Flexible Knowledge Acquisition Through Explicit Representation of Knowledge Roles

Bill Swartout
Yolanda Gil

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
swartout@isi.edu, gil@isi.edu

Abstract

A system that acquires knowledge from a user should be able to reflect upon the knowledge that it has—at each moment—and understand what kinds of new knowledge it needs to learn. For the past two decades, research in the area of knowledge acquisition has been moving towards systems that have access to richer representations of knowledge about their task. This paper reviews some well-known knowledge acquisition tools representative of this trend. It also describes our recent work in EXPECT, a system with explicit representations of knowledge about the task and the domain that supports knowledge acquisition for a wider range of tasks and applications than its predecessors. We hope our observations will be useful to researchers in user interfaces and in machine learning concerned with acquiring information from users.

Introduction

The acquisition of knowledge about a task can be viewed as a process of incorporating new knowledge into some existing knowledge structure [Rosenbloom 1988]. The existing knowledge can guide and constrain the search for new knowledge, and the process of integrating the new knowledge with the old may identify additional opportunities for learning. An acquisition system that takes this view needs to represent and understand the knowledge about the task as well as the process of finding and integrating new knowledge.

A general trend in research on knowledge acquisition has been to make the knowledge structures that guide acquisition increasingly explicit. In early acquisition tools many of the requirements that needed to be satisfied when adding a new piece of knowledge were not stated. The result was that they could not provide very precise guidance for acquisition. Later, more of these requirements were made explicit, but they were embedded within the acquisition tools themselves. This allowed the tools to provide more specific guidance, but the requirements that were embedded in the tools could not be changed, which meant that some important aspects of the knowledge-based system being built could not be changed. More recent work has focused on making these requirements explicit and represented outside of the tool itself. A result of this has been more flexible acquisition tools that allow users to make a greater variety of changes to the knowledge-based system being built.

To make this more concrete, we will begin by briefly reviewing three well-known acquisition systems: TEIREISIAS [Davis 1976], SALT [Marcus 1988], PROTÉGÉ-II [Musen and Tu 1993]. We will then describe EXPECT, the acquisition framework we have been developing [Swartout and Gil 1995, Gil 1994, Gil and Paris 1994]. Each of these four systems represents a point along the trend we outlined above. We conclude with a discussion about how these systems make the knowledge structures they use for acquisition more or less explicit, and summarize the implications of these differences in terms of the kinds of knowledge acquisition they can support.

Symbol-Level Approaches

TEIREISIAS [Davis 1976] was one of the earliest knowledge acquisition systems. It was designed to help a user correct and extend MYCIN's knowledge base [Buchanan and Shortliffe 1984] for diagnosing infections. If MYCIN either incorrectly concluded that a disease was present, or missed a correct diagnosis, TEIREISIAS would walk the user through the trace of rule firings to determine where the error arose. If an incorrect diagnosis was concluded, TEIREISIAS would display the rule that led to the conclusion, and ask if any of the conditions on the rule needed to be changed. If so, the user was given the

opportunity to make the change. If the rule was correct, but fired because some if its conditions were incorrectly asserted, the process would recurse and the user could look at the rules that made those assertions to determine if they were correct. Similarly, if a correct diagnosis was missed, TEIREISIAS would display the rule that could have caused that diagnosis to be reached, and would walk the user through the process of determining why those rules did not fire and correcting the problem, either by changing the conditions on the rules or by adding additional rules. When a new rule was added, TEIREISIAS provided guidance based on rule models derived from the existing rule base through statistical conceptual clustering. If rules that concluded about a particular parameter *x* frequently mentioned some other parameter *y* in their antecedents, then TEIREISIAS would point out a possible error if the user left the parameter *y* out in a rule concluding about *x*. This is useful, but a long way from actually understanding the role that the rule plays in problem solving.

