The New World of Mechanisms

Angel R. Puerta, Samson W. Tu, and Mark A. Musen
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
Stanford, CA 94305-5479, USA
puerta@camis.stanford.edu

ABSTRACT

A goal of second-generation expert systems is to supply knowledge engineers with common frameworks to develop expert systems, thus, eliminating the need to build entirely new systems for each application. A mechanism is a *unit* of problem-solving knowledge from which such a framework can be developed. We define what a mechanism is and how it can be engineered to compose models of problem solving for expert systems. We also introduce PROTÉGÉ-II, an expert-system development environment based on mechanisms. Using PROTÉGÉ-II, knowledge engineers can implement all the elements of an expert system, from the problem-solving specifications, through the definition of knowledge-acquisition tools, to the editing of the domain knowledge in the knowledge base. The use of PROTÉGÉ-II is demonstrated with an example from the medical domain of AIDS therapy.

Keywords: Artificial intelligence, expert systems, knowledge engineering, knowledge acquisition, problem-solving methods, generic tasks.

1. Second-Generation Expert Systems

Research on knowledge-based systems during the 1970s resulted, during the following decade, in the first generation of commercially available expert systems. The construction of expert systems in this period concentrated on two implementational aspects of knowledge-based systems: (1) the representation of knowledge (e.g., as rules, frames, or semantic networks), and (2) the reasoning strategy of the inference engine. One of the crucial shortcomings of this generation of expert systems was the *brittleness* of the resulting implementations; that is, a system designed for a particular task could not be adapted easily to provide expertise for a different task. Consequently, researchers now developing the second generation of expert systems are examining how knowledge is *used* in an expert system, as opposed to how knowledge is *implemented.* One of the paradigms being studied is that of using models of problem solving, or problem-solving methods, as a foundation to build expert systems. Problem-solving methods determine how a task is to be solved, and can be applied to a class of tasks, thus partially overcoming brittleness.

There are numerous examples of models of problem solving, and of knowledge-based systems implemented using those models. *Skeletal-plan refinement* [Friedland and Iwasaki, 1985] is a method that solves problems by elaborating a general (skeletal) solution plan, and by then recursively refining the general plan into more detailed plans until a final, fully detailed plan is produced. *Heuristic classification* [Clancey, 1985] is a method that determines solutions to given tasks by evaluating to which category the solution belongs. SALT [Marcus and McDermott, 1989]

is a knowledge-acquisition tool that generates problem solvers based on the *propose-and-revise* model of problem solving. ROGET [Bennett, 1985] is a system that performs diagnostic tasks using a version of the heuristic-classification model.

Although expert systems built around problem-solving methods are not limited to performing a single task, they are still restricted to the class of tasks that can be modeled under the particular problem-solving method. There are two alternatives to overcome this limitation. The first one is to define more general models of problem solving, which would result in methods that can solve a larger class of tasks. Since there is a certain cost in effort in specializing from a general method to a specific implementation, the drawback with this option is that making models more general also makes them more difficult to apply to specific problems [Klinker et al., 1991]. The second alternative—the one that we advocate in this paper—is to develop architectures that, with a single inference engine, can support multiple models of problem solving, and that thus potentially place no limit on the kinds of tasks that they can perform. But because each method requires a different inference engine, to build such architectures, we need a *unit* of knowledge different from the problem-solving method. We call such a unit a *mechanism.* The unit serves both as a *building block* for assembling problem-solving methods, and as a basis for developing an inference engine that can process mechanisms, or combinations of mechanisms.

In this paper, we introduce the concept of mechanisms and their usefulness to the knowledge-engineering endeavor. In Section 2, we explain how mechanisms can be combined to form problem-solving methods. Section 3 contains an overview of PROTÉGÉ-II—a shell to build problem solvers for methods assembled with mechanisms that is under development at our laboratory. Section 4 demonstrates the use of PROTÉGÉ-II in building an advice system for AIDS therapy. Finally, in Section 5, we discuss how the availability of mechanisms, and of architectures based on them, changes the role of knowledge engineers in developing knowledge-based systems.

