Results of Encoding Knowledge with Tutor Construction Tools[1]

Tom Murray[2] and Beverly Park Woolf
Computer Science
University of Massachusetts
Amherst, Mass. 01003
{tmurray, bev}@cs.umass.edu

Abstract

We have developed and evaluated a set of tutor construction tools which enabled three computer-naive educators to build, test and modify an intelligent tutoring system. The tools constitute a knowledge acquisition interface for representing and rapid prototyping both domain and tutoring knowledge. A formative evaluation is described which lasted nearly two years and involved 20 students. This research aims to understand and support the knowledge acquisition process in education and to facilitate browsing and modification of knowledge. Results of a person-hour analysis of throughput factors are provided along with knowledge representation and engineering issues for developing knowledge acquisition interfaces in education.

Motivation

This research addresses issues of scaling up and shaking down the tutor construction process. It asks how educators' concepts of subject matter and teaching methods can be transformed into a detailed yet flexible conceptualization consistent with computer based tutoring. We report on the usability and efficiency of a knowledge acquisition interface in the form of a set of tutor construction tools.[3] Our formative evaluation of the tools includes a user-participatory design process [Blomberg & Henderson, 1990] and a case study of three educators using the tools to develop a knowledge based tutor for statics. In this paper we give data suggesting that educators (with with appropriate tools and the assistance of a knowledge engineer) can build a tutor with an effort comparable to that required to build traditional computer aided instructional (CAI) systems. We report on one of the first evaluations of a knowledge acquisition interface for a tutor and the first such interface tailored exclusively for use by practicing educators. We expect that only a few master teachers will participate on tutor design teams, but the tools will allow a wide range of educators to easily customize the resulting systems.

Our study is not an evaluation of the effectiveness of either a specific tutor or a particular instructional approach. Since this research area is new and few methodological guidelines or prior results exist, it is an "exploratory" study. We focus on design and knowledge acquisition issues of knowledge based tutors, and, though the statics tutor was tested on 20 students, we did not measure student learning, but rather observed how the teacher collected and analyzed data to update the tutor.

Other generic shells for AI-based tutoring systems have been built, including SIIP [Macmillan et al., 1988], IDE [Russell et al., 1988], ID Expert [Merrill, 1989] and Byte-sized Tutor [Bonar et al., 1986]. These systems are *intended* to be used by educators or instructional experts who are not programmers. However, they are focused more on generality than on usability. They do not clearly address the issues encountered when educators actually use these systems. Few researchers (none that we know of) have completed a user-based analysis of a generic tutor shell and the associated knowledge acquisition issues. Though we have focused on teachers, the tools for rapid prototyping of ITS domain content and tutoring strategies should also be valuable to industry trainers, instructional designers, and educational researchers.

Knowledge Acquisition Issues

The knowledge acquisition component of this research is similar to other research which addresses techniques for eliciting domain and heuristic knowledge from experts (e.g., [Bareiss, et. al., 1989; Mullarkey, 1990]). However, this work focused on several knowledge acquisition issues specific to education, beginning with the difficulty of enlisting *domain expert participation*

[1] This work was supported by the National Science Foundation under grant number MDR 8751362 and by External Research, Apple Computer, Inc., Cupertino, CA.

[2] Murray is currently working in the Interactive Multimedia Group at ABB Combustion Engineering, Research and Technology, Power Plant Laboratory, Windsor CT.

[3] Also called KAFITS, Knowledge Acquisition Framework for ITS.

