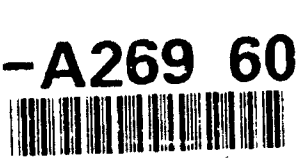
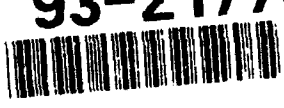


(12)

EXPECT: Intelligent Support for Knowledge Base Refinement

Cecile Paris and Yolanda Gil
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292

January 1993
ISI/RR-93-339



REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1993	Research Report

4. TITLE AND SUBTITLE

EXPECT: Intelligent Support for Knowledge Base Refinement

5. FUNDING NUMBERS

DABT63-91-C-0025

6. AUTHOR(S)

Cecile Paris and Yolanda Gil

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

RR-339

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

ARPA
3701 Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

T

12A. DISTRIBUTION/AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Effective knowledge acquisition amounts to having good sources of expectations that can provide guidance about what knowledge needs to be acquired from users. Current approaches to knowledge acquisition often rely on strong models of the problem-solving method used in the task domain to form expectations. These methods are often implicit in the tool, which is a strong limitation for their use in different domains. Additionally, these tools require an understanding of the method to be used that most experts find difficult to overcome. In this paper we present EXPECT, a novel approach to knowledge acquisition based on the EES architecture that forms expectations based on the current knowledge contained in the system about the task, and are not hard-coded in the tool. We show how the explicit representation of domain principles and its relation to compiled procedural knowledge enables a system to form expectations as to what knowledge is missing or incorrect. This capability coupled with a dialogue-based explanation facility makes communication with the knowledge acquisition tool more natural to domain experts.

14. SUBJECT TERMS

examples, natural language generation, descriptions

15. NUMBER OF PAGES

15

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

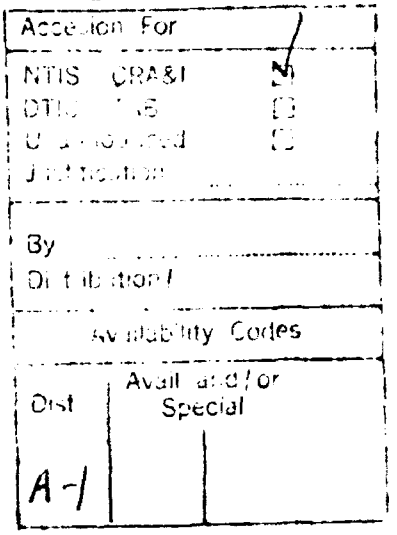
Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

EXPECT: Intelligent Support for Knowledge Base Refinement

Cécile Paris and Yolanda Gil

Information Sciences Institute
and Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
U.S.A.



Abstract

Effective knowledge acquisition amounts to having good sources of expectations that can provide guidance about what knowledge needs to be acquired from users. Current approaches to knowledge acquisition often rely on strong models of the problem-solving method used in the task domain to form expectations. These methods are often implicit in the tool, which is a strong limitation for their use in different domains. Additionally, these tools require an understanding of the method to be used that most experts find difficult to overcome. In this paper we present EXPECT, a novel approach to knowledge acquisition based on the EES architecture that forms expectations based on the current knowledge contained in the system about the task, and are not hard-coded in the tool. We show how the explicit representation of domain principles and its relation to compiled procedural knowledge enables a system to form expectations as to what knowledge is missing or incorrect. This capability coupled with a dialogue-based explanation facility makes communication with the knowledge acquisition tool more natural to domain experts.

1 Introduction

Domain experts are very seldom proficient users of computers. People who build tools for knowledge-based systems, however, are proficient. This might be why interactions with these tools are not always very easy for domain experts. Knowledge acquisition tools are no exception, and, even though our systems' "social skills" are continuously improving, field experiences report things such as:

> "Automated (knowledge acquisition) tools are often fairly complex and require the skills of a knowledge engineer for their use." [Kitto, 1989]

Previous work in the Explainable Expert System (EES) project [Neches *et al.*, 1985, Swartout *et al.*, 1991, Swartout and Smoliar, 1987b] moved toward improving communication with experts by giving them information in terms they can relate to, highlighting domain principles and domain facts as opposed to programming details. We find that the architecture of EES allows two mechanisms

that we believe of central importance for knowledge acquisition: *content-based expectations* and *interaction at the knowledge level.*

Some systems form expectations based on the syntactic structure of the current knowledge (e.g., [Davis, 1976]). Others commit to a particular problem-solving method and form expectations based on the method (e.g., [Eshelman and McDermott, 1986, Marcus and McDermott, 1989]). These expectations are fixed. Instead, our approach creates expectations based on the content of the current knowledge base by understanding the function of each piece of information in the reasoning process. Most knowledge-based systems contain operational knowledge extracted from experts. Experts' operational knowledge is usually in the form of compiled procedures, however. As a result, reasons for specific actions are not readily available [Anderson, 1983]. Instead, we take domain principles (factual and problem-solving knowledge) entered by the user and automatically transform them into a more efficient procedural representation. Consequently, the system has all the information about this compilation in hand, so it has a handle on the role of each piece of knowledge. This allows it to form expectations as to what should be acquired based on the content of its existing knowledge base. We show in this paper that this is a tremendous advantage for a knowledge acquisition tool.