2. What Is a Mechanism?

The concept of a mechanism is closely related to that of a task. The word *task* is defined differently throughout the literature, and its meaning is dependent on the context in which it is applied. We call a *task* a representation of a real-world problem. A task has a set of inputs and a set of outputs. A *mechanism* is then a procedure that solves the problem represented by the task. Thus, a task specifies *what* is the problem to be solved, whereas a mechanism specifies *how* to solve that problem (Fig. 1a). Note that, in the definition of a mechanism, there is no restriction regarding the implementation form that the procedure takes (e.g., as rules or frames). The relationship between mechanisms and tasks is many-to-many. A mechanism can solve any task whose inputs and outputs can be mapped to those of the mechanism, and vice versa.

The task–mechanism correspondence constitutes the basis for the idea of using mechanisms as building blocks to compose models of problem solving. By definition, a problem-solving method decomposes a task into subtasks, and then solves each subtask separately to reach a final solution. Consequently, if each one of these subtasks can be mapped to a mechanism, we can state that methods are in fact decomposable into mechanisms (Fig. 1b). The result is that we have an unit of control knowledge that is appropriate for assembling problem-solving methods. In addition, since each mechanism is a black box that encloses its own control flow, an inference engine for mechanisms would need only to execute mechanisms as dictated by the decomposition set by the method, and to have access to a knowledge base that satisfies the input requirements of the mechanisms.

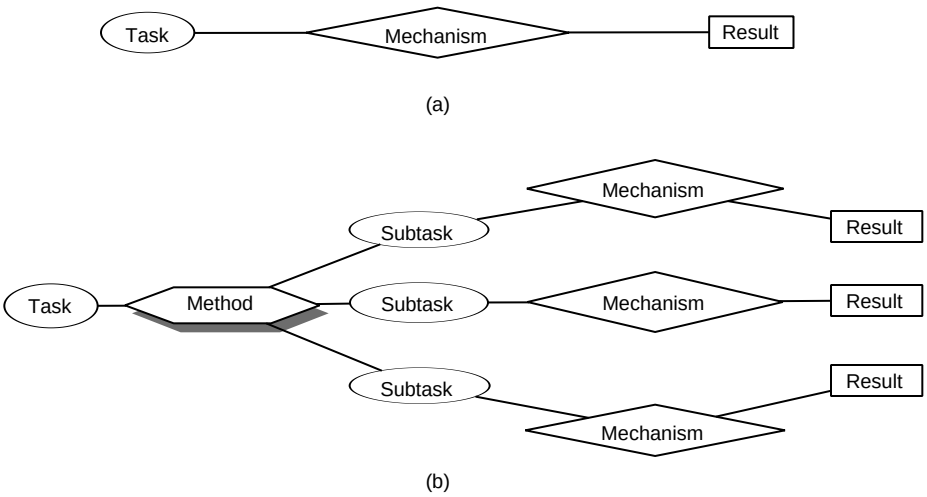

Figure 1. Tasks, methods, and mechanisms. (a) A mechanism is a unit of control knowledge that completely solves the problem represented by a task. (b) A method is a *composite* unit of control knowledge that subdivides tasks into subtasks. The subtasks are then performed by mechanisms. Consequently, methods can be viewed as being composed of mechanisms.

To formalize how mechanisms and problem-solving methods relate to each other, we declare two operations: method configuration and method assembly. These operations correspond respectively to top-down and bottom-up approaches to the implementation of methods. In *method configuration*, we start with an existing method; examine the decomposition of tasks into subtasks that such method imposes; and configure that method by finding, for each subtask, a mechanism whose inputs and outputs map to those of the subtask. On the other hand, in *method assembly*, we begin with individual mechanisms, and assemble them by matching the output of one to the input of another, thus producing a new method. Obviously, the assembly of methods is a much more complex process than is that of configuration because assembly involves the creation of a new decomposition of tasks into subtasks that must be applicable to a whole class of tasks if the created method is to be useful.

