Using Interdependency Models to Acquire Problem Solving Knowledge

Jihie Kim and Yolanda Gil

Information Sciences Institute
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA
jihie@isi.edu, gil@isi.edu

Abstract

A knowledge acquisition approach that has been very effective to develop tools that acquire knowledge from users is to use expectations of what users have to add or may want to add next based on how new knowledge fits within a knowledge base that already exists. When a knowledge base is first created or undergoes significant extensions and changes, these tools cannot provide much support. This paper describes our approach to developing knowledge acquisition tools that guide users in creating problem-solving knowledge. We analyzed the user tasks during knowledge base development and found several ways of guiding users by helping them understand the relationships among the individual elements in the knowledge base (called the *Interdependency Models*). We implemented a tool called EMeD that exploits these Interdependency Models in several ways: pointing out missing pieces at a given time, predicting what pieces are related and how, and detecting inconsistencies among the definitions of the various elements in the knowledge base. We have evaluated our implementation with different user groups, including end users with no formal computer science training and the results show an average over thirty percent time savings. The results also show that the tool that exploits Interdependency Models helped end users enter more knowledge. We discuss additional lessons we have learned that may be useful to knowledge-acquisition researchers.

Some material in this paper is also reported in the Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-1999) or in the Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000).

1 Introduction

Knowledge acquisition (KA) is recognized as an important research area for making knowledge-based AI succeed in practice (Feigenbaum, 1993). In recent years, researchers have investigated a variety of new approaches to develop KA tools. An approach that has been very effective is to use *expectations* of what users have to add or may want to add next (Eriksson *et al.*, 1995; Birmingham & Klinker, 1993; Marcus & McDermott, 1989; Davis, 1979). Most of these expectations are derived from the *interdependencies* among the components in a knowledge-base system (KBS). EXPECT (Gil & Melz, 1996; Swartout & Gil, 1995) and Protégé-II (Eriksson *et al.*, 1995) use interdependencies between factual knowledge and problem-solving methods to find related pieces of knowledge in their KBS and create expectations from them. To give an example of these expectations, suppose that the user is building a KBS for a configuration task that finds constraint violations and then applies fixes to them. When the user defines a new constraint, the KA tool has the expectation that the user should specify possible fixes for cases where the constraint is violated, and helps the user do so. These tools can successfully build expectations because there is already a body of knowledge where the new knowledge added by the user must fit in. In the configuration example, there would be problem-solving knowledge about how to solve configuration tasks (how to describe a configuration, what is a constraint, what is the relation between a constraint and a fix, how to apply a fix, etc.)

However, much of the work was done to support users in modifying already existing knowledge bases or extending factual knowledge for pre-defined problem-solving methods(Eriksson *et al.*, 1995; Birmingham & Klinker, 1993; Marcus & McDermott, 1989; Tallis & Gil, 1999). When a new knowledge base (KB) is created (or when an existing one is significantly extended) there is little or no pre-existing knowledge in the system to draw from. How can a KA tool support the user in creating a large body of new knowledge? Are there any sources of expectations that the KA tool can exploit?

We present our approach to developing KA tools that derive expectations from the KB in order to guide users during KB creation. We performed an analysis of why KB creation is hard and we found several ways of guiding users by helping them understand the relationships among the individual elements in the knowledge base (called the *Interdependency Models*). We implemented a tool called EMeD that exploits these Interdependency Models in several ways: pointing out missing pieces at a given time, predicting what pieces are related and how, and detecting inconsistencies among the definitions of the various elements in the knowledge base. As the user defines new *KB elements* (i.e., new concepts, new relations, new problem-solving knowledge), the knowledge acquisition tool derives more frequent and more reliable expectations.

We performed a set of preliminary experiments with different types of users, including experienced knowledge engineers, AI researchers with some background on knowledge base development, users with strong computer science background, and end users without programming experience. We analyzed how the tools affected the performance of the different groups of the users. The results show that our tool was able to reduce the time by over 30% on average, with less experienced users spending more time to finish the acquisition tasks that they were given.

The paper begins by describing why KB creation is hard and presents our approach to developing KA tools that derive expectations from the KB in order to guide users during KB creation. The following section presents EMeD, the acquisition tool we developed illustrating the features we have implemented. Then we describe our experimental design and setup, and show the results from our experiments with several different subject groups. We then describe results with domain experts in the field. Finally, we extract some general conclusions that motivate future directions of knowledge acquisition research.

2 Approach

This section begins by describing difficulties in KB creation based on our experience. Then we map the difficulties to a set of interdependency models that can guide users.

2.1 Difficulties in Knowledge Bases Development

There are several reasons why creating or significantly extending a knowledge base is hard:

- **Developers have to design and create a large number of KB elements.** KB developers have to turn models and abstractions about a task domain into individual KB elements. When they are creating an individual KB element, it is hard to remember the details of all the definitions that have already been created. It is also hard to anticipate all the details of the definitions that remain to be worked out and implemented. As a result, many of these KB elements are not completely flawless from the beginning, and they tend to generate lots of errors that have unforeseen side effects. Also, until a KB element is debugged and freed from these errors, the expectations created from it may not be very reliable.
- **There are many missing pieces of knowledge at a given time.** Even if the developers understand the domain very well, it is hard to picture how all the knowledge should be expressed correctly. As some part of the knowledge is represented, there will be many missing

pieces that should be completed. It is hard for KB developers to keep track of what pieces are still missing, and to take them into account as they are creating new elements.

- **It is hard to predict what pieces of knowledge are related and how.** Since there is not a working system yet, many of the relationships between the individual pieces are in the mind of the KB developer and have not been captured or correctly expressed in the KB.
- **There can be many inconsistencies among related KB elements that are newly defined.** It is hard for KB developers to detect all the possible conflicts among the definitions that they create. Often times they are detected through the painful process of trying to run the system and watching it not work at all. The debugging is done through an iterative process of running the system, failing, staring at various traces to see what is happening, and finally finding the cause for the problem.

As intelligent systems operate in real-world, large-scale knowledge intensive domains, these problems are compounded. As new technology enables the creation of knowledge bases with thousands and millions of axioms, KB developers will be faced an increasingly more unmanageable and perhaps impossible task. Consider an example from our experience with a Workaround Generation domain selected by DARPA as one of the challenge problems of the High-Performance Knowledge Bases program that investigates the development of large-scale knowledge based systems. The task is to estimate the delay caused to enemy forces when an obstacle is targeted by reasoning about how they could bypass, breach or improve the obstacle. After several large ontologies of terms relevant to battlespace reasoning were built (military units, engineering assets, transport vehicles, etc.), we faced the task of creating the problem-solving knowledge base that used all those terms and facts to actually estimate the workaround time. We built eighty-four problem-solving methods from scratch on top of several thousand defined concepts, and it took two intense months to put together all the pieces. Figure 1 shows some examples of our methods. Each method has a *capability* that describes what goals it can achieve, a *method body* that specifies the procedure to achieve that capability (including invoking subgoals to be resolved with other methods, retrieving values of roles, and combining results through control constructs such as conditional expressions), and a *result type* that specifies the kind of result that the body is expected to return (more details of their syntax are discussed below). Creating each method so that it would use appropriate terms from ontologies was our first challenge. Once created, it was hard to understand how the methods were related to each other, especially when these interdependencies result from the definitions in the ontologies. Despite the modular, hierarchical design of our system, small errors and local inconsistencies tend to blend together to produce inexplicable results making it very hard to find and to fix the source of the problems. Although some portions of the knowledge base could be examined locally by testing

```
(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 earthmoving rate.")
 (capability (estimate (obj (?t is (spec-of time)))
                       (for (?s is (inst-of narrow-gap)))))
 (result-type (inst-of time))
 (body (divide (obj (compute (obj (spec-of dirt-volume))
                             (for ?s)))
               (by (compute (obj (spec-of earthmoving-rate))
                            (for ?s))))))

(define-method M2
 (documentation "The earthmoving rate for a workaround step that involves bulldozing region is
the combined bulldozing rate of the bulldozer specified as bulldozer-of of the step.")
 (capability (compute (obj (?r is (spec-of earthmoving-rate)))
                      (for (?s is (inst-of bulldoze-region)))))
 (result-type (inst-of rate))
 (body (find (obj (spec-of standard-bulldozing-rate))
             (of (bulldozer-of ?s)))))

(define-method M3
 (documentation "The standard bulldozing rate of a military bulldozer can be found by
looking up the military-dozer-rate-table for the bulldozer.")
 (capability (find (obj (?r is (spec-of standard-bulldozing-rate)))
                   (for (?b is (inst-of military-bulldozer)))))
 (result-type (inst-of bulldozing-rate))
 (body (look-up (obj (military-dozer-rate-table))
                (for ?b))))
```

FIGURE 1: Methods in a simplified workaround generation domain.

subproblems, we often found ourselves working all the way back to our own documentation and notes to understand what was happening in the system.