TEIREISIAS understood MYCIN at the symbol level [Newell 1982]. It understood rule patterns and why a particular rule fired or did not fire, but it did not have a global view of the overall algorithm that MYCIN was following. Indeed, work on TEIREISIAS pre-dated Clancey's analysis of MYCIN that showed that it was following the general problem solving strategy that he identified as heuristic classification [Clancey 1985]. TEIREISIAS did not capture the distinctions between rules for data abstraction, heuristic match, and solution refinement that charaterize a heuristic classification system. Since TEIREISIAS did not understand the roles that the rules played in a system, it could not provide much help in guiding the user concerning the content of those rules.

Role Limiting Approaches

The next generation of knowledge acquistion tools represents what is called a role-limiting approach. It was based on the observation that the kind of problem solving method that a system uses determines the kind of domain information the system needs [McDermott 1988]. Put another way, the role that a particular kind of knowledge plays in problem solving strongly constrains how that knowledge should be expressed — what is required for the system to function. A research goal during this stage was to try to understand a number of general methods used by knowledge based systems [Chandrasekaran, 1986], such as propose-and-revise and heuristic classification, and then to construct a knowledge acquisition tool for a particular method. Such tools could then be used to build knowledge based systems that used that particular problem solving approach. Examples of such tools include MORE [Eshelman 1988], SALT [Marcus 1988], and ROGET [Bennett 1985]. Because these tools understood how knowledge would be used in problem solving, they could provide much more guidance in helping the user formulate the knowledge he was adding to a system correctly.

SALT is a good example of a knowledge acquisition tool based on a role-limiting method. SALT was used to develop systems that use the propose-and-revise method to construct a solution to a problem. Examples included a configurer for elevators and a flow-shop scheduler. In the propose-and-revise method, a system constructs an initial approximate solution to a problem which may have a number of aspects that still remain to be determined. It then proposes a design extension to fill in missing parts of the design, and looks for possible constraints that may be violated. If a constraint violation is detected, the system has knowledge of fixes that may be used to revise the solution and correct the problem. SALT captures this general algorithm, and understands that the knowledge that needs to be acquired to support this method is knowledge of ways of extending a design, the constraints that the design must satisfy, and ways of correcting constraint violations. In other words, these are the roles that new knowledge plays within the propose-and-revise strategy. Because SALT was specifically designed for building propose-and-revise systems, it includes tools that detect and correct problems that can arise in building these systems, such as cycles in the fixes and constraints.

Role-limiting approaches center knowledge base construction on filling the roles that knowledge plays in the particular problem-solving method that they are designed for. However, the problem solving method is built into the knowledge acquisition tool itself. Thus, when one selects a tool, one also determines the problem-solving method that is employed by the application. One problem with this is that many large-scale systems are not homogeneous. That is, they do not use a single problem-solving method throughout—some of the application can use one technique but different techniques are needed for other parts. As a result, an acquisition tool that only supports a single method has limited applicability [Musen 1992].

Composable Role-Limiting Methods

Rather than embodying a single problem solving method, some knowledge acquisition environments contain a library of problem solving methods of smaller size. New applications are built by composing the overall problem-solving strategy from