Based primarily on method configuration, expert systems can be developed that apply the notions of mechanisms and problem-solving methods. A method provides a proper decomposition of tasks, a method-configuration operation assigns mechanisms to each subtask, a knowledge-acquisition tool obtains the knowledge required by the mechanisms, and an inference engine reasons about the knowledge base according to the control flow dictated by the mechanisms and the task decomposition. Perhaps even more important, those same notions can be the foundation for complete development environments that support all facets of expert-system construction from mechanisms. In the next section, we describe such an environment: PROTÉGÉ-II, a system under development at our laboratory.

3. Overview of PROTÉGÉ-II

The PROTÉGÉ-II environment [Puerta et al., 1992; Tu et al., 1992] is a descendant of PROTÉGÉ [Musen, 1989], which was a knowledge-acquisition tool founded on a single problem-solving method, but that did not incorporate the concept of a mechanism. PROTÉGÉ-II is designed

primarily for use by knowledge engineers; it provides for them the necessary tools to build knowledge bases and problem solvers. It includes the following functionality:

1. Library facilities for methods and mechanisms
2. Method configuration
3. Generation of knowledge-acquisition tools
4. Knowledge editing

Figure 2 depicts an overview of PROTÉGÉ-II with its four main subcomponents: the library subsystem, the user-interface management system, the whiteboard, and the application (i.e, the resulting advice system).

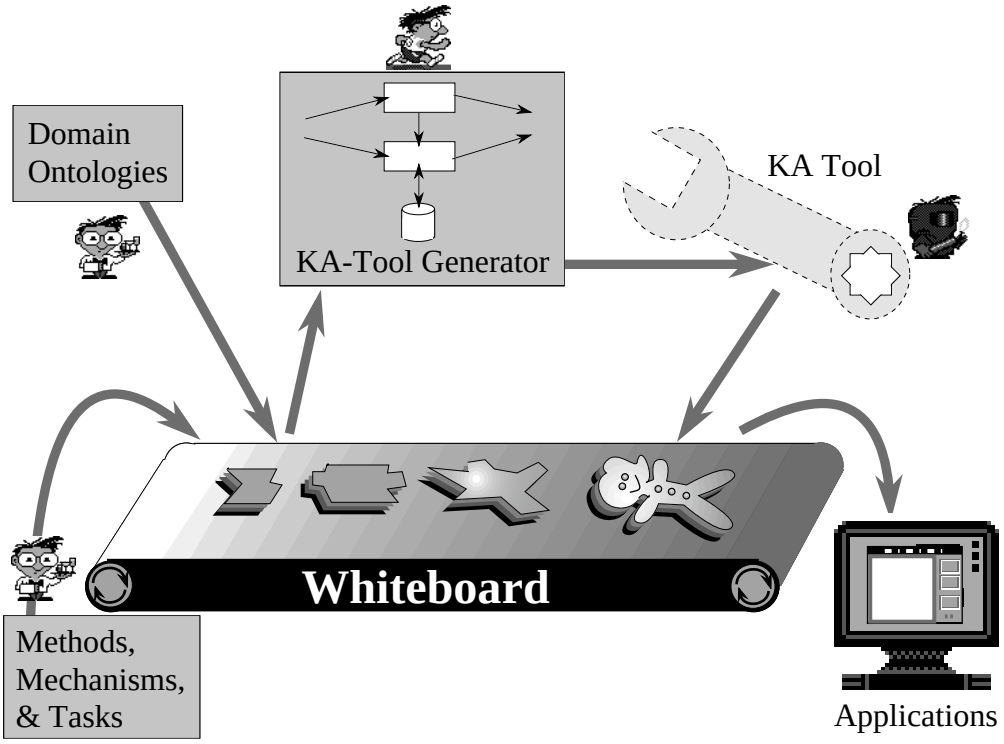

Figure 2. An overview of PROTÉGÉ-II. The system includes library-like storage for methods, mechanisms, and domain ontologies; a user-interface management system that controls the generation of knowledge-acquisition tools, and the use of such tools by domain experts; and a whiteboard that serves as a central repository for configured methods and acquired domain knowledge. The resulting application uses a common inference engine that operates on the knowledge stored in the whiteboard.