Furthermore, the system can take advantage of this explicit representation of the domain principles and their relation to the compiled knowledge to communicate with the user at the knowledge level. That is, it presents to the user justifications of the reasoning process at various levels of abstraction, independently of the specific steps that appear in the execution trace [Moore and Paris, 1991, Moore, 1989, Paris, 1991]. These explanations are given in natural language, in an interactive fashion.[1] This allows the user to navigate through the problem solving traces more easily. Consequently, the detection of faults in the knowledge base is guided by the user in natural language, abstracted away from the implementation, and closer to the way an end-user is accustomed to communicate with colleagues. In summary, during knowledge acquisition, the system both *learns* at the knowledge level [Newell, 1981, Dietterich, 1986] (acquiring new knowledge from the expert) and *interacts* at the knowledge level (communicating in terms of the domain and not of the implementation).

The rest of the paper runs as follows. We begin with an overview of the EXPECT framework, describing how the knowledge base is structured and how problem solving works. Then we show two hypothetical scenarios to illustrate how the knowledge acquisition tool will form expectations about missing information. The examples are drawn from a transportation domain that evaluates proposed routings, taking into account restrictions on objects transported, destination points, and vehicles used. The remaining sections present a discussion of our approach and future directions of research.

2 EXPECT

EXPECT builds upon previous work on the Explainable Expert System (EES) framework [Neches *et al.*, 1985, Swartout *et al.*, 1991, Swartout and Smoliar, 1987b], which allows for the construction of expert systems that can provide good explanations of their behavior. In this section, we briefly

[1] Our use of the word explanation is in reference to how a system justifies its reasoning, as opposed to the use of the word for explanation-based learning methods where explanations refer to proofs.

describe the features of the framework that were found necessary to support good explanations and explain why the same features also allow for the construction of intelligent knowledge acquisition tools. The next section then illustrates our points.

In order to produce good explanations, a system must represent its knowledge in a structured way, separating the different kinds of knowledge (e.g., domain facts, problem solving knowledge, etc). Furthermore, a system must have the knowledge necessary to justify its behavior (e.g., [Clancey, 1983a, Clancey and Letsinger, 1981, Clancey, 1983b, Swartout and Smoliar, 1987a, Swartout *et al.*, 1991, Swartout, 1981, Chandrasekaran and Swartout, 1991]). Finally, it must be able to produce coherent text and to clarify and elaborate on its own explanations [Moore, 1989, Moore and Swartout, 1989, Moore and Swartout, 1991]. This is indeed needed as, often, a user has follow-up questions (e.g., [Pollack *et al.*, 1982, Moore, 1989]). To have good explanations, then, the knowledge bases and execution traces of conventional expert systems have to be considerably enriched.

In EXPECT, the different kinds of knowledge that comprise an expert system are specified distinctly 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 design rationale needed. The resulting architecture is shown in Figure 1.

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 domain of the expert system. The domain model is written in the LOOM knowledge representation formalism [MacGregor, 1988].

- A problem-solving knowledge base, which contains an organized collection of plans. The plan language allows for an explicit representation of *intent* (what is to be done) and supports a wide range of control structures [Project, 1993].

As an example, consider Figures 2 and 3, which contain samples of these knowledge bases for a system in the 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 its `location`, `piers`, `berths`, and `storage areas`.[2]

Figure 3 shows two of the 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 by (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, an *automatic program writer* (APW) 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 APW records all its steps and decisions in an annotated *design history*. For example, given the

