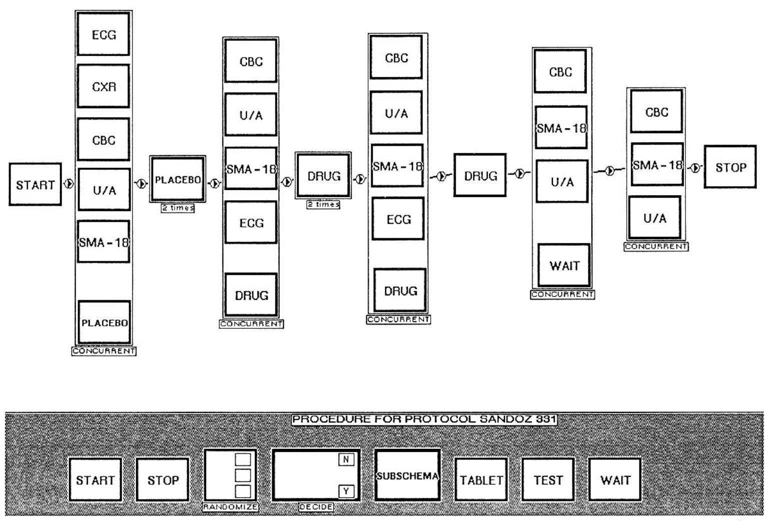
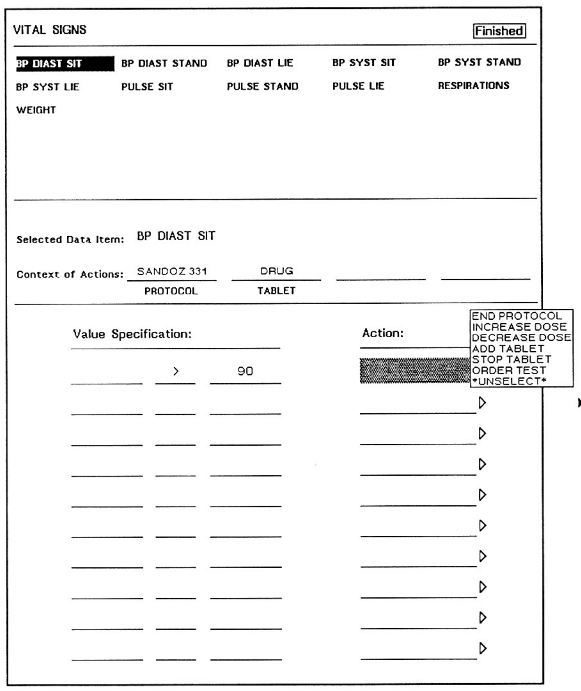


Figure 9. HTN flowchart environment. The HTN knowledge-editing tool includes a graphical language with which physicians draw out the sequence of steps in a protocol for antihypertensive drugs. All task-specific features of this language were derived from the explicit task model that knowledge engineers created previously using PROTÉGÉ. Compare this flowchart with the diagram constructed using OPAL in Figure 4.

When knowledge engineers use PROTÉGÉ to generate knowledge-editing tools for other application areas, similar flowcharting environments are created; the task-dependent aspects of those environments, of course, reflect the models created at the PROTÉGÉ level. Unlike the flowchart language in OPAL, the PROTÉGÉ-generated languages can be modified easily by the knowledge engineer. The developer needs only to edit the task model using PROTÉGÉ and then to regenerate the corresponding knowledge editor. The PROTÉGÉ-derived tools transparently convert the flowchart diagrams that users draw on the workstation screen into augmented transition networks (ATNs) that are incorporated within the knowledge bases of the target expert systems. The e-ONCOCIN inference engine uses these ATNs to determine how instances of skeletal-planning entities (for example, specific hypertension protocols) should be refined into their component skeletal plans from one consultation to the next, Thus, the ATN constructed from the flowchart in Figure 9 would specify that, on the first e-ONCOCIN consultation for a particular patient, the *protocol* should be refined to include the administration of an electrocardiogram (ECG), a chest X-ray study (CXR), a complete blood count (CBC), a urinalysis (U/A), and a blood-chemistry panel (SMA-18)—all of which are instances of *tests*—and that the administration of a placebo *tablet* also should occur. On the occasion of the subsequent consultation for the patient, the ATN would indicate that refinement of the *protocol* plan requires only the administration of *placebo.*

