Submitted for publication.

Memory-Based Meta-Level Reasoning for Interactive Knowledge Capture

Jihie Kim
University of Southern California/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292 USA
+1 310 448 8769
jihie@isi.edu

ABSTRACT

Current knowledge authoring interfaces lack understanding of what users are doing and how well they are doing, and cannot provide effective assistance in organizing various knowledge authoring tasks. Users have to make up for these shortcomings by keeping track of the status, progress, potential problems and possible courses of actions by themselves. We present a novel extension to existing systems that 1) keeps track of past problem solving episodes and relates them to user entered knowledge, 2) assesses the current status of the knowledge and the problem solving using such relations, and 3) provides assistance to the user based on the assessment. We applied the approach in developing an intelligent assistant for decision making tasks. The resulting interaction shows that the system helps the user understand the progress and guides the knowledge authoring process in terms of making the knowledge more useful, adapting the knowledge to dynamic changes over time, and making the overall problem solving more successful.

Keywords
Knowledge acquisition, knowledge bases, problem solving episodes, memory, meta-level reasoning

1. INTRODUCTION

End user programming is a challenging area of research for intelligent user interfaces. There has been a wide range of approaches taken, including programming by demonstration [6,17], case-based reasoning [1], and learning apprentices [21]. These approaches make use of observed examples and generalize them into a task representation. While these approaches work well for simple tasks, for capturing complex problem solving activities, direct knowledge authoring interfaces seem more useful [5,4,2,8].

In the past we have participated in developing and evaluating various knowledge authoring interfaces. We have analyzed various types of tasks involved in entering complex problem solving knowledge and examined the challenges end users face in performing such tasks[14,15,25,22]. These results show a common pattern: end users have difficulties in understanding what they are doing and how well they are doing, and they can easily become lost in the process of performing various tasks involved in knowledge authoring. Most existing knowledge authoring interfaces do not explicitly model this type of meta-level process and cannot provide effective help.

In existing systems, knowledge entered by the user (called *k-items*) are treated equally and systems do not reflect on how each knowledge has been built and how well it has been used over time. The systems cannot provide effective assistance in making the k-items more useful and the resulting problem solving more successful. For instance, when there is an inconsistency detected between two k-items, existing systems provide uniform error messages. However, in order to develop actually useful k-items, users may need to know which k-item should be modified and how the modification improves the overall problem solving. For example, while one of the k-items has been successfully used over time and the system has *high confidence* in it, the other k-item may conflict with some of the past successful problem solving results and its estimated confidence level is lower. In such a case, the system may suggest modifying the less confident item instead of the other.

When captured k-items are used in problem solving, the k-items can be applied according to the level of confidence assessed for them. That is, more confident items are preferably used.

When confident k-items are overridden or modified, the system notices them as unexpected event and may also recognize that there are some changes in the problem solving. In such cases, the system will assist users in making appropriate k-item modifications. For example, if a k-item becomes obsolete, the problem solving steps where the k-item was applied may be also out-of-date and other related k-items may become less reliable. This capability is particularly important for the applications where there are dynamic changes in the problem solving over time and the associated knowledge needs to be adjusted accordingly. Many practical applications in business and science require this type of capability.

Here we propose a novel framework for knowledge authoring interfaces that 1) keeps track of past problem solving episodes and **relates** them to each k-item 2)

assesses the current status of k-items and problem solving using such relations and 3) provides assistance to the user based on the assessment. We have built a system called Echo (mEta-Cognitive History analysis and Organization) that provides these capabilities. Echo can be used as an extension to existing knowledge-based systems that support knowledge authoring and problem solving. Echo builds a *memory model* that relates each k-item to the problem solving episode it was built from, the episodes it matched, the episodes where it was actually used, and the episodes it can potentially contribute to. This model is used in assessing a k-item, checking if it can be confidently used, it conflicts with some successful results and needs some modifications, or it needs significant changes including deactivation. Based on this assessment, Echo guides the user in improving and maintaining the confidence levels of the k-items over time.

