Acquiring Procedural Knowledge in EXPECT

Yolanda Gil, Jim Blythe, Jihie Kim and Surya Ramachandran
University of Southern California / Information Sciences Institute
Marina del Rey, CA 90292
{gil,blythe,jihie,surya}@isi.edu
http://www.isi.edu/expect

Abstract

The EXPECT project has focused on acquiring problem-solving knowledge for users for the last decade, using an expressive language that is open to inspection. Our aim has been to alleviate the bottleneck in creating knowledge-based systems by providing support for both knowledge engineers and end users to specify problem-solving knowledge. In this paper we summarize selected areas of current research where we focus on end users, and highlight some of the research questions that arise from them. We also summarize some of the results from experiments with end users using our tools.

Introduction

Most machine learning approaches for acquiring knowledge from users reduce the user's participation in the learning process to a bare minimum. Learning apprentices, for example, acquire knowledge by analyzing a user's *normal* interaction with a system (Mitchell, Mahadevan, & Steinberg 1985). Some systems learn from solutions given by the user, while others display a proposed solution that the user can accept, correct, or rate (Cypher 1993; Bareiss, Porter, & Murray 1989). An advantage of these approaches is that they are non-intrusive, and users do not need to understand the underlying learning process beyond sometimes the fact that it is analyzing their examples. To learn under this limited style of interaction, the learning system must figure out on its own knowledge about what features of the example are important, what new abstract features (not appearing in the examples) need to be defined, what combinations of those features are relevant to the task, and what parts of the example correspond to the user's preferences in that specific case. This approach may be impractical in complex tasks where there are many different features in an example, many alternative values of these features, and many combinations thereof can account for user behavior.

An alternative approach is represented by learners that can request information from the user when faced with a situation where they do not know how to proceed. A drawback of this approach is that it puts more burden on users in that they are required to determine what information the learner needs and to figure out what is the best way to provide it, often by designing good examples that facilitate learning. As a system acquires more knowledge about a task, information about which knowledge the system already has and how it is organized may be unknown to the user and often hard to keep track of, posing increasingly stronger requirements on the user.

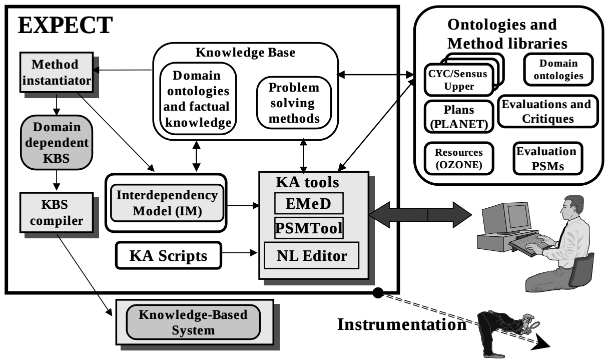


Figure 1: The EXPECT Architecture

Ideally, learning systems should place less demands on the user and become increasingly more competent at the learning task as their knowledge bases grow. They should become more responsible for their internal organization of knowledge and of their own learning process, perhaps by asking more specific questions to the user or by pointing out areas in their knowledge base where they detect the need for additional information. In order to achieve this, they must be able to reflect upon the knowledge that they have about the task—at each moment—and understand what every piece of knowledge, old and new, contributes to the task. This understanding would allow them to be responsible for ensuring harmonious interactions between any new knowledge provided by the user and existing knowledge, thereby preventing redundancies, inconsistencies, and knowledge gaps that may arise inadvertently. This is the approach taken in our work on EXPECT (Swartout & Gil 1995). In this paper we summarize selected areas of current research where we focus on knowledge acquisition of procedural knowledge from end users, describe some of the tools that we have developed for EXPECT, and summarize some of the results from experiments that we have conducted with a variety of end users.