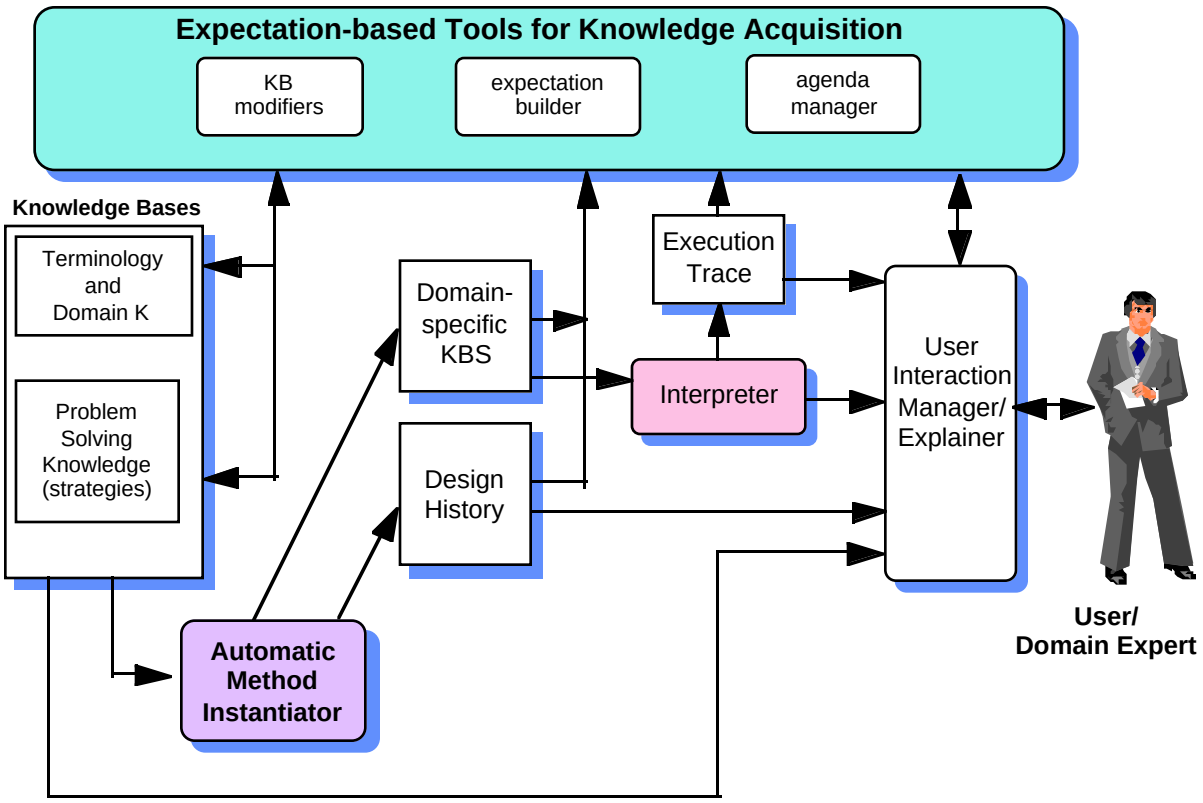

Figure 1: EXPECT Architecture

the smaller components of the library. Examples of systems that take this approach are SBF [Klinker et al. 1991], COMET [Steels 1990], and PROTÉGÉ-II [Musen and Tu 1993].

PROTÉGÉ-II guides acquisition based on fine-grained role-limiting methods. To build an application, a knowledge engineer configures the overall problem-~solving method using the components in the method library. At the same time, he or she builds a method ontology that contains the terms that are used by the method being configured. For example, "constraint" would be one of these terms and it would be identified as one of the kinds of knowledge needed by the component that represents the revision stage of propose-and-revise. A domain ontology that contains domain-specific knowledge is represented separately. The knowledge engineer then links the knowledge roles in the method ontology to the domain ontology, identifying how terms like "constraint" map onto domain-specific terms. Once this mapping has been done, an automatic interface builder uses the mapping to generate a knowledge acquisition tool that allows an end user to enter domain-specific knowledge about the domain. Through this mapping the system understands just how domain-specific knowledge will actually be used, and the interface that is constructed thus requests the knowledge that will actually be needed for problem solving.

Flexibility through Derived Interdependencies

A remaining problem is that while acquisition systems allow a user to make changes to a system's factual knowledge, they do not allow the user to make changes to the problem solving methods that the system employs. Tools like SALT don't allow these changes because the problem solving method is implicitly encoded in the tool itself. PROTÉGÉ-II doesn't allow such changes because its methods are pre-configured and the mapping between method and domain ontologies is fixed at system design time. Since PROTÉGÉ-II's knowledge acquisition tool is derived from that mapping, it too is fixed at design time. The idea in EXPECT [Swartout and Gil 1995, Gil 1994, Gil and Paris 1994] is to derive the interdependencies between domain knowledge and problem solving methods automatically, and to be able to re-derive the dependencies as needed when changes are made to the problem solving knowledge. This approach allows a user to modify either the domain knowledge or the problem solving knowledge. We have used EXPECT to create systems in several domains. One of these domains is

military transportation planning, where EXPECT was used to construct a system to evaluate transportation plans. We will use that domain to illustrate points in this paper.