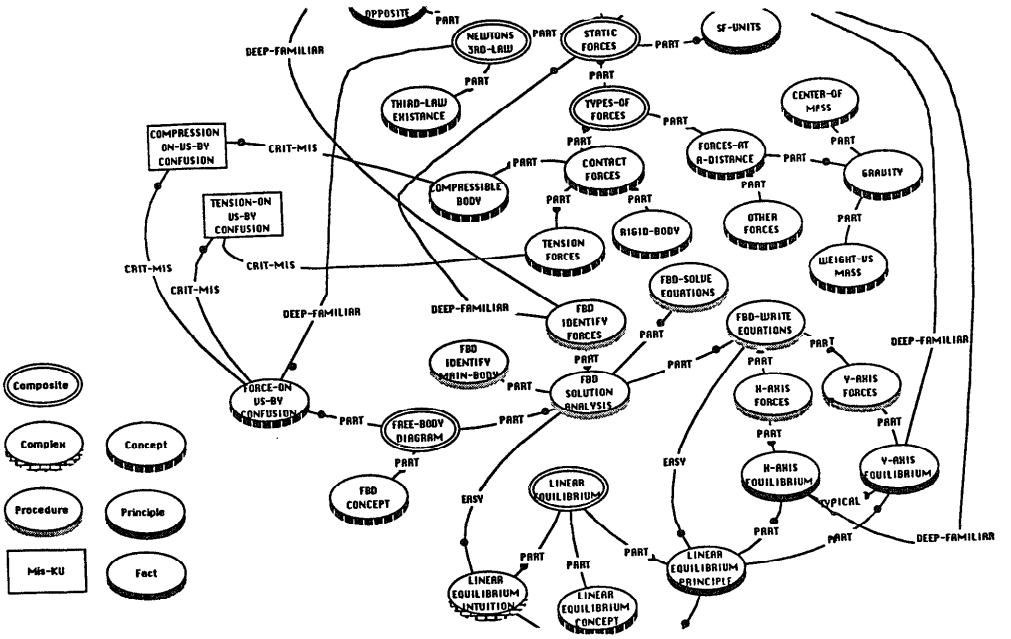

Figure 1: Topic Net Editor, Showing Part of the 41-node Statics Topic Network

in the process of building teaching systems. In fact, understanding, acceptance and use of AI research by educators has been much slower than research progress [Clancey & Joerger, 1988], due in part to the lack of tools for constructing these programs. Increased coordination between domain experts and evolving tutors is needed because the insights, principles and rules built into these systems should originate from a rich synthesis of the experience of practicing teachers, learning and instructional theories, and on-line experimentation. Yet, complex knowledge bases present a tremendous challenge to domain experts and knowledge engineers, and often results in causing them to be lost in data space [Carroll et al., 1990] or to make inappropriate or ineffective changes [Terveen and Wroblewski, 1990].

In this paper we discuss design principles and give results of a quantitative nature regarding the design of a physics tutor using the development tools, e.g., How much effort is required to build a knowledge based tutoring system?[4]

Design principles for a visual knowledge editor. The tutor construction tools are an example of a *visual knowledge editor*, here defined as a system which makes knowledge visible and communicable. Such an interface is based on the highly asymmetrical abilities of people and computers [Terveen and Wroblewski, 1990] and is designed to reduce cognitive load on human working memory and to reify a person's underlying domain knowledge structure. Such an editor in an educational application should visualize domain knowledge and tutoring heuristics and communicate complex conceptual relationships instantiating an educator's mental model of a domain.

We found a number of design principles which were key to the usability of the tools. The interface for a knowledge based tutor authoring system should:

1. Provide the user with a clear and accurate cognitive model of the underlying framework;
2. Make it hard for the user to inadvertently change the knowledge base;
3. Give feedback on the operation just completed;
4. Help the user "self-locate" and "navigate" within the knowledge base;
5. As much as possible, anticipate the cognitive capabilities and limitations of the typical user;
6. Provide features which reduce cognitive load on working memory (by reifying information and structure) and long term memory (by providing reminders);
7. Allow the user to view (*static*) knowledge base in multiple ways and to view relationships between items;

[4] In [Murray & Woolf, 1992] we describe our tutor design process, and how the development tools were used to assist in this process. The representational framework and student model are described in [Murray & Woolf, 1990], and in [Murray, 1991] we discuss issues for representing curriculum and tutoring strategy knowledge, and how the conceptual models and cognitive limitations of the practicing teacher affect his/her ability to conceptualize curriculum and the instructional processes.

8. Provide 'monitors' which allow the user to view the changes in *dynamic* processes and data structures;
9. Provide on line assistance and reference information;
10. Facilitate opportunistic (i.e. top down and bottom up) design by not forcing the user to make decisions in a rigid order;
11. Allow quick movement and iteration between test and modification.