An Overview of EXPECT

The main goal of the EXPECT project (Swartout & Gil 1995) is to make knowledge-based systems more accessible to end users that are not programmers. This section gives a

```
(define-method M1
(documentation "In order to estimate the time that it takes to narrow
a gap with a bulldozer, combine the total dirt volume to be moved
and the standard earthmoving rate.")

(capability (estimate (obj (?t is (spec-of time)))
              (for (?s is (inst-of narrow-gap)))))
 (result-type (inst-of number))
 (body (divide (obj (find (obj (spec-of dirt-volume))
                    (for ?s)))
         (by (find (obj (spec-of standard-bulldozing-rate))
                    (for ?s))))))

(define-method M2
(documentation "The amount of dirt that needs to be moved in any
workaround step that involves moving dirt (such as narrowing a gap
with a bulldozer) is the value of the role earth-volume for that step.")

(capability (find (obj (?v is (spec-of dirt-volume)))
                   (for (?s is (inst-of move-earth)))))
 (result-type (inst-of number))
 (body (earth-volume ?s)))

(define-method M3
(documentation "The standard bulldozing rate for a workaround step
that involves earthmoving is the combined bulldozing rate of the doz-
ers specified as dozer-of of the step.")

(capability (find (obj (?r is (spec-of standard-bulldozing-rate)))
                   (for (?s is (inst-of bulldoze-region)))))
 (result-type (inst-of number))
 (body (find (obj (spec-of standard-bulldozing-rate))
                (of (dozer-of ?s))))
```

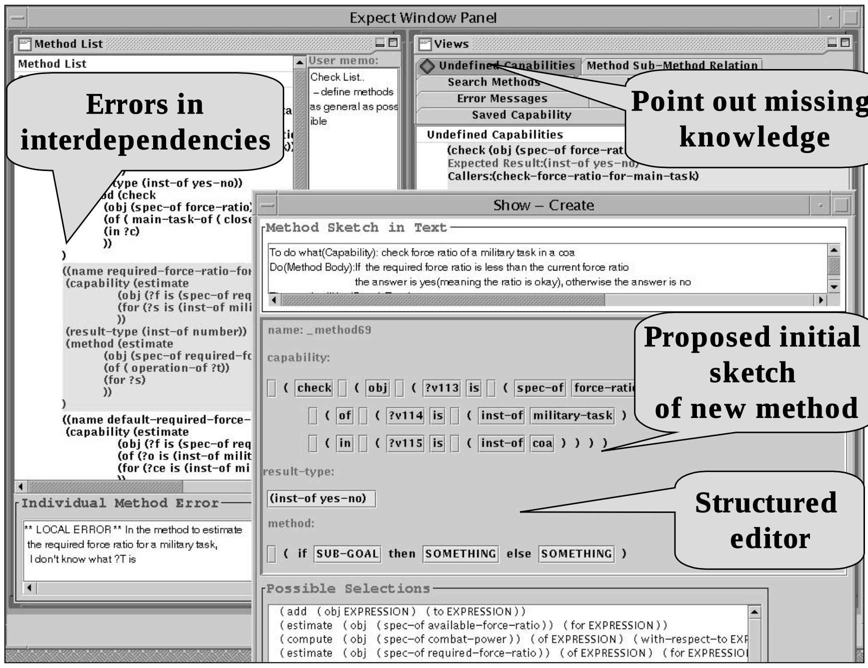
Figure 2: Example EXPECT Methods.

brief overview of the architecture, more details can be found in (Swartout & Gil 1995; Gil & Melz 1996; Gil 1994).

An overview of the architecture is shown in Figure 1. EXPECT provides a highly declarative framework for building knowledge-based systems. Ontologies are used to represent domain objects as well as general theories about domain-independent classes, while task knowledge is represented as problem solving methods. Ontologies are represented in LOOM (MacGregor 1991), a knowledge representation system based on description logic. Every concept or class can have a definition that intensionally describes the set of objects that belong to that class. Concept descriptions can include type and number restrictions of the fillers for relations to other concepts. Relations can also have definitions. LOOM uses the definitions to produce a subsumption hierarchy that organizes all the concepts according to their class/subclass relationship. Problem-solving knowledge is represented in a procedural-style language that is tightly integrated with the LOOM representations (Swartout & Gil 1995). Subgoals that arise during problem solving are solved by methods. Each method description specifies: 1) the goal that the method can achieve, 2) the type of result that the method returns, and 3) the method body containing the procedure that must be followed in order to achieve the method's goal. The language includes expressive parameter typing, conditional statements, and some forms of iteration. Some problem solving knowledge is domain-independent, such as general principles of plan evaluation described in (Blythe & Gil 2000). Problem solving knowledge can also be domain specific. Figure 2 shows some examples of such methods from a system to assess enemy workarounds to target damage. Given a bridge as a target, a possible workaround if the river bed is dry is to use bulldozers to narrow the gap. EXPECT can import external ontologies and knowledge bases.

