Towards Method-Independent Knowledge Acquisition

Yolanda Gil and Cécile Paris

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
U.S.A.
email: gil@isi.edu, paris@isi.edu

Abstract

Rapid prototyping and tool reusability have pushed knowledge acquisition research to investigate method-specific knowledge acquisition tools appropriate for predetermined problem-solving methods. We believe that method-dependent knowledge acquisition is not the only approach. The aim of our research is to develop powerful yet versatile machine learning mechanisms that can be incorporated into general-purpose but practical knowledge acquisition tools. This paper shows through examples the practical advantages of this approach. In particular, we illustrate how existing knowledge can be used to facilitate knowledge acquisition through analogy mechanisms within a domain and across domains. Our sample knowledge acquisition dialogues with a domain expert illustrate which parts of the process are addressed by the human and which parts are automated by the tool, in a synergistic cooperation for knowledge-base extension and refinement. The paper also describes briefly the EXPECT problem-solving architecture that facilitates this approach to knowledge acquisition.

Keywords: knowledge acquisition; knowledge-base refinement; learning by analogy; explanation.

1 Introduction

The science of artificial intelligence started with and continues to build on general principles. Newell and Simon's pioneering work on understanding human problem solving shows us that the same mechanisms come to play when trying to find solutions to problems that may seem very different [Newell and Simon, 1972]. GPS implemented this principle and demonstrated that, given domain-specific knowledge, a general-purpose reasoning engine can be used to solve problems in that specific domain, and that the same engine can be used for different problem-solving applications. Decades later, a constellation of expert systems was possible thanks to the expert system shell version of GPS (e.g., [Buchanan and Shortliffe, 1984]). Inference is understood, said common AI wisdom, the hard part is getting the knowledge right. General-purpose inference mechanisms, general-purpose representation mechanisms (a similar argument can be shown with Minsky's frames for the field of knowledge representation) gave AI, like any other science, the beauty of building on principles. Researchers turned then to the next issue in the automated reasoning agenda: machine learning and the acquisition of knowledge. The work ranges from learning heuristic knowledge [Minton, 1988; Knoblock, 1993] to acquiring factual knowledge [Gil, 1992], and shows us that complete automation of the learning or acquisition task in domains of technical expertise is far from reach. Semi-automated general-purpose knowledge acquisition (KA) tools [Davis, 1980] are not enough to achieve efficient prototyping. Generality does not provide the tool with much basis to support the knowledge engineering process.

At the same time, we witnessed the surge of special-purpose mechanisms to support real-world applications. General-purpose tools seemed incapable of performing well in applications such as manufacturing or robot path planning [Lozano-Perez, 1987; Chang and Wysk, 1985]. The new common AI wisdom reflects that general-purpose is good, but may not be enough as we currently understand it. The field of knowledge acquisition has rightly taken this lesson. General-purpose KA tools are augmented with application-independent method-specific inference structures [Chandrasekaran, 1986; McDermott, 1988]. For example, SALT [Marcus and McDermott, 1989] provides support for propose-and-revise tasks. PROTEGE [Musen, 1989] helps in acquiring knowledge in temporal skeletal planning applications. Although the resulting tools have less generality, they allow the semi-automated production of fast prototypes in novel domains once the appropriate inference mechanism is manually determined. The inference mechanisms provide expectations about the role of each piece of knowledge that are powerful guidance in the knowledge acquisition process.

However, this approach to building KA tools has limitations that arise from the need of more flexibility than they provide in adapting them to an application [Musen, 1992]. The problem-solving structure of an application cannot always be defined in domain-independent terms. Furthermore, these method-specific inference mechanisms may not address some of the particulars of an application simply because they were designed with generality in mind. Another problem with the method-specific KA tools is that they raise the non-trivial issue of determining a library of possible methods [Chandrasekaran, 1986; McDermott, 1988]. The work involves handcrafting this library of methods making sure to provide both wide coverage of tasks and well-understood characterizations of the inference capabilities of each method.

To address these limitations, some researchers [Klinker *et al.*, 1991; Puerta *et al.*, 1992] are developing libraries of problem-solving methods that handle finer-grained inference structures than the ones above. These approaches provide much more flexibility in building a knowledge-based system but two major issues remain. One is the task of building method libraries. Another issue is the accessibility of KA tools to domain experts. The users of these tools still need to be knowledge engineers [Kitto, 1988].