We claim that this memory based meta-level reasoning and reflection helps end users understand what they are doing, and how well they are doing in the process of k-item development and use. The resulting interactions help users build better k-items that make the overall problem solving more successful. The approach also facilitates adaptation of k-items when there are changes in problem solving.

The contributions of this work are twofold. First we present a novel architecture of intelligent systems where memory based meta-level reasoning supports knowledge authoring and problem solving, which defines a new class of cognitive interfaces. Second, we demonstrate how the architecture supports developing useful intelligent assistants. We use a domain where the problem solving is traditionally done manually but acquisition of simple problem-solving knowledge from the user incrementally automates the process and improves the problem solving results.

This paper begins with example interactions with Echo, illustrating how its meta-level reasoning helps users. We use a simple decision making domain where captured k-items help the user make better decisions. Then we present the architecture of the system. We show each component of Echo and describe a set of general algorithms supporting meta-level reasoning. In particular we demonstrate how the memory model is used in assessing the status of the knowledge base and providing assistance to the user. Next, we describe the details of the knowledge representation and the problem solving of the system we have developed, summarizing the benefits of Echo. Finally we discuss related work and future extensions.

2. EXAMPLE INTERACTIONS

Here we illustrate our approach with a simple application of visitor hosting where the user *manually* makes decisions on visitor activities and meeting schedules. That is, the user instead of a problem solver performs decision making tasks, choosing talk time, talk location, meeting attendees, meeting times, etc. The user can enter constraints in making decisions as k-items based on failures and successes of decisions that are observed. These k-items are used in providing suggestions for new decisions, aiming at improving the quality of the decisions over time.

In the following examples, we show example interactions between the user and Echo. The k-items are paraphrased in English for readability. The representation that we use for these examples are described in Section 4.

Assisting k-item authoring

In the following, the user enters a new k-item that describes how Sam's meetings with visitors should be scheduled.

* k-item-1: Meetings with Sam who is a division director should be scheduled in the late afternoon (5pm, 5:30pm) because he often cancels meetings.

Echo: I am not so confident about k-item-1. It conflicts with some of the recent decisions. For example, do you remember Ken's visit? Sam met Ken at 3pm. …

Do you want to modify k-item-1 instead of ignoring these conflicts? For example, you can do one of these:
- specify why 3pm was chosen for Ken's visit
- include 3pm in the suggestion

In this case, the user specifies that 3pm was chosen for Ken's visit because Ken had to leave early. The k-item is modified accordingly.

* k-item-1': Meetings with Sam should be scheduled in the late afternoon (5pm, 5:30pm) *if the visitor doesn't leave early*.

Echo: I am now confident about k-item-1.

The confidence level of the modified k-item is higher because it is now consistent with past decisions.

Unlike in existing knowledge authoring interfaces, Echo relates a new definition to the problems solved over time, assesses the confidence level of the item using the relations, and makes suggestions in the context of improving the confidence level and making the overall problem solving more successful. Without Echo's memory-based meta-level reasoning, the system may not recognize the inconsistencies the user needs to pay attention to and also may not provide feedback on how user's k-item modifications improve the knowledge base.

Assisting problem solving with user entered k-items

When the user has a new visitor and starts a new decision on Sam's meeting time, the system looks for the k-items that can provide confident suggestions on the decision. When it is known that the visitor does not leave early, the system uses k-item-1 with high confidence.

Echo: suggest selecting late afternoon (5pm, 5:30pm) <u>with high confidence.</u>

If the user's decision conflicts with confident k-items, the system notices it as an unexpected event and suggests examining how relevant decisions were made in the past before moving onto other tasks. For less confident k-items, modification or deactivation of the k-items is considered with higher priority. For example, if the user selects a meeting time before 4:30pm, the system generates this output.

Echo: Are you sure about your decision? Do you remember Sam's visit and Ben's visit? You have selected late afternoon times.

If there were actual changes (e.g., Sam is not a division director anymore and he is now less busy), k-item-1 is no longer useful, and the user may choose to modify or deactivate it. Based on the user actions, the system may recognize that there are some changes and analyze further potential modifications. For example, the system suggests examining past decisions where k-item-1 was used and check whether they are now invalid. This may affect other k-items that were actively used in the invalid episodes since they may be now less reliable.