Our research in knowledge acquisition has concentrated on the acquisition of problem-solving knowledge. We have focused on several central issues that users of a knowledge acquisition tool are concerned with:

- *How do users know that they are adding the right things?*
  Users need to know that they are providing knowledge that is useful to the system and whether they have given the system enough knowledge to do something on its own. Our approach is to use **Interdependency Models** that capture how the individual pieces of knowledge provided work together to reason about the task (Kim & Gil 1999). These Interdependency Models are derived automatically by the system, and are used to detect inconsistencies and missing knowledge that turn into follow-up questions to the user. Users are often not sure whether they are on the right track even if they have been making progress, so we have found that it is very useful to show the user some aspects of this Interdependency Model (for example, showing the substeps involved in doing a task) and how it changes over time.
- *How do users know where to start and what to do next?*
  Intelligent systems use knowledge to perform tasks for the user. If the acquisition tool has a model of those tasks, then it can reason about what kinds of knowledge it needs to acquire from the user. We have developed acquisition tools that **reason about general task models** and other kinds of pre-existing knowledge (such as domain-specific knowledge that is initially included in the system's knowledge base) in order to guide users to provide the knowledge that is relevant to those tasks (Blythe & Gil 2000). Our work has concentrated on plan evaluation and assessment tasks, but could be used with other task models. We have also developed a framework to guide users through typical sequences of changes or **Knowledge Acquisition Scripts** (KA Scripts) (Tallis & Gil, 1999). As the user interacts with a KA Script and enters knowledge step by step, additional KA Scripts may be activated that contain follow-up questions about that new knowledge. We have develop a library of general-purpose KA Scripts (Tallis & Gil, 1999), as well as with scripts customized to the general task models mentioned above (Blythe & Gil 2000).
- *Users do not know formal languages.*
  We have developed **English-based editors** that allow users to modify English paraphrases of the internal, more formal representations (Blythe & Ramachandran 1999). Users can only select the portions of the paraphrase that correspond to a valid expression in the internal language, and pick from a menu of suggested possible replacements for that portion. This approach enables the system to communicate with the user in English while circumventing



User memo:
Check List..
– define methods as general as possible

```
type (inst-of yes-no))
od (check
      (obj (spec-of force-ratio)
      (of ( main-task-of ( close
      (in ?c)
      ))
)
((name required-force-ratio-for
 (capability (estimate
          (obj (?f is (spec-of req
          (for (?s is (inst-of mili
          ))
 (result-type (inst-of number))
 (method (estimate
       (obj (spec-of required-fo
       (of ( operation-of ?t))
       (for ?s)
       ))
)
((name default-required-force-
 (capability (estimate
          (obj (?f is (spec-of req
          (of (?o is (inst-of milit
          (for (?ce is (inst-of mi
```

Undefined Capabilities
```
(check (obj (spec-of force-rat
Expected Result:(inst-of yes-no)
Callers:(check-force-ratio-for-main-task)
```

Individual Method Error
** LOCAL ERROR ** In the method to estimate the required force ratio for a military task, I don't know what ?T is

Method Sketch in Text
To do what(Capability): check force ratio of a military task in a coa
Do(Method Body):If the required force ratio is less than the current force ratio
the answer is yes(meaning the ratio is okay), otherwise the answer is no

```
name: _method69
capability:
( check ( obj ( ?v113 is ( spec-of force-ratio
    ( of ( ?v114 is ( inst-of military-task )
    ( in ( ?v115 is ( inst-of coa ) ) ) )
result-type:
(inst-of yes-no)
method:
( if SUB-GOAL then SOMETHING else SOMETHING )
```

Possible Selections
```
(add (obj EXPRESSION) (to EXPRESSION))
(estimate (obj (spec-of available-force-ratio)) (for EXPRESSION))
(compute (obj (spec-of combat-power)) (of EXPRESSION) (with-respect-to EXP
(estimate (obj (spec-of required-force-ratio)) (of EXPRESSION) (for EXPRESSIO
```

Figure 3: A screenshot from the EMeD Interface.