The goal of the EXPECT project is to provide an environment for the development of knowledge-based systems that aids in the acquisition, maintenance, and documentation of the knowledge about a task. Our KA tool is independent of the inference structure or problem-solving method of the task. This paper concentrates on EXPECT as a tool for knowledge refinement and describes our work on how to correct and extend an existing knowledge base. Our tool uses machine learning techniques to support knowledge refinement independently of the inference structure of the task. Our KA work focuses on the integration of machine learning methods in KA tools that can automatically (or semi-automatically) come up with generalized versions of methods that express domain-specific inference structures and support their reuse in new domains and new situations through analogy. This paper shows our first results in this direction and illustrates through examples the practical advantages of this approach. In particular, we show how EXPECT currently uses the same general-purpose mechanism (analogy) to support knowledge refinement within a domain and across domains. The system and the user work in synergy: unlike other machine learning approaches, our system does not attempt to deduce all the information needed to form an analogy by itself. Rather, it relies on input from the user to guide its analogical learning. On the other hand, the user does not need to know a lot about the knowledge base or enter all the new necessary knowledge explicitly, but can rely on the tool to guide the knowledge acquisition process.

The issue of accessibility to domain experts has been central in the design of our architecture [Neches *et al.*, 1985; Swartout *et al.*, 1991; Swartout and Smoliar, 1987]. The explicit representation of factual and problem-solving knowledge and the ability to produce flexible explanations in an interactive dialogue provide the basis for building a KA tool that communicates with an expert much in the way a colleague in their field of expertise would. Our work in this area is not the main focus of this paper, and it is discussed elsewhere [Paris and Gil, 1993; Moore and Paris, 1993].

The examples throughout the paper are from a logistics transportation application that evaluates proposed routings, taking into account restrictions on objects transported, destination points, and vehicles used. Users are specialists in sea transportation planning. EXPECT is given knowledge about ships, seaports, packages, berths, etc., as well as plans for transporting objects. Users are interested in estimating how long it takes to transport some number of objects from one location to another with the resources available.

The paper runs as follows. We first describe briefly our problem-solving architecture and our knowledge representation language. Then we illustrate how EXPECT makes use of analogy to facilitate KA within a domain and across domains. Later, we describe our plans for future work through an example scenario that uses the same analogy techniques augmented with generalization to facilitate KA in a different domain. Our scenarios illustrate the user's involvement during knowledge acquisition, as well as which parts of the process are automated by the tool.

Insert Figure 1 here

Figure 1: A schematic representation of EXPECT. The design history and the knowledge base components are used to form expectations for knowledge acquisition.

2 The EXPECT Architecture

EXPECT builds upon previous work on the Explainable Expert System (EES) framework [Neches *et al.*, 1985; Swartout *et al.*, 1991; Swartout and Smoliar, 1987], which allows for the construction of expert systems that can provide good natural language explanations of their behavior [Moore and Paris, 1993; Swartout and Moore, 1993; Moore, 1989; Moore and Swartout, 1989; Moore and Swartout, 1991]. EXPECT's architecture is shown in Figure 1. In EXPECT, different kinds of knowledge are specified in distinct knowledge bases in a high-level specification language. The specific actions that are to be executed by the system to solve a specific problem are *derived* from these knowledge bases by the system, and a record of the derivation is stored to provide the rationale for these actions. The knowledge acquisition tool uses this rationale and the knowledge bases themselves to form expectations to support the interaction with the user [Gil, 1994]. A natural language generation module produces justifications and explanations of the system's behavior to the user.

The knowledge bases capture what the system knows about the domain and how to solve problems in that domain. They comprise:

- A domain descriptive knowledge base (or *domain model*), which stores definitions and facts in the task domain. The domain model is written in LOOM [MacGregor, 1988; MacGregor, 1991], a knowledge representation formalism of the KL-ONE family. LOOM uses a descriptive logic representation language, and includes a classifier for inference.

- A problem-solving knowledge base, which contains an organized collection of plans. A plan in EXPECT is an abstract and generic description of how a goal can be achieved, instead of a sequence of operators [Fikes and Nilsson, 1971]. The plan language allows for an explicit representation of *intent* (what is to be done to achieve a goal) and supports a wide range of control structures to combine subgoals (e.g., conditionals, variable assignments, iteration).

As an example, consider Figures 2 and 3, which contain samples of these knowledge bases for the logistics transportation domain. In that application, the domain model includes descriptions of `ports`, `seaports`, and `airports`. The representation of a `seaport` is shown in Figure 2: A `seaport` is a type of `port`, and it has attributes such as the ships available at the seaport, its piers, its berths, and the available storage areas.[1]

Figure 3 shows two very simple plans of the problem-solving knowledge base. The capability describes the goals that the plan can achieve, the method represents the body of the plan, and the result type indicates what is returned by the method. The first plan can be used to determine whether a specific type of ship is supported (or "fits in") a given seaport. This is done by testing whether the length of the ship is less than the maximum vessel length allowed in that particular seaport. The second plan finds the maximum ship length a seaport supports based on the length of its berths.