In summary, it is hard for KB developers to keep in mind all the definitions that they create and to work out their interdependencies correctly. KB developers generate and resolve many errors while they build a large body of knowledge. Our goal is to develop KA tools that help users resolve these errors and, more importantly, help them avoid making the errors in the first place.

2.2 Approach: Interdependency Models

We identified several sources of Interdependency Models that KA tools can exploit in order to guide users in creating a new knowledge base. We explain our approach in terms of the problems and examples described in the previous subsection.

- *Difficulty in designing and creating many KB elements ⇒ Guide the users to avoid errors and look up related KB elements.*

  First, each time a KB element is created by a user, we can check the dependencies within the element and find any potential errors based on the given representation language. For

example, when undefined variables are used in method body, this will create an expectation that the user needs to define them in the method.

In our example, Method M1 has two variables, ?t and ?s, defined in its capability, and if the method body uses a different variable, the system can send a warning message to the user. Likewise, if a concept definition says that a role can have at most one value but also at least two values, then this local inconsistency can be brought up. By isolating these local errors and filtering them out earlier in the KB development process, we can prevent them from propagating to other elements in the system.

However small the current KB is, if there are KB elements that *could be* similar to the one being built, then they can be looked up to develop expectations on the form of new KB element. For example, developers may want to find existing KB elements that are related with particular terms or concepts based on the underlying ontology. If there is a concept hierarchy, it will be possible to retrieve KB elements that refer to superconcepts, subconcepts, or given concepts and let the user develop expectations on the current KB element based on related KB elements. For example, if a developer wants to find all the methods related to moving earth, the system can find the above methods, because narrow-gap and bulldoze-region are subtypes of move-earth. When the user adds a new method about moving earth to fill a crater, instead of to narrow a gap, then it may be useful to take them into account. Specifically, M1 can generate expectations on how a method for estimating time to fill a crater should be built.

- *Many pieces of knowledge are missing at a given time: ⇒ Compute surface relationships among KB elements to find incomplete pieces and create expectations from them*

  The KA tool can predict relationships among the methods based on what the capability of a method can achieve and the subgoals in the bodies of other methods. For example, given the three methods in Figure 1, method M1 can use M2 for one of its subgoals — compute earthmoving rate. These relationships can create method-submethod trees that are useful to predict how methods will be used in problem solving. In the process of building this kind of structure, the system can expose missing pieces in putting the methods together. For example, *unmatched subgoals* can be listed by collecting all the subgoals in a method that cannot be achieved by any of the already defined methods. The user will be reminded to define the missing methods and shown the subgoals that they are supposed to match. In Figure 1, if a method for the other subgoal of method M1 (compute dirt volume) is not defined yet, the user is asked to define one.

  Similarly, if a concept is used in a KB element definition but not defined yet, then the system will detect the *undefined concept*. Instead of simply rejecting such definition, if the

developer still wants to use the term, the KA tool can collect undefined concepts and create an expectation that the developer (or other KB developers) will define the term later.

- *Difficulty in predicting what pieces of knowledge are related and how ⇒ Use surface relationships to find unused KB elements and propose potential uses of the elements*

  The above surface relationships among KB elements, such as method-submethod relationship can also help detect unused KB elements. If a method is not used by any other methods, then it can be collected into an *unused-method list*. In addition to finding such unused methods, the KA tool can propose potential uses of it. For example, if the capability of a method is similar to one of the unmatched subgoals (e.g., same goal name and similar parameter types), then a potential user of the method will be the method that has the unmatched goal. For example, method M3 can be potentially used to solve a subgoal of M2 (find standard-bulldozing rate of a given bulldozer), because it can find standard-bulldozing rate of military bulldozers (a sub-type of bulldozer).

  In the same way, concepts created but not referred to in any other definitions can be collected into an *unused-concept list*. The KA tool can develop expectations of KB elements that will use the definitions or perhaps even deleting these concepts if they end up being unnecessary.

- *Inconsistencies among newly defined KB elements ⇒ Help users find them early and propose fixes*

  The KA tool can check if the user-defined result type of a method is inconsistent with what the method body returns based on the results of the subgoals. If there are inconsistent definitions, the system will develop an expectation that user has to modify either the current method or the methods that achieve the subgoals.

  Also, for concept definitions, there can be cases where a user wants to retrieve a role value of a concept, but the role is not defined for the concept. In addition to simply detecting such a problem, the system may propose to define the role for the concept or to change the method to refer to a different but related concept that does have that role.

  Finally, once the KA tool indicates that there are no errors, inconsistencies, or missing knowledge, the user can run the inference engine, exposing additional errors in solving a given problem or subproblems. The errors are caused by particular interdependencies among KB elements that arise in specific contexts. If most of the errors are detected by the above analyses, users should see significantly fewer errors at this stage.

Notice that as the KB is more complete and more error-free it becomes a stronger basis for the KA tool in creating expectations to guide the user.

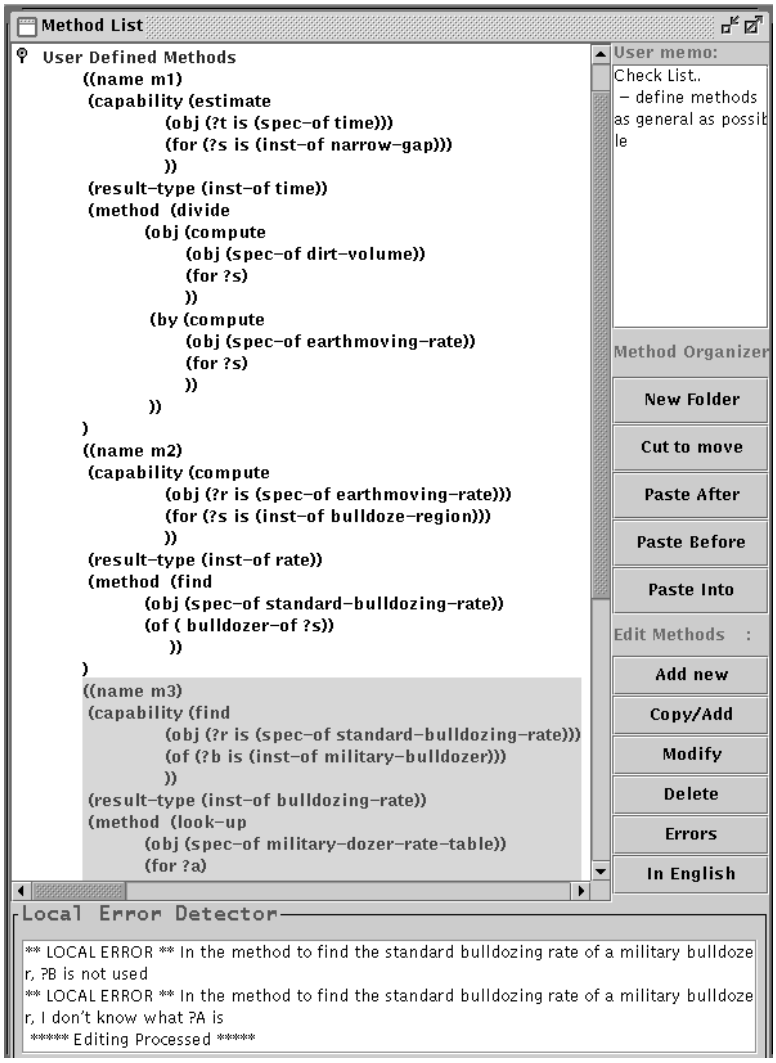


FIGURE 2: EMeD Interface (Editing window).

3 EMeD: Expect Method Developer

We have concentrated our effort in developing a KA tool that uses these kinds of expectations to support users to develop problem-solving methods. We built a KA tool called *EMeD* (EXPECT Method Developer) for the EXPECT framework for knowledge-based systems (Gil & Melz, 1996; Gil, 1994; Swartout & Gil, 1995).