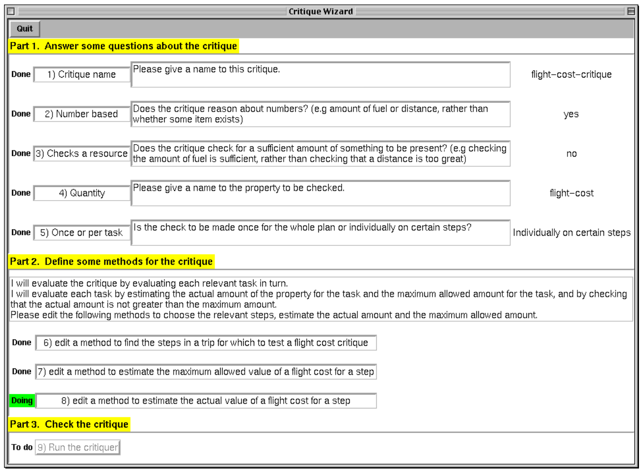
the challenges of full natural language processing.

EXPECT has several acquisition interfaces that exploit each of these approaches, and are implemented in a client-server architecture within the overall system. The rest of the paper describes some of these tools in more detail.

Exploiting Interdependency Models to acquire procedural knowledge

A theme of our KA research has been how KA tools can exploit *Interdependency Models* (Swartout & Gil 1995) that relate individual components of the knowledge base in order to develop expectations of what users need to add next. EMeD (EXPECT Method Developer) (Kim & Gil 1999; 2000), a knowledge acquisition tool to acquire problem-solving knowledge, exploits the Interdependency Models to guide users by helping them understand the relationships among the individual elements in the knowledge base. Our hypothesis is that Interdependency Models allow users to enter more knowledge faster, particularly for end users.

We analyzed the user tasks during knowledge base development and found several areas where an acquisition tool could help: (1) pointing out missing pieces at a given time; (2) predicting what pieces are related and how; (3) detecting inconsistencies among the definitions of the various elements in the knowledge base. We then expanded Interdependency Models, developing a set of techniques and principles that could guide users in both knowledge base creation and modification. The expectations result from enforcing constraints in the knowledge representation system, working out incrementally the interdependencies among the different components of the KB. As the user defines new *KB elements* (e.g., new problem-solving knowledge), the KA tool can form increasingly more frequent and more reliable



Part 1. Answer some questions about the critique

Done	1) Critique name	Please give a name to this critique.	flight–cost–critique
Done	2) Number based	Does the critique reason about numbers? (e.g amount of fuel or distance, rather than whether some item exists)	yes
Done	3) Checks a resource	Does the critique check for a sufficient amount of something to be present? (e.g checking the amount of fuel is sufficient, rather than checking that a distance is too great)	no
Done	4) Quantity	Please give a name to the property to be checked.	flight–cost
Done	5) Once or per task	Is the check to be made once for the whole plan or individually on certain steps?	Individually on certain steps

Part 2. Define some methods for the critique

I will evaluate the critique by evaluating each relevant task in turn.
I will evaluate each task by estimating the actual amount of the property for the task and the maximum allowed amount for the task, and by checking that the actual amount is not greater than the maximum amount.
Please edit the following methods to choose the relevant steps, estimate the actual amount and the maximum allowed amount.

Done	6) edit a method to find the steps in a trip for which to test a flight cost critique
Done	7) edit a method to estimate the maximum allowed value of a flight cost for a step
Doing	8) edit a method to estimate the actual value of a flight cost for a step

Part 3. Check the critique

To do	9) Run the critiquer

Figure 4: A screenshot from the PSMTool interface.

expectations. Figure 3 shows a screenshot from the EMeD interface.