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

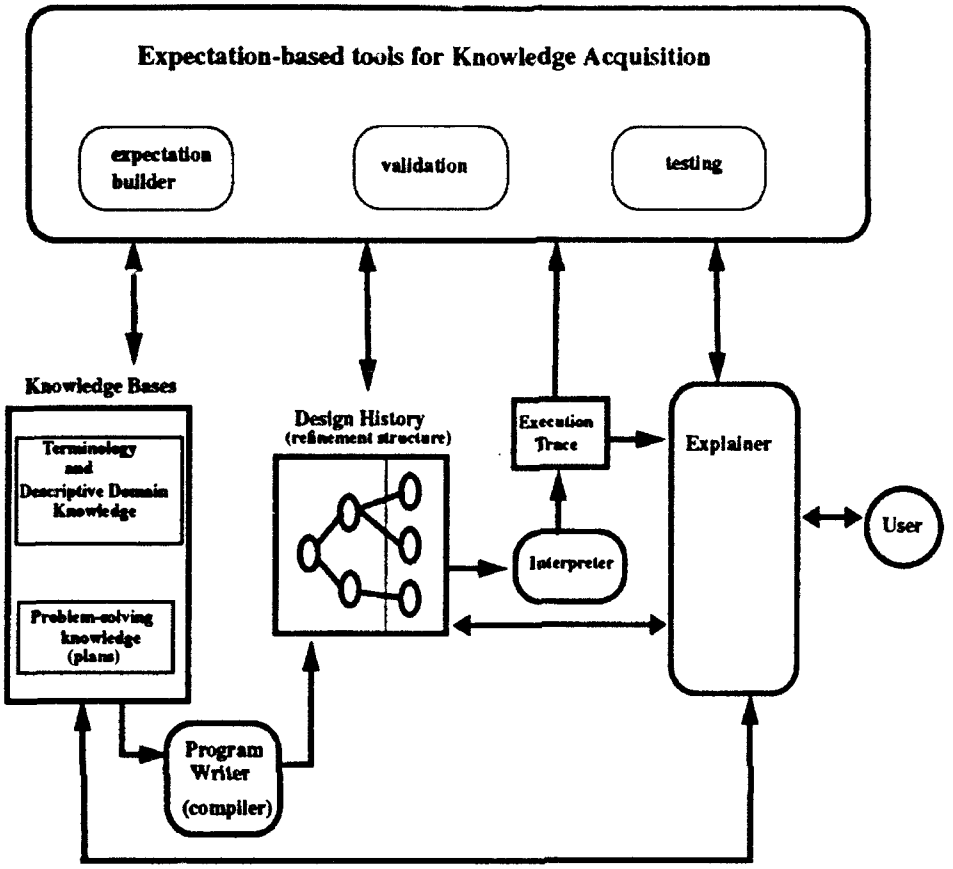


Figure 1: A schematic representation of EXPECT. The design history and the knowledge base components are used to form expectations for knowledge acquisition

general goal to transport a set of objects to a location using a set of ships, the design history produced by the APW is shown in Figure 4. This design history reflects both the goal/ subgoal expansion as well as *goal reformulations*. For example, in Figure 4, we see that 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 is then available for the system to introspect and explain its reasoning.

To solve a concrete goal, an interpreter instantiates the design history, expanding it where necessary, to produce a solution. The interpreter also records its decision process in an execution trace, also available to EXPECT for introspection.

Finally, in order to be able to provide good coherent texts in an interactive fashion, we have developed a *text planning system* that selects and organizes information from the underlying knowledge sources to construct a coherent text. This text planner supports dialogue with a user [Moore and Paris, 1989, Moore and Paris, 1992, Moore, 1989]. User input can be stylized English, menu based, or a hypertext-like interaction mousing parts of the explanation to request

```
(defconcept seaport
   :is (:and port
             (:exactly 1 r-location)
             (:exactly 1 r-piers)
             (:exactly 1 r-available-berths)
             (:exactly 1 r-covered-storage-area))
```

