

# User Studies of an Interdependency-Based Interface for Acquiring Problem-Solving Knowledge

**Jihie Kim and Yolanda Gil**  
USC/Information Sciences Institute  
4676 Admiralty Way  
Marina del Rey, CA 90292  
{jihie,gil}@isi.edu

## **ABSTRACT**

This paper describes a series of experiments with a range of users to evaluate an intelligent interface for acquiring problem-solving knowledge to describe how to accomplish a task. The tool derives the interdependencies between different pieces of knowledge in the system and uses them to guide the user in completing the acquisition task. The paper describes results obtained when the tool was tested with a wide range of users, including end users. The studies show that our acquisition interface saves users an average of 32% of the time it takes to add new knowledge, and highlight some interesting differences across user groups. The paper also describes what are the areas that need to be addressed in future research in order to make these tools usable by end users.

## **INTRODUCTION**

End user programming is an active area of research that poses many challenges for intelligent user interfaces. There have been a wide range of approaches, including programming by demonstration [Cypher 1993], case-based reasoning [Bareiss et al 1989], and learning apprentices [Mitchell et al. 1985]. These systems observe users perform a task and use various induction techniques to generalize from the observed examples and create a representation of the task that can be used to automate it for the user in the future. These approaches work well for relatively simple tasks. However, for complex problem solving activities example-based acquisition of procedural knowledge may be tedious and ultimately impractical, since a very large amount of examples may be required in order to provide enough information to draw adequate generalizations.

We are investigating an alternative and complementary approach to develop acquisition interfaces for problem-solving knowledge that enables users to specify new knowledge directly by using a knowledge editor and associated tools. This paper reports our work on EMeD [Kim and Gil 1999], an acquisition interface that supports users in adding problem-solving knowledge to a knowledge-based system. The tool helps users understand the relationships among the individual elements in the knowledge base, keep track of missing knowledge that still needs to be added, suggest potential uses of newly defined elements, and detect and resolve errors early on.

Since our ultimate goal is to create an acquisition interface for end users (i.e., without formal training in computer science, artificial intelligence, or knowledge bases), we decided to perform some experiments to gather data regarding the usability of our interface for end users. This kind of study would let us learn more about the needs of end users. Would they understand how to express procedural knowledge into methods and sub-methods? Could they do this without the context of examples? Would they be able to manipulate our formal language, or would they require a completely different kind of interface such as an English-based editor? Would they need a radically different interaction (perhaps with much stronger guidance) to be able to perform the same kinds of tasks? Would they need additional functionality that users with formal computer science background do not need? We did not know what to expect from this group of users, and were prepared to see that they are unable to perform the tasks using our current version of the acquisition interface. We conducted experiments with a wide range of users: experienced knowledge engineers, users familiar with knowledge based systems, users with formal computer science training but little background in knowledge bases, and finally end users with no formal training in any of the above areas.

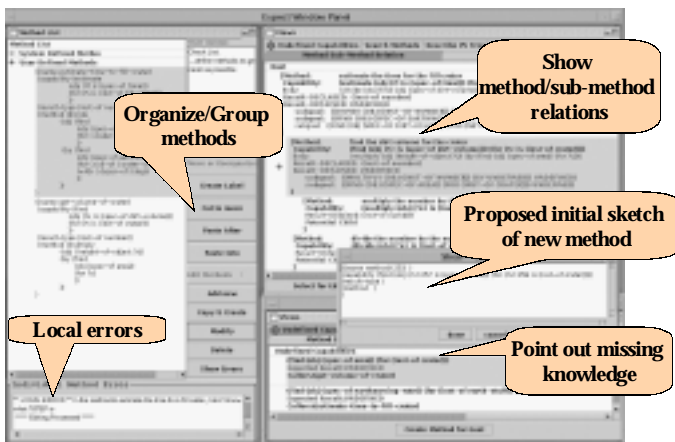
This paper describes the results of these experiments and summarizes some of the insights that we obtained about interfaces for editing procedural knowledge. The results should be relevant to researchers of end user programming, knowledge-based systems environments, and intelligent interfaces at large.

## **EMeD: A METHOD ACQUISITION INTERFACE FOR THE EXPECT ARCHITECTURE**

*EMeD* (EXPECT Method Developer) [Kim and Gil 1999] is a knowledge acquisition interface that allows users to specify problem-solving knowledge within the EXPECT framework [Gil 1994; Swartout and Gil 1995; Gil and Melz 1997]. This section provides a short overview of EMeD, more details can be found in [Kim and Gil 1999].