In addition to the flowcharting environments, the tools created by PROTÉGÉ incorporate a variety of graphical forms that are much like those in OPAL (see Figure 3). The domain-specific features of the forms in the PROTÉGÉ-generated system, however, are derived from the explicit task models that knowledge engineers create using PROTÉGÉ. Figure 10, for example, shows one of the graphical forms in HTN. This form allows hypertension


VITAL SIGNS Finished

BP DIAST SIT	BP DIAST STAND	BP DIAST LIE	BP SYST SIT	BP SYST STAND
BP SYST LIE	PULSE SIT	PULSE STAND	PULSE LIE	RESPIRATIONS
WEIGHT				

Selected Data Item: BP DIAST SIT

Context of Actions: SANDOZ 331 PROTOCOL DRUG TABLET

Value Specification: > 90

Action:

Figure 10. HTN form for vital-sign measurements. This form allows physician experts to enter actions to take within hypertension protocols in response to changes in a patient's vital signs. Here, the expert is about to specify actions for e-ONCOCIN to recommend whenever the treating physician notes that a patient's diastolic blood presure (when measured with the patient in the sitting position) is greater than 90 mm Hg. All task-specific features of this form were derived from the explicit task model created at the PROTÉGÉ level. Compare with the OPAL form in Figure 3. (SYST stands for *systolic*; DIAST stands for *diastolic*; STAND, SIT, and LIE indicate whether the patient is standing, sitting, or lying down when the corresponding measurement is taken.)

specialists to indicate how therapy should be modified in response to changes in a patient's vital signs. A list of possible vital-sign measurements that knowledge engineers previously entered into PROTÉGÉ appears at the top of this form. The HTN form in Figure 10 allows physician-experts to indicate actions that e-ONCOCIN should recommend if the end user notes that any of a patient's vital signs (blood pressure, pulse, weight, or respiratory rate) is elevated, depressed, or within a particular range. In the figure, the expert is about to enter the specification that, if a patient's diastolic blood pressure (measured with the patient in the sitting position) is greater than 90 mm Hg, then the dose of the drug that the patient is taking should be increased. (The expert indicates by how much to increase the dose using another form that HTN subsequently displays.) The menu of permitted actions shown in Figure 10 includes choices such as *end protocol*, *add tablet*, and *order test*. When knowledge engineers created the hypertension task model, the meanings of these actions were specified. Although the hypertension-related actions are relatively simple, in domains such as cancer chemotherapy, task actions can be quite complex and can affect a variety of plan components simultaneously [Musen 1989a]. An application specialist who enters knowledge into a

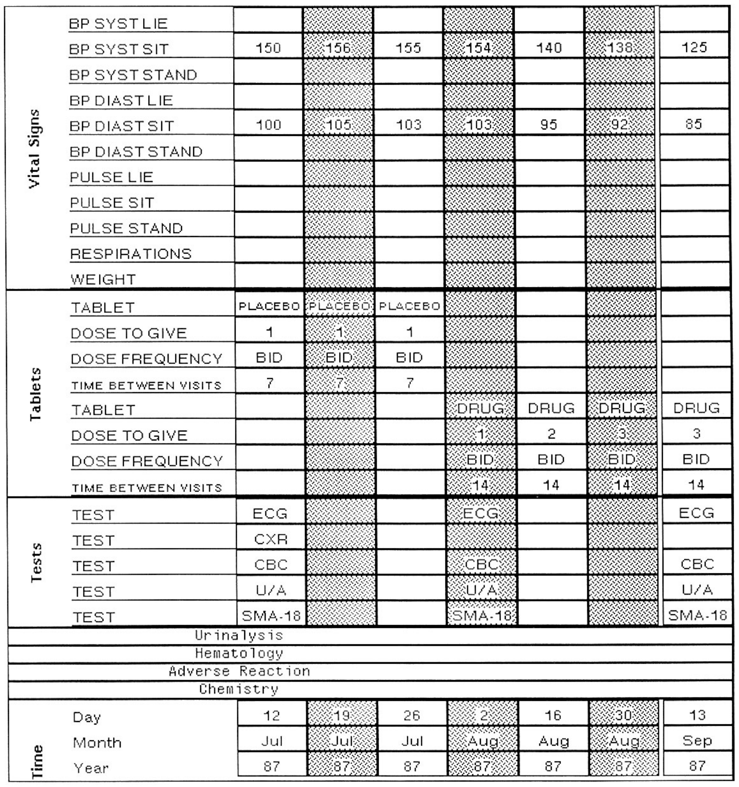
PROTÉGÉ-generated tool does not need to be concerned with the often-thorny issues of working out the semantics of such actions; the user merely selects the predefined actions from the menu. The user, however, still must understand and agree to the semantics established by the developers who created the relevant task model in the first place.