An EXPECT knowledge base is composed of factual knowledge and of problem-solving knowledge. The factual knowledge includes concepts, relations, and instances in Loom (Macgregor 1990), a knowledge representation system of the KL-one family. The problem-solving knowledge is represented as a set of problem-solving methods such as those shown in Figure 1. As described earlier, each method has a capability, a result type, and a method body. Within the capability and body sections, each goal is expressed as a goal name followed by a set of parameters. Also, each parameter

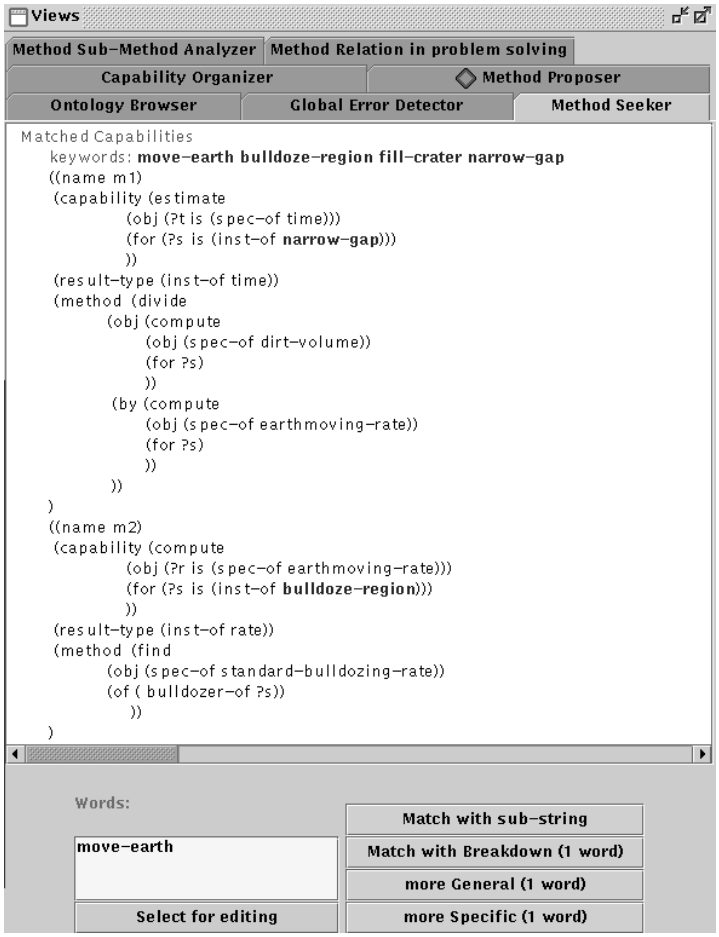


FIGURE 3: Search Methods in EMeD.

consists of a parameter name and a type.

Figure 2 shows the editing window from the EMeD user interface. There is a list of current methods and buttons for editing methods. Users can add, delete, or modify the methods using these buttons. (Other buttons and windows will be explained later.) Users often create new methods that are similar to existing ones so the tool has a copy/edit facility.

Every time a new method is defined, the *Local-Error Detector* checks for possible parsing errors based on the method representation language. If there are interdependencies among the subparts of a method, they are also used in detecting errors. For example, if a variable is used but not defined for the method, the same variable is defined more than once, or there are unused variables, the system will produce warning messages and highlight the incorrect definitions in red so that the user can be alerted promptly. Also, if there were terms (concepts, relations, or instances) used in a method but not defined in the KB yet, error messages will be sent to the developer. When the term definition is obvious, as the verbs used in capabilities, a definition will be proposed by the tool. In Figure 2, the small panel in the bottom left corner with the label "Local Error Detector"

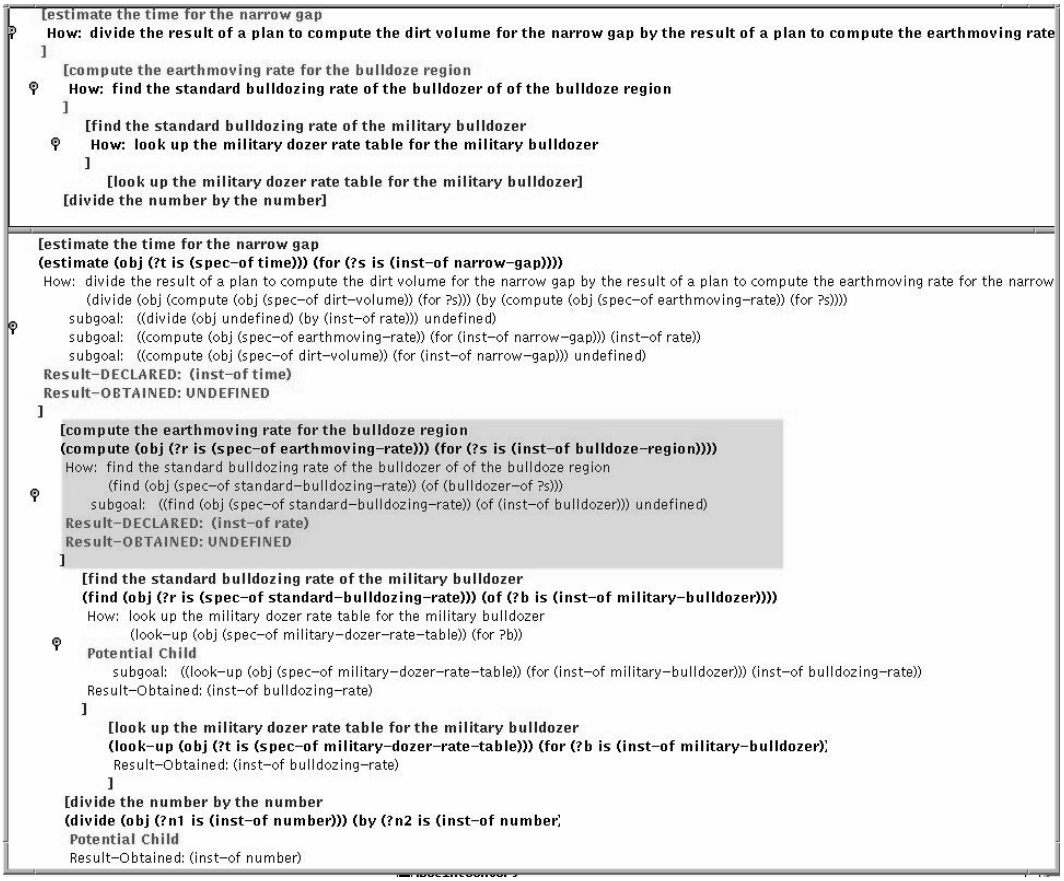


FIGURE 4: The Method Sub-method Analyzer.

displays these errors. Using the results, users can detect the local errors earlier, separating them from other types of errors.

Users can use the *Method Seeker* to find existing methods related with particular terms in concepts, relations or instances through the Loom ontology. The KA tool can retrieve methods that refer to subconcepts, superconcepts, or a given concept and let the user create new methods based on related methods. Figure 3 shows the result from retrieving methods about moving earth. The system was able to find the methods in Figure 3, narrow-gap and bulldoze-region are subtypes of move-earth.

The *Method Sub-method Analyzer* in EMeD analyzes interdependencies between each method and its sub-methods as well as the interdependencies among the problem-solving methods and factual knowledge. Figure 4 shows relationships among methods based on how the subgoals of a method can be achieved by other methods. The top window shows a succinct description of a tree, displaying only the natural language paraphrase of the capability and the body (how part). The bottom one shows a detailed description of the tree, including the formal definitions of capability and body, subgoals in the body, user defined result type (result-declared), and result obtained by evaluating

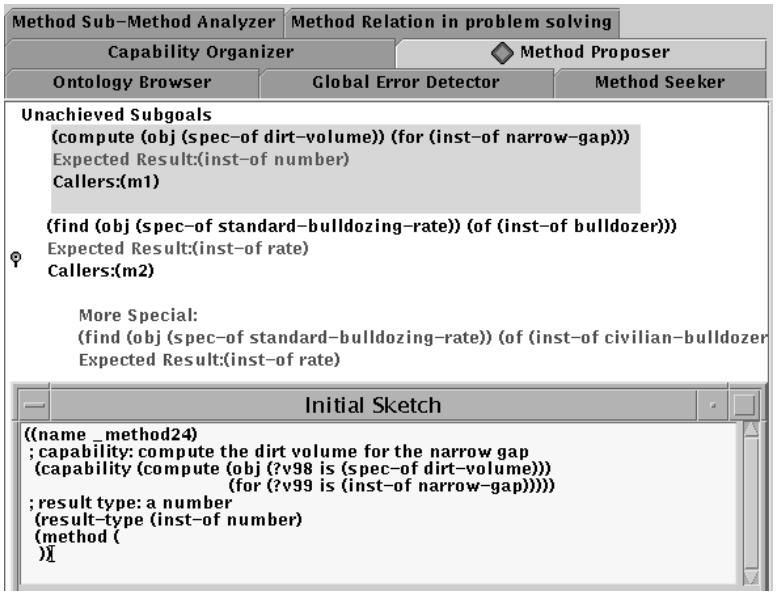


FIGURE 5: The Method Proposer.

the subgoals.