When users add new knowledge to a system, an intelligent acquisition interface should help them understand 1) how the new knowledge fits given the knowledge that is already there, and 2) what additional knowledge is still missing and needs to be provided by the user. EXPECT addresses these

issues by analyzing the knowledge in the system and *automatically deriving the interdependencies* between all the elements of the knowledge base. The key to knowledge acquisition is then to guide the user in understanding these interdependencies and in providing enough information to make all the individual pieces work together as intended. EMeD was developed to support users in different knowledge acquisition activities, from making small changes to significantly extending an existing knowledge base or creating new knowledge base. We analyzed the user's tasks during knowledge base development and found several areas where an acquisition interface 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 developed a set of techniques and principles that could guide users in both knowledge base creation and modification.



**Figure 1: A screenshot of EMeD's interface**

Figure 1 shows snapshots from the EMeD interface. Users can use the *method seeker* to find existing methods related with particular terms or functionality. The *method organizer* allows users to define classes and groups of methods. When a new method is defined, the method is checked for *local errors*. If there is no error, the *method sub-method analyzer* creates links between the new method and existing methods and shows the user the dependencies among them in terms of which super-method can call which sub-method. Whenever there is any missing knowledge detected, it is highlighted with a red diamond. The *undefined method proposer* generates an initial sketch of the missing knowledge based on an analysis of what is expected to be added. The *global error detector* analyzes the knowledge base further and detects more subtle errors in the context of problem solving.

### EXPERIMENTAL DESIGN AND SETUP

An overview of the methodology that we are developing to evaluate acquisition interfaces is described in [Tallis et al. 1999].

The hypotheses that we wanted to test were:

1. Users will be able to complete tasks in **less time** using the EMeD acquisition interface.
2. The reduction in completion time will be more noticeable for **less experienced users**.

The subjects were divided into four groups:

1. **users familiar with the knowledge base environment** (Group 1), i.e., who had previous experience in developing knowledge-based systems in EXPECT.
2. **users familiar with related AI technology** (Group 2), i.e., who were familiar with ontologies and knowledge-based systems, but who had never developed systems with EXPECT.
3. **users trained in CS** (Group 3) who were not familiar with either knowledge-based systems or EXPECT.
4. **users with no formal CS background** (Group 4), who are familiar with software tools such as spreadsheets and HTML editors (our project assistants).

Notice that **end users** were in Group 4. We tested four subjects in Group 1, two subjects in Group 2, four subjects in Group 3, and two subjects in Group 4. None of the subjects had used or seen EMeD before the experiment. Each subject was tested under two conditions: using EMeD and using an ablated version of EMeD that only allowed them to edit methods and did not have any other functionality from EMeD. In order to prevent a transfer effect, each subject used a different scenario for each condition. Different subjects were given the scenarios and tools in different orders to reduce the influences from familiarity with tools or fatigue. The two scenarios were comparable in order to make the results of the experiment meaningful, and involved adding a different set of methods to the same initial knowledge base. This knowledge base is of considerable complexity, and was developed by our group to participate in the evaluation of the DARPA High-Performance Knowledge Bases program that investigates the development of large-scale knowledge based systems [Cohen et al 1998]. The problem that we addressed was analyzing enemy workarounds to a damaged target.

Each subject was given a tutorial of the tools with simpler scenarios and was allowed a period of practice with both tools. During the practice, each subject was asked to perform a simple acquisition task with both tools. We instrumented the tools to record the actions performed by the subjects. We took detailed transcripts of their activities and the comments they voiced out loud as they were performing the tasks.

### RESULTS

Before we report on the results concerning the performance during the experiment, it is worth describing the differences across subject groups during the training and practice periods. The training time we measured includes the time spent on the tutorial and the practice with the tools. Group 1 users were not given the tutorial because they were already familiar with the language, and it took them an

average of two hours to practice with EMeD. Group 2 needed some time to learn the representation language (3.5 hrs), and Group 3 needed more time for both phases (4 hrs). The training time for Group 4 was almost twice the time (7.7 hrs) for Group 3 and almost 4 times of what Group 1 had.

A few observations about training the subjects in Group 4 are worth mentioning. First, teaching the representation language was not easy although this was hardly a surprise for us. Because of their lack of background on organizing and structuring algorithms, we decided that these subjects would need support in breaking down the statements of the new knowledge that they were given and organizing them into methods. We suggested that they write on paper first what they intended to put in each method. (We are now developing an interface to support this.)