Research Goals and Methodology. Our goals were to identify effective knowledge acquisition interface *components* and to research the *process* of working with educators to build domain and tutoring strategy knowledge-bases. In order to identify interface components, we maintained a sixteen month user-participatory design methodology along with an iterative design/formative evaluation, i.e., design and implementation were iterative and concurrent with use by a domain expert and two "knowledge base managers" (see below). This 'participatory design' methodology supported considerable teacher involvement in the construction of the system and shifted the role of educators from software 'users' or research 'subjects' to co-researchers or co-developers [Blomberg & Henderson, 1990; Whiteside et al., 1988].[5]

To research the *process* by which educators can design, modify and evaluate a tutoring system we used the case study methodology. The work reported here describes the methodology used as well as a portion of the data collection, results and analyses. The participants were three educators who had little or no programming experience before the project: one exemplary high school physics teacher took on the role of domain expert,[6] and two education graduate students took on the role of **knowledge base managers**, responsible for the initial entry of curriculum content, for non-content-specific debugging of the knowledge base, and for the supervision of student test runs. The first author acted as knowledge engineer. The three participants suggested changes to the system and saw those changes manifest. The experimental data included 93 pages of field notes (in which the knowledge engineer recorded information and impressions from interviews with the domain expert and observations of system use), a daily record of all code changes, data files tracing knowledge base editing sessions and student tutorial sessions, post-session interviews with students, and a post-study interview with the domain expert.

Results of Case Study

Brief description of the tutor. The tutor was designed to convey a qualitative, intuitive understanding of relationships among forces in static (non motion) situations [Murray & Woolf, 1990]. Its objectives in the area of linear equilibrium, for instance, are for a student to predict the motion or non-motion of objects with forces exerted on them and to isolate the X and Y components of forces in static situations. Figure 1 shows a portion of the curriculum. The tutor is able to diagnose and remediate several common physics misconceptions, including the belief that stationary or solid objects do not exert forces, and confounding the forces *acting on* an object with the forces *exerted by* an object. The tutor asks qualitative questions about real-world objects and situations. Additionally, an interactive "crane boom simulation" can be presented, allowing the student to manipulate connected objects and to observe resulting force magnitudes and vectors. The student can interrupt the tutorial session at any time, skip or repeat the current presentation, explore and inquire freely, change the teaching style, ask for hints, definitions, explanations, etc., or visit another part of the curriculum.

	Step	Develmt	Training	Total
1.	Introduction	0	3.5	3.5
2.	Brainstorm	12	6.2	18.2
3.	Class. Script	9	5	14
4.	Topic Net	9	5	14
5.	CAI Script	45	5	50
6.	Strategies	7	3	10
7.	Work Sheets	80	5	85
8.	Data Entry	148	3	151
9.	Debugging	83	4	87
10.	Testing	81	3	84
Design (steps 1-4, 6)		37	23.7	59.7
Implement. (5, 7-10)		438	19	457
Totals	all	474	42.7	517

Figure 2: Person-hour Analysis for Building the Statics Tutor

Analysis of the curriculum and knowledge base. The driving questions for the quantitative analysis of the data were:

- How can the size and structure of the knowledge base be measured to allow for comparative analysis of knowledge bases from different knowledge based tutors?
- How much effort is required by each participant in each step of designing the statics tutor?
- Approximately how much time did it take (per hour of instruction and per curriculum topic) to design the statics tutor?

Instructional units such as 'topics' and 'lessons' can have different meanings for different knowledge based tutoring systems, and system-independent metrics are

[5] See [Murray, 1991] for a discussion of how user participation affected the design of the tools.

[6] Dr. Charles Camp is a physics teacher at the Amherst Regional High School, Amherst, Massachusetts. We will refer to him as the "domain expert" or "teacher."

needed to compare terminology and the size and complexity of tutors. We propose three such metrics: *student tasks per instructional unit,*[7] *curriculum links per instructional unit,* and *transactions per instructional unit. Transactions* are defined as the smallest functional unit of discourse, such as a motivation paragraph, an explanation, a question, a hint, etc. The average number of student tasks per instructional unit is a rough indication of the amount of interaction in a tutoring system. The 'curriculum links per instructional unit' (links per topic network node in our system) is an indication of curriculum interconnectedness and complexity. The average 'transactions per instructional unit' (transactions per topic in our system) gives a rough indication of the amount of content represented by an instructional unit. The *total* number of transactions is a relatively system-independent metric for comparing the size of tutor knowledge bases encoded within different representational frameworks.