When the super-sub relation is not complete, the potential children are marked as 'Potential Child'. For example, one of the subgoals of M1 'divide' has two parameters with parameter arguments: 'compute earthmoving-rate for a narrow-gap step' and 'compute dirt-volume for a narrow-gap step'. Since the first one has a defined result-type (inst-of rate), the goal can be represented as 'divide (obj UNDEFINED) (by (inst-of rate))'. However, one of the built-in methods in EXPECT has capability of 'divide (obj Number) (by Number)', the Method Sub-method Analyzer creates a link between this and the subgoal as a *potential interdependency*. Also, the method to find standard-bulldozing rate of a military bulldozer is not enough to find standard-bulldozing rate of any type of bulldozer, as required for computing the earthmoving rate. The user has to either add another method for civilian bulldozer or to generalize the method for military bulldozer, so the system can find standard-bulldozing rate of any bulldozer in general.

There can be multiple trees growing in the process of building a number of methods when they are not fully connected. These method-relation trees are incomplete problem-solving trees to achieve some intermediate subgoal. The (sub)trees should be eventually put together to build a problem-solving tree for the whole problem. For example, given these three methods, the system can build a method-relation tree, as shown in Figure 4.

The Method Sub-method Analyzer can detects undefined methods based on the subgoals in a method that are not achieved by any existing methods. The *Method Proposer* in EMeD notifies the user with a red diamond (a diamond shown in Figure 5 on the top) and displays the ones needed to be defined. If there are constraints imposed on the methods to be built, such as the expectations

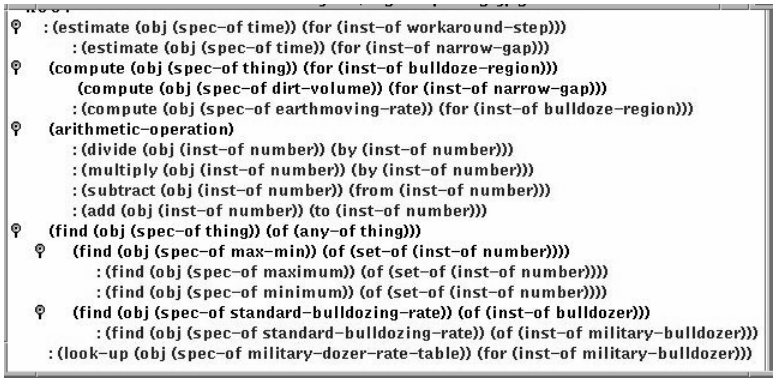


FIGURE 6: The Capability Organizer.

coming from the methods that invoke them, then these can be also incorporated. For example, the method to estimate time to narrow-gap step calls a subgoal to find the dirt volume of the gap, which is undefined yet. Since the result or the undefined method should be a number to be divided, the system can expect (through an interdependency analysis) the same result type for the undefined method. Figure 5 shows the list of unachieved subgoals, to compute the dirt volume of the gap and to find the standard-bulldozing rate of any bulldozer. For the latter one, the system shows one more option to define more special method for only unachieved subpart (civilian bulldozer). The figure also shows the initial sketch for the first unachieved subgoal. The system proposes a capability and a expected result type based on the interdependency analysis described above.

There are other relationships among problem-solving methods based on their capabilities that the KA tool can exploit. For example, a hierarchy of the goals based on the subsumption relations of their goal names and their arguments can be created. In the hierarchy, if a goal is to estimate the time for a workaround step, and another goal is to estimate the time for a kind of workaround step, such as narrow-gap step, then the former subsumes the latter. The *Capability Organizer* builds this dependency among the capability descriptions of the methods. The resulting structure is useful in that it allows the user to understand the kinds of sub-problems the current set of methods can solve. To make their relationships more understandable, EMeD also computes potential capabilities. For example if there are super-concepts defined for the parameters of a capability, a capability with these concepts is created as a parent of the capability. The organized capabilities for the given example methods is in Figure 6. The system-defined methods for arithmetic operations are grouped together by their super-concept (arithmetic-operation). Also, the methods for finding objects (finding standard-bulldozing rates and finding max-min) are grouped together.

Finally, there are user-expected dependencies among the problem-solving methods, which are usually represented in comments or by grouping of methods in the files where they are built. They do not directly affect the system, but they often become the user's own instrument to understand

the structure of what they are building. Also, it can be the user's own interpretation of additional interdependencies among methods. The *Method Organizer* in EMeD provides a way of organizing methods into hierarchies and groups, and allows users to provide documentation for the methods. In Figure 2, the "Method Organizer" buttons (New Folder, Cut to move, etc.) support these functions.

In addition, EMeD can use the expectations derived from running the problem solver, detecting problems that arise while attempting to build a complete problem-solving tree, by the *Global-Error Detector*.

4 Experimental Design

Current KA research lacks evaluation methodology. In recognition of the need for evaluation, the community started to design a set of standard task domains that different groups would implement and use to compare their work. These Sisyphus experiments (Schreiber & Birmingham, 1996; Sisyphus, 2000) show how different groups would compare their approaches for the same given task, but most approaches lacked a KA tool and no user evaluations were conducted. Other evaluations have tested the use and reuse of problem-solving methods, but they measure code reuse rather than how users benefit from KA tools (Runkel & Birmingham, 1995; Eriksson *et al.*, 1995). Other KA work evaluated the tool itself. TAQL's performance was evaluated by comparing it with some basic data that had been reported for other KA tools (Yost, 1993). There were some user studies on ontology editors (Terveen, 1991). In contrast with our work, these evaluations were done with knowledge engineers. Also since the experiments were not controlled studies, the results could not be causally linked to the features in the tools.

Our research group has conducted some of the few user studies to date (Tallis & Gil, 1999; Kim & Gil, 1999), and as a result we have proposed a methodology (Tallis *et al.*, 1999) that we use in our own work. It turns our that the lack of user studies is not uncommon in the software sciences (Zelkowitz & Wallace, 1998). In developing a methodology for evaluation of KA tools, we continue to draw from the experiences in other areas (Self, 1993; Basili *et al.*, 1986; Olson & Moran, 1998).

Our goal was to test two main hypotheses, both concerned with Interdependency Models (IMs):

> **Hypothesis I**: A KA tool that exploits IMs *enables users to make a wider range of changes* to a knowledge base because without the guidance provided with IMs users will be unable to understand how the pieces of knowledge being built fit each other and complete the development of KB elements.
>
> **Hypothesis II**: A KA tool that exploits IMs *enables users to enter knowledge faster*

because it can use the IMs to point out to the user at any given time what additional knowledge still needs to be provided.

There are three important features of our experiment design:

- The most adequate method to test these hypotheses is a controlled experiment because it allows us to collect data that is comparable across users and tasks. Thus, we designed the tasks to be given to the participants based on typical tasks that we encountered ourselves as we developed the initial knowledge base.
- Given the hypotheses, we needed to collect and compare data about how users would perform these tasks under two conditions: with a tool that exploits IMs and with a tool that does not (this would be the control group). It is very important that the use of IMs be the only difference between both conditions. We designed an ablated version of EMeD that presented the same EMeD interface but did not provide any of the assistance based on IMs.
- Typically, there are severe resource constraints in terms of how many users are available to do the experiments (it typically takes several sessions over a period of days). In order to minimize the effect of individual differences, we performed within-subject experiments. Each subject performed two different but comparable sets of tasks, one with each version of the tool.

In order to determine when a KA task was completed, the subjects were asked to solve some problems and examine the output to make sure they obtained the expected results. In addition, after each experiment, we checked by hand the knowledge added by the subjects.

Participants were given different combinations of tools and tasks and in different order, so as to minimize transfer effects (i.e., where they would remember how they did something the second time around).