5.3. *The Performance Element: e-ONCOCIN*

The e-ONCOCIN shell has been derived from the ONCOCIN cancer-chemotherapy advice system [Tu, et al. 1989] in much the same way that EMYCIN was distilled from the MYCIN program [Buchanan and Shortliffe 1984]. The shell comprises (1) an inference mechanism that instantiates frame hierarchies using methods such as production rules, ATNs, attached procedures, and queries to the expert-system user, (2) a database for storing time-dependent information that was either entered by the end user or concluded by the system during previous consultations [Kahn, Ferguson, Shortliffe, and Fagan 1986], and (3) a graphical user interface that acquires data from the user and that displays the recommendations concluded by the system. The systems created by PROTÉGÉ therefore must deliver to e-ONCOCIN (1) a knowledge base, (2) a database schema, and (3) specifications for constructing the user interface. These three functional components are encoded as a set of objects in an object-oriented programming language [Lane 1986]. Representation of the simple hypertension protocol described in Section 5.2 (see Figures 9 and 10) required HTN to generate 177 objects.

Users interact with e-ONCOCIN much as they do with the original ONCOCIN system [Lane, et al. 1986]. Each time that a consultation is run on a particular case, the user enters data into a time-oriented spreadsheet (Figure 11). Because the complete spreadsheet is typically too large to be displayed on the workstation screen in its entirety, the interface is divided into sections, such that each section refers to a specific class of input data or to a different portion of e-ONCOCIN's recommendation. With the mouse, users select specific sections of the spreadsheet to examine and then enter current input data into the rightmost column of the indicated sections. (The interface makes it convenient for the users to examine data from previous consultations and to review the recommendations that e-ONCOCIN suggested during these past encounters, because the data are displayed chronologically by column.) After all the current data have been entered, e-ONCOCIN completes its refinement of the relevant skeletal plan and displays the system's recommendation in the corresponding portion of the spreadsheet. In Figure 11, the recommendation appears in the sections labeled *tablets* and *tests*.

The e-ONCOCIN system, like any expert-system shell, assumes a particular knowledge-representation syntax (namely, a hierarchy of frames with attached productions rules and ATNs). The semantics of e-ONCOCIN knowledge bases are determined operationally by the behavior that results when the inference engine is applied to those frames, production rules, and ATNs. At the same time, e-ONCOCIN's behavior can be described in terms of the skeletal-planning model that is built into PROTÉGÉ. When a PROTÉGÉ-generated tool is used to build an e-ONCOCIN knowledge base, the tool automatically constructs the frames and other symbols that will cause e-ONCOCIN's activity during a consultation to match the task model that the knowledge engineer first created with PROTÉGÉ and that the application specialist extended using the resultant tool.



Vital Signs	BP SYST LIE							
	BP SYST SIT	150	156	155	154	140	138	125
	BP SYST STAND							
	BP DIAST LIE							
	BP DIAST SIT	100	105	103	103	95	92	85
	BP DIAST STAND							
	PULSE LIE							
	PULSE SIT							
	PULSE STAND							
	RESPIRATIONS							
	WEIGHT							
Tablets	TABLET	PLACEBO	PLACEBO	PLACEBO				
	DOSE TO GIVE	1	1	1				
	DOSE FREQUENCY	BID	BID	BID				
	TIME BETWEEN VISITS	7	7	7				
	TABLET				DRUG	DRUG	DRUG	DRUG
	DOSE TO GIVE				1	2	3	3
	DOSE FREQUENCY				BID	BID	BID	BID
	TIME BETWEEN VISITS				14	14	14	14
Tests	TEST	ECG			ECG			ECG
	TEST	CXR						
	TEST	CBC			CBC			CBC
	TEST	U/A			U/A			U/A
	TEST	SMA-18			SMA-18			SMA-18
	Urinalysis							
	Hematology							
	Adverse Reaction							
	Chemistry							
Time	Day	12	19	26	2	16	30	13
	Month	Jul	Jul	Jul	Aug	Aug	Aug	Sep
	Year	87	87	87	87	87	87	87