Given a high-level goal, a *Program Writer* integrates these structured knowledge bases by refinement and reformulation from that goal [Neches *et al.*, 1985; Swartout *et al.*, 1991] to produce a system that will be capable of solving specific instances of that goal. The goal is matched against the problem-solving knowledge base. The method of the plan found causes additional subgoals to be posted, and the process is iterated. EXPECT's matcher unifies goals and plans according to the semantic representation of the goal's arguments. For example, a goal to find the distance between two cities matches a plan to find the distance between two locations, since cities are locations. Type constraints are also propagated through the method of plans. The Program Writer also reformulates goals when no plan is found to match them.

The Program Writer records all its steps and decisions in an annotated *design history*. For example, Figure 4 shows the most relevant parts of the design history for the general goal to transport a set of objects to a location using a set of ships. The design history reflects both the goal/subgoal expansion as well as some goal reformulations. For example, the subgoal `determine-whether-fits-in` was reformulated into three goals, one for each type of ship that the system knows of, as indicated in the domain model.

The design history tells the system whether a definition is used and where, what it is used for, and how. The system understands that the type of a ship will require taking a different path in the tree, because there are different procedures to determine if a ship fits in a seaport

[1]We use the prefix "r-" to indicate names of relations.

```
(defconcept seaport
    :is (:and  port
               (:some r-ships ship)
               (:some r-berths berth)
               (:some r-piers number)
               (:some r-covered-storage-area number)))
```

Figure 2: Definition of a seaport in the domain model.

```
(define-plan FIND-IF-SHIP-FITS-IN-SEAPORT
   :capability (determine-whether-fits-in (OBJ (?s is (inst-of ship)))
                                          (IN (?p is (inst-of seaport))))
   :result-type boolean
   :method (less-than (r-ship-length ?s)
                      (compute-max-vessel-length-in-seaport ?p)))


(define-plan COMPUTE-MAX-VESSEL-LENGTH-IN-SEAPORT
   :capability (compute-max-vessel-length-in-seaport
                                   (OBJ (?s is (inst-of seaport))))
   :result-type number
   :method (let ((?berth-types (r-berth-type (r-berth-availability ?s))))
             (max (r-berth-length ?berth-types))))
```

Figure 3: Two plans from the transportation domain.

depending on what type of ship it is. The same can be said about each plan: the design history records whether they are used and where, for what goals, and with what arguments. In essence, it represents the *functionality of the knowledge that the domain model contains*. It thus allows the system to reason about *how* knowledge will be used. This is crucial as one cannot indiscriminately add new knowledge into a system but rather needs to understand how that knowledge will be used. Otherwise, there is a danger that the knowledge added be useless or incomplete to achieve the task for which the system is designed. The design history also provides an important piece of information: what knowledge is not used for problem solving. The system is aware of what plans are not being matched with any goals and which parts of the domain model are not currently relevant for the task. These are signs that the user needs to correct the system's current knowledge to make all of it useful for the task at hand. The scenarios in the next sections illustrate in more detail how these features of the architecture support knowledge acquisition.

Figure 5 shows EXPECT's user interface. The user can examine the current contents of knowledge bases by pulling menus and choosing to display the concept hierarchy or the

Insert Figure 4 here

Figure 4: Relevant portions of the design history produced by the problem solver.

design history. When the user clicks on a concept, the system provides a description of what is currently known about it. The user can also examine instances. When the user clicks on a node in the design history, the system describes in detail both what goal is accomplished in a node and how (a reformulation, a grammar construct, a plan). The user can also look up plans from the problem-solving knowledge base. In short, the interface allows the user to inspect the knowledge currently available to the system, which is the first step toward detecting errors that can be corrected through knowledge acquisition.

When the user detects a knowledge fault through this inspection process, the interface offers several possibilities to change the contents of the knowledge bases, and at different levels of detail. The user can change existing information about instances or add new instances (e.g., new locations, new seaports). The user can also change any plan by manipulating its components (adding, removing, or substituting substeps), or add new plans either created by the user or created by the system by analogy with existing plans. Finally, the user can change portions of the design history by manipulating its nodes. Adding or removing a step in a node causes the plan in that node to be changed, as well as any plans that appear in the subtree originating at that node. If any new subgoals cannot be matched with existing plans, the system creates new plans by analogy with the plans in the original subtree. The next section shows an example of this behavior.

Insert Figure 5 here

Figure 5: EXPECT's user interface.

3 Transfer of Problem-Solving Knowledge Within a Domain

Figure 6 shows a scenario in which existing problem-solving knowledge is corrected. The user is presented with menus (instead of being allowed to type in free English), but we paraphrase here the user's intervention in English for clarity and indicate this with italics. EXPECT generates language descriptions of plans of the type shown in this figure and carries out the modifications to its knowledge base as indicated by the user.