EMeD was instrumented to collect data about the user's performance, including actions in the interface (e.g., commands invoked and buttons selected), the knowledge base contents at each point in time, and the time at which each user action takes place. These provide objective measurements about task completion time and the use of specific features. Since this data was insufficient to understand what things users found hard and difficult to do with the tool or why a certain action was not taken, we collected additional information during the experiment. We asked users to voice what they were thinking and what they were doing and recorded them in transcripts and in videotapes (during the experiments with domain experts). We also prepared a questionnaire to get

their feedback, where instead of questions with free form answers we designed questions that could be answered with a grade from 1 (worst) to 5 (best).

5 Preliminary Study

Since it is expensive to run user studies and hard to get domain experts in the field, we wanted to filter out distractions which are unrelated with our claim, such as problems with the tool that are not related to Interdependency Models. We also wanted to understand whether our interface and KA tool are appropriate for end users and how different types of users interact with the system and what common types of errors they experience, so that we can improve our tools and our experimental methodology. For these reasons, we performed a preliminary study before the actual evaluation.

The study used a spectrum of users that had gradually less background in AI and CS (Kim & Gil, 2000). We had

1. users familiar with the knowledge base environment (Group 1), i.e., who had previous experience in developing knowledge-based systems in EXPECT. These subjects can be considered experienced knowledge engineers.

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. These subjects can be considered to be acquainted but not proficient with knowledge engineering techniques.

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 use computers on a daily basis (for tasks such as email, meeting scheduling, text editing, etc.) and are familiar with software tools such as spreadsheets and HTML editors. They are project assistants in our research institute, with no formal training in computer science.

None of the subjects had used or seen EMeD before the experiment. We tested four subjects in Group 1, two subjects in Group 2, four subjects in Group 3, and two subjects in Group 4. Because it is hard to recruit participants for this kind of experiment, we had to work with the pool of people that were willing to volunteer for our study. The subjects in the third group were first and second year graduate students that work on AI projects and thus were willing to participate in our

time-consuming experiments as a way to learn more about AI, but we made sure that they had no real experience in knowledge-based systems. One of the subjects in the fourth group had taken a few introductory programming classes but dropped out early in the course, although it was very clear that in practice this subject had no real experience with programming.

Each subject was tested under two conditions: using EMeD and using a version of EMeD that only allowed them to edit methods (the buttons to add, delete, or modify 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 challenge problem that we addressed was concerned with analyzing enemy workarounds to a damaged target. The subjects were not experts in this domain and were not familiar with the details of the knowledge base.

Each subject was given a tutorial of the tools with simpler scenarios and was allowed a period of practice with both tool. During the practice, each subject was asked to perform a simple acquisition task with both tools. Finally, the subject was given a tool and a high-level description in English of the new knowledge that they needed to add and asked to edit the knowledge base until the system could correctly solve problems using the new knowledge. Then, the subject was given another such scenario and was asked to use the other interface. Each of these activities took from a half hour to several hours, so the experiment was often spread over several days.

We instrumented the tools to record the actions performed by the subjects. We also took detailed transcripts of their activities and the comments they voiced out loud as they were performing the task.

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. Figure 7 shows the training time for different groups of users. This 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. Group 2 needed some time to learn the representation language, and Group 3 needed more time for both phases. The training time for Group 4 was almost twice the time for Group 3 and almost four times of what Group 1 had needed.

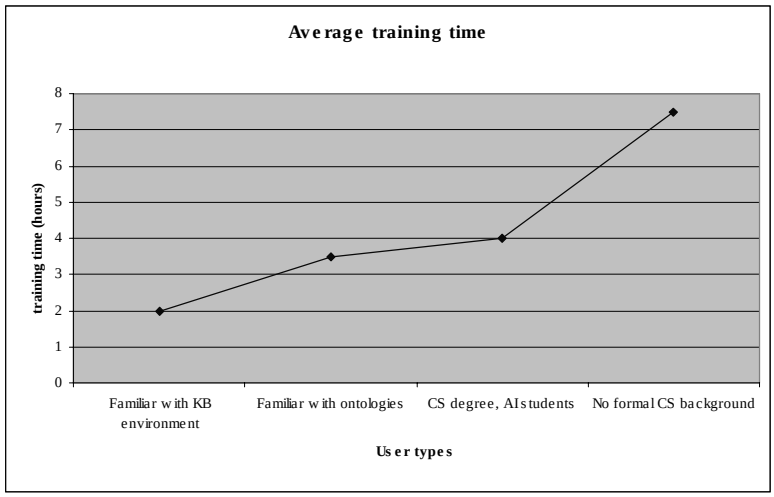


FIGURE 7: Average training time.

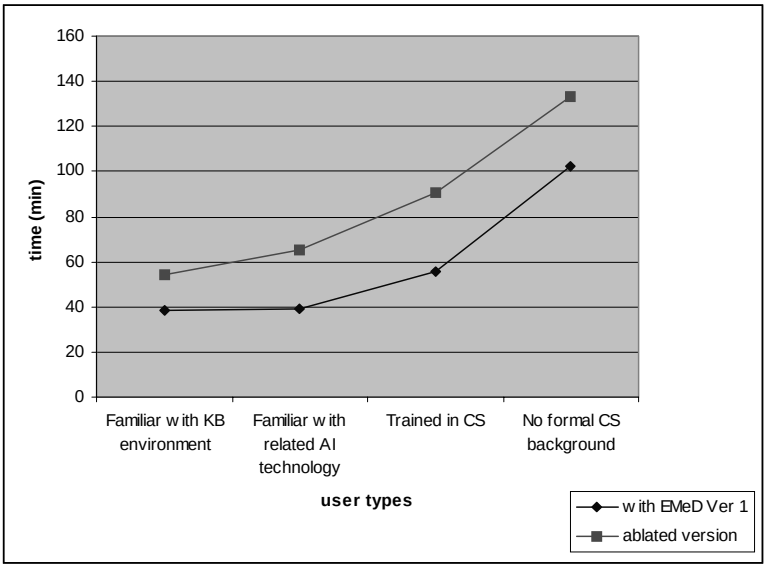


FIGURE 8: Average time to complete KA tasks.

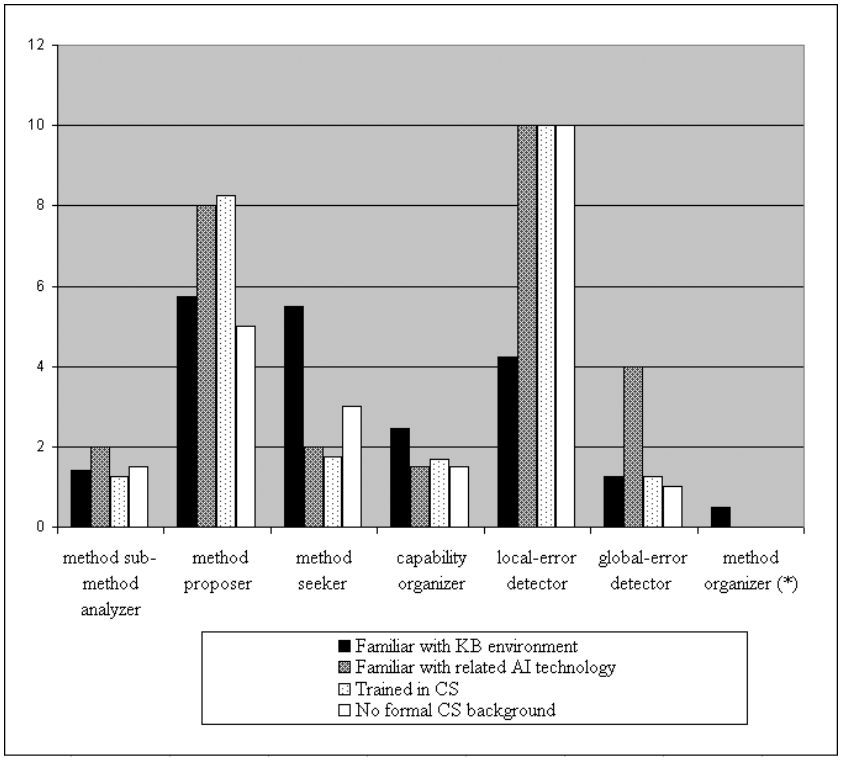


FIGURE 9: Average number of times EMeD features used.

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. Although the subjects had used e-mail and some other commercial software for text editing and spreadsheet before, they had no exposure to concepts such as nested goal expressions, defining and using variables, retrieving values, etc. Also, 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 let them write on paper first what they intended to put in each method. The subjects took approximately the same time to do this with and without EMeD. (We are developing an interface that would provide help with this task and will take advantage of the fact that EXPECT's method language was design with intelligibility in mind (Swartout & Moore, 1991).)