Figure 11. Interface for e-ONCOCIN expert systems. In addition to a knowledge base describing a particular hypertension protocol, HTN generates a user interface for e-ONCOCIN based both on the general task model entered into PROTÉGÉ and on the specific hypertension protocol entered into HTN. The interface consists of a spreadsheet, with each column representing the occurrence of a different e-ONCOCIN consultation regarding the same patient case. In the figure, the sequence of tablets and tests that have been administered corresponds with the HTN flowchart diagram in Figure 9.

The model of skeletal planning that knowledge engineers extend at the PROTÉGÉ level to create new task models ultimately is constrained by the limitations of the e-ONCOCIN shell. Thus, a plan described with PROTÉGÉ can be refined only in a top-down manner, because the current e-ONCOCIN architecture does not include a general mechanism for performing backtracking to satisfy constraints [Tu, et al. 1989]. Similarly, the input data described at the PROTÉGÉ level must be associated with only discrete time intervals that correspond with elements of past or current plans—a restriction that reflects the semantics of the e-ONCOCIN temporal data model [Kahn, Ferguson, et al. 1986]. Future work in our laboratory to enhance the capabilities of the e-ONCOCIN shell ultimately will allow refinement of the problem-solving model built into PROTÉGÉ and will expand the applicability of the system. In the absence of a *meta*-metalevel editor to alter PROTÉGÉ's method-specific assumptions, such changes will require manual reprogramming.

6. Discussion

For over 20 years, many workers in AI have viewed knowledge acquisition as a problem in the transfer of expertise. Concentrating on the issue of knowledge transfer, these researchers have tried to identify impediments to successful knowledge acquisition and have suggested that automated tools can help to improve knowledge flow. Historically, the knowledge engineer is perceived as an intermediary who must interview the expert and then transform the expert's rules of thumb into representations that can be interpreted by the computer. Because the knowledge engineer is inexperienced in the application area and because the expert is unable to envision how his knowledge might be captured within the knowledge base, failures in communication are inevitable. In the traditional view, the knowledge-acquisition bottleneck occurs because of these communication difficulties; if the application experts could somehow record their knowledge directly, without having to explain everything to the knowledge engineers, the development and maintenance of knowledge bases would be accelerated.

From the time of TEIRESIAS, the knowledge-acquisition community has struggled to build tools that might allow application specialists to work alone, bypassing the need for knowledge engineers. Although the knowledge-base–maintenance features of TEIRESIAS were never put into practical use, the program set a standard for how most researchers believed automated knowledge-acquisition tools should function. At conferences and in the literature, developers of new tools boast whenever application specialists have been successful in encoding portions of their knowledge without assistance from human intermediaries. Although such examples are laudable, what often is missing from these reports is careful evaluation of the results that have been achieved. It is often impossible to know how to assign credit for a tool's apparent success. What features of the tool, of the application specialist, or of the situations in which the tool was used were most relevant? More important, knowledge entered directly by application specialists themselves is unlikely to be *authentic* (see Section 2.2). Whenever domain experts use knowledge-acquisition tools without the mediating influence of a knowledge engineer, system builders must be willing to accept that the entered knowledge may not reflect the behavior that the experts actually exhibit in practice. Whether the discrepancy significantly degrades the performance of the target expert system almost never is assessed.

OPAL, for example, is a tool that cancer specialists often use alone without the aid of knowledge engineers. Like most knowledge-acquisition programs, there are many aspects of OPAL that never have been evaluated formally. Once the system was put into routine use, however, the obvious rapidity with which oncology protocols could be encoded using OPAL made knowledge engineers unenthusiastic, to say the least, about engaging in academic experiments that required manual knowledge-engineering techniques. At the same time, because the knowledge that users entered into OPAL was never tacit (but rather entailed content knowledge about the doses of drugs and the sequencing of chemotherapies), system builders never saw the need to question the authenticity of the physicians' specifications. Indeed, knowledge bases created with OPAL have been shown to achieve expert-level performance [Shwe, et al. 1989].