The whiteboard acts as a central repository for the whole system. It contains the configured problem-solving method as well as the knowledge base, and it is accessed by all components. The whiteboard is so named to distinguish it from the traditional view of a blackboard [Nii, 1986]. As in the blackboard architecture, the subsystems in PROTÉGÉ-II contribute incrementally to the whiteboard in building a solution (i.e., a problem solver). In contrast to blackboard systems, however, the system components cannot access the whiteboard opportunistically. Instead, access

must follow a fixed order of events. This sequence of actions is as follows: (1) method selection, (2) method configuration, (3) task modeling, (4) knowledge-acquisition tool generation, (5) knowledge-base editing, and (6) application-level interaction. The first four actions are performed by a knowledge engineer, the acquisition and editing of knowledge is done by a domain expert, and the interaction at the application level involves the end user of the expert system.

Method selection, method configuration, and task modeling are conducted within the library subsystem. The library is a storage–retrieval tool for methods and mechanisms. The goal of this subsystem is to supply the knowledge engineer with search utilities that facilitate the selection of a preassembled problem-solving method, and to direct the assignment of individual mechanisms to subtasks of the selected method (i.e., method configuration). The selection and configuration of a method rely on the skills of the knowledge engineer to interpret the information available in the library about each method, and on the analysis of the task that the engineer performs. This task analysis may include, among other activities, consultations with domain experts, and identification of the inputs and outputs of the task for which an advice system is to be developed with PROTÉGÉ-II.

By definition, problem-solving methods, configured or not, are domain independent. Therefore, to be applied to a specific task, a method must be specialized for the domain of interest. This process is known as *task modeling* in PROTÉGÉ-II. It embodies the definition of domain terms in the format dictated by the configured problem-solving method (e.g., as a hierarchy of terms, or as frames). For example, in the medical domain, terms such as *patients*, *treatments*, and *medications* are described as a hierarchy for the skeletal-plan refinement method of the PROTÉGÉ-II library.

It is because of the need for task modeling that the library subsystem also includes storage of domain ontologies. These ontologies are structured representations of domain terms, and their interrelationships, that can be edited by the knowledge engineer to fit given configured methods, thereby minimizing the work required to specialize a method. Note that task modeling is the factor limiting how general problem-solving methods can be Making a method more general implies more complex task modeling for specialization, thus reducing the usefulness of the method itself despite its applicability to a larger class of tasks.

After task modeling, the whiteboard contains a configured method and an edited domain ontology. Those two elements are sufficient for the inference engine to control the reasoning process of the advice system. However, the knowledge required by such process is still to be acquired and placed on the whiteboard. Under PROTÉGÉ-II, the knowledge engineer is not responsible for eliciting such domain knowledge from domain experts. Instead, domain experts supply knowledge directly through a knowledge-acquisition tool. Because it would be extremely difficult to implement a knowledge-acquisition tool useful for every possible domain, PROTÉGÉ-II generates a custom-tailored tool for each domain. A user-interface management system called Mecano [Puerta, Egar, and Musen, 1991] controls the generation of the knowledge-acquisition tools, the use of these tools by domain experts, and the storage of the acquired knowledge in the whiteboard.

To generate a new knowledge-acquisition tool, the utilities in Mecano examine the structure of the edited domain ontology that resides on the whiteboard, identify which domain terms must be represented in the generated tool, and select interaction styles for each identified term. Currently, there are two interaction styles supported in PROTÉGÉ-II: forms and graphical editors. We have chosen to implement these two styles first because they are most relevant to applications in the

medical domain. Forms are a natural medium for expert physicians to enter facts (e.g., symptoms, laboratory-test results) into a knowledge base, and graphical editors are appropriate for procedural knowledge (e.g., the flowchart of a treatment plan). Once a tool is generated, Mecano manages the editing session with the domain expert, and translates the acquired knowledge from the input format of the tool to the format required for storage in the whiteboard. For example, in the case of a flowchart, the translation would involve taking a graph, converting it into a textual description, and compiling that description into the target code of the whiteboard [Egar, Puerta, and Musen, 1992].