The next section presents the Echo system architecture and shows how Echo works while the user interacts with the system.

3. MEMORY-BASED META-LEVEL REASONING

Echo's meta-level analysis is driven by these meta-level goals:

- ensure that the user entered k-items are consistent and complete
- ensure that k-items are useful in problem solving
- ensure that k-items adapt to changes in the world

Figure 1 shows architecture of a knowledge-based system where Echo's meta-level reasoning supports knowledge authoring and problem solving. The highlighted boxes in the figure represent the main components that support meta-level reasoning. The knowledge authoring interface makes use of the results from the meta-level reasoner and provides proactive assistance in making k-items more useful. Likewise, during problem solving, the system makes use of the results from the meta-level reasoner and provides assistance based on the level of confidence assessed through memory-based analysis.

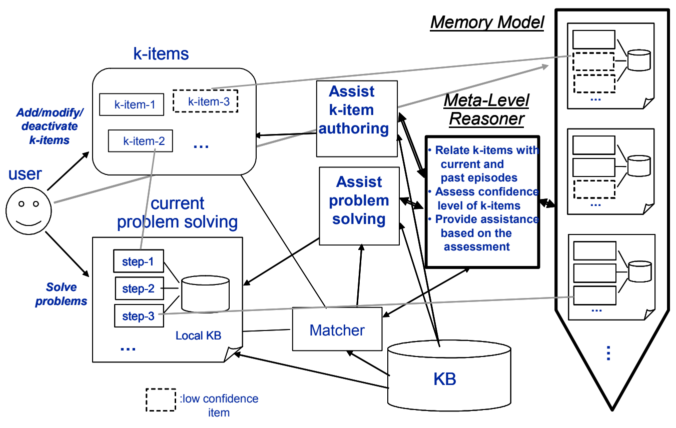

Figure 1: Architecture

The following introduces the memory model and describes how it is used in assisting the user.

Memory model

We assume that each problem solving episode consists of a set of problem solving steps. A k-item is applicable to a problem solving step when its trigger conditions match the features that describe the problem and the problem solving context. For example the above k-item-1 is used when the user tries to schedule a new meeting with Sam.

Echo keeps track of past problem solving episodes and relates them to each k-item. Each problem solving episode consists of:

- problem features that describe a problem
  (e.g. schedule activities for a new visitor Carl)
- definitions and terms used during problem solving
  (e.g. Carl's research interests, visit date, etc.)
- a set of problem solving steps
  (e.g. decide talk time, decide talk location, ...)

Each problem solving step maintains:

- problem solving context
  (e.g. deciding meeting time for Sam)
- problem solving results
  (e.g. selected 5pm for Sam's meeting time)
- k-items matched
- k-items that are actually used in the problem solving
- timestamp of the step
- whether it is valid

The user can indicate out-of-date problem solving steps as invalid. The invalid steps are ignored in assessing the confidence-levels of the k-items.

Echo relates new k-items to the problem solving steps and computes these:

- steps it matches
- steps it can potentially contribute to

- steps it conflicts with
- steps where it was used

This structure is incrementally updated for new problem solving steps, as described below.

Assessing confidence level of k-items

Based on the memory model described above, Echo infers whether a k-item can be confidently used, whether it conflicts with some successful results and needs some modifications, or it needs significant changes including deactivation.

Echo calculates the *confidence level* of k-item i using a simple equation as follows:

$$w1*P(i) + w2*R(i) + w3*U(i) \quad (1)$$

where P(i) is the performance level of i, R(i) is the application recency of i, and U(i) is the application frequency of i.

$$P(i) = \begin{cases} 0, & \text{if } \#\text{matched} = 0 \\ \#\text{ used} / \#\text{ matched}, & \text{otherwise} \end{cases}$$

$$R(i) = \begin{cases} 0, & \text{if } \#\text{matched} = 0 \\ 1 / (t(i) + 1), & \text{otherwise} \end{cases}$$

where t(i) is time elapsed since the most recent match

$$U(i) = \begin{cases} 0, & \text{if } \#\text{ problem solving steps} = 0 \\ \#\text{ matches} / \#\text{ problem solving steps}, & \text{otherwise} \end{cases}$$

In each equation, the division normalizes the metric so that the value is bound between 0 and 1. The relative weights of the metrics can be chosen depending on the strategies a given application takes.