The acquisition of authentic knowledge becomes an issue when system builders create new task models. It is during this early stage of knowledge acquisition that developers formulate their initial theories of how experts solve problems. It is also during this early stage that knowledge engineers—and computer-based tools—can greatly facilitate the modeling process.

Many workers in AI have described expert-system knowledge bases as unstructured collections of rules that correspond with the problem-solving heuristics actually used by experts. In this traditional view, the rules are considered to be modular and independent; each rule thus lacks relationships with other rules in the knowledge base and is devoid of any preordained role in problem solving. Recently, however, the elucidation of heuristic classification [Clancey 1985] and of other problem-solving methods [Chandrasekaran 1986; McDermott 1988] has provided an alternative perspective that offers much more guidance to the programmers who develop and maintain complex knowledge bases. In this new light, expert-system behavior need not be caused by the seemingly random results of one "modular" rule triggering the invocation of another; rather, such behavior can result from the application of coherent, domain-independent strategies. Emphasizing these problem-solving methods allows sytem builders to clarify the roles that elicited knowledge plays in arriving at a task solution and provides a structure by which to direct further knowledge-elicitation work. The use of an explicit model of problem solving (such as that of heuristic classification) when creating the incipient task model in no way guarantees that knowledge engineers will obtain authentic knowledge from application specialists. The model's framework simply helps system builders to structure the elicited knowledge and to determine where there still may be gaps.

Models of problem-solving methods vary in the assumptions that they make about the tasks to which the method can be applied. Very general methods, such as heuristic classification, make few assumptions and, therefore, have tremendous applicability. A great many diagnostic tasks and plan-selection operations, for example, can be represented as extensions of the heuristic-classification model. The generality of the model, however, limits the structure that the heuristic-classification model can impose on the way that knowledge engineers represent domain tasks. There is a direct tradeoff between the applicability of a problem-solving model and the guidance that the model can provide for system builders. The more specialized, less widely applicable models incorporated within programs such as MOLE [Eshelman 1988] and SALT [Marcus 1988] have, in practice, been more helpful to developers attempting to structure domain tasks than have more abstract models such as heuristic classification. The advantage of the more specialized models is that they provide greater assistance in distinguishing the different ways in which a problem solver may use domain knowledge to arrive at a solution. To apply these more detailed models, however, system builders must be able to foresee whether a proposed method will be successful in addressing the task at hand, or whether that method will prove to be too restrictive.

Models of problem solving, when embodied within a computer-based tool, are much more useful to system developers than are models that are mapped out only on paper. The ability to translate a user's extensions of the model into machine-readable knowledge bases is an obvious advantage. A more subtle, but perhaps more important, benefit arises because automated tools can facilitate the presentation of complex systems. The graphical forms in PROTÉGÉ, for example, group together related data and emphasize the relationships entered by the user. Each transition from one form to another moves the user's view of the task model that he is creating to a different level within an abstraction hierarchy. The forms help to break up a knowledge engineer's entries into manageable portions, and the relationships among the forms emphasize the relationships among the components of the user's specifications. The same advantages in knowledge presentation accrue in model-extending tools as well. Users of programs such as OPAL and those generated by PROTÉGÉ benefit from graphical presentation formats that accentuate the relationships among large numbers of entries and that organize those entries coherently.

PROTÉGÉ offers the additional advantage that users can extend a predefined model of problem solving in two discrete stages. Knowledge engineers first extend a model of skeletal-plan refinement to create a task model. Domain experts then extend that task model (itself an extension of the model of the method) to define individual applications. By viewing knowledge acquisition as the process of task-model formation followed by the process of task-model extension, system builders can think critically about these two phases of the expert-system lifecycle and can identify features of knowledge-acquisition tools best suited for each phase. Rather than concentrating on whether the need for knowledge engineers has been obviated by a particular tool—and implicitly assuming that eliminating the knowledge engineers is a necessary and sufficient metric of success—developers can consider the *roles* that knowledge engineers might play in helping application specialists to build models. The knowledge engineer should be regarded as a potential partner, rather than as an inherent marplot, allowing workers in AI to develop more effective strategies for acquiring and representing the tacit knowledge that separates experts from novices. At the same time, by recognizing the ease with which application specialists can enter the content knowledge that extends pre-existing models, developers can build tools such as OPAL that experts can indeed use independently.