When the editing of the knowledge base is complete, the advice system is ready to be run by an end user. In the next section, we shall illustrate the use of PROTÉGÉ-II with an example from the medical domain.

4. An AIDS-Protocol Advisor

The final stage for the development of new medications is the performance of clinical trials in which patients receive either the new drug or a standard therapy—in some cases, simply a placebo. By comparing the response of patients to whom the new drug was administered with those to whom the standard therapy was administered, researchers can draw conclusions about the effectiveness of the new drug. The treatment plan that specifies how each patient enrolled in the clinical trial is cared for is detailed in a document called a *protocol.* This document, which can normally be represented as a flowchart, often is complicated. It must detail the symptomatology that makes a patient eligible to be enrolled in the clinical trial, the procedure by which patients are assigned to the standard therapy or to the actual drug, the dosages, and the changes to the treatment plan that must be made when the patient has adverse reactions to the treatment. Protocols are normally written by a committee of physicians and administered at various hospitals by many different physicians. Often, however, the administering physicians have difficulty understanding the instructions given in the protocol; as a result, they may make variations to the treatment plan that can render the results of the clinical trial invalid. Given that administering physicians generally cannot readily consult the authors of the protocol, they would find useful an expert system that could provide advice on protocols.

In developing an expert system for clinical trials with PROTÉGÉ-II, the first step is to formalize the task in terms of inputs an outputs. We will designate the task of advising a physician regarding a protocol as *clinical-trial management.* Such a task will take as inputs a treatment plan (i.e., a protocol), the case data for a patient, and the current time, and will produce as output the next action to be taken in the treatment plan for the given patient. The second step is to find a method in the library that is applicable to the defined task. One of the indexing schemes in this library allows the retrieval of methods that match the input–output characteristics of the task. In our case, one of the methods that can be retrieved in this manner is *episodic* skeletal-plan refinement, a version of skeletal-plan refinement that accepts temporal data, as is required in our problem. This method receives the following inputs:

1. A hierarchy of planning entities
2. A set of plan data
3. A time value

It delivers a single output:

1. A fully detailed solution plan for the given time value

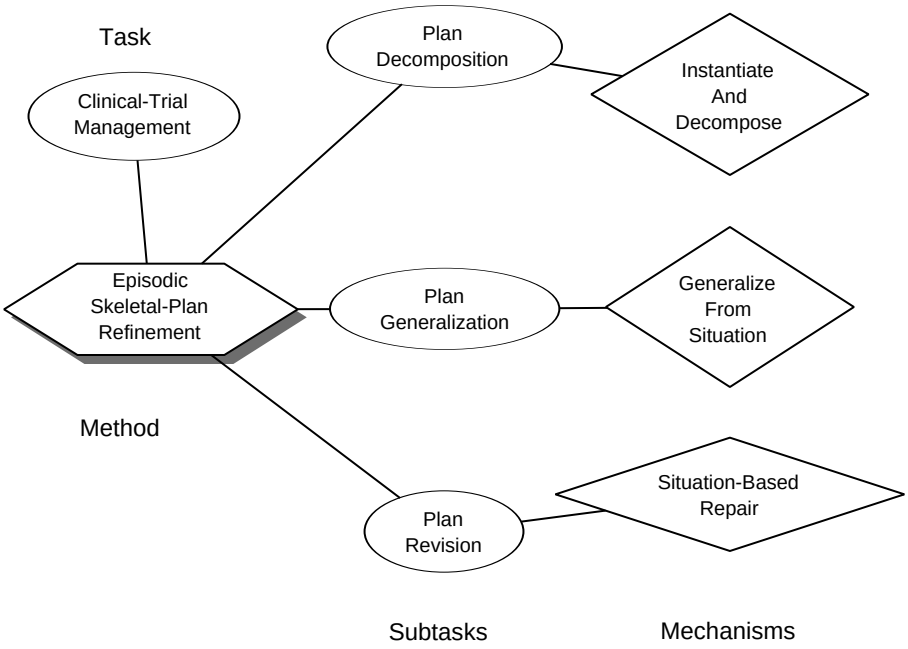

Figure 3. Method configuration for the AIDS-protocol advisor. Episodic skeletal plan refinement decomposes clinical-trial management into the three subtasks shown. Three mechanisms have been selected from the PROTÉGÉ-II library to perform the subtasks. The selected method iteratively decomposes a plan into subplans and, based on the plan data, generalizes these subplans and revises the generalizations made.