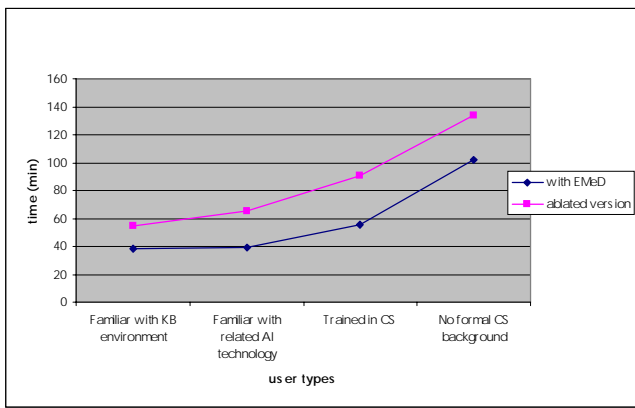


Figure 2: Average time to complete KA tasks

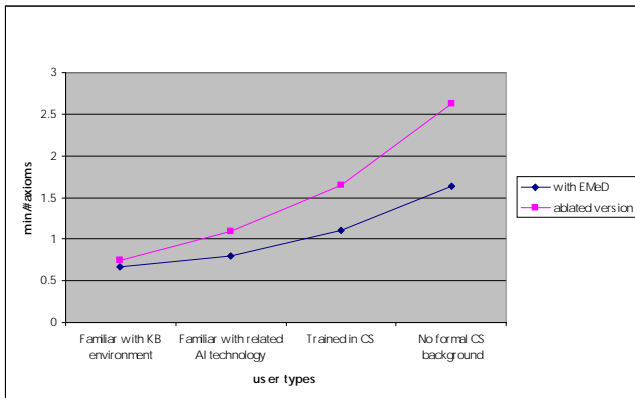


Figure 3: Average time per axiom

Figure 2 shows that EMeD reduced the time to complete the tasks by an average of 32%. Because the subjects performed the task in different ways, they ended up adding a different number of methods with different numbers of axioms and of different kinds. The subjects added 10 to 14 methods for the two tasks, building 102 to 133 axioms for them. We decided to measure the average amount of knowledge added in terms of new axioms in the knowledge base, following common practice in knowledge base

research. Figure 3 shows the average time to add an axiom. The overall time and the overall time savings increase when the subjects have less experience in knowledge-based systems and computer science. The time savings provided by the tool is more acute for users with no CS background.

For some subjects in Group 1, the time per axiom was not very different in both conditions. Our hypothesis is that since they were proficient in the language and the KB environment and were given relatively small sized tasks, they were able to keep track of the interdependencies in their minds without needing the help that the tool provides. We expect that the time difference will be larger when tested with more realistic scenarios, where the users would need to add more knowledge and keep track of many more interdependencies.

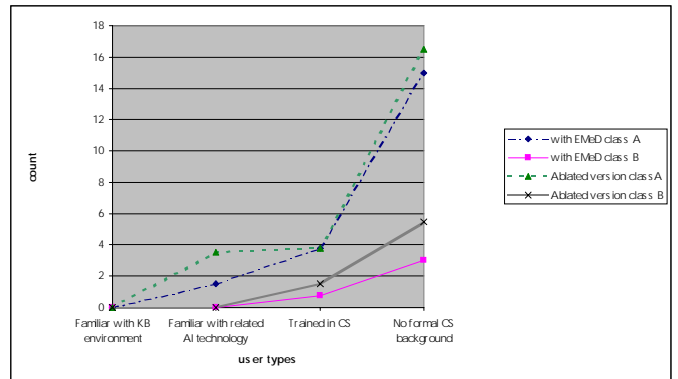


Figure 4: Average number of interventions

We allowed some interventions of the experimenter so that the users would not go way off track and take more time to finish the task than they had patience and inclination to spend. There were various types of interventions that we divided into two major classes. Class A interventions were simple hints on language and syntax, and were considered less grave. For example, the experimenter would indicate where to put parentheses in a goal description. It is worth noting that had the experimenter not intervened, the tool would have indicated a syntax error and the users would have had to look up the syntax and fix the error, something we believe they would be capable of doing. Detecting and fixing these errors is not a major goal of the tool, and we have developed form-based interfaces for this purpose that have not yet been integrated in EMeD. Class B interventions were of more serious nature and were done when the user was not making any progress during the experiment. For example, Group 4 users often asked for help to compose sub-goal descriptions to represent a goal. We predict that, if we had not given the help, the times in Figure 2 for this user group would have grown to a very large number. These interventions point to new functionality that future versions of our interface should provide to users.

Figure 4 shows the number of interventions of Class A and Class B during the experiment. Overall, subjects not using

EMeD required a larger number of Class B interventions. As shown in Figure 4, the increase from Group 1 to Group 2 and Group 3 was moderate, but Group 4 needed a significantly larger amount when not using EMeD (about four times more than Group 3.)