In this scenario, the system is given the problem of transporting a unit to the seaport of Cabra (line [1]). The system solves the problem and reaches the conclusion that it takes 3 days (line [2]). The user is surprised that it could be done so fast in such a small seaport and asks why (line [3]). The system explains its conclusion by retracing its reasoning (line [4]). Notice that, at this point, only a summary of the whole reasoning is included in the explanation, thus providing a high-level justification for the conclusion. The user can however "zoom-in" on particular part of the problem solving by asking further questions. This is illustrated in lines [5-6].

The justifications provided by the system allow the user to detect a potential error: the

U: *How long does it take to transport that unit to Cabra with the ships available?* [1]

S: 3 days [2]

U: *Why?* [3]

S: To transport unit X to a location, first I calculated that the total weight of the cargo is 300,000 tons, then I determined that I could use 3 container ships and 10 breakbulk, and distributed the cargo among the ships. [4]

U: *Why breakbulk ships?* [5]

S: I used the Cabra seaport, and determined that breakbulk ships fit in the Cabra seaport. [6]

U: *This is wrong. Cabra is too small for breakbulk ships.* [7]

S: I concluded that breakbulk ships fit in the Cabra seaport. To do this, I checked that the ship length is less than the maximum vessel length in that seaport. Is this right? (Yes; No; Yes with constraints) [8]

U: *Yes, with constraints: you have to check both the length and the draft.* [9]

S: I know how to check the length. Do I check the draft in a similar way? (Yes; No; Yes with constraints) [10]

U: *Yes* [11]

S: What corresponds to (1) the ship length (2) less than (3) the maximum vessel length in that seaport? [12]

U: *(1) ship draft (2) less than (3) maximum vessel draft in the seaport* [13]

S: Ok. So to determine what ship types fit in a seaport, I determine if the ship length is less than the maximum vessel length in that seaport, and then I determine if the ship draft is less than the maximum vessel draft allowed in the seaport. Both have to be true in order for a ship type to fit in a seaport. Correct? (Yes or No) [14]

U: *Yes* [15]
[...]

S: Re-solving the problem with the updated method for determining what ship types fit in a seaport. It takes 10 days to transport the unit to Cabra. [16]

⋮

Figure 6: Example scenario: Modifying existing plans.

user disagrees with some of the information given by the system and provides conflicting information (line [7]). Given this new information, EXPECT now reasons about the design history to localize the problem, and a dialogue to debug the system is initiated.

First, EXPECT finds the exact place in the design history that led to the conclusion questioned by the user and explains that part of the reasoning (line [8]). This allows the user to detect the problem more precisely. In this case, the user realizes that the plan to determine whether a ship fits in a port is incorrect (or incomplete). The user thus provides the additional information, namely that the draft of a ship must also be taken into consideration (line [9]) at a particular node of the design history. The system asks the user how to check the draft, and whether the analogy with checking the length is appropriate. This is possible because the system already has a plan to check the length, and it can use it to form an expectation for how to check the draft (line [10]). Upon positive reaction (line [11]), EXPECT prepares the elements that can be the source of the analogy, and prompts the user for their correspondents (line [12]). After the user's response (line [13]), EXPECT summarizes to the user the changes made to the knowledge base (line [14]). Notice that since the results of both checks are boolean, the system also has a suggestion about how to combine them. The plan is now corrected. After the problem has been fixed, EXPECT re-works the original goal (line [16]). Notice that since the system did not find a plan to compute the maximum vessel depth on a seaport, it created a new plan by analogy with the plan that achieves the goal of computing the maximum vessel length. The acquisition of this new plan is done autonomously by the system, based on the user's suggested changes on how the problem was solved. This is possible because the design history reflects the fact that this plan is used in the part of the reasoning that the user wants to modify. If the new plan had any inconsistencies, EXPECT would notify the user. This can be caused by incompletenesses in the mapping that the user gives the system to do the analogy. It is important to realize that had the system found an existing plan to solve the new goal, it would have used this plan to expand the new subtree. EXPECT is acquiring new plans for its problem-solving knowledge base, effectively learning at the knowledge level [Newell, 1982].

Figure 7 shows a different scenario where new plans are added to the knowledge base by the user. Much like in the previous scenario, the user communicates to the system the analogy of ships and ports with aircrafts and hangars: what to retrieve (line [2]), what the correspondences are (line [4]), and what the exceptions are (lines [6-10]). The system takes charge of the rest, including updating the corresponding parts of the different knowledge sources.