In order to determine when a KA task was completed, the subjects were asked to solve some problems and examine the output to make sure they obtained the expected results. In addition, after each experiment, we checked by hand the knowledge added by the subjects. Because the subjects performed the task in different ways, they ended up adding a different number of methods with different number of axioms and of different kinds. The subjects added 10 to 14 methods for the two tasks, building 102 to 133 axioms for them. Figure 8 shows that EMeD reduced the time

to complete the tasks by an average of 32%. That is, the users were able to finish the tasks faster with the help provided by Interdependency Models.

We also analyzed the use of each of the components of EMeD throughout the experiments. Figure 9 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. The Local-Error Detector 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 Method Generator 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.

The capability index in the Capability Organizer was found useful to find system-defined methods, such as arithmetic and logic operations, but the search functionality in the Method Seeker 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 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 (in EMeD Version 1, only detailed view in Figure 4 was supported). With more exposure to the interface, we hoped it can become easier to understand. A more summarized view seemed more adequate.

The Method Organizer functionality was only provided to subjects in Group 1, because we thought that providing too many features would divert the attention and or confuse other subjects and because we thought they would not really use it anyway. 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 the methods as a group without being distracted by other methods.

Since a major goal of this preliminary study was to understand our interface and KA tool, we allow the subjects to ask for hints when they were not able to make progress (this was not allowed in the final evaluation). These hints allow us to categorize the basic types of difficulties experienced by users and adjust the tool based on them.

Figure 10 shows the number of hints given to the subjects. More hints were always needed with the ablated version. The number of hints increases dramatically when subjects lack CS background. We analyzed all the hints, and separated them into two major categories. Class A hints consist

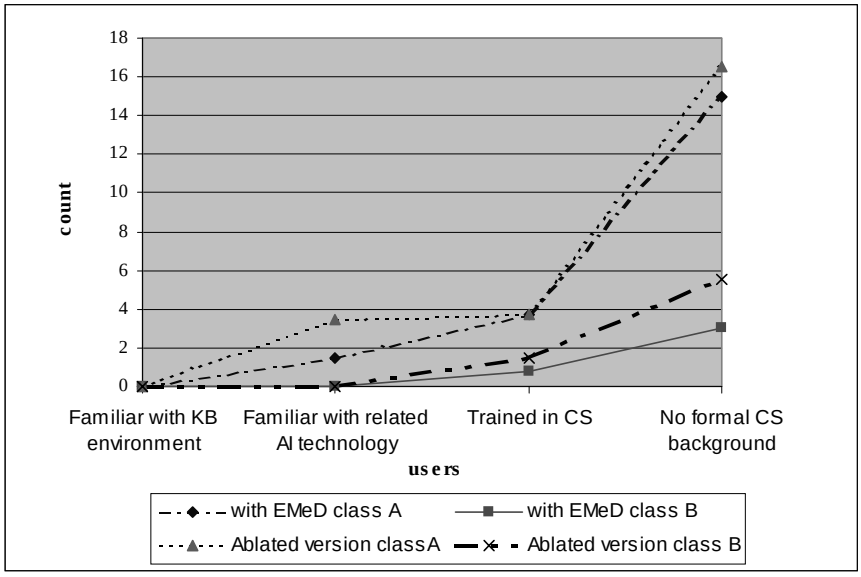


FIGURE 10: Average number of hints given to each group of subjects during the preliminary user study.

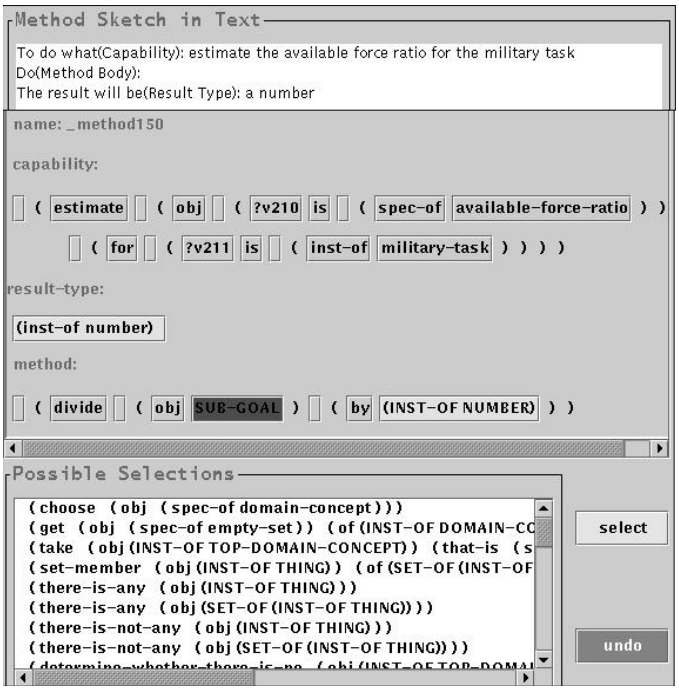


FIGURE 11: Structured Editor.

of simple help on language and syntax, or clarification of the tasks given. Since syntax errors are unrelated to our claims about IMs, we developed a Structured Editor for the new version of EMeD (version 2) that guides users to follow the correct syntax. Figure 11 shows the new editor which guides the users to follow the correct syntax. Users can build a method just using point and click operations without typing.

Class B hints were of a more serious nature. For example, users asked for help to compose goal descriptions, or to invoke a method with the appropriate parameters. Although the number of times these hints were given is smaller and the number is even smaller with EMeD, they suggest new functionality that future versions of EMeD should eventually provide to users. The subjects indicated that sometimes the tool was showing too many items, making it hard to read although they expected this would not be a problem after they had used the tool for a while and had become used to it. Since these presentation issues were affecting the results of the experiment and are not directly evaluating the IMs, the new version of EMeD (version 2) has more succinct views of some of the information, showing details only when the user asks for them. Other hints pointed out new ways to exploit IMs in order to guide users and would require more substantial extensions to EMeD that we did not add to the new version. One area of difficulty for subjects is expressing composite relations (e.g., given a military task, retrieve its assigned units and then retrieve the echelons of those assigned units). Although EMeD helps users in various ways to match goals and methods, in some cases the users still asked the experimenters for help and could have benefited from additional help. The fundamental difficulties of goal composition and using relations still remained as questions for the real experiment.

In addition to improving the tool, we debugged and examined our experimental procedure, including tutorial, instrumentation, questionnaire, etc., especially based on the the results from end users.

The most remarkable result of these experiments is that subjects of Group 4 were able to complete the 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, who represented real end users in our study, were able to take charge of the task and execute it to completion is in itself a very encouraging result. The result implies that even the end users were able to finish complex tasks, and that the KA tool saves time for users with technical background.

As described above, we extended our tool based on the pre-test results, creating a new version of EMeD (version 2). The next section describes the evaluation with domain experts with this new version of EMeD.

6 Experiment with Domain Experts

The participants in this experiment were Army officers facilitated by the Army Battle Command Battle Lab at Ft Leavenworth, KS. They were asked to use our KA tools to extend a knowledge based system for critiquing military courses of action (the performance of this critiquing system was done in a separate evaluation). Each subject participated in four half-day sessions over a period of two days. The first session was a tutorial of EXPECT and an overview of the courses of action critiquer. The second session was a tutorial of EMeD and a hands-on practice with EMeD and with the ablated version. In the third and fourth sessions we performed the experiment, where the subjects were asked to perform the KA tasks, in one session using EMeD and in the other using the ablated version. Only four subjects agreed to participate in our experiment, due to the time commitment required.

An important difference with the previous study is that during this experiment subjects were not allowed to ask for hints, only clarifications on the instructions provided. As soon as a participant would indicate that they could not figure out how to proceed, we would terminate that part of the experiment. In order to collect finer-grained data about how many tasks they could complete, we gave each subject four knowledge base development tasks to do with each version of the KA tool. The reason is that if we gave them one single task and they completed almost but not all of it then we would not have any objective data concerning our two initial hypotheses. The four tasks were related, two of them were simpler and two more complex. The easier tasks required simple extension to an existing method (e.g., generalize the existing methods that compute the required force ratio for "destroy" tasks into methods that can compute the ratio for any military tasks in general). The more complex tasks required adding new methods. For example, one of the tasks involved adding a new method that checks the force ratio for a military task, given some methods that compute required force-ratio and available force-ratio.