The PROTÉGÉ system demonstrates a divide-and-conquer strategy that separates the model-building work that application specialists best perform with the aid of knowledge engineers from the model-extending work that application specialists easily can perform independently. At the PROTÉGÉ level, knowledge engineers work with domain experts to build models of tasks that can be solved using the method of skeletal-plan refinement. These models can then be used as the foundation for custom-tailored knowledge-editing tools. PROTÉGÉ is used to map out the structure of the task and, consequently, the *process* by which a problem solver might arrive at a recommendation. The tools that PROTÉGÉ generates, on the other hand, acquire knowledge about the *content* of specific plans. Although these two phases of knowledge acquisition sometimes may be strictly sequential in nature, attempts to enter content knowledge frequently point out deficiencies in the initial task model; PROTÉGÉ's division of labor allows knowledge engineers to alter the task model easily whenever application specialists encounter problems during their model-extension work. (With OPAL, changes to the task model always required cumbersome reprogramming of LISP code.)

The decision regarding the optimum way to separate task knowledge into a fixed, reusable portion and a variable, application-specific portion is an important judgment that all PROTÉGÉ users must face. The declaration of the classes of entities in the domain and the attributes of those entities is necessarily part of the task model entered into PROTÉGÉ. The values of those attributes, however, may either be predefined as part of the task model (or have predefined methods by which the attributes' values may be concluded) or be identified as content knowledge to be entered by the user of the tool that PROTÉGÉ generates.

Whether an attribute's value should be considered a constant element of the task model or part of the application-specific content knowledge is determined by the nature of the task domain and by the role that that attribute plays in problem solving.

Although demonstrated within the context of the skeletal-planning method, the PROTÉGÉ approach also should apply to other methods of problem solving. For example, if the system were adapted for an inference engine that is well-suited for solving problems using heuristic classification (such as EMYCIN), knowledge engineers then would use PROTÉGÉ to create models of classification tasks, rather than models of planning tasks. A knowledge engineer, for instance, might use PROTÉGÉ to describe the set of classification problems that is encountered during geological mineral exploration, as was done in the Prospector system [Reboh 1981]. A knowledge-acquisition tool generated by PROTÉGÉ then could be used by expert geologists to enter specific ore-deposit models. The ore-deposit models could be converted to knowledge bases for expert systems that workers in the field would use to detemine the most favorable drilling sites for particular minerals.

Where there are multiple, related tasks within an application area—and when there is thus the need to construct multiple knowledge bases—the PROTÉGÉ approach offers a considerable advantage. The difficult problem of creating a computational model of the domain task does not disappear; the need for knowledge engineers to help application specialists to build such a model does not disappear either. Nevertheless, the methodology allows system builders to confront only a single bottleneck. If knowledge engineers and domain experts first use tools such as PROTÉGÉ to build the required task models, those experts then can go to work on their own, extending those task models to define multiple knowledge bases. The models incorporated within the tools that PROTÉGÉ generates, however, may not always account for all the professional behaviors that system builders ultimately may observe in an application area. When the user of a PROTÉGÉ-generated tool is unable to extend the given task model to specify a required action (that is, if he must unexpectedly describe an entity that is not within the original model), the task model may have to be augmented at the PROTÉGÉ level.

Like natural theories that are proposed, tested, and revised, the models constructed by knowledge-acquisition tools display a distinct life cycle. Workers in AI have built a variety of tools, each addressing different aspects of this modeling process. Tools such as ROGET assist developers with the initial model-building phase when the task still may be ill defined. Tools such as OPAL aid in the final model-extending phase, when the task area is well understood and end users require multiple, related knowledge bases. The new challenge is to integrate these approaches, allowing model building to be followed by model extension, providing continuous assistance from the time that the application task is first identified to the time that the final knowledge base is disseminated to end users.

PROTÉGÉ is the first step toward that integration. Workers in AI, however, have not yet identified an optimal technology for acquiring knowledge for expert systems, and even less is known about acquiring domain knowledge for the purposes of building *knowledge-acquisition tools*. Consequently, there will be substantial opportunities for research as the PROTÉGÉ approach is broadened to other task areas, to other problem-solving methods, and to other knowledge-system architectures. In the process of expanding the techniques demonstrated by PROTÉGÉ, we shall be able to learn more about the structure and applicability of new problem-solving methods and about the modeling of domain tasks.