The following metrics were calculated for the statics tutor knowledge base (see [Murray 1991] for a more thorough analysis):[8]

- **Average presentations (student tasks) per topic (instructional unit):** 3.
- **Average curriculum links per topic (instructional unit):** 1.2.
- **Average transactions per topic (instructional unit):** 12.

Though we have not yet performed a detailed comparison of our curriculum knowledge base with those from other systems, we used these characteristics to compare the linear equilibrium part of the statics curriculum (the 25% that was tested with students) to the entire curriculum so that we could justify extrapolating some data from the linear equilibrium portion to the entire curriculum.

Productivity analysis. Figure 2 shows a person-hour analysis for the various steps in designing the statics tutor. It is organized with project steps in the columns and project activities in the rows. Project steps include *design*, e.g., brainstorming, classroom script design, topic network design, tutoring strategies design, and *implementation*, e.g., generating a CAI-like script, completing work sheets, data entry, debugging the knowledge base, and student trial runs (see [Murray, 1991] for a description of each design step). Within each step there are two kinds of 'activities:' *development* (including production and guidance) and *training.* These numbers do not include the following: the effort spent programming the simulation, the effort spent (by an artist) creating graphic illustrations, nor the time to build the development tools. Some of the figures are based on estimates and our numerical analysis is only intended to give order of magnitude figures. We believe the figures can be used to estimate production time for similar projects using the tutor construction tools.

We analyzed the productivity data in several ways (see Figure 3[9]), including: as a function of design step (design vs. implementation), as a function of design activity (training vs. development), and as a function of participant role (domain expert vs. knowledge engineer and knowledge base manager). Following are some summary figures from this analysis:

- Total design time for the 6-hour curriculum was 596 hours, or about 15 hours per topic.
- The effort per hour of instruction time was about 100 hours (the entire curriculum represents about six hours of teaching). Factoring out training, it took about 85 hours per hour of instruction.
- The domain expert's time was about the same as the knowledge base manager's time, and the knowledge engineer's time was about one third of the others. The knowledge engineer's effort represented about 15% of the total effort.
- Implementation was about six times the effort of design, i.e. design was about 15% of the total effort.
- Training took about 15% of the total effort. Most of this training was one-time training that would *not* have to be repeated for the next tutor built by the same team. The training required for the knowledge base managers was small, taking only one sixth as much time as the domain expert's training (see [Murray, 1991] for a discussion of training content).
- We also analyzed such things as the average time to enter curriculum information to create a new object in the knowledge base (25 minutes), and the average time for making knowledge base modifications during test runs (6 minutes).
- Over the course of the project we saw a two-fold increase in efficiency using the knowledge base Browser. We can not determine, however, how much of this change was due to improvements in the knowledge base and how much was due to increased user experience.

[7] A student 'task' is an activity (question, request, etc.) that the tutor gives to the student. These are represented by 'presentations' in our system. An 'instructional unit' is any unit of chunking content, and will differ according to the tutoring system, e.g. topic, lesson, concept, rule, etc. 'Topics' are the main instructional unit for our system.

[8] Other characteristics were determined. The topic network contains 41 nodes (39 topics and 2 misconceptions), each topic representing (very roughly) about 9 minutes of instruction. There were 252 curriculum objects in the knowledge base, including 81 presentations and 48 pictures.

[9] The sum of the time for the domain expert and knowledge base manager from Figure 3 corresponds to the grand total in Figure 2. Figure 2 does not include the knowledge engineer's time.

	Domain Expert		KB Managers		Knowledge Engineer		All		Total
	Train.	Devel.	Train.	Devel.	Train.	Guidance	Train.	Devel.	
Design	22.7	36.8	0	0	22.7	3.7	46	40.5	86.5
Implem.	14	203	6	234	20	32	40	469	509
Totals	36.7	240	6	234	42.7	35.7	86	510	596
	277		240		79		596		

Figure 3: Time vs. Participant Role

The most interesting figure is the 85 hours of development effort per hour of instruction. This number must be interpreted cautiously, in part because it is based on a single case, and in part because 'an hour of instruction' has questionable meaning as a metric in highly interactive computer based instruction that tailors content according to individual students. However, it seems to be the best metric we have thus far, and as a benchmark, the results are very encouraging, being an order of magnitude better that what had been previously proposed for knowledge based tutor development time. It compares quite favorably with the estimated 100 to 300 hours required to build traditional CAI [Gery, 1987]. Additionally, construction time for tutor development beyond the first hours might be significantly reduced after the initial curriculum is built, since generic rules for representing pedagogy, student knowledge and dialogue can be re-used and existing domain knowledge can be presented from different perspectives. On the other hand, CAI is not "generative" and instruction time still requires multiple hours development time for multiples hours of on-line instruction since each interaction and each new screen must be encoded explicitly by the author.