The main results are shown in Figure 12. Figure 12-(a) shows the average time to complete tasks (for the completed tasks only). None of the subjects was able to do the more complex tasks with the ablated version of EMeD. For the easier tasks, subjects were able to finish the tasks faster with EMeD. Figure 12-(b) shows the number of tasks that the subjects completed with EMeD and with the ablated version, both by task category and overall. The solid part of the bars show the number of tasks completed. We show with patterned bars the portion of the uncompleted tasks that was done when the subjects stopped and gave up (we estimated this based on the portion of the new knowledge that was added). Figure 12-(c) shows the same data but broken down by subject[1].

[1]We had noticed early on that Subject 4 had a different background from the other three, but unfortunately we were not able to get an alternative subject.

The results show that on average subjects were able to accomplish with EMeD almost twice as many tasks as they accomplished with the ablated version. The results support our claims that Interdependency Models can provide significant help to end users in extending knowledge bases.

It would be preferable to test additional subjects, but it is often hard for people (especially domain experts) to commit the time required to participate in this kind of study. Given the small number of subjects and tasks involved it does not seem appropriate to analyze the statistical significance of our results, although we have done so for some of the initial experiments with EMeD (Kim & Gil, 1999) with a t-test showing that they were significant at the 0.05 level with $t(2)=7.03$, $p < .02$ (Tallis *et al.*, 1999). Gathering data from more subjects may be more reassuring than using these tests for validation.

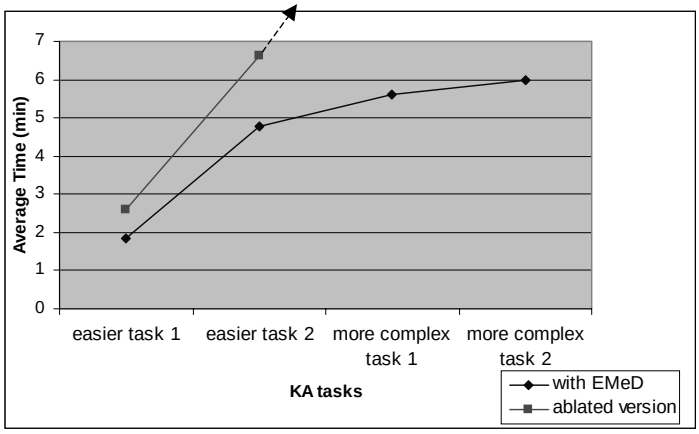
Our experience with these experiments also motivates us to share a few of the lessons that we have learned about knowledge acquisition research:

- **Can end users use current KA tools to develop 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?**

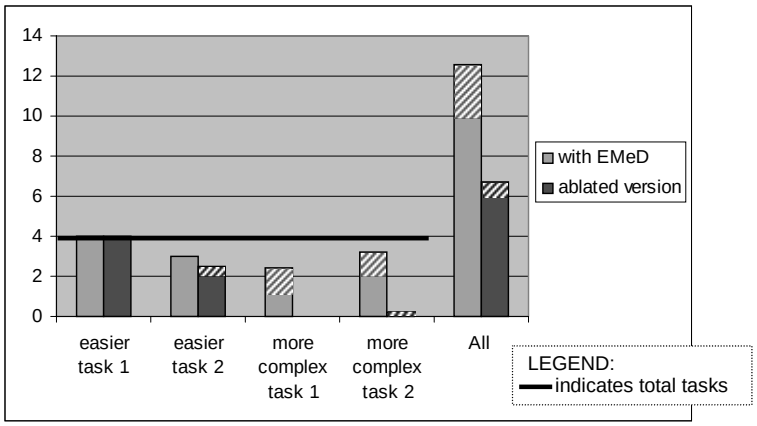
  As we described earlier, we spent 8 hours (two half-day sessions) for training. They spent roughly half of that time learning EXPECT's language and how to put the problem-solving methods together to solve problems. The rest of the time was spent learning about the KA tool and its ablated version. We believe that this time can be reduced by improving the tool's interface and adding on-line help. We also recognize that 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 problem-solving methods 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.

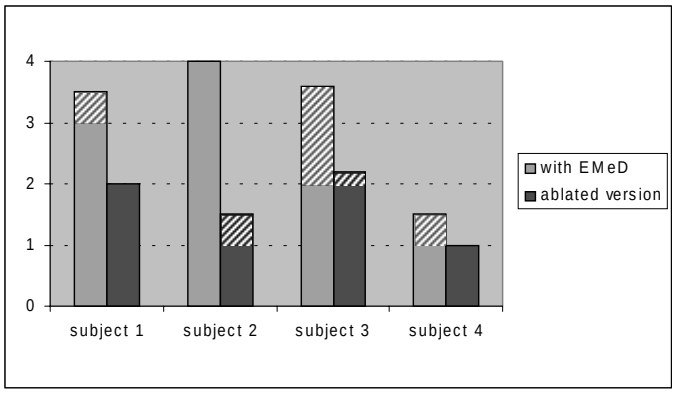
  As a result of this experiment, we believe that with current technology it is possible to develop KA tools that enable end users to add relatively small amounts of new problem



(a) Average time to complete tasks



(b) Tasks completed



(c) Tasks completed (for each subject)

FIGURE 12: Results of the evaluation with domain experts.

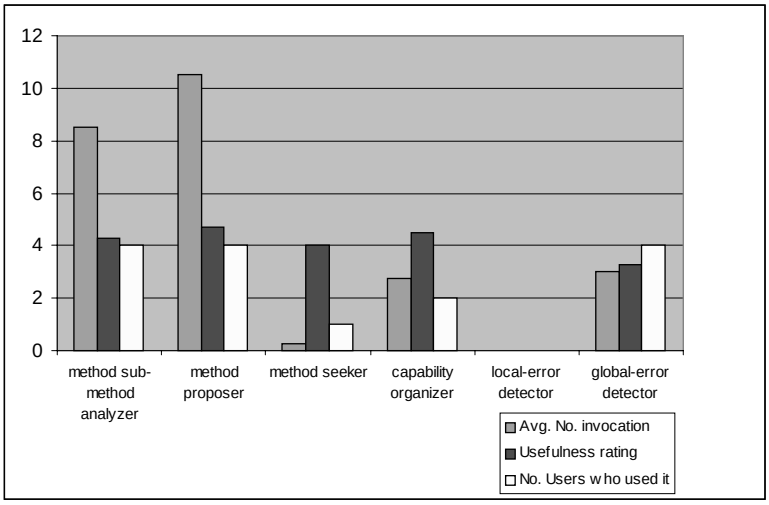


FIGURE 13: Average Uses of EMeD's Features.

solving knowledge, and that they can be trained to do so in less than a day.

- **How much do Interdependency Models help? What additional features should be added to our KA tools?**

  Overall, the Interdependency Models exploited via different features in EMeD were useful for performing KA tasks. Table 13 shows the average use of each of the Components of EMeD, in terms of the number of times the user invoked them[2]. The subjects were very enthusiastic about the tool's capabilities, and on occasion would point out how some of the features would have helped when they were using only the ablated version.

  According to the answers to the questionnaire, using EMeD it was easier to see what pieces are interrelated. That is, visualizing super/sub method relations using Method Sub-method Analyzer was rated as useful (4.3/5). Also detecting missing knowledge and adding it was easier with EMeD's hints. Highlighting missing problem-solving methods and creating initial sketch based on interdependencies (by Method Proposer) were found to be the most useful (4.7/5).

  The Structured Editor used in this version of EMeD provided very useful guidance, and there were less errors for individual method definitions. The Local-Error Detector was not used for the given tasks.

- **What aspects of a KA task are more challenging to end users?**

  Almost everyone could do simple extensions to the KB, which required that the subjects

[2]We show the number of times the users selected them, except for the Method Proposer where we show the number of times the system showed it automatically as well as the number of time selected (in parenthesis).

browse and understand the given methods to find one method to be extended and then changing it.

Some subjects had difficulties starting the KA tasks, when EMeD does not point to a particular element of the KB to start with. Although they could use the Method Seeker in EMeD or look up related methods in the Capability Organizer, this was more difficult for them than when the tool highlighted relevant information.

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. A KA tool that keeps track of what they are doing in the context of the overall task and lets them know about their progress would be very helpful. Some of the research in using Knowledge Acquisition Scripts to keep track of how individual changes to the KB contribute to complex changes (Tallis & Gil, 1999) could be integrated with EMeD.