4 Transfer of Problem-Solving Knowledge Across Domains

The analogies in the above scenarios involve plans within the same domain, but exactly the same mechanism applies if the new plans are in a different domain. For example, the user can suggest the similarities with checking if a package fits in a truck, or if any physical object fits in another. Because the analogy is not worked out automatically and the user is in charge of guiding the KA tool through all the stages (retrieval, mapping, and adaptation), the analogy can be as far fetched as the user finds suitable. For example, one can imagine using the plan to check if a ship fits in a port to check if an event happens during another one by checking if the corresponding time intervals fit within one another. Thus, the same general mechanism can be used throughout any stage of knowledge acquisition and throughout any application domain at hand. Notice that this requires that knowledge acquisition does not happen in a vacuum, and by that we mean that the system should have access to as many

S: . . . [1]

U: *To service a type of aircraft, first check if it fits in a hangar. Similar to how a ship fits in a seaport.* [2]

S: To determine what ship types fit in a seaport, I determine if the ship length is less than the maximum vessel length in that seaport, and then I determine if the ship draft is less than the maximum vessel draft allowed in the seaport. Both have to be true in order for a ship type to fit in a seaport. We want to describe a way to determine if a type of aircraft fits in a hangar. What corresponds to (1) the ship length (2) less than (3) the maximum vessel length in that seaport (4) the ship draft (5) less than (6) the maximum vessel draft allowed in the seaport (7) both have to be true? [3]

U: *(1) aircraft length (2) less than (3) hangar length (4) aircraft height (5) less than (6) hangar height (7) both have to be true.* [4]

S: Ok. So in order to check if a type of aircraft fits in the hangar, I determine if the aircraft length is less than the hangar length, and then I determine if the aircraft height is less than the hangar height. Both have to be true in order for an aircraft type to fit in the hangar. [Yes; No; Yes with constraints] [5]

U: *Yes, with constraints: you have to check length, height, and width.* [6]

S: Ok. I know the length, width, and height of hangars. I know the length and height of an aircraft. What is the width of an aircraft? [7]

U: *Wing span?* [8]

S: I understand. I know how to check the length and the height. Do I check the width in a similar way? (Yes; No; Yes with constraints) [9]

U: *Yes* [10]

S: Ok. So in order to check if a type of aircraft fits in a hangar . . . [11]

Figure 7: Example scenario: Adding new plans.

knowledge bases in different domains as possible. Some efforts in very large knowledge bases already aim in this direction [Neches *et al.*, 1991; Lenat and Guha, 1989]. This is much as humans communicate and learn things from each other: by first establishing backgrounds and then using constructs that the other person is accustomed to. For example, doctors may not teach new things to biologists in the same way that they would explain them to engineers, since they would have common ground concepts with the former to build upon. Having knowledge bases shared by different knowledge-based systems will some day provide this common ground for our knowledge acquisition tools.

5 Next Step: Generalization of Problem-Solving Methods

Of course, acquiring knowledge by analogy has its limits. For example, a user who is entering knowledge in a medical domain may not know in detail what problem-solving knowledge is involved in our transportation domain. This is also common in humans: the more we know about the other person's areas of knowledge, the easier it is to explain things to them. Again, we can turn to machine learning techniques for an answer. This section describes our plans for future work regarding the use of generalization and induction techniques to extend the current analogical reasoning in EXPECT.

Let us consider an example from our transportation domain. There is a plan to calculate the throughput (takeoffs per day) that an airport can handle. To do that, the given throughput is adjusted by a percentage factor determined by the presence of bad weather, and by how many docks are available for unloading.[2] There is also a different plan to calculate how many planes are available for a transportation problem. To do so, the amount of planes assigned for the task is adjusted by a percentage factor that takes into account the rate at which each type of plane tends to break down and how much time it takes to repair it. Induction techniques, and in particular, generalization from examples could be extended and applied in these cases to provide the system with the concept of a plan to do adjustments of values, as shown in Figure 8. The following example illustrates how these generalized plans could be used to cooperate with an expert during KA.

In summary, this scenario illustrates how EXPECT could use plan generalization and analogical reasoning to guide knowledge acquisition. The knowledge about adjustments in the transportation domain is used to acquire adjustments for drug therapy. We used two very different domains to illustrate the potential of this approach, but the impact should be greater when the nature of the domains is not so diverse.

Suppose that a medical expert is using EXPECT for administering drugs, in particular to advice with digitalis therapy [Swartout, 1983]. The default dose of a drug has to be reduced if the patient has certain conditions (called sensitivities to the drug). To treat a patient with digitalis, the normal dose (1 μg/Kg) needs to be adjusted by a factor of 0.8 if the patient has a high level of serum calcium and by a factor of 0.7 when serum potassium is low. If both sensitivities are present, the default dose is adjusted by the product of both factors $(0.8 * 0.7)$.