The user can also explicitly indicate whether a k-item is obsolete and should not be used in problem solving.

Reflection: analyzing k-items with memory model and providing assistance

The following algorithms show how memory model changes through knowledge authoring (add new k-item, modify a k-item, and deactivate/delete a k-item) and problem solving. They also show how Echo makes use of the memory model in providing assistance to the user. These algorithms are general and independent of a given application or the knowledge representation used.

```
Add-new-k-item (k-item i)
  1) find-matching-problem-solving-steps-in-memory (i)
  2) check whether the matching steps are consistent with i
  3) compute the confidence level of cl of i based on 1) & 2)
  4) When there are inconsistent steps,
     if cl is low, suggest modifying i to make it
      consistent with the matching steps
       using modify-k-item-to-make-it-consistent-with-step(i, s)
      rather than ignoring or invalidating the inconsistent steps
    otherwise (cl is high), for each inconsistent step s,
      suggest checking whether s is still valid
      rather than modifying i
```

```
Modify-k-item(k-item i-old, k-item i-new)
  1) if the confidence-level of i-old was high
       (Echo notices that there are some changes)
        2-1) find steps where i-old was used
        2-2) suggest checking whether the steps are still valid
  2) find changes in the matching steps using
       find-matching-problem-solving-steps-in-memory (i-new)
  3) detect any inconsistencies between i-new and
       the matching steps
  4) compute the confidence level cl of i-new based on 2) & 3)
  5) When there are inconsistent steps,
       if cl is low, suggest modifying i-new further to make it
        consistent with the matching steps using
         modify-k-item-to-make-it-consistent-with-step(i, s)
        rather than ignoring or invalidating the inconsistent steps
      otherwise (cl is high), for each inconsistent step s,
       suggest checking whether s is still valid
       rather than modifying i-new
```

```
Deactivate-k-item (k-item i)
  1) find past problem solving steps where i was used
  2) for each step s, unless s is consistent with other k-items
      whose confidence-level is high,
      suggest checking whether s should be invalidated
  3) deactivate i
```

```
Perform-problem-solving-step (step s)
1) find-matching-k-items(s)
2) find potentially applicable high-confidence k-items
   using find-potentially-matching-k-items(s)
    e.g. identify potentially missing problem features for match
      "is the visitor staying until late afternoon?"
3) assist the problem solving with the matching k-items based on
    their confidence levels
4) observe actual problem solving of s
5) check whether the results are consistent with the k-items that
    are brought up
6) if there are inconsistent k-items whose confidence level is high,
     (Echo notices unexpected event)
      suggest examining s rather than
      modifying or deactivating confident k-items
   otherwise, for each inconsistent k-items i,
      suggest modifying or deactivating i to make it
      consistent with s using
       modify-k-item-to-make-it-consistent-with-step(i, s)
```

```
Invalidate-past-problem-solving-step (step s)
  1) find k-items that match s
  2) for each matching k-item i,
       2-1) update the confidence-level cl of i ignoring s
       2-2) if cl becomes very low,
              suggest modifying or deactivating it
  3) invalidate s in the memory
```

When the user creates a new k-item, Echo relates it to past problem solving episodes and analyzes whether it conflicts with any past episodes. When the estimated confidence level of the new k-item is low, Echo suggests modifying the k-item to avoid the conflicts. For example, the k-item can be specialized to avoid matches with conflicting steps. In the example in Section 2, k-item-1 was specialized to be applicable only when the visitor does not leave early. The system can propose different modifications depending on the domain and the representation language used as described below.

A modification of a k-item may also result in changes in the memory model. When a confident k-item is modified, Echo notes that there are potential changes in problem solving and suggests checking whether the old episodes are now invalid. When there are new conflicts noticed, Echo may suggest further modifications of the k-item.