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-available-berths ?s))))
             (max (r-berth-length ?berth-types))))
```

Figure 3: Two plans from the transportation domain.

clarifications [Moore and Swartout, 1990].

As we show in the scenario below, the design history plays a crucial role in the knowledge acquisition process. 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.

To do this type of reasoning, EXPECT creates links between the domain model and the design history. Every relation or concept type referenced in a plan recorded in the design history is linked to that relation or concept in the domain model.

As an example, Figure 5 shows the links created for the first plan shown in Figure 3. Given these links and their place in the design history, EXPECT understands how the factual knowledge is used and why. As we illustrate in the next section, this allows the system to create expectations for knowledge acquisition, and thus guide the user in augmenting and debugging an existing system.

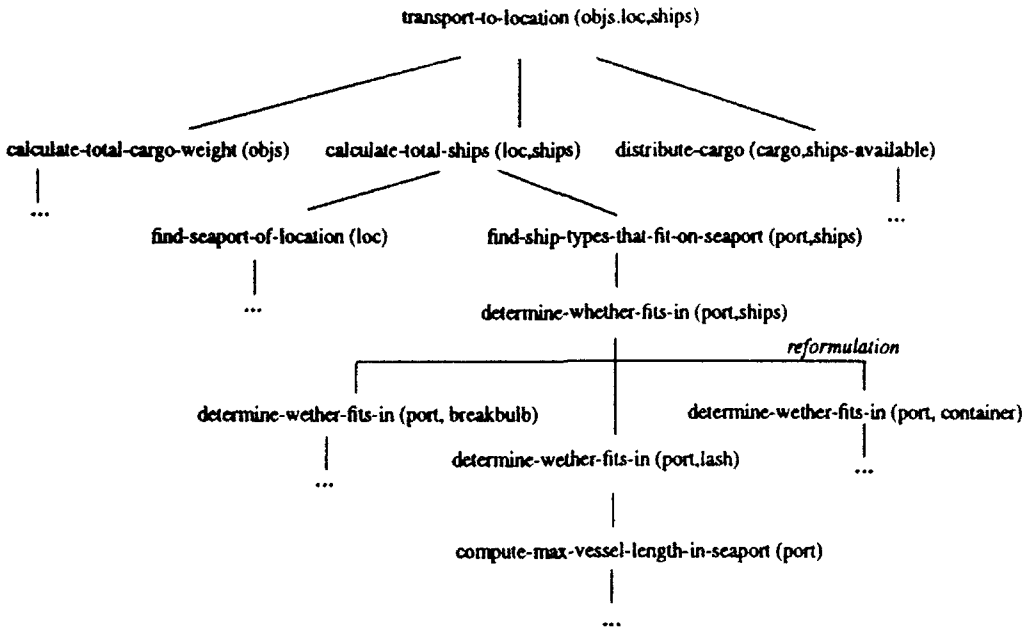


Figure 4: The design history

Importantly, because the system includes an explanation facility capable of producing coherent explanations of the system's behavior, the system can justify to the user why it is asking specific questions and provide feedback as to what is being added or changed. Furthermore, the user can then at any point request documentation about any part of the knowledge base.

3 Augmenting a Knowledge Base

In this section, we show how EXPECT provides guidance based on expectations it forms from its current knowledge. We illustrate the behavior we expect from this tool with two hypothetical scenarios in our transportation domain.

3.1 Adding a New Instance

Figure 6 shows a scenario in which a user wants to extend the system to cover a new seaport. The user starts the dialogue in line [1]. By reasoning about the existing domain model, EXPECT realizes that a port can be either a seaport or an airport. It thus asks the user to be more specific (line [2]).

At this point, EXPECT reasons about its procedural knowledge and forms expectations as to what other information about a seaport is needed. By understanding that its primary goal is to transport objects and by examining the design history (Figure 4), EXPECT determines that it needs to know

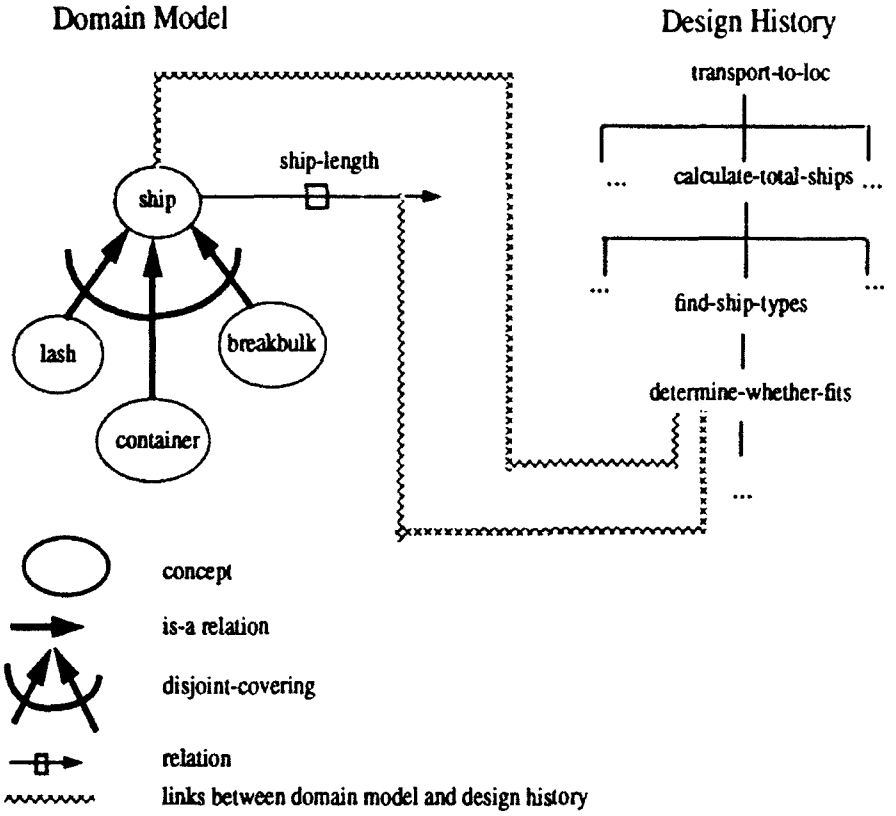


Figure 5: EXPECT creates links between the knowledge base and the design history that represent the functionality of its knowledge. The knowledge acquisition tool makes use of them to form expectations.

the types of the berths in a seaport in order to compute the maximum vessel length in that seaport, as this step is required to figure out whether a ship fits in a seaport. This allows it to prompt the user for that information (line [4]). Furthermore, because it knows why this information is needed and because interactions are conducted through the explanation facility, EXPECT is able to justify to the user the need for this question (lines [5-8]). Finally, again because of its explanation capabilities, EXPECT can also provide help to the user by providing information about the current contents of the knowledge bases, as illustrated in lines [9-11].

Notice that EXPECT is able to focus the knowledge acquisition process and only ask from the user what is actually necessary for the task at hand (i.e., it does not automatically prompt the user for all the roles defined on a seaport, shown in Figure 2). (The user is informed that entering this information is optional, however, as shown in line [12]). This would not be possible by reasoning only about the domain model, as a pure concept editor would. EXPECT can do so because it understands how the declarative knowledge base is used during problem solving.

U:	I want to define a new port called "Long Beach", whose location is "Los Angeles".	[1]
S:	Is Long Beach a seaport or an airport?	[2]
U:	Seaport. It has 2 piers and 6 berths.	[3]
S:	Ok. Now I need to know the types of the berths.	[4]
U:	Why?	[5]
S:	This is needed to compute the length of a berth. I need to know the length of a berth in order to compute the maximum ship length supported by a seaport.	[6]
U:	What for?	[7]
S:	In this context, we are trying to transport objects to locations by using ships. To do so, [...] This requires finding the ship types that fit in the seaport. This is why I must compute the maximum ship length supported by a seaport.	[8]
U:	Ok. What types of berths are there?	[9]
S:	I know of five types of berths. [...]	[10]
U:	Ok. The berths of Long Beach are [...]	[11]
S:	You may also tell me other information about Long Beach, like ..., but it is not necessary. Do you want to specify any of those?	[12]
	⋮	

Figure 6: Hypothetical scenario – Adding a new instance