Suppose now that the system knows only about one sensitivity (low serum potassium), as well as how to adjust the dose of digitalis if the patient has this condition. Figure 9 shows a scenario where the user wants to add a new sensitivity: high serum calcium. The user enters whatever information she is aware that the system needs about drug sensitivities in line [1]. EXPECT updates the domain model accordingly. Now EXPECT examines the problem-solving knowledge to determine what sensitivities are used for. This causes the question in line [2] of how to adjust the dose for this new sensitivity.

The system knows to acquire an adjustment for this sensitivity based on how sensitivities are used for problem solving. Because a general plan for doing adjustments is available, the

[2]If there are not enough unloading docks the planes would be idle waiting for a dock to become available.

Insert Figure 8 here

Figure 8: A hierarchy of plans for adjusting values.

system knows that when more than one adjustment is present it needs a method to recombine the results (line [6]). The system could justify the reason for the questions to the user (lines [7-8]), as well as to produce examples that undo the generalization and facilitate a ground analogy to the user in terms of airport takeoffs (line [10]).

This example illustrates how, through plan generalization, the system could further support analogies across domains. The generalization hierarchy would provide the grounds for transferring knowledge across different domains through its abstraction from domain-specific knowledge. The type of language that we use in EXPECT's plans combined with the LOOM knowledge representation system would facilitate this, but in addition we plan to research on general-purpose machine learning methods to induce these generalized plans automatically or semi-automatically. In the later case, a knowledge engineer would be involved, but we believe this is a much lighter burden than other current KA environments would impose. Either way, these machine learning methods could prove more useful for automating KA than a general KA tool for a particular generic method.

6 Related Work

NeoDisciple [Tecuci, 1992] integrates several machine learning techniques in a knowledge acquisition tool. NeoDisciple takes a user-given answer to a problem and applies explanation-based learning to build a plausible proof tree, abduction to complete the proof, and several other learning techniques to generalize the proof. Its predecessor, DISCIPLE [Tecuci and Kodratoff, 1990], built an analogy with an existing proof when the system lacked domain knowledge to build the proof for a new input. Our approach automates different parts of the analogical process. The user suggests the source of the analogies, the mapping, and any necessary adaptations. The tool provides the *supporting environment* by navigating through the system's reasoning and carrying out the user's corrections through

U: *Add a new sensitivity for digitalis: high serum calcium. [...]* [1]

S: Ok. What method should I use to adjust the dose for high serum calcium sensitivity [Take percentage; Do computation; Other]? [2]

U: *Take percentage* [3]

S: Ok. Which percentage? [4]

U: *0.8* [5]

S: Ok. Now I need to know how to combine the results of the adjustments. Choose one of [MIN;MAX;ADD;MULTIPLY;Other]. [6]

U: *Why?* [7]

S: We are adjusting digitalis dose. To do an adjustment, I need to know: (1) value to be adjusted, (2) factors, (3) methods to adjust the value, and (4) how to combine the results. In this case, (1) digitalis dose (2) drug sensitivities (3) take percentage 0.8 for high serum calcium, take percentage 0.7 for low serum potassium. Now I need to know (4) how to combine the results. [8]

U: *How to combine the results?* [9]

S: For example, when I compute the takeoffs per day in an airport, I adjust the default value of takeoffs for the airport according to weather conditions and to available unloading docks. To do an adjustment, I need to know: (1) value to be adjusted, (2) factors, (3) methods to adjust the value, and (4) how to combine the results. In this case, (1) takeoffs per day (2) weather, unloading docks (3) take percentage, compute docks throughput (4) take minimum value. In the digitalis dose adjustment, I need to know how to combine the results of the adjustments. Choose one of [MIN;MAX;ADD;MULTIPLY;Other]. [10]

U: *multiply* [11]

S: Ok. High serum calcium is now a new sensitivity for digitalis. [12]

Figure 9: Hypothetical scenario: Adding new plans in a different domain.

analogical reasoning. EXPECT provides a framework for analogical reasoning where the tool takes responsibility for suggesting corrections and for making the adequate changes in the knowledge base.

Other work on analogy uses the derivational traces of the problem solver to guide the system in solving a new problem [Veloso, 1992; Carbonell, 1986]. EXPECT uses the design history as a source of analogies as well, and its problem solver will find plans to match any new goals that arise as a result of the mapping. However, the crucial difference is that EXPECT will create new problem-solving knowledge when no plan is found to achieve a goal. A new rule is created in our first scenario to compute the maximum depth of a berth, without which the system could not have solved the problem. In other words, EXPECT is learning at the knowledge level [Newell, 1982].