We also analyzed the use of each of the components of EMeD throughout the experiments. Figure 5 shows the average number of times that each component was used for each user group. The local error detector was the component most used, mostly for errors within a method. It effectively displayed errors within a method definition, and subjects in all groups used it whenever they introduced an error in creating or modifying a method. The undefined method proposer was the second most used in the experiment. Subjects used it to see what methods remained to be built and to create new methods called by already defined methods. This component seems more useful when the users build methods in top-down fashion, and was used more often for users with some AI background. It is interesting to notice that some users in Group 4 took what seemed like a bottom-up approach to develop the methods.

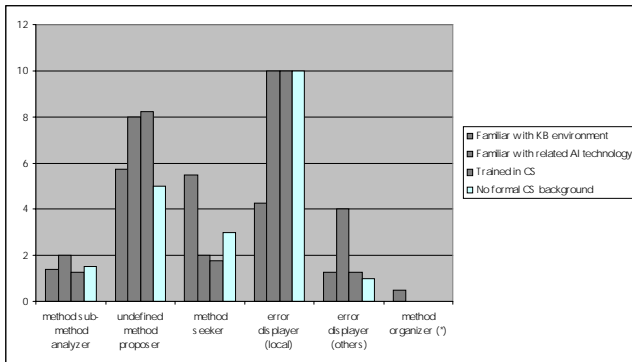


Figure 5: Average use of EMeD features

The method seeker was found useful to find system-defined methods, such as arithmetic and logic operations, but the search capability was not used very often. We believe that this is because of the small number of methods involved in the experiments. We expect that this functionality will be more useful in practice when larger numbers of methods are added.

The method sub-method analyzer was used, but not as many times as we expected. The subjects felt that it was hard to understand the way these relationships are displayed. With more exposure to the interface, we hope it can become easier to understand. Perhaps a more graphical view, or a more summarized view would be more adequate.

The method organizer was only provided to Group 1, because we thought others would not really use it and because we thought that providing too many features would divert their attention and possibly cause confusion. However, subjects in the other groups ended up asking for ways to do this, because they thought it would help them to

collect related methods in one place and look at methods in groups without being distracted by other methods.

## CONCLUSIONS

Perhaps the most remarkable result of these experiments is that end users were able to complete the complex tasks they were given. The interventions of the experimenters were a crucial factor for this, but they were punctual and concerned with very specific steps during the process and that we hope to be able to support in future versions of our acquisition interfaces. The fact that these subjects were able to take charge of the task and execute it to completion is in itself a very encouraging result.

The results reported here provided us with very important feedback about how to build our acquisition interfaces. Based on these results, we have already developed a new version of EMeD and have successfully tested it with domain experts (Army officers) extending a knowledge base as part of the DARPA High Performance Knowledge Bases Knowledge Acquisition Critical Component Experiment. The kinds of user evaluations that we are conducting and that we report here are crucial to understand how intelligent interfaces can help end users in the complex task of extending a system's knowledge.

## ACKNOWLEDGMENTS

We would like to thank Marcelo Tallis for his tremendous help during the experiment. Also, we are indebted to the many subjects that have participated in our experiments for their time and their patience. 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 contract DABT63-95-C-0059 as part of the DARPA/Rome Laboratory Planning Initiative.

## REFERENCES

- Bareiss, R., Porter, B., and Murray, K. (1989). Supporting start-to-finish development of knowledge bases. *Machine Learning*, 4:259-283.
- 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).
- Cypher, A. (1993) *Watch what I do: Programming by demonstration*. Allen Cypher, Ed. MIT Press.
- Gil, Y. and Melz, E. (1996) Explicit representations of problem-solving strategies to support knowledge acquisition. In *Proceedings of AAAI-96*.
- Gil, Y. (1994) Knowledge refinement in a reflective architecture. In *Proceedings of AAAI-94*.
- Kim, J. and Gil, Y. (1999) Deriving expectations to guide knowledge base creation. In *Proceedings of AAAI-99*.
- Mitchell, T., Mahadevan, S. and Steinberg, L. (1982) LEAP: A learning apprentice for VLSI design. In *Proceedings of IJCAI-85*.
- Tallis, M., Kim, J. and Gil, Y. (1999) User studies of knowledge acquisition tools: Methodology and lessons learned. In *Proceedings of KAW-99*.
- Swartout B. and Gil, Y. (1995) EXPECT: Explicit representations for flexible acquisition. In *Proceedings of KAW-95*.