When users define new procedural knowledge (problem-solving methods), EMeD first finds the interdependencies and inconsistencies within that element, such as if any undefined variable is used in the body of the procedure. If there are any errors within a definition, the *Local-Error Detector* displays the errors and it also highlights the incorrect definitions in red so that the user can be alerted promptly. The *Global-Error Detector* analyzes the knowledge base further and detects more subtle errors that occur in the context of problem solving. By keeping the interdependencies among the problem-solving methods and factual knowledge, and analyzing interdependencies between each method and its sub-method, the *Method Sub-method Analyzer* in EMeD can detect missing links and can find undefined problem-solving methods that need to be added. EMeD highlights those missing parts and proposes an initial version of the new methods. When the system is missing the knowledge, the *Method Proposer* in EMeD notifies the user and displays the ones needed to be defined. It can also construct an initial sketch of the new method to be defined. Users can search for existing methods that can achieve a given kind of capability using the *Method-Capability Hierarchy*, a hierarchy of method capabilities based on subsumption relations of their goal names and their parameters. Finally, EMeD can propose how the methods can be put together. By using the Method Sub-method Analyzer for analyzing the interdependencies among the KB elements, it can detect still unused problem-solving methods and propose how they may be potentially used in the system.

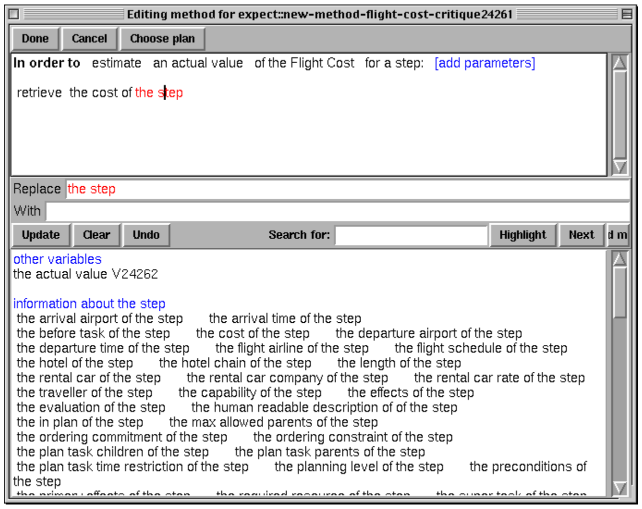
Our evaluations show an average over thirty percent time savings for different user groups, including end users with no formal computer science training. The results also show that KA tool that exploits Interdependency Models helped end users enter more knowledge.

Exploiting background knowledge about tasks

End users who are not programmers are often unable to modify procedural knowledge directly, even with interdependency-based tools. Often they are unsure where to even begin when specifying how to perform some task and understanding a language for procedural information is also an obstacle. We address these issues by guiding the user through the process of adding problem-solving knowledge using background knowledge about a generic task type such as a problem-solving method library. We also use an English-based method editor so that users can modify task knowledge without having to see and understand the EXPECT programming language. Using the combination of these two approaches, we have been able to acquire procedural knowledge from end users who were not programmers that was directly executable (Blythe & Gil 2000).

We use background knowledge about a task area in several different ways to guide users adding new knowledge in that area. One way it is used is to identify which of a set of generic procedures might be appropriate to specialize for a given task. In this case, a tool can help a user to get started with a question-answer dialog that concentrates on features of the task to be performed rather than on programming constructs. Then it can recommend a generic KBS from a library that goes some of the way to performing the task that the user wants to define. Here the required knowledge includes a set of generic procedures that are useful within a domain, information about appropriateness conditions for each one, about the information they need and about the relative strengths and weaknesses of alternative procedures. This information is in a form that can be operationalized as easily understandable questions for the user.

We also use background knowledge about a procedural task domain to help the user specialize a procedure once it has been selected. Some parts of a procedure should usually not be changed by a naive user since they define the way the generic task is performed. (These are often at the upper



Editing method for expect::new-method-flight-cost-critique24261
Done
Cancel
Choose plan

In order to estimate an actual value of the Flight Cost for a step: [add parameters]

retrieve the cost of the step

Replace the step
With

Update
Clear
Undo
Search for:
Highlight
Next
d m

other variables
the actual value V24262

information about the step
the arrival airport of the step the arrival time of the step
the before task of the step the cost of the step the departure airport of the step
the departure time of the step the flight airline of the step the flight schedule of the step
the hotel of the step the hotel chain of the step the length of the step
the rental car of the step the rental car company of the step the rental car rate of the step
the traveller of the step the capability of the step the effects of the step
the evaluation of the step the human readable description of of the step
the in plan of the step the max allowed parents of the step
the ordering commitment of the step the ordering constraint of the step
the plan task children of the step the plan task parents of the step
the plan task time restriction of the step the planning level of the step the preconditions of the step

Figure 5: A Screenshot from the English-based editor.

levels of a procedure definition.) Other parts are designed to be changed for a specific task, and are often represented as stubs. Knowledge about which parts of a task description can easily be changed and which should stay fixed can allow a tool to guide the user in modifying the procedure. One can also provide knowledge about the way that certain parts of the procedure are expected to be changed.