The Spark/Burn/FireFighter framework [Klinker *et al.*, 1991] treats knowledge acquisition as a programming effort, and aims to provide a set of *mechanisms* as basic blocks for building knowledge-based systems. The goal is to design such mechanisms to be both *usable* (understandable by domain experts) and *reusable* (applicable to several tasks and domains). The PROTEGE-II system [Puerta *et al.*, 1992] is also based on reusable mechanisms as building blocks for knowledge acquisition, and addresses usability by integrating domain-dependent knowledge in the process of putting these mechanisms together. EXPECT shares both goals, and the aim is to achieve them without explicitly building such mechanisms. If at all, these mechanisms would be a by-product of the tool's interaction with the user and its learning capabilities. Problem-solving knowledge in EXPECT is usable because it is accessible to domain experts through explanations. Knowledge is reused when the user proposes analogies with new situations, as well as through any generalizations over existing cases.

Another general-purpose knowledge acquisition approach is that proposed by KADS [Wielinga *et al.*, 1992], a methodology for building knowledge based systems, which, unlike EXPECT, emphasizes the initial building of a system (as opposed to its refinement). KADS views knowledge acquisition as a modelling activity and proposes a number of models that a knowledge engineer needs to build. Each model emphasizes a specific aspect of the system to construct and contains part of the knowledge needed. KADS also separates the domain model from the control knowledge, which is in turn divided into the inference model, the task model, and the strategic knowledge. These last three models together indicate how knowledge from the domain model is to be used in problem solving, and could thus theoretically be also used to guide refinement of a knowledge-based system once an initial prototype has been built. This is not currently done, however. KADS's current aim is to help the knowledge engineer faced with the task of building a new knowledge-based system by providing an explicit set of building blocks (or models) that the knowledge engineer should be concerned about, thus using a divide-and-conquer approach to the initial knowledge acquisition task. Our aim in EXPECT is a complementary one: it is to aid in refining and debugging a knowledge-based system automatically once a prototype system exists. Because EXPECT derives code from the specification automatically (i.e., the domain model and the plans), changes done at the specification level are automatically reflected in the executable code. In contrast, until executable systems are automatically derived from the models designed with the KADS methodology, guidance from the models would only apply to refining the models themselves. The changes would then need to be reflected in the code by hand.

7 Conclusion

We have shown our approach to building KA tools to achieve knowledge re-use based on general-purpose machine learning methods. In particular, we showed how a learning by analogy mechanism can be incorporated in a knowledge acquisition tool to facilitate the knowledge acquisition process. In our system, the current content of the knowledge base is used to create expectations dynamically for what knowledge is to be acquired. These expectations are based on the functionality of each piece of knowledge in the overall system, and on analogies derived from the current knowledge base. Importantly, the system and the user work in synergy: unlike other machine learning approaches, our system does not attempt to deduce all the information needed to form the analogy by itself. Rather, it relies on input from the user to guide its analogical learning. On the other hand, the user does not need to know a lot about the knowledge base or to enter all the new necessary information explicitly, but can rely on the tool to guide to process. We showed how this approach can be employed to acquire information both within a domain and across domains.

We also outlined how other general learning mechanisms (such as generalization and induction techniques) might be incorporated into our system. We hope to support with this work our claim that powerful, versatile and yet practical knowledge acquisition tools can be built by coupling general machine learning mechanisms with more traditional knowledge acquisition techniques.

Acknowledgments

We would like to thank Kevin Knight, Bing Leng, Vibhu Mittal, and Bill Swartout for their feedback during the writing of this paper. We would also like to thank the anonymous reviewers for their helpful comments on earlier drafts of this paper, and especially the editors of this journal issue, Smadar Kedar, Yves Kodratoff, and Gheorghe Tecuci for their detailed and valuable feedback. This work was possible thanks to the efforts of current and previous members of the EXPECT group, in particular Pedro Gonzalez, Bing Leng, Vibhu Mittal, Ramesh Patil, Bill Swartout, and Marcelo Tallis. Finally, we would also like to thank Sheila Coyazo for her prompt support on the keyboard.

We gratefully acknowledge the support of the Advanced Research Projects Agency under contract no. DABT63-91-C-0025. 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.

References

Buchanan, B. G. and E. H. Shortliffe. 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley.

Carbonell, J. G. 1986. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In *Machine Learning, An Artificial Intelligence Approach, Volume II.* San Mateo, CA: Morgan Kaufmann.

Chandrasekaran, B. 1986. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* 1(3):23–30.

Chang, T. C. and R. A. Wysk. 1985. *An Introduction to Automated Process Planning Systems.* Englewood Cliffs, NJ: Prentice-Hall.

Davis, R. 1980. *Knowledge-based systems in artificial intelligence.* New York, NY: McGraw-Hill.

Fikes, R. E. and N. J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Gil, Y. 1992. *Acquiring Domain Knowledge for Planning by Experimentation.* PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.