3.2 Modifying Existing Plans

Figure 7 shows a scenario in which the system is given a problem to solve.[3] This problem involves 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]), exploiting both the actual execution trace for this specific problem and the design history. 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 line [5-6].

The justifications provided by the system allows the user to detect a potential error: he or she disagrees with some of the information given by the system and provides conflicting information

[3]This interaction could occur some time after the new device has been added to the knowledge base and involve a different user. It could also occur if the system developer simply wanted to give the system some problems as a way of debugging and refining the knowledge bases after some modifications.

(line [7]). Given this new information, EXPECT now reasons about its execution trace, attempting to localize the problem, and a dialogue to debug the system is initiated.

First, EXPECT finds the exact point in the execution trace 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]).

As it turns out, the term draft is unknown to the system. Because the factual knowledge is explicitly represented, EXPECT is able to go further in its debugging and hypothesizes that a domain fact may be missing. The system presents the user with the relevant knowledge that is currently available (both ships and berths have length, depth, and width) and asks the user if a new concept is to be added (line [10]). The user could now add a new concept "draft". In this case, however, a new concept is not needed as the word "draft" is being used to refer to the concept "depth" which already exists in the knowledge base (line [11]). The user indicates to the system that the word "draft" is used to refer to the "depth". From this, the system can add the word "draft" as a new lexical item for the relation "depth". This will then allow the system to refer to this relation in terms understandable by experts in the domain (as opposed to system builders). Having added this new lexical item, the system asks the user how to check the draft. Because the system already has a plan to check the length, it uses it to form an expectation for how to check the draft, and presents it as a suggestion to the user (line [12]). Upon positive reaction (line [13]), EXPECT summarizes to the user the changes made to the procedural knowledge base (line [14]). Notice that since the results of both checks are boolean, the system also has an expectation about how to combine them. The plan is now corrected.

The next step for EXPECT is to check that the domain model has the information to complete the task. The draft of the ships was never used before in a plan, and as a result, EXPECT never bothered the user to acquire it. However, it now becomes relevant with the acquisition (or modification) of the plan. The user is thus now asked for this information in line [16]. Finally, after the problem has been fixed, EXPECT re-works the original goal (line [17]).

4 Discussion

The two scenarios above illustrate several important points about EXPECT's knowledge acquisition capabilities:

- It acquires information that the user may have forgotten to include in a definition either by oversight or because lack of understanding as to what needs to be included in the knowledge base. For example, the system decides that it needs a concept more specific than "port" for Long Beach ("airport" or "seaport"). Thus, users (like the user in our scenario) do not have to be system developers, but could be instead end-users, who need to enter new information recently available.

- It is able to focus the knowledge acquisition process and only ask for the information that is relevant to the task, thus avoiding burdening the user with many irrelevant questions. For example, in the first scenario, the system did not ask the user for all the roles that were defined

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 can use 3 container ships and 10 breakbulk ships, and distributed the cargo among the ships. I then calculated how long it takes to transport this cargo with these ships. It takes 3 days. [4]

U: Why breakbulk ships? [5]

S: Caboverde is the seaport of Cabra, and determined that breakbulk ships fit in it. [6]

U: This is wrong. Caboverde is too small a port for breakbulk ships. [7]

S: I concluded that breakbulk ships fit in Caboverde. 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 the length, width, and depth of ships and berth types. Should I add the draft? (Yes; No) [10]

U: No. Draft is the depth for both ships and berths. [11]

S: I understand. I know how to check the length. Do I check the draft in a similar way? (Yes; No; Yes with constraints) [12]