Deactivation of k-items occurs when they are no longer needed or relevant to the problem solving. It may trigger invalidation of the problem solving steps where they were actively used.

The user authored k-items are used in solving new problems. Echo searches for potentially useful high-confidence k-items as well as the k-items that actually match. During problem solving, Echo re-assesses the confidence level of the k-items based on their actual usages in the problem solving and analyzes whether the k-items need any modifications.

Invalidation of problem solving episodes and their steps occur when the way problems are solved changes and the old episodes are no longer compatible with the new approach. The invalidated episodes are not used in assessing k-items or making suggestions.

The supporting sub-procedures, such as *find-matching-problem-solving-steps-in-memory*, *find-matching-k-items, find-potentially-matching-k-items, and modify-k-item-to-make-it-consistent-with-step* are defined based on the given knowledge representation and the domain they are used for. The procedures that we use for visitor hosting domain are described in Section 4.

In the current Echo memory model, k-items are related to each other through problem solving steps that they match. Inconsistencies between k-items are addressed by making them consistent with the same problem solving steps. The system can predict an overlap between two k-items when they match the same steps and provide the same suggestions.

4. APPLICATION OF ECHO: ADAPTIVE DECISION AID

In this section we show the details of how Echo has been applied to a visitor hosting domain. First we describe the characteristics of manual decision making applications and then show how we map Echo's functions to the knowledge authoring and problem solving tasks.

Supporting manual decision making

There are many decision making tasks that are done *manually* in everyday life including business planning, engineering, and event scheduling. Each decision has many options, some are better than others. People often pick bad options because they are unaware the good options or they don't remember how the decisions were made in the past. Some of the decisions are inter-related each other and there can be many constraints across the decisions. Often one or a small number of final solutions are derived from multiple decisions.

Many of these tasks are done manually because it is very expensive to build a complete knowledge base that supports automatic tools. Dynamic changes in the world, such as new business rules and changes in engineering parts, also pose challenges in developing such knowledge base.

We took an alternative approach where the user incrementally authors simple decision constraints based on failures and successes of decisions that are observed. These constraints (k-items) are used in providing suggestions for new decisions, improving the quality of the decisions over time. As more k-items are captured, the degree of automated assistance increases. We have used the Echo architecture in developing such intelligent assistant. Echo's meta-level reflection capability supports adaptation of k-items that is essential in these applications. Here we focus on the visitor hosting domain that is described above. Echo is being applied to other knowledge authoring applications including capturing source recommendations from information analysts.

In the following, we first describe the basic components that support knowledge authoring and problem solving in the visitor hosting domain, and then show how Echo's meta-level reasoning improves the assistance provided by the system.

In this domain, each problem consists of 1) a set of domain features that describe the situation and 2) a set of decisions that should be made to solve the problem. For example, a visitor hosting problem consists of a set of features that describe a particular visit (e.g. visitor, host, visit-date, visit-purpose etc.) and a set of decisions on visitor activities including talk, meetings, and lunch. In our implementation, we use simple *triples* for representing domain features, which is compatible with semantic web standards such as RDF and OWL. For example, (Visit1 visitor Carl) means that the current problem of Visit1 has Carl as the visitor.

The following shows the application specific sub-procedures that support the general Echo algorithms described in Section 3.

```
------------------------------------------------------------------------
find-matching-problem-solving-steps-in-memory (k-item i)
   1) find matching past decisions using domain features
      in the episode and the decision type
modify-k-item-to-make-it-consistent-with-step (k-item i, decision d)
      - specialize i to avoid matches with d or
      - add options chosen in d as good options or
      - remove bad options avoided in d or
      - deactivate i
find-matching-k-items (decision d)
   1) find matching k-items using the domain features introduced
      for the current decision problem and the decision type

find-potentially-matching-k-items (decision d)
   1) find high confidence k-items that can match d
      with an additional problem feature
   2) create temporary definitions of the missing features
      based on other features used in the match
   3) query the user whether the features are actually satisfied
------------------------------------------------------------------------
```

Assisting k-item authoring

Each k-item provides suggestions by reminding potentially good options and bad options for a decision. The user may enter new k-items to avoid the same mistakes or repeat successes in similar decisions. The user can create a new k-item by selecting a decision (e.g. deciding meeting time of an activity) and a set of features that are relevant to making the decision (e.g. the type of visit is AI seminar) and the recommendations (what are good and bad options). The user can also enter a free text that describes the details of why such suggestions were made. We found that often formal representations cannot capture the details of the decision rationale and free text helps the user understand how he/she can make use of the k-item suggestions during the decision making process.