The last two inputs and the output of the method map easily to the respective inputs and outputs of the clinical-trial management task. The first input is a little trickier because episodic skeletal-plan refinement does not input plans directly. It is in these situations that the skills of the knowledge engineer become crucial in determining the applicability of the selected method. In this case, the method inputs a hierarchy of primitives (i.e., planning entities) with which plans can be constructed, thus creating a knowledge-acquisition requirement. The knowledge engineer must ascertain that it will be possible, through task modeling, to edit a domain ontology from which an appropriate knowledge-acquisition tool can be generated. A protocol author should be able to use the generated tool to put together the protocols that satisfy the knowledge-acquisition requirement of the episodic skeletal-plan refinement method.

Task modeling proceeds after the selected method is configured. Figure 3 shows the configured method for our example. Through a process of search, selection, and evaluation—similar to that done for the method—mechanisms are applied to each of the three subtasks stipulated by the method. Now the goal of the knowledge engineer is to identify primitives in the domain of AIDS therapy, and to structure these primitives into a hierarchy as required by episodic skeletal-plan refinement. A three-level tree, as shown in Figure 4, suffices in this instance. Based on the edited domain ontology, and on the fact that procedural knowledge is best acquired graphically, the knowledge-acquisition tool shown in Figure 5 is generated. Protocols are entered through this tool into the knowledge base enclosed in the whiteboard. After the knowledge base is complete, the advice system is ready to be run by the administering physicians.

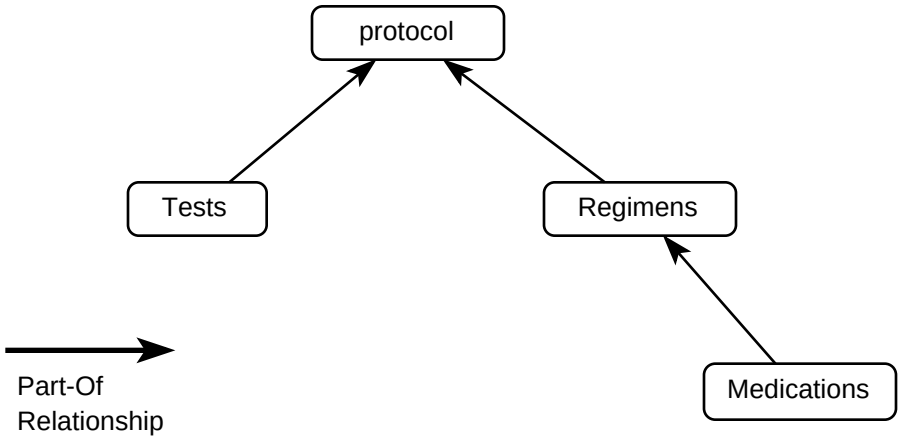

Figure 4. The domain ontology for AIDS therapy under the episodic skeletal-plan refinement method. The hierarchy states that AIDS protocols include *tests* and *regimens*, and that each regimen includes the administration of various *medications.*

5. Discussion

Knowledge engineers are the principal developers of knowledge-based systems, particularly of expert systems. Often, however, their work concentrates on implementational details of these systems, such as control flow, knowledge representation, and reasoning strategies. They put far less effort into actually manipulating knowledge as an abstract entity. Furthermore, considerable duplication of work takes place because these systems are useful generally for a single task, and cannot be adapted easily to different domains.