Notice that this is complementary to the Interdependency Model described in the last section. As a user modifies a generic task, a tool such as EMeD can help the user with information about missing pieces of the modified knowledge and about other parts that may need to be changed for consistency. Here, the background knowledge has solved the problem of how to start defining a task.

We built a problem-solving method library for plan evaluation and a KA tool called PSMTool that uses background knowledge in the ways we have described to guide users building new KBs for plan evaluation tasks (Blythe & Gil 2000). Figure 4 shows a screenshot from the PSMTool interface. In experiments in two different domains, end users were able to add new procedural knowledge for evaluating plans that could be directly executed. We found that users could complete on average 60% more tasks when using a full version of PSMTool than when using an ablated version that did not use background knowledge. They completed tasks on average 3 times as quickly with the full version.

Editing methods with an English-based editor

We also use an English-language based editor that presents problem-solving methods in an English paraphrase. The editor allows the user to select parts of the method as phrases of English and automatically creates a set of probable new fragments that the part can be changed to. These are presented for the user to select from, also in an English paraphrase. Thus users to modify task knowledge without having to see and understand the EXPECT programming language. Figure 5 shows a screenshot from this editor. The power of this editor comes from the combination of hiding the underlying language and providing a browsable list of useful program elements and can be used to specify arbitrary procedural knowledge (Blythe & Ramachandran 1999).

Evaluations with End Users

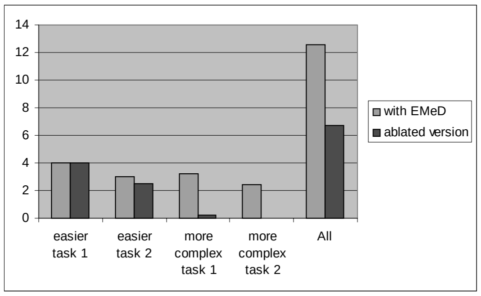
We wanted to put our KA tools to the test and understand how accessible they are to end users. Could end users use current KA tools to modify the problem-solving knowledge of a knowledge-based system? How much training do they need to start using such KA tools? Would they be able to understand and use a formal language? We performed several suites of experiments with different kinds of users that had no background in AI and programming. The study presented here shows results from testing Army officers using our KA tools to extend a Course of Action critiquer, and was part of the Knowledge Acquisition Critical Component Experiment that was conducted as part of the DARPA High Performance Knowledge Bases program (Cohen *et al.* 1998). More details on these and other studies acn be found in (Kim & Gil 2000; 2000; Blythe & Gil 2000; Tallis & Gil, 1999).

With EMeD, we spent 8 hours (two half-day sessions) for training. Users spent roughly half of that time learning EXPECT's language and how to put the procedural knowledge together. The rest of the time was spent learning about the KA tool and its ablated version. We believe that this time can be reduced by improving the tool's interface and adding on-line help. More training may be needed if users are expected to make much more complex changes to the knowledge base. At the same time, if they did not need to be trained on how to use an ablated version of the tool they would not need to learn as many details as our subjects did.

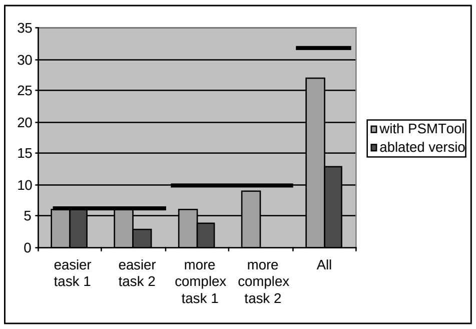
Our subjects got used to the language and could quickly formulate new procedural knowledge correctly. They did not seem to have a problem using some of the complex aspects of our language, such as the control structures (e.g., if-then-else statement) and variables. It took several examples to learn to express procedural knowledge into methods and sub-methods and to solve problems. EMeD helps this process by automatically constructing sub-method sketches and showing the interdependencies among the methods. Retrieving role values through composite relations was also hard. Providing a better way to visualize how to find this kind of information would be very useful.