A diagram of the architecture is shown in Figure 1. Starting at the lower left in the figure, EXPECT's knowledge bases separate out factual knowledge and problem solving knowledge. Problem solving knowledge in EXPECT consists of a mix of general, domain independent strategies as well as some that are domain specific. EXPECT's factual knowledge includes facts about the domain and terminology that describes the domain, as well as domain-independent terms that are used by problem-solving methods. In EXPECT, each problem solving method has a *capability description* that describes what the method can do. Each method also has a body, which is a step or sequence of steps for achieving the method's capability. EXPECT's method language supports conditionals, sequences of steps and embedded steps in method bodies.

The automatic method instantiator uses partial evaluation and reformulation (see [Swartout and Gil 1995]) to derive the interdependencies between the problem solving methods and domain knowledge that are needed to guide knowledge acquisition. The method instantiator starts with a high level goal that specifies a *class* of problems for which one would like to create a system. For example, one might specify that one wanted to create a system to evaluate a transportation plan from the perspective of logistics. This high-level problem description contains "generic instances" which are placeholders for actual data that will be used when solving specific problems. Starting with this goal, the method instantiator searches the method library for a method whose capability matches the goal. Among those that match, the most specific one is chosen, its method body is instantiated, and the generic instances replace variables in the method. If the method body contains any subgoals, these are recursively expanded and the process continues.

During method instantiation, the system also records how domain relations and concepts are being used by the various problem solving methods. Figure 2 shows this linkage between the domain knowledge and the expansion of the problem solving methods. In the figure, the location of a seaport is used in one of the steps in the expanded method tree and berths of a seaport is used in another. Notice that there are some relations that are defined by the domain concepts but they are not used in any of the problem solving steps, such as *piers.* We expect that it will often be the case that domain ontologies will contain some relations that are useful for some problem solving methods but not for others. In particular, this would occur if we wanted to re-use a domain ontology to support a different sort of problem solving. For example, one might want to use the same domain ontology in the related (but different) problem of scheduling transportation movements. Much of the domain information needed by the two applications would be similar, but some would be different. EXPECT can support such re-use because its knowledge acquisition routines can focus acquisition on just the information that is actually needed to support the particular problem solving methods in use.

EXPECT also allows a user to modify problem solving methods. For example, in some transportation planning situations, one might want to take into account additional resources that might be available. For example, in figuring out the throughput of a particular location, one might want to take into account not only the capacity of its seaports, but also of its marinas. In EXPECT, that could be done by modifying the method that finds the seaports of a location so that it finds the marinas as well. When that modification is made, EXPECT would re-derive the dependency structure shown in Figure 2, and it would determine that information about the marinas of a location was no longer optional, but required. Accordingly, EXPECT would ask the user for information about the marinas for all the locations where information about marinas had not been specified, like Los Angeles.

Summary

Table 1 summarizes the major issues that we have pointed out in this paper. As the knowledge structures we use for learning and acquisition become richer and more explicit, our acquisition tools can support a broader range of problem solving methods, provide more guidance to the user and help the user make a wider variety of changes to a system.

Acknowledgements

We gratefully acknowledge the support of ARPA with the contract DABT63-95-C-0059 as part of the ARPA/Rome Laboratory Planning Initiative. The view and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ARPA or the U.S. Government.