U: Yes [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: Ok. However, I don't know the draft of all ship types. Could you tell me [...]? [16]
[...]

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. [17]

⋮

Figure 7: Hypothetical scenario – Modifying existing plans

on a seaport. This capability is especially important when dealing with large knowledge bases

that might be shared among various systems. By reasoning about how knowledge is to be used, EXPECT can focus on the information relevant for the task at hand.

- Because the system represents the domain principles and reasons about them to derive the procedural knowledge, it is enough for the user to inspect and change information in the domain principles, without having to know exactly what in the procedural knowledge that change affects.

- The system is capable of explaining its behavior to the user. Explanations are central to the design of our system. Specifically, in this case, the system can explain to the user why it is requesting further information about berth types. The user can also zoom-in to specific parts of the explanation, which allows selective inspection of the system's reasoning and leads to detect the exact point that lead to the wrong conclusion. This is especially important if the knowledge acquisition tool is to be used by users other than the system developer.

- The expectations described above are based on existing declarative and procedural knowledge about the task domain. Expectations are not formed based solely on structure but also on the contents of the knowledge base, and as a consequence EXPECT can justify the need for data required from the user.

- The system, and not the user, is both responsible and aware of what is in the system, how knowledge is structured and how it is being used. Knowledge acquisition is guided directly by the current structure of the system and its contents. Structure and contents are the basis for the dynamic creation expectations, instead of being predefined in the acquisition tool.

- Knowledge acquisition is tightly coupled with the explanation facility, which also relies on the current structure and content of the knowledge sources. This capability provides explanations and feedback to the user in appropriate English and in an interactive manner.

- It is also important to note that help can be provided even when the system has wrong expectations. The explanation facility provides a means for the user to examine the knowledge base in search of the problem. Once the problem is located, the user will be able to enter the information necessary to fix the bug directly.

- Our scenarios show that users can either extend or correct the existing knowledge. Thus, EXPECT can be used with knowledge bases that are incorrect or incomplete.

5 Related Work

Using available knowledge to create expectations is not a novel approach. Already in TEIRESIAS [Davis, 1976] rule models captured expectations based on syntactic regularities of existing rules to predict the likely patterns of new ones. A stronger source of expectations is the inference structure behind the task domain, which may be common to many different applications [Chandrasekaran, 1986, McDermott, 1988, Clancey, 1985]. Most current tools for knowledge acquisition exploit this type of expectations and are tailored to a specific problem-solving method [Eshelman and McDermott, 1986, Marcus and McDermott, 1989, Musen *et al.*, 1988,

Bareiss *et al.*, 1989]). For example, a KA tool for heuristic classification expects to acquire mappings from input data into predefined classes and uses these expectations to guide the acquisition process. Whether KA tools base their expectations on syntactic or inference structure, the expectations are implicit in the tool. In EXPECT , the inference structure is not implicit: it is put together by the APW from the different knowledge sources. However, neither the inference or the syntactic structure are used by EXPECT to assist in KA. Instead, EXPECT forms expectations based on the content of the KBs and how the knowledge is used for problem solving.

Automated knowledge refinement tools cooperate with users in taking charge of some part of the process of correcting a knowledge base. For learning apprentices, an expert provides problem-solving traces [Wilkins, 1988] or solutions [Mitchell *et al.*, 1990, Tecuci, 1992] and the system takes responsibility for adjusting its knowledge base to cover that desired behavior with minimal user interaction. It is not practical in many cases to put the burden on the user to provide solutions and/or traces of when problems are complex. For example, in our domain, calculating how many days it takes to transport a unit is a complicated and detailed procedure. Furthermore, it probably would not help much if the user indicates that the correct answer is 10. We take a different approach as to what part of the learning process to automate. Our tool will provide help to the user for navigating through its reasoning and pinpoint erroneous steps. It will take responsibility for suggesting corrections (based on expectations) and for making the adequate changes in the knowledge base. In other words, the user's mission is to understand what was at fault and the system's to suggest and carry out any repairs.