Given these, the system performs simple generalization to create a rule form. A k-item consists of 1) a set of general domain features that describe the kinds of problems where the k-item can be used, 2) a decision type that it can provide suggestion on, 3) suggestions: what options to choose and what options to avoid, and 4) free text the user entered.

For example, k-item-1 is represented as:

```
In deciding (ACTIVITY_0 when ACTIVITY-TIME)
  If  (VISIT_0 activities ACTIVITY_0)
      (VISIT_0 type AI-seminar)
     (ACTIVITY_0 who Sam).
  Then CHOOSE {17:00, 17:30}
  "Sam is very busy and often cancels meetings."
```

A more sophisticated representation can be used but it is outside the scope of the current work.

In this type of domain, the initial knowledge base often lacks definitions of domain terms, and the system needs to handle uses of undefined terms during knowledge authoring flexibly. Our system automatically creates initial definitions of undefined terms using the context where they are used and assist users in creating new definitions. For example, a visitor's research interest will be defined as an instance of a research topic.

In the basic knowledge authoring interface without Echo, when the user enters k-item-1 shown in Section 2, the system cannot estimate its confidence level and cannot recognize conflicting decisions that the user needs to pay attention to more closely. Even if the system detects some errors using some available debugging capabilities and produces some suggestions[1], it may not be able to point which k-item modification is more desirable. When the user modifies a k-item, the basic system can neither provide feedback on how user's k-item modifications improve the knowledge, nor relate it to other changes needed for the k-items that participate in the same decision steps.

Assisting problem solving (decision making)

The user created k-items are applied whenever their decision context and feature descriptions match the current decision step. In order to cope with incompleteness of given problem descriptions that often appears in interactive problem solving, the system makes use of high confidence k-items that are potentially applicable as well as the k-items that match the problem exactly. This is done by identifying potentially missing problem features during the match and querying the user whether they are actually satisfied.

Without Echo, all the suggestions are provided uniformly as in most existing problem solving systems. With Echo, the system can suggest how the k-item should be applied. Depending on the level of confidence assessed for the k-item used, the system can provide different suggestions in a different degree of strength.

Supporting Adaptation

Without Echo, the system cannot recognize there are changes in how the decisions are made and cannot help users make appropriate k-item modifications. With Echo, the system notes significant changes when confident items are modified or deactivated, and predicts further modifications needed. For example, when confident k-item-1's suggestion is overridden, the system notices it as an unexpected event and suggests examining whether the past decisions that were consistent with the old definition are still valid. When some of the decisions become invalid, the confidence levels of the matching k-items change and the system may predict further modifications of the k-items

[1] Additional source of user guidance in the basic interface may be combined with Echo's results as discussed in Section 5.

whose confidence level drop significantly. Table 1 summarizes these additional capabilities provided by Echo.

System Assistance	Without Echo	With Echo
k-item creation	-Detect errors in new definitions - Suggest modifications that resolve the errors	- Detect conflicts with past problem solving results as well as the errors in the definitions - Estimate the confidence level of the new definition - Suggest modifications that improve the overall problem solving as well as resolve the errors
k-item modification	-Do not provide feedback on how the modification changes the knowledge base - Do not recognize changes in problem solving	- Provide feedback on how the modification improves the knowledge base - Recognize whether there is a significant change in problem solving and predict further modifications - Relate it to other changes needed for the k-items that participate in the same episodes
problem solving	- K-items are used uniformly - do not recognize changes in problem solving	- Control the use of k-items based on their confidence level - Recognize whether there is a significant change in problem solving and predict further modifications

Table 1: Additional assistance provided by Echo.