In one experiment with PSMTool, users worked with the tool after having taken the training for EMeD, with an extra 45 minutes of training on PSMTool. In a second experiment users only saw PSMTool and received one hour of training that showed the tool and how to put procedural knowledge together using the English-based editor. These users were able to effectively use the tool to define new tasks, indicating that procedural knowledge can be directly specified by users with very little training. Again, the more complex the task knowledge, the more training may be required.

For each tool we created an ablated version that lacks Interdependency Models in the case of EMeD or background knowledge in the case of PSMTool. The differences in the number of tasks that users could complete are shown in Figure 6, both by task category and overall. Averaged across both tools, users could complete roughly twice as many tasks with the full versions as with the ablated versions.



(a) Number of tasks completed by users for EMeD



(b) Number of tasks completed by users for PSMTool

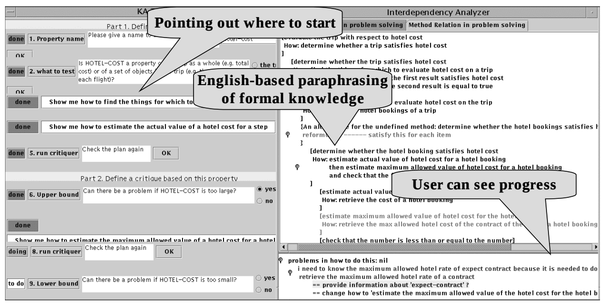
Figure 6: Number of tasks completed by users for EMeD (a) and PSMTool (b), with the full and ablated versions of the tools.

Discussion

One important issue is to identify what aspects of a knowledge base modification task are more challenging to end users when using KA tools. Some subjects using EMeD had difficulties in starting the KA tasks, when it does not point to a particular element of the KB to start with. Although they could use the search capability or look up related methods in the tool, this was more difficult for them than the cases when the elements or problems to be resolved were already present in the KB and could be highlighted by the tool. PSMTool is able to provide more help for starting a task, but only in cases where pre-specified generic task knowledge can be appropriately modified.

Typically, a KA task involves more than one step, and sometimes subjects are not sure if they are on the right track even if they have been making progress. Here PSMTool is helpful because it keeps track of what users are doing in the context of the overall task and lets the user visualize their progress. KA tools that help with this in a broader range of situations would be very helpful.

Another important issue is whether KA tools need to be different for users with different degrees of background in computer science. We did not know whether end users would need a completely different interface from knowledge engineers. It seems that a few improvements to the presentation in order to make the tool easier to use was all


Pointing out where to start
English-based paraphrasing of formal knowledge
User can see progress

```
Part 1. Defi
done  1. Property name   Please give a name t        otel-cost
OK
done  2. what to test    Is HOTEL-COST a property o      p as a whole (e.g. total   ○ the t
                         cost) or of a set of objects      trip (e.g.
                         each flight)?
OK
done  Show me how to find the things for which to
done  Show me how to estimate the actual value of a hotel cost for a step
done  5. run critiquer   Check the plan again   OK
Part 2. Define a critique based on this property
done  6. Upper bound     Can there be a problem if HOTEL-COST is too large?   ● yes  ○ no
done
Show me how to estimate the maximum allowed value of a hotel cost for a hotel
doing 8. run critiquer   Check the plan again   OK
to do 9. Lower bound     Can there be a problem if HOTEL-COST is too small?   ○ yes  ○ no
```

```
Interdependency Analyzer
n problem solving | Method Relation in problem solving
[evaluate the trip with respect to hotel cost
 How: determine whether a trip satisfies hotel cost
]
   [determine whether the trip satisfies hotel cost
                    hich to evaluate hotel cost on a trip
                    he first result satisfies hotel cost
                    e second result is equal to true

                    evaluate hotel cost on the trip
      Ho            hotel bookings of a trip
      ]
      {An al        e for the undefined method: determine whether the hotel bookings satisfies h
       reform       ------ satisfy this for each item
      }
         [determine whether the hotel booking satisfies hotel cost
          How: estimate actual value of hotel cost for a hotel booking
                then estimate maximum allowed value of hotel cost for a hotel booking
                and check that the f
         ]
            [estimate actual value
             How: retrieve the cost of a hotel booking
            ]
            [estimate maximum allowed value of hotel cost for the hotel
             How: retrieve the max allowed hotel cost of the contract of the        a hotel booking
            ]
            [check that the number is less than or equal to the number]
problems in how to do this: nil
  i need to know the maximum allowed hotel rate of expect contract because it is needed to do
   retrieve the maximum allowed hotel rate of a contract
        == provide information about 'expect-contract' ?
        == change how to 'estimate the maximum allowed value of the hotel cost for the hotel b
```