	TEIREISIAS	SALT	PROTÉGÉ-II	EXPECT
Composing the overall problem-solving strategy	no problem-solving strategy expressed, rule chaining determines strategy	manually by KA tool's designers	manually by KBS builder by selecting methods from library	automatically assembled by method instantiator
Representation of problem-solving strategy	no problem-solving strategy expressed	hard-coded in KA tool	explicit structure manually created by KBS builder	explicit structure automatically derived
Abstraction	rule templates statistically derived from existing rule base	model of problem solving (i.e., propose-and-revise), manually designed, hard-coded in KA tool	library components correspond to models of problem-solving, manually designed, hard-coded in component's KA tool	generic problem-solving methods, manually designed, and explicitly represented
When are interdependencies factual knowledge and problem-solving methods determined?	when rule statistics are gathered	when KA tool is created	when application KBS is created	whenever the application is modified
Derivation of interdependencies between domain-dependent knowledge and problem-solving knowledge	through predicates and rule models	domain-dependent knowledge corresponds to pre-determined knowledge roles of the problem-solving method	correspondences between domain ontology and method ontology are manually specified	one single ontology represents correspondences
Range of problem solving methods supported	problems solveable by backward chaining rule-based system	propose and revise	methods that can be composed from the methods in library	methods that can be expressed in EXPECT's method language
Modifications Supported	rule modification & addition	add new data (design extensions, constraints, fixes) used by propose and revise	add/modify factual information used by problem solving methods in library	add/modify factual information; add/modify problem solving methods

Table 1: Comparing knowledge acquisition tools

References

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

[Buchanan and Shortliffe 1984] Buchanan, B. and Shortliffe, E. *"Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, 1984.

[Chandrasekaran, 1986] B. Chandrasekaran. Generic tasks in knowledge-based reasoning. *IEEE Expert* , 1(3):23-30, 1986.

[Clancey 1985] Clancey, W.J., "Heuristic classification" *Artificial Intelligence*, 27(3):289-350, 1985.

[Davis 1976] Davis, R. Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases, Phd Thesis, Stanford University, 1976 (also available as SAIL AIM-283.

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

[Gil 1994] Gil, Y. Knowledge Refinement in a Reflective Architecture. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-94)*, 1994.

[Gil and Paris 1994] Gil, Y., and Paris, C.L. Towards Method-Independent Knowledge Acquisition. In *Knowledge Acquisition*, 6 (2), pp. 163-178, 1994.

[Klinker et al. 1991] Klinker, G., Bhola, C., Dallemagne, G., Marques, D., and McDermott, J. "Usable and reusable programming constructs," *Knowledge Acquisition*, 3(2):117--135. 1991.

[Marcus 1988] Marcus, S. "SALT: a knowledge-acquisition tool for propose-and-revise systems," in *Automating Knowledge-acquisition for expert systems* S.Marcus (ed), pp. 81-121. Kluwer Academic Publishing. 1988

[McDermott 1988] McDermott, J, "Preliminary steps toward a taxonomy of problem solving methods," in *Automating Knowledge-acquisition for expert systems* S.Marcus (ed), Kluwer Academic Publishing. 1988

[Musen 1992] Musen, M. A. "Overcoming the limitations of role-limiting methods," *Knowledge Acquisition*, 4(2):165--170. 1992.

[Musen and Tu 1993] Musen, M. A., and Tu, S. W. Problem-solving models for generation of task-specific knowledge acquisition tools. In J. Cuena (Ed.), *Knowledge-Oriented Software Design*, Elsevier, Amsterdam, 1993.

[Newell 1982] Newell, A. "The knowledge level," *Artificial Intelligence*, 18(1):87--127. 1982.

[Rosenbloom 1988] Rosenbloom, P.S., "Beyond generalization as search: Towards a unified framework for the acquisition of new knowledge," in G.F. DeJong (ed.) Proceedings of the AAAI Symposium on Explanation-Based Learning, pp. 17-21, Stanford, CA. 1988.

[Swartout and Gil 1995] Swartout, B. and Gil, Y. "EXPECT: Explicit Representations for Flexible Acquisition" in Proceedings of the Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'95) Banff, Canada, February 26-March 3, 1995.