The Tutor Construction Tools

This section briefly describes components of the construction tools, indicating first the needs of the users and then the functionality of the components. Domain experts require support in locating relevant information quickly and adding new information effectively, in harmony with existing representation conventions. Effective interfaces must make concepts and structures within the framework visually apparent. Experts also need to notice, modify and save incremental changes to the knowledge base or tutoring strategies. We describe two types of tools: *editing tools*, used to inspect and modify static component of the knowledge base; and *monitoring tools*, which allow the user to visually trace the current state of the dynamic components of the system during tutorial runs. Several of the tools serve as both editing tools and monitoring tools.

Editing tools. The *Tutor Strategy Editor* allows multiple tutoring strategies to be created and modified as editable graphic procedural networks [Murray & Woolf, 1991]. Figure 4 shows the strategy "Give Follow up" being modified. Creating, deleting, repositioning and testing nodes and arcs is done by clicking on a node or arc and using a menu of operations. The Tutoring Strategy Editor traces the flow of control through the networks during trial runs of the tutor so the educator can observe strategy choices by the tutor and action choices within a single strategy. This facilitates modification of tutoring strategies and provides a concrete visualization of what can be, at times, a complex control structure. We are just beginning to work with experts to modify tutoring strategies and have no results as yet.

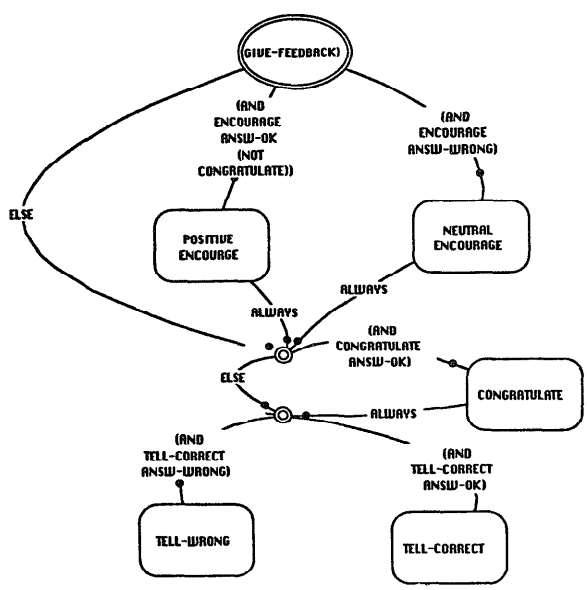

Figure 4: Tutoring Strategy Editor

The *Domain Browser* supports a domain expert in viewing and editing textual domain information (see Figure 5). All three tables of the figure, Object Type, Instance, and Slot, scroll through an alphabetical listing of available elements. The figure shows the teacher examining the Hints slot of a presentation he named LE-Intuition-Easy1. Below the three tables are three pop-up operations menus, one for each table, which allow the user to copy, edit, test, browse, and view components of the knowledge base. The large window at the bottom allows standard operations, such as