A second generation of expert systems is now being studied by researchers. It will focus more on how knowledge is used as opposed to how knowledge is implemented. These systems will reduce the duplication of effort by defining common frameworks from which expert systems are derived. These frameworks are based on ideas such as *generic tasks* [Chandrasekaran, 1986] and *problem-solving methods* [McDermott, 1988], which have spawned several development environments for expert systems [Puerta, Tu, and Musen, 1992; Steels, 1990; Marques et al., 1992]. PROTÉGÉ-II is one of such environments that uses a unit of knowledge, called a mechanism, to build problem-solving methods, to generate knowledge-acquisition tools, and to produce problem solvers for a potentially unlimited number of domains.

The benefits of using mechanisms as a basis for expert-system construction are manyfold. First, mechanisms provide a simple way to compose problem-solving methods. Second, as used in PROTÉGÉ-II, mechanisms create a structured representation of domain terms from which knowledge-acquisition tools can be generated automatically. Because these tools are geared for use by domain experts, they free knowledge engineers from certain elicitation tasks that must be normally performed. Third, by embodying their own control-flow configuration, mechanisms allow the development of a generic inference engine that can be applied to any problem solver built from mechanisms, thereby shielding the knowledge engineer from several of the problems of implementation of a reasoning strategy.

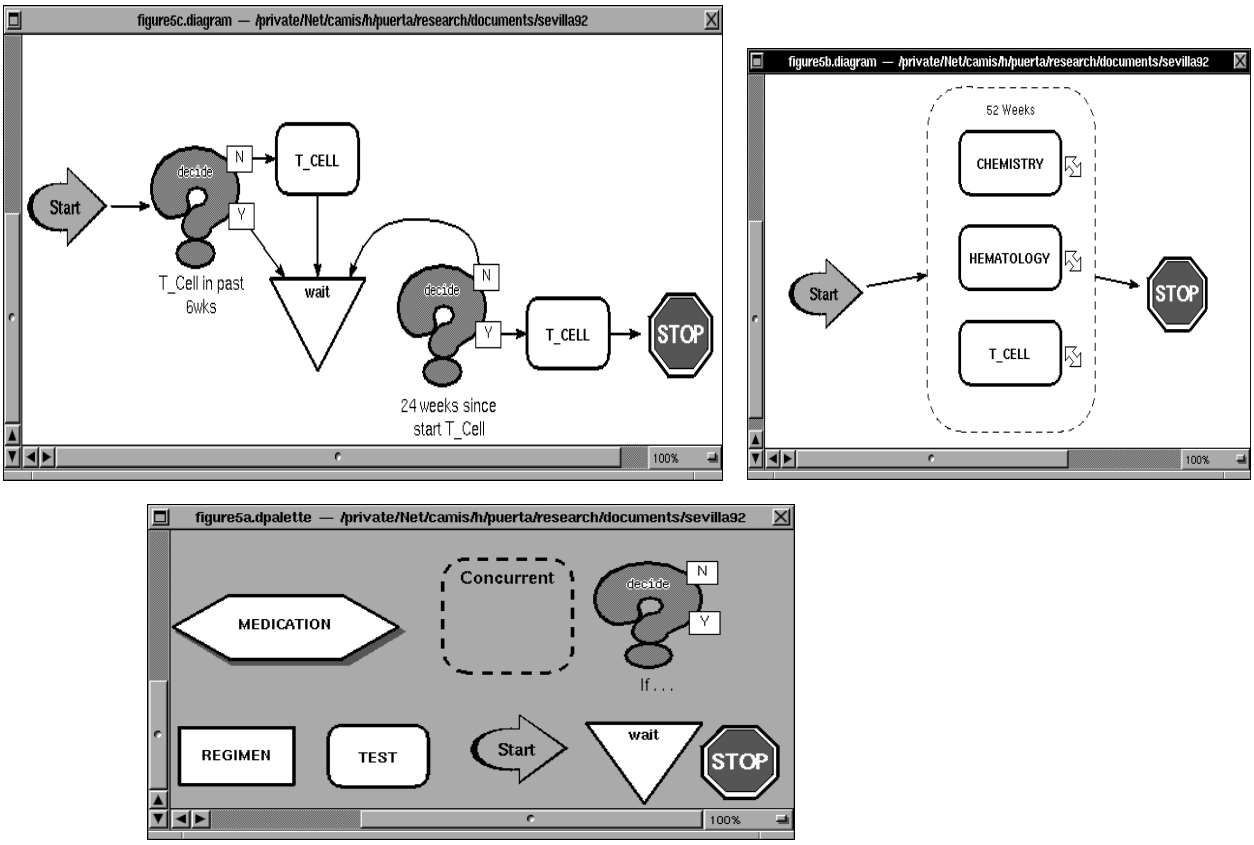

Figure 5. A knowledge-acquisition session with a domain expert. The generated knowledge-acquisition tool provides a palette of graphical elements that the protocol author uses to draw the protocol flowcharts. The screens shown detail the procedure to follow to administer tests to determine T-cell counts on the patient.

PROTÉGÉ-II provides a library of problem-solving methods, preassembled from mechanisms, that allows knowledge engineers to redefine expert-system development as an activity that requires a careful search for appropriate methods for the given problem, and the specialization of the selected method to the domain of interest. The library saves the PROTÉGÉ-II users from the difficult task of assembling problem-solving methods. Based on studies of expert systems previously implemented in our laboratory, we are proceeding with the assembly of methods and the definition of mechanisms for the PROTÉGÉ-II library.