Figure 7: Screenshot from the new integrated EXPECT interface

they needed. The editors that help the users avoid syntax problems were important. In interviews we found that end users consistently preferred a version that used an English paraphrase, but we have not tested whether this leads to improved performance.

On the other hand, we were surprised that end users found some of the features useful when we had expected that they would cause confusion. For example, a feature we thought would be distracting is organizing procedural knowledge into a hierarchy, but it seemed more useful for the end users than for knowledge engineers.

Although EMeD and PSMTool are proactive in providing guidance, we believe that some users would perform better if we used better visual cues or pop-up windows to show the guidance. As the users are more removed from the details, the KA tool needs to do a better job at emphasizing and making them aware of what is important.

We are currently integrating the tools described here into a unified acquisition interface. Figure 7 shows a screenshot of a preliminary version of the new integrated acquisition interface for EXPECT. The complementary capabilities of the different approaches will help us create a more comprehensive environment for acquiring procedural knowledge. We are also beginning research into verification and validation techniques that will extend the scope of our existing approaches.

Acknowledgments

We gratefully acknowledge the support of DARPA with grant F30602-97-1-0195 as part of the DARPA High Performance Knowledge Bases program, and with grant F30602-00-2-0513 as part of the DARPA Active Templates program.

References

Bareiss, R.; Porter, B.; and Murray, K. 1989. Supporting start-to-finish development of knowledge bases. *Machine Learning* 4:259–283.

Blythe, J., and Gil, Y. 2000. Extending the role-limiting approach: Supporting end-users to acquire problem-solving knowledge. In *Proc. ECAI 2000 workshop on Applications of Ontologies and Problem-Solving Methods*.

Blythe, J., and Ramachandran, S. 1999. Knowledge acquisition using and english-based method editor. In *Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

Cohen, P.; Schrag, R.; Jones, E.; Pease, A.; Lin, A.; Starr, B.; Gunning, D.; and Burke, M. 1998. The DARPA High Performance Knowledge Bases Project. *AI Magazine* 19(4).

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

Gil, Y., and Melz, E. 1996. Explicit representations of problem-solving strategies to support knowledge acquisition. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Cypher, A. 1993. *Watch what I do: Programming by demonstration*. MIT Press.

Kim, J., and Gil, Y. 1999. Deriving expectations to guide knowledge-base creation. In *Proc. Sixteenth National Conference on Artificial Intelligence*. AAAI Press.

Kim, J., and Gil, Y. 2000. User studies of an interdependency-based interface for acquiring problem-solving knowledge. In *Proceedings of the Intelligent User Interface Conference*, 165–168.

Kim, J., and Gil, Y. 2000. Acquiring problem-solving knowledge from end users: Putting interdependency models to the test. In *Proc. Seventeenth National Conference on Artificial Intelligence*. AAAI Press.

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

Mitchell, T.; Mahadevan, S.; and Steinberg, L. 1985. LEAP: A learning apprentice for VLSI design. In *Proceedings of the 1985 International Joint Conference on Artificial Intelligence*.

Swartout, W. R., and Gil, Y. 1995. Expect: Explicit representations for flexible acquisition. In *Proc. Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

Tallis, M. and Gil, Y. 1999. Designing scripts to guide users in modifying knowledge-based systems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence,* Orlando, FL.