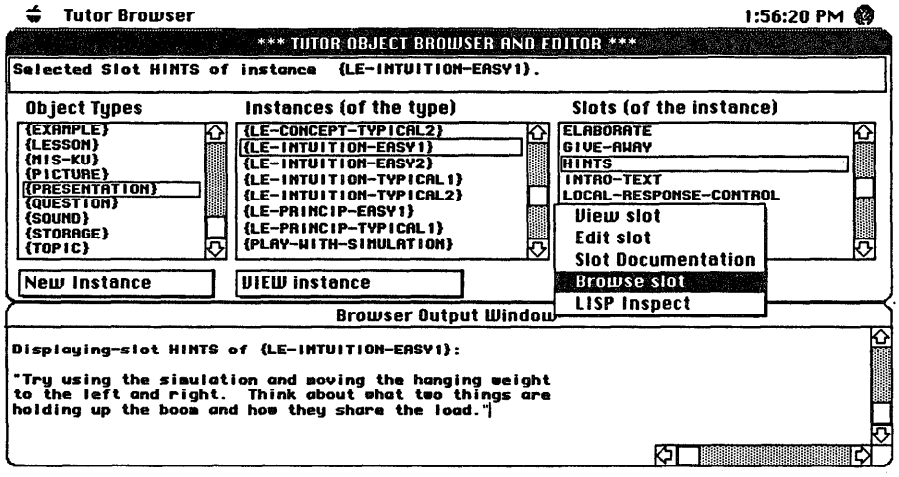

Figure 5: The Domain Knowledge Browser

viewing, copying, creating, editing, and deleting.

The *Topic Network Editor* (see Figure 1) is used to define and graphically view domain topics and their relationship to each other. It is similar to a curriculum network or concept network. If this editor is active during a trial tutor session, domain topics are highlighted to monitor the tutor's traversal of the curriculum.

Monitoring Tools. Session monitoring tools are invaluable in helping educators understand the progress of a tutorial session in terms of curriculum traversal, flow of control through tutoring strategies and changes in the student model. The Topic Net Editor and Strategy Editor function as editing tools *and* monitoring tools.[10] Other session monitoring tools include an Event Log, a Topic Level Display and status/achievement reports on students which can be used by the educator to evaluate presumed areas of strength and weakness or to determine advanced or remedial needs for non-computer parts of the course.

Lessons Learned

This research focused on understanding and supporting knowledge acquisition/editing activities for building educational knowledge bases. It addressed issues around tuning a knowledge acquisition interface to the special needs of educators to tease apart knowledge of what they teach and how they teach it. Several lessons were learned and several principles were confirmed:

1. Educators can become an integral part of development of a knowledge based system and their expert hands-on participation can be made practical through supportive interface tools.

2. An intelligent tutor curriculum can be developed with an effort comparable to that required to develop a CBT system and additionally the tutor has the benefits of a knowledge-based system.

3. Expert educators' mental models of curriculum tend to be procedural in nature, and this tendency should be addressed in their training and in the tools used to build the *declarative* knowledge bases of tutors.

4. An interface using widespread and well designed Macintosh standard pointing and clicking tools is simple enough to reveal the underlying knowledge representation of the tutor and yet expressive enough to enable expert educators with differing backgrounds to tease apart their domain and tutoring knowledge. In this sense the interface representation lies midway on a simplicity-expressiveness continuum [Mullarkey, 1990].

5. Session-tracking and tracing mechanisms reassure computer-naive experts that they can follow the impact of system changes without loosing sight of the flow of control through the curriculum or tutoring strategies. They can experiment without "breaking" the system and this enticed experts to become more involved, resulting in a better tutoring system.

6. [Gould, 1988] observes that "developing user-oriented systems requires living in a sea of changes [and therefore] development is full of surprises." A user-participatory design process is crucial to designing an interface that includes what users need and *does not* include what they find extraneous. Also, seeing domain experts as collaborators and co-experimentors rather than subjects or clients adds depth and richness to the design process and product.

[10] The current topic is highlighted in the Topic Net Editor and the current action is highlighted in the Strategy Editor.

7. Working on a knowledge based system benefits educators. Experts we worked with (both within and outside this case study) have commented that they have a better understanding of the structure of the domain and of students reasoning and misconceptions as a result of designing a tutoring system.

Contributions and Future Research

This research focused on the need for communication and collaboration between educators and software engineers or scientists evolving large and complex knowledge-based systems for tutoring. Programmers can not act in isolation to construct knowledge-based systems by encoding sequences of operations that perform certain goals. Rather, computer-naive domain experts will need to be involved for great lengths of time to develop and maintain specialized knowledge bases. This research provided a case study of one such development process.

This research has contributed to the important issue of facilitating educator collaboration in building tutoring systems by: 1) placing highly usable tools in the hands of practicing educators; 2) drawing educators into the development process as domain experts; 3) documenting a user-participatory iterative design process; and 4) evaluating a qualitative case study of tutor knowledge base development. Time and resource requirements for building a knowledge base were analyzed including construction time per topic, per participant, per activity (e.g., data entry vs. bug detection) and per construction goal (e.g., design vs. implementation). Several principles of a visual knowledge editor were articulated along with a description of a knowledge acquisition interface built based on those principles.