Acknowledgments

Development of OPAL and PROTÉGÉ was supported in part by grants LM-07033 and LM-04420 from the National Library of Medicine. Computing facilities were provided by the SUMEX-AIM resource under NIH grant RR-00785. Lyn Dupré, Tom Gruber, and Sandra Marcus provided valuable comments on an earlier draft of this paper.

Notes

1. We also could refer to each situation in which the model applies as an *instantiation*, although many authors reserve that word for descriptions of symbols within a knowledge-representation language. In this paper, therefore, I use the term *extension*.

References

Addis, T.R. 1987. A framework for knowledge elicitation. In *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, Reading, England.

Anderson, J.R. 1987. Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review*, *94*: 192–210.

Belkin, N.J., Brooks, H.M., and Daniels, P.J. 1987. Knowledge elicitation using discourse analysis. *International Journal of Man-Machine Studies*, *27*: 127–144.

Bennett, J.S. 1985. ROGET: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning*, *1*: 49–74.

Boose 1989. A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition*, 1: 3–37.

Breuker, J., and Wielinga, B. 1987. Use of models in the interpretation of verbal data. In A.L. Kidd (Ed.), *Knowledge acquisition for expert systems: A practical handbook*. London: Plenum.

Buchanan, B.G., and Shortliffe, E.H. 1984. *Rule-based expert systems: The MYCIN experiments of the Stanford heuristic programming project*. Reading, MA: Addison-Wesley.

Chandrasekaran, B. 1986. Generic tasks for knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, *1*: 23–30.

Clancey, W.J. 1985. Heuristic classification. *Artificial Intelligence*, *27*: 289–350.

Clancey, W.J. 1986. Viewing knowledge bases as qualitative models. (Technical Report KSL-86-27). Stanford, CA: Knowledge Systems Laboratory, Stanford University.

Cleaves, D.A. 1987. Cognitive biases and corrective techniques: Proposals for improving elicitation procedures for knowledge-based systems. *International Journal of Man-Machine Studies*, *27*: 155–166.

Conklin, J. 1987. Hypertext: An introduction and survey. *Computer*, *20*: 17–41.

Cooke, N.M., and McDonald, J.E. 1987. The application of psychological scaling techniques to knowledge elicitation for knowledge-based systems. *International Journal of Man-Machine Studies*, *26*: 533–550.

Davis, R. 1976. *Applications of Meta Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases*. Ph.D. thesis, Technical Report STAN-CS-76-564, Stanford University, Stanford, CA.

Ericsson, K.A., and Simon, H.A. 1984. *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.

Eshelman, L. 1988. MOLE: A knowledge-acquisition system for cover-and-differentiate systems. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems*. Boston: Kluwer Academic Publishers.

Fitts, P.M. 1964. Perceptual-motor skill learning. In A. Melton (Ed.), *Categories of human learning*. New York: Academic Press.

Fodor, J.A. 1968. The appeal of tacit knowledge in psychological explanation. *Journal of Philosophy*, *65*: 627–640.

Friedland, P.E. 1969. *Knowledge-Based Experiment Design in Molecular Genetics*. Ph.D. thesis, Technical Report STAN-CS-79-771, Stanford University, Stanford, CA.

Friedland, P.E., and Iwasaki, Y. 1985. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, *1*: 161–208.

Freiling, M.J., and Alexander, J.H. 1984. Diagrams and grammars: Tools for mass producing expert systems. In *The First Conference on Artificial Intelligence Applications* (pp. 537–543). Denver, CO: IEEE Computer Society Press.

Gale, W.A. 1987. Knowledge-based knowledge acquisition for a statistical consulting system. *International Journal of Man–Machine Studies*, *13*: 81–116.

Johnson, P.E. 1983. What kind of expert should a system be? *Journal of Medicine and Philosophy*, *8*: 77–97.

Kahn, G., Nowlan, S., and McDermott, J. 1985. Strategies for knowledge acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-7*: 511–522.

Kahn, M.G., Ferguson, J.C., Shortliffe, E.H., and Fagan, L.M. 1985. Representation and use of temporal information in ONCOCIN. In *Proceedings of the Ninth Annual Symposium on Computer Applications in Medical Care*. Baltimore, MD: IEEE Computer Society Press.

Kitto, C.M. 1989. Progress in automated knowledge acquisition tools: How close are we to replacing the knowledge engineer? *Knowledge Acquisition*, in press.