- **How do KA tools need to be different for different kinds of users**

  We did not know whether end users would need a completely different interface altogether. It seems that a few improvements to the presentation in order to make the tool easier to use was all they needed. We did not expect that syntax errors would be so problematic, and developing a structured editor solved this problem easily. 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 in the original EMeD that we thought would be distractive and disabled is organizing problem-solving methods into a hierarchy. However, the feedback from the end users indicates that they would have found it useful.

  Although EMeD is pro-active 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.

7 Related Work

There are other KA tools that take advantage of relationships among KB elements to derive expectations. TEIRESIAS (Davis, 1979) uses *rule models* to capture relationships among the rules based on their general structure and guide the user in following that general structure when a new rule is added that fits a model. Some of the capabilities of EMeD are similar in spirit to the rule model (e.g., the method capability tree), and are also used in EMeD to help developers understand potential dependencies among the KB elements. Other KA tools (Eriksson *et al.*, 1995;

Birmingham & Klinker, 1993; Marcus & McDermott, 1989) also use dependencies between factual knowledge and problem-solving methods to guide users during knowledge acquisition. These tools help users to populate and extend a system that already has a significant body of knowledge, but they are not designed to help users in the initial stages of KB development. More importantly, these tools are built to acquire factual domain knowledge and assume that users cannot change or add problem-solving knowledge. In this sense, EMeD is unique because it guides users in adding new problem-solving methods.

In the field of software engineering, it has been recognized that it is generally better to focus on improving the process of software development rather than on the output program itself (Dunn, 1994). Our approach embraces this view and tries to improve the initial phases of KB development. Some previous work on using formal languages to specify knowledge bases (Fensel *et al.*, 1998) is inspired by software engineering approaches. This work provides a framework for users to model and capture the initial requirements of the system, and require that users are experienced with formal logic. Our approach is complementary in that it addresses the stage of implementing the knowledge-based system, and we believe that our formalism is more accessible to users that have no formal training. Other approaches (Wielinga *et al.*, 1992; Gaines & Shaw, 1993) support users in the initial stages of development by providing a methodology that can be followed systematically to elicit knowledge from experts and to design the new system. These methodologies can be used in combination with our approach.

There is also related research in developing tools to help users build ontologies (Fikes *et al.*, 1997; Terveen & Wroblewski, 1990; Clark & Porter, 1997). Unlike our work, these tools do not tackle the issue of using these ontologies within a problem-solving context. Many of the research contributions in these tools concern the reuse of ontologies for new problems, collaborative issues in developing knowledge bases, and the visualization of large ontologies. We believe that integrating our approach with these capabilities will result in improved environments to support KB creation.

Many knowledge acquisition approaches target knowledge engineers (Wielinga *et al.*, 1992; Yost, 1993; Fikes *et al.*, 1997), and those that have been developed for end users (Eriksson *et al.*, 1995; Marcus & McDermott, 1989) only allow them to specify certain kinds of knowledge, i.e., domain-specific knowledge regarding instances and classes but not problem-solving knowledge about how to solve tasks. Alternative approaches apply learning and induction techniques to examples provided by users in a natural way as they are performing a task (Mitchell *et al.*, 1985; Cypher, 1993; Bareiss *et al.*, 1989). Although these tools are more accessible to end users, they are only useful in circumstances where users can provide a variety of examples. When examples are not readily available, we may need knowledge acquisition (KA) tools for direct authoring.

8 Conclusion

We analyzed the process of KB development to support KB creation and found a set of expectations to help KA tools guide users during the development process. We have classified the sources of errors in the KB development process based on their characteristics, and found ways to prevent, detect, and fix errors earlier. These expectations were derived from the dependencies among KB elements. Although EMeD aims to provide support for KB creation, its functionality is also useful for modifying existing knowledge or populating a KB with instances.

The evaluation results show that EMeD was able to provide useful guidance to users reducing KB development time by more than 30%. EMeD was even more beneficial for domain experts who don't have much KA experience. EMeD also opens the door to collaborative tools for knowledge acquisition, because it captures what KA tasks remain to be done and that may be done by other users.

Now we are examining the uses of Interdependency Models in other phases of knowledge-base development. For example, the EXPECT problem solver provide an integrated framework for partial evaluation of generic problem-solving goals that allow the detection of inconsistencies and missing knowledge for knowledge base verification. By analyzing the traces we will be able to detect inconsistencies and missing knowledge for knowledge base verification, as well as mismatches between system's knowledge and the users' knowledge. Also, the tools can keep mechanisms to record, update, and analyze a suite of test cases and use them to validate new knowledge added by users. The tools can let the users examine the resulting traces from the tests and help users find the causes of mismatches between the expected result and the system's answer.

We are also extending the EMeD framework to be able to derive expectations in solving particular problems. Currently EMeD computes relationship among the KB components regardless of the context. Depending on what problem episode we are solving, the relationships may show different patterns, since the problem-solving methods may become specialized.

Acknowledgments

We gratefully acknowledge the support of DARPA with grant F30602-97-1-0195 as part of the DARPA High Performance Knowledge Bases Program. We would like to thank Marcelo Tallis, Surya Ramachandran, and Jim Blythe for their help during the evaluation. We are indebted to the many subjects, who have participated in the experiments for their time and their patience.

References

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

Basili, V., Selby, R. W., & Hutchens, D. H. (1986). Experimentation in software engineering. *IEEE Transactions in Software Engineering*, SE-12(7).

Birmingham, W. & Klinker, G. (1993). Knowledge-acquisition tools with explicit problem-solving methods. *The Knowledge Engineering Review*, 8(1):5–25.

Clark, P. & Porter, B. (1997). Building concept representations from reusable components. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 369–376.

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

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

Davis, R. (1979). Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 12:121–157.

Dunn, R. H. (1994). Quality assurance, Encyclopedia of software engineering. 2.

Eriksson, H., Shahar, Y., Tu, S. W., Puerta, A. R., & Musen, M. (1995). Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79:293–326.

Feigenbaum, E. (1993). The tiger in the cage. A Plenary Talk in Fifth Innovative Applications of AI Conference.

Fensel, D., Angele, J., & Studer, R. (1998). The knowledge acquisition and representation language KARL. *IEEE Transactions on Knowledge and Data Engineering*, 10(4):527–550.

Fikes, R., Farquhar, A., & Rice, J. (1997). Tools for assembling modular ontologies in Ontolingua. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 436–441.

Gaines, B. R. & Shaw, M. (1993). Knowledge acquisition tools based on personal construct psychology. *The Knowledge Engineering Review*, 8(1):49–85.

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

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

Kim, J. & Gil, Y. (1999). Deriving expectations to guide knowledge base creation. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pp. 235–241.

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

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

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

Olson, G. M. & Moran, T. P. (1998). Special issue on experimental comparisons of usability evaluation methods. *Human-Computer Interaction*, 13.

Runkel, J. T. & Birmingham, W. P. (1995). Knowledge acquisition in the small: Building knowledge-acquisition tools from pieces. *Knowledge Acquisition*, 5(2):221–243.

Schreiber, A. T. & Birmingham, W. P. (1996). The Sisyphus-VT initiative. *International Journal of Human-Computer Studies*, 44(3/4).

Self, J. (1993). Special issue on evaluation. *Journal of Artificial Intelligence in Education*, 4(2/3).

Sisyphus (2000). Sisyphus projects. http://ksi.cpsc.ucalgary.ca/KAW/Sisyphus/.

Swartout, B. & Moore, J. (1991). Explanation in second generation expert systems. In *Second Generation Expert Systems*.

Swartout, W. & Gil, Y. (1995). EXPECT: Explicit representations for flexible acquisition. In *Proceedings of the Ninth Knowledge-Acquisition for Knowledge-Based Systems Workshop*.

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

Tallis, M., Kim, J., & Gil, Y. (1999). User studies of knowledge acquisition tools: Methodology and lessons learned. In *Proceedings of the Twelfth Knowledge-Acquisition for Knowledge-Based Systems Workshop*.

Terveen, L. (1991). *Person-Computer Cooperation Through Collaborative Manipulation*. PhD thesis, University of Texas at Austin.

Terveen, L. G. & Wroblewski, D. (1990). A collaborative interface for editing large knowledge bases. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 491–496.

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

Yost, G. R. (1993). Knowledge acquisition in Soar. *IEEE Expert*, 8(3):26–34.

Zelkowitz, M. & Wallace, D. (1998). Experimental models for validating computer technology. *IEEE Computer*, 31(5):23–31.