6 Conclusions and Future Work

EXPECT 's knowledge acquisition tool can be used in early stages of development of a KBS. At that time, the knowledge bases are not very populated, so EXPECT does not have a good basis to form expectations but it will do so whenever possible. The more knowledge there exists in the system, the more helpful the tool is in the acquisition of additional knowledge. We also believe that knowledge from other task domains can be brought about to aid in the acquisition of knowledge early on in a new application [Gil and Paris, 1993].

We plan to develop a full-fledged analogy component to facilitate the acquisition of new plans. In our second scenario, the new plan for checking the depth has a very direct analogy to the existing plan for checking the length. The correspondence is most of the times not so direct. Yet, existing plans can be used as a starting point for new plans. As for providing good suggestions for repairs, we will use as a starting point the techniques developed in [Gil, 1991] to correct planning domains.

Finally, we plan to incorporate validation mechanisms to ensure that proposed modifications will not corrupt other parts of the system, or alternatively, check whether the modification could expose previously undetected problems in the knowledge bases. The rich execution traces of problem solving activities will serve as a history of past experience. EXPECT would maintain a set of test cases from this experience, and we plan to look into criteria for selecting which execution traces to include in such a test suite.

Acknowledgments

We would like to thank Kevin Knight, Craig Knoblock, Vibhu Mittal, Eric Meltz, Bill Swartout and the anonymous reviewers for their helpful comments on earlier drafts of this paper as well as some of the research described here. We gratefully acknowledge the support of ARPA with the contract DABT63-91-C-0025.

References

[Anderson, 1983] J. R. Anderson. Knowledge Compilation: The General Learning Mechanism. In R. S. Michalski, editor, *Proceedings of the International Machine Learning Workshop*, pages 203–212, Monticello, IL, June 1983. University of Illinois at Urbana-Champaign Department of Computer Science.

[Bareiss *et al.*, 1989] Ray Bareiss, Bruce W. Porter, and Kenneth S. Murray. Supporting Start-to-Finish Development of Knowledge Bases. *Machine Learning*, 4(3/4):259–283, 1989.

[Chandrasekaran and Swartout, 1991] B. Chandrasekaran and William Swartout. Explanations in Knowledge Systems: The Role of Explicit Representation of Design Knowledge. *IEEE Expert*, 6(3):47–50, June 1991.

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

[Clancey and Letsinger, 1981] William J. Clancey and Reed Letsinger. NEOMYCIN: Reconfiguring a Rule-Based Expert System for Application to Teaching. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 829–836, Vancouver, B. C., Canada, 1981.

[Clancey, 1983a] William J. Clancey. The Advantages of Abstract Control Knowledge in Expert System Design. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 74–78, Washington, D.C., August 22-26, 1983.

[Clancey, 1983b] William J. Clancey. The epistemology of a rule-based expert system: a framework for explanation. *Artificial Intelligence*, 20(3):215–251, 1983.

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

[Davis, 1976] Randall Davis. *Applications of Meta-level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases*. PhD thesis, Stanford University, 1976.

[Dietterich, 1986] Thomas G. Dietterich. Learning at the Knowledge Level. *Machine Learning*, 1(3):287–316, 1986.

[Eshelman and McDermott, 1986] L. Eshelman and J. McDermott. MOLE: A Knowledge Acquisition Tool That Uses its Head. In *Proceedings of the National Conference on Artificial Intelligence*, pages 950–955, Philadelphia, PA, August 1986. AAAI.

[Gil and Paris, 1993] Yolanda Gil and Cécile L. Paris. Towards general-purpose knowledge acquisition. To appear in the *Proceedings of the IJCAI-93 workshop on Machine Learning and Knowledge Acquisition*, 1993.

[Gil, 1991] Yolanda Gil. A Domain-Independent Framework for Effective Experimentation in Planning. In *Proceedings of the Eight International Workshop on Machine Learning*, Evanston, IL, 1991. Morgan Kaufmann.