Since this work was a case study, the results are suggestive and descriptive rather than conclusive. Also, since the study involved a fairly curriculum-based model of instruction (and our tools were tweaked to support this type of model), our results may not be applicable to expert system based tutors such as 'model tracing' tutors or coaches. This study represents the "exploratory" beginnings of understanding how knowledge acquisition tools can be used to involve practicing educators in tutor design. Future work includes evaluating the tutoring strategy interface, extending and fleshing out existing tutoring strategies, evaluating the tutor's ability to choose among alternative strategies, and studying their impact on student learning. We also intend to evaluate our approach in a classroom setting, addressing some of the limitations of the construction tool interface by exploring its usability in classroom contexts and with multiple domain experts. In addition, although tutors build with our tools allow substantial student flexibility and control, we have not yet studied characteristics of the interface that encourage such control, and nor have we studied whether students will have difficulty utilizing the available options. Finally, we also need to test our approach with diverse instructional domains and pedagogical styles.

References

Bonar, J., Cunningham, R., & Schultz, J. (1986). An Object-Oriented Architecture for Intelligent Tutoring Systems. *Proceedings of OOPSLA-86.*

Barciss, R., Porter, B. & Murray, K. (1989). Supporting Start-to-Finish Development of Knowledge Bases. *Machine Learning* 4: pp. 259-283.

Blomberg, J. L. & Henderson, A. (1990). Reflections on Participatory Design: Lessons from the Trillium Experience. *Proceedings of The Computer-Human Interface Conference–1990.*

Carroll, J. M., Singer, J., Bellamy, R. & Alpert, S. (1990). A View Matcher for Learning Smalltalk. *Proceedings of the 1990 ACM Conference on Human Factors in Computing Systems*, ACM Press, pp. 431-437.

Clancey, W. J. & Joerger, K. (1988). A Practical Authoring Shell for Apprenticeship Learning. *Intelligent Tutoring Systems-88 Proceedings*, University of Montreal.

Gery, G. (1987). *Making CBT Happen.* Boston, MA: Weingarten Publ.

Gould, J. D. (1988). How to Design Usable Systems. In Helander, M. (Ed.), *Handbook of Human-Computer Interaction.* Amsterdam: Elsevier Science North Holland, pp. 272-298.

Macmillan, S., Emme, D., & Berkowitz, M. (1988). Instructional Planners, Lessons Learned. In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned*, Hillsdale, NJ: Lawrence Erlbaum Assoc.

Merrill, M. D. (1989). An Instructional Design Expert System. *Computer-Based Instruction*, 16(3), pp. 95-101.

Mullarkey, P. (1990). An Experiment in Direct Knowledge Acquisition, *Proceedings Eighth National Conference on Artificial Intelligence*, pp. 498-503.

Murray, T. (1991). *A Knowledge Acquisition Framework Facilitating Multiple Strategies in an Intelligent Tutor.* Ed.D. Dissertation, University of Massachusetts at Amherst, Computer Science Tech. Report 91-95.

Murray T. & Woolf, B.P. (1990). A Knowledge Acquisition Framework for Intelligent Learning Environments. *SIGART (Special Interest Group in Artificial Intelligence) Bulletin*, 2: 2, pp. 1-13.

Murray, T. & Woolf, B.P. (1992). Tools for Teacher Participation for ITS Design. *Proceedings of the ITS-92 conference*, Montreal.

Russell, D., Moran, T. P., & Jordan, D. S. (1988). The Instructional Design Environment. In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems: Lessons Learned.* Hillsdale, NJ: Lawrence Erlbaum.

Suchman, L. (1987). *Plans and Situated Action: The Problem of Human-Machine Communication.* Cambridge: Cambridge Press.

Terveen, L., and Wroblewski, D. (1990). A Collaborative Interface for Editing Large Knowledge Bases. *Proceedings Eighth National Conference on Artificial Intelligence*, pp. 491-496.

Whiteside, J., Bennett, J., & Holtzblatt, K. (1988). Usability Engineering: Our Experience and Evolution. In M. Helander (Ed.), *Handbook of Human-Computer Interaction.* Amsterdam: Elsevier Science North-Holland Press.