Although the assembly of methods clearly remains a challenge, we have gained considerable experience through our use of PROTÉGÉ-II. That experience makes us optimistic about the future of development environments using mechanisms. Clearly, we need to continue building up the PROTÉGÉ-II library, and to solve a wider range of problems. Nevertheless, we have demonstrated that PROTÉGÉ-II has the potential to allow knowledge engineers to reach one of their most desired goals: to engineer knowledge rather than software.

Acknowledgments

This work has been supported in part by grant LM05157 from the National Library of Medicine, by grant HS06330 from the Agency for Health Care Policy and Research, and by a gift from

Digital Equipment Corporation. Computer support was provided in part by the CAMIS resource, supported by grant LM05305 from the National Library of Medicine.

We thank John Egar for preparing the artwork, and Lyn Dupré for editing a previous version of this document. We recognize the contributions of John Egar, Henrik Eriksson, and Yuval Shahar to the architecture of PROTÉGÉ-II.

References

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.

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

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

Egar, J.W., Puerta, A.R., and Musen, M.A. (In press). Graph-grammar assistance for modeling of decisions. In *Proceedings of the Seventh Banff Knowledge-Acquisition for Knowledge-Based Systems Workshop,* Boose, J. H., and Gaines, B. R., editors. Banff, Alberta, Canada.

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

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., and McDermott, J. (1991). Usable and reusable programming constructs. K*nowledge Acquisition,* **3**, 117–135.

Marcus, S. and McDermott, J. (1989). SALT: A knowledge acquisition tool for propose-and-revise systems. *Artificial Intelligence*, **39**, 1–37.

Marques, D., Dallemagne, G., Klinker, G., McDermott, J., and Tung, D. (1992). Easy programming: Empowering people to build their own applications. *IEEE Expert,* 7(3), 16–29.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems*, Marcus S., editor, pp. 225–256. Boston: Kluwer Academic.

Musen, M.A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*. London: Pitman.

Nii, H.P. (1986). Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine,* **7**(2), 38–53.

Puerta, A.R., Egar, J.W., and Musen, M.A. (1991). *Automated Generation of Adaptable Knowledge-Acquisition Tools with Mecano.* Report No. KSL–91–62, Knowledge Systems Laboratory, Stanford University, October 1991.

Puerta, A.R., Egar, J.W., Tu, S.W., and Musen, M.A. (1992). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge Acquisition,* **4**, 171–196.

Steels, L. (1990). Components of expertise. *AI Magazine* **11**(2), 30–49.

Tu, S.W., Shahar, Y., Dawes, J., Winkles, J., Puerta, A.R., and Musen, M.A. (1992). A problem-solving model for episodic skeletal-plan refinement. *Knowledge Acquisition,* **4**, 197–216.