[Kitto, 1989] C.M. Kitto. Progress in Automated Acquisition Tools: How close are we to replacing the knowledge engineer. *Knowledge Acquisition*, 1, 1989.

[MacGregor, 1988] Robert MacGregor. A Deductive Pattern Matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota, August 1988.

[Marcus and McDermott, 1989] S. Marcus and J. McDermott. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39(1):1–37, May 1989.

[McDermott, 1988] John McDermott. Preliminary steps towards a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for KBS*. Kluwer, 1988.

[Mitchell *et al.*, 1990] T. M. Mitchell, S Mahadevan, and L. Steinberg. LEAP: a learning apprentice for VLSI design. In *Machine Learning: An Artificial Intelligence Approach*, volume 3. Morgan Kaufmann, San Mateo, CA, 1990.

[Moore and Paris, 1989] Johanna D. Moore and Cécile L. Paris. Planning Text For Advisory Dialogues. In *Proceedings of the Twenty-Seventh Annual Meeting of the Association for Computational Linguistics*, pages 203–211, Vancouver, B.C., Canada, June 26-29 1989.

[Moore and Paris, 1991] Johanna D. Moore and Cécile L. Paris. Requirements for an Expert System Explanation Facility. *Computational Intelligence*, 7(4), 1991.

[Moore and Paris, 1992] Johanna D. Moore and Cécile L. Paris. Planning Text for Advisory Dialogues: Capturing Intentional, Rhetorical and Attentional Information, 1992. Technical Report from the University of Pittsburgh, Department of Computer Science (Number 92–22) and USC/ISI; Submitted for publication.

[Moore and Swartout, 1989] Johanna D. Moore and William R. Swartout. A Reactive Approach to Explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1504–1510, Detroit, Michigan, August 20-25 1989.

[Moore and Swartout, 1990] Johanna D. Moore and William R. Swartout. Pointing: A Way Toward Explanation Dialogue. In *Proceedings of the National Conference on Artificial Intelligence*, pages 457–464, Boston, MA, July 29 - August 3 1990.

[Moore and Swartout, 1991] Johanna D. Moore and William R. Swartout. A Reactive Approach to Explanation: Taking the User's Feedback into Account. In Cécile L. Paris, William R. Swartout, and William C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 3–48. Kluwer Academic Publishers, Boston, 1991.

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

[Musen *et al.*, 1988] M.A. Musen, L. M. Fagan, D.M. Combs, and E. H. Shortliffe. Use of a Domain Model to Drive an Interactive Knowledge Editing Tool. *International Journal of Man-Machine Studies*, 26:105–121, 1988.

[Neches *et al.*, 1985] Robert Neches, William R. Swartout, and Johanna D. Moore. Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development. *IEEE Transactions on Software Engineering*, SE-11(11):1337–1351, November 1985.

[Newell, 1981] A. Newell. The knowledge level. *AI Magazine*, 2(2):1–20,33, Summer 1981.

[Paris, 1991] Cécile L. Paris. Generation and Explanation: Building an Explanation Facility for the Explainable Expert Systems Framework. In Cécile L. Paris, William R. Swartout, and William C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 49–81. Kluwer Academic Publishers, Boston, 1991.

[Pollack *et al.*, 1982] Martha E. Pollack, Julia Hirschberg, and Bonnie Lynn Webber. User Participation in the Reasoning Processes of Expert Systems. In *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, August 18-20 1982. A longer version of this paper is available as a Technical Report from the University of Pennsylvania, Report Number CIS-82-10.

[Project, 1993] The EES Project. The Explainable Expert System: user's manual, 1993. Working notes.

[Swartout and Smoliar, 1987a] William R. Swartout and Stephen W. Smoliar. Explaining the link between causal reasoning and expert behavior. In *Proceedings of the Symposium on Computer Applications in Medical Care*, Washington, D. C., November 1987. (also to appear in "Topics in Medical Artificial Intelligence"; Miller, P.L. (ed), Springer-Verlag).

[Swartout and Smoliar, 1987b] William R. Swartout and Stephen W. Smoliar. On Making Expert Systems More Like Experts. *Expert Systems*, 4(3):196–207, August 1987.

[Swartout *et al.*, 1991] William R. Swartout, Cécile L. Paris, and Johanna D. Moore. Design for Explainable Expert Systems. *IEEE Expert*, 6(3):58–64, June 1991.

[Swartout, 1981] W. Swartout. Explaining and Justifying Expert Consulting Programs. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 815–823, Vancouver, B. C., Canada, August 1981.

[Tecuci, 1992] Gheorghe D. Tecuci. Automating Knowledge Acquisition as Extending, Updating, and Improving a Knowledge Base. *IEEE transactions on Systems, Man, and Cybernetics*, 22(6):1444–1460, 1992.

[Wilkins, 1988] David C. Wilkins. Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St.Paul, MN, 1988.