5. RELATED WORK

There have been various techniques developed to help end users directly author problem solving knowledge. Some tools exploit strong background knowledge [2, 11] and or specific tasks and problem solving strategies [3,19] to guide the user. Some use constraints from knowledge representation language and prototypical knowledge authoring steps [7,24]. Other tools focus on detecting errors in the knowledge entered by the user [4,20]. The Echo algorithms can be used in combination of other sources of user guidance available in the system. For example, when the system detects errors in new k-item definitions, the results may be combined with Echo's analysis and used in further prioritizing possible k-item modification actions. Other language dependent or task dependent features may be also exploited in constraining the knowledge authoring actions.

Some knowledge authoring tools make use of test cases in validating or synthesizing new definitions [1,10]. However these systems do not exploit the history of how k-items are built and how they are used, and cannot associate them with the progresses the user makes over time.

There are some case-based reasoning efforts concerned with utility of cases and case maintenance [23,26,18,16]. Although their focus was not interactive knowledge authoring and the analysis does not explicitly use the history of knowledge changes, their case evaluation techniques can be used in combination with the existing k-item confidence metrics.

In the past, we have developed a dialogue tool for interactive knowledge authoring [15,9]. The tool incorporates the dynamics of tutor-student interactions in order to support users in their role of tutors of computers, making authoring interfaces better students. Assessment of k-items and their progresses over time in Echo can be combined with other dialogue strategies and be used in structuring the front-end interactions for knowledge authoring.

6. SUMMARY AND FUTURE WORK

This paper presents a novel architecture for knowledge-based interfaces that exploits memory-based meta-level analysis of k-items in guiding knowledge authoring and problem solving. Unlike existing knowledge authoring interfaces, Echo assesses the status of knowledge authored by the user and exposes the assessment results, allowing the user to understand what the system has learned and how it can be further improved. The architecture is used in developing an intelligent assistant for manual decision making tasks. The resulting system guides the knowledge authoring process in terms of making the k-items more useful, adapting k-items to dynamic changes in the problem solving over time, and making the overall problem solving more successful.

We are currently extending the Echo algorithms in order to assess the space of problems covered by the k-items and predict what types of new k-items are needed. It will be based on the problems solving steps where there are no applicable k-items. Furthermore, the system will assess its competence level in solving various types of problems.

We are also building a more explicit model of k-item relations, combining the existing memory model and the model of interdependencies between k-items [12]. This will allow us to analyze evolution of relevant k-items and to predict related changes more directly.

We plan to perform more intensive analysis of the user interactions with the system and evaluate the performance of Echo in terms of effectiveness of assistance and quality of k-items built. An ablation experiment, where the baseline tool resulting from eliminating Echo's meta-level reasoning is compared with the full Echo system, is considered.

7. REFERENCES

1. Bareiss, R., Porter, B., Murray, K.: Supporting Start-to-Finish Development of Knowledge Bases. Machine Learning 4: 259-283 (1989).
2. Barker, K., Clark, P. and Porter, B., A Library of Generic Concepts for Composing Knowledge Bases.

Proceedings of the First International Conference on Knowledge Capture (K-CAP-2001), pp. 14-21, 2001.