Gil, Y. 1994. Knowledge refinement in a reflective architecture. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, 1994.

Kitto, C. M. 1988. Progress in automated acquisition tools: How close are we to replacing the knowledge engineer? In *Proceedings of the Third Knowledge-Acquisition for Knowledge-Based Systems Workshop.* Banff, Alberta, Canada.

Klinker, G., C. Bhola, G. Dallemagne, D. Marques, and J McDermott. 1991. Usable and reusable programming constructs. *Knowledge Acquisition* 3(2):117–135.

Knoblock, C. A. 1993. *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning.* Norwell, MA: Kluwer Academic Publishers.

Lenat, D. B. and R. V. Guha. 1989. *Building Large Knowledge-Based Systems.* Reading, MA: Addison-Wesley.

Lozano-Perez, T. 1987. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal on Robotics and Automation* 3(3):224–238.

MacGregor, R. 1988. A deductive pattern matcher. In *Proceedings of the 1988 Conference on Artificial Intelligence.* St Paul, MN.

MacGregor, R. 1991. The evolving technology of classification-based knowledge representation systems. In *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, ed. J. Sowa. San Mateo, CA: Morgan Kaufmann.

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

McDermott, J. 1988. Preliminary steps towards a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Knowledge-Based Systems*, ed. S. Marcus. Boston, MA: Kluwer Academic Publishers.

Minton, S. 1988. *Learning Search Control Knowledge: An Explanation-based Approach.* Boston, Massachusetts: Kluwer Academic Publishers.

Moore, J. D. 1989. *A Reactive Approach to Explanation in Expert and Advice-Giving Systems.* PhD thesis, University of California, Los Angeles.

Moore, J. D. and C. L. Paris. 1993. Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics* 19(4):651–694.

Moore, J. D. and W. R. Swartout. 1989. A reactive approach to explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1504–1510. Detroit, MI.

Moore, J. D. and W. R. Swartout. 1991. A reactive approach to explanation: Taking the user's feedback into account. In *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, 3–48. Boston, MA: Kluwer Academic Publishers.

Musen, M. A. 1989. Automated support for building and extending expert models. *Machine Learning* 4(3/4):347–375.

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

Neches, R., R. Fikes, T. Finin, T. Gruber, R. Patil, and W. R. Swartout. 1991. Enabling technology for knowledge sharing. *AI Magazine* 12(3):36–56.

Neches, R., W. R. Swartout, and J. D. Moore. 1985. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering* SE-11(11):1337–1351.

Newell, A. 1982. The knowledge level. *Artificial Intelligence* 18(1):87–127.

Newell, A. and H. A. Simon. 1972. *Human Problem Solving.* New Jersey: Prentice-Hall.

Paris, C. L. and Y. Gil. 1993. EXPECT: Intelligent support for knowledge-based system development. In *Proceedings of the Seventh European Knowledge Acquisition for Knowledge-Based Systems Workshop.* Toulouse, France. Also published in 'Knowledge Acquisition for Knowledge-Based Systems', N. Aussenac, G. Boy, M. Linster, J-G. Ganascia and Y. Kodratoff (eds), New York, NY: Springer Verlag, 1993. ISI Technical Report # RR-93-339.

Puerta, A. R., J. W. Egar, S. W. Tu, and M. A Musen. 1992. A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge Acquisition* 4(2):171–196.

Swartout, W. R. 1983. XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence* 21(3):285–325.

Swartout, W. R. and J. D. Moore. 1993. Explanation in Second Generation Expert Systems. In *Second Generation Expert Systems*, J.M. David, J.P. Krivine, and R. Simmons (Eds), 543–585. New York, NY: Springer Verlag.

Swartout, W. R., C. L. Paris, and J. D. Moore. 1991. Design for explainable expert systems. *IEEE Expert* 6(3):58–64.

Swartout, W. R. and S. W. Smoliar. 1987. On making expert systems more like experts. *Expert Systems* 4(3):196–207.

Tecuci, G. and Y. Kodratoff. 1990. Apprenticeship learning in imperfect domain theories. In *Machine Learning: An Artificial Intelligence Approach*, vol. 3. San Mateo, CA: Morgan Kaufmann.

Tecuci, G. D. 1992. Automating knowledge acquisition as extending, updating, and improving a knowledge base. *IEEE transactions on Systems, Man, and Cybernetics* 22(6):1444–1460.

Veloso, M. M. 1992. *Learning by Analogical Reasoning in General Problem Solving.* PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.

Wielinga, B. J., A. Th. Schreiber, and A. Breuker. 1992. KADS: a modelling approach to knowledge acquisition. *Knowledge Acquisition* 4(1):5–54.