Klinker, G. 1988. KNACK: Sample-driven knowledge acquisition for reporting systems. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems*. Boston: Kluwer Academic Publishers.

LaBerge, D., and Samuels, S.J. 1974. Toward a theory of automatic information processing in reading. *Cognitive Psychology*, *6*: 293–323.

Lane, C.D. 1986. *Ozone reference manual*. (Technical Report KSL-86-40). Stanford, CA: Knowledge Systems Laboratory, Stanford University.

Lane, C.D., Walton, J.D., and Shortliffe, E.H. 1986. Graphical access to medical expert systems: II. Design of an interface for physicians. *Methods of Information in Medicine*, *25*: 143–150.

Lyons, W. 1986. *The disappearance of introspection*. Cambridge, MA: MIT Press.

Marcus, S. 1988. SALT: A knowledge acquisition tool for propose-and-revise systems. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems*. Boston: Kluwer Academic Publishers.

McCarthy, J. 1968. Programs with common sense. In M. Minsky (Ed.), *Semantic information processing*, Cambridge, MA: MIT Press.

McDermott, J. 1988. Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems*. Boston: Kluwer Academic Publishers.

Meyer, M.A., Mniszewski, S.M., and Peaslee, A. 1989. Use of three minimally-biasing elicitation techniques for knowledge acquisition. *Knowledge Acquisition*, 1: 59–71.

Michalski, R.S., and Chilausky, R.L. 1980. Knowledge acquisition by encoding expert rules versus computer induction from examples: A case study involving soybean pathology. *International Journal of Man–Machine Studies*, *12*: 63–87.

Musen, M.A., Fagan, L.M., Combs, D.M., and Shortliffe, E.H. 1987. Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Man-Machine Studies*, *26*: 105–121.

Musen, M.A., Fagan, L.M., and Shortliffe, E.H. 1988. Graphical specification of procedural knowledge for an expert system. In J. Hendler (Ed.), *Expert systems: The user interface*. Norwood, NJ: Ablex.

Musen, M.A. 1989a. *Automated generation of model-based knowledge-acquisition tools*. London: Pitman.

Musen, M.A. 1989b. An editor for the conceptual models of interactive knowledge-acquisition tools. *International Journal of Man–Machine Studies*, in press.

Musen, M.A. 1989c. Conceptual models of interactive knowledge-acquisition tools. *Knowledge Acquisition*, 1: 73–88.

Newell, A. 1982. The knowledge level. *Artificial Intelligence*, *18*: 87–127.

Nisbett, R.E., and Wilson, T.D. 1977. Telling more than we can know: Verbal reports on mental processes. *Psychological review*, *84*: 231–259.

Reboh, R. 1981. *Knowledge engineering techniques and tools in the Prospector environment*. (Technical Report 243). Menlo Park, CA: SRI International.

Regoczei, S., and Plantinga, E.P.O. 1987. Creating the domain of discourse: Ontology and inventory. *International Journal of Man–Machine Studies*, *27*: 235–250.

Rumelhart, D.E., and Norman, D.A. 1983. *Representation in memory*. (Technical Report CHIP 116). San Diego, La Jolla, CA: Center for Human Information Processing, University of California.

Silverman, H.A. 1975. *A digitalis therapy advisor.* (Technical Report MAC/TR-143). Cambridge, MA: Massachusetts Institute of Technology.

Slovic, P., and Lichtenstein, S. 1971. Comparison of Bayesian and regression approaches to the study of information processing in judgment. *Organizational Behavior and Human Performance, 6*: 649–744.

Shwe, M.A., Tu, S.W., and Fagan, L.M. 1989. Validating the knowledge base of a therapy-planning system. *Methods of Information in Medicine, 28*: 36–50.

Swartout, W.R. 1981. *Producing Explanations and Justifications of Expert Consulting Programs*, Ph.D. thesis, Technical Report MIT/LCS/TR-251, Massachusetts Institute of Technology.

Tu, S.W., Kahn, M.G., Musen, M.A., Ferguson, J.C., Shortliffe, E.H., and Fagan, L.M. 1989. Episodic monitoring of time-oriented data for heuristic skeletal-plan refinement. *Communications of the ACM*, in press.

Winograd, T., and Flores, F. 1986. *Understanding Computers and Cognition: A New Foundation for Design*. Norwood, NJ: Ablex.