3. Birmingham, W and Klinker, G., Knowledge-acquisition tools with explicit problem-solving methods, The Knowledge Engineering Review, 8 (1), pp. 5-25, 1993.
4. Blythe, J. Kim, J., Ramachandran, S., and Gil, Y., An Integrated Environment for Knowledge Acquisition. Proceedings of the International Conference on Intelligent User Interfaces (IUI-2001), 2001.
5. Clark, P., Thompson, J., Barker, K., Porter, B., Chaudhri, V., Rodriguez, A., Thomere, J., Mishra, S., Gil, Y., Hayes, P. and Reichherzer, T., Knowledge Entry as the Graphical Assembly of Components. Proceedings of the First International Conference on Knowledge Capture (K-Cap-2001), pp. 22-29, 2001.
6. Cypher, A. Watch what I do: Programming by demonstration. Allen Cypher, Ed. MIT press, 1993.
7. Davis, R., Interactive Transfer of Expertise: Acquisition of New Inference Rules. Artif. Intell. 12(2): 121-157 (1979).
8. Friedland, N., Allen, P., Witbrock, M., Matthews, G., Salay, N., Miraglia, P., Angele, J., Staab, S., Israel, D., ,Chaudhri, V., Porter, B., Barker, K., and Clark, P., Towards a Quantitative, Platform-Independent Analysis of Knowledge Systems. KR 2004: 507-515.
9. Gil, Y. and Kim, J., Interactive Knowledge Acquisition Tools: A Tutoring Perspective, Proceedings of the 24th Annual Meeting of the Cognitive Science Society (COGSCI-2002), pp. 357-362, George Mason University, Fairfax, Virginia, 2002.
10. Ginsberg, A., Weiss, S., Politakis, P., SEEK2: A Generalized Approach to Automatic Knowledge Base Refinement. IJCAI 1985: 367-374.
11. Huffman, S., Laird, J.,Flexibly Instructable Agents. J. Artif. Intell. Res. (JAIR) 3: 271-324 (1995).
12. Kim, J. and Gil, Y., Deriving Expectations to Guide Knowledge Base Creation. Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-1999), 235-241, 1999.
13. Kim, J. and Gil, Y., Acquiring Problem-Solving Knowledge from End Users: Putting Interdependency Models to the Test. Proceedings of the Seventeenth National Conference on Artificial Intelligence, pp. 223-229 (AAAI-2000).
14. Kim, J. and Gil, Y., User Studies of an Interdependency-Based Interface for Acquiring Problem-Solving Knowledge. Proceedings of the Intelligent User Interface Conference (IUI-2000), 165-168, 2000.
15. Kim, J. and Gil, Y. Deriving Acquisition Principles from Tutoring Principles, Proceedings of the Intelligent Tutoring Systems Conference (ITS-2002) , pp. 661-670, 2002.
16. Kira, Z. and Arkin, R., Forgetting Bad Behavior: Memory Management for Case-Based Navigation, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).
17. Lau, T., Wolfman, S., Domingos, P., and Weld, D. Programming by Demonstration using Version Space Algebra, Machine Learning, 2003.
18. Leake, D., and Wilson, D., Guiding case-base maintenance: Competence and performance. In Proceedings of the 14th European Conference on Artificial Intelligence Workshop on Flexible Strategies for Maintaining Knowledge Containers, 2000.
19. Marcus, S., SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In Marcus, S., editor, Automating Knowledge Acquisition for Expert Systems, pages 81—123, 1988.
20. McGuinness, D., Fikes, R., Rice, J., and Wilde, S., An Environment for Merging and Testing Large Ontologies, Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), 2000.
21. Mitchell, T., Mahadevan, S. and Steinberg, L., LEAP: A learning apprentice for VLSI design. Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85), pp. 574-580, 1985.
22. Pool, M., Murray, K.. Fitzgerald, J,. Mehrotra, M., Schrag, R., Blythe, J., Kim, J., Chalupsky, H., Miraglia, P., Russ, T., and Schneider D., Evaluating SME-Authored COA Critiquing Knowledge, Proceedings of the International Conference on Knowledge Capture (K-CAP 2003), 2003.
23. Smyth, B. and Keane, M., Remembering to Forget: A Competence-Preserving Deletion Policy for Case-Based Reasoning". In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, 1995.
24. Tallis, M., and Gil, Y.,. "Designing Scripts to Guide Users in Modifying Knowledge-based Systems". Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), 1999.
25. Tallis, M., Kim, J. and Gil, Y., User Studies Knowledge Acquisition Tools: Methodology and Lessons Learned. Journal of Experimental and Theoretical Artificial Intelligence, Vol 13. No 4., 2001.
26. Zhu, J. and Yang Q., Remembering to Add: Competence-preserving Case-Addition Policies for Case Base Maintenance. (IJCAI-1999): 234-241, 1999.