

Researchers at Worcester Polytechnic Institute (WPI) and at Carnegie Mellon University (CMU) propose to do the following research collaboratively. Ken Koedinger at CMU is also submitting the main text of this proposal under separate cover. This document is the WPI version. The CMU version has the same proposal text, but a different statement of work and budget.

Worcester Polytechnic Institute

**Proposal Title:
Affordable Cognitive Modeling Authoring Tools using HCI Methods:
Worcester Polytechnic Institute Portion**

This proposal is submitted pursuant to BAA 02-020
Affordable Human Behavior Modeling
August 29, 2002

Amount Requested: \$203,304
Period of Performance: October 1, 2002 to September 30, 2005

Principal Investigator Information:

Neil T. Heffernan III
Computer Science Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609-2280
Phone: (508) 831-5569
Fax: (508) 831-5776
Email: nth@wpi.edu

Approved for the University:

Francois D. Lemire, Director
Office of Research Administration
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609-2280
Tel. (508) 831-5359
Fax. (508) 831-5789
Email: flemire@wpi.edu

Abstract

We propose to develop a suite of Cognitive Modeling Authoring Tools (CMAT) that will make modeling both *easier and faster* for experienced modelers and *possible* for potential modelers who are not experts in cognitive psychology or artificial intelligence programming. Our concrete goal is to reduce modeling time by a factor of three and to experimentally demonstrate this reduction. We can achieve this ambitious goal because we have a proven track record of using Human-Computer Interaction (HCI) methods and Cognitive Science principles to build software systems that meet measurable objectives and are in nation-wide use. For instance, using HCI methods to redesign an intelligent tutoring authoring tool, we demonstrated a decrease in programming time by a factor of 2.6. Preliminary analytic and empirical analyses of CMAT are promising. Keystroke Level Model analyses comparing our early CMAT designs with our existing cognitive modeling tools predict a reduction in time by a factor of 1.5 to 3. A preliminary empirical comparison found a reduction in time of a factor of 2. These early quantitative results are less important than the specific guidance that such analyses have and will provide as we iteratively converge on demonstrably more cost-effective cognitive modeling tools.

Problem

Accurate models of human behavior and knowledge have wide and significant applications in addition to their important role in advancing basic theory of human thinking and learning. These applications include agents in training simulations, automated methods for testing and improving computer and device interfaces, agents in games and other entertainment systems, and intelligent tutoring systems for education and training.

We have created and employed cognitive models in an approach to intelligent tutoring systems called Cognitive Tutors. The use of cognitive models is critical to the dramatic success these systems have achieved. A Cognitive Tutor for programming reduced training time by a factor of three and yielded higher learning outcomes (Anderson, Corbett, Koedinger, & Pelletier, 1995). In evaluations of the Algebra Cognitive Tutor, we demonstrated that students in classes with the model-based tutor outperform students in control classes by 50-100% on targeted complex problem-solving skills and by 10-25% on standardized tests (Koedinger, Anderson,

Hadley, & Mark, 1997). This tutor is now being marketed nationwide and is use in over 800 schools across the country (c.f., Corbett, Koedinger, & Hadley, 2001; see also www.carnegielearning.com).

Despite the great potential of cognitive and behavioral modeling to create powerful systems, development of such systems is currently costly and has been limited to just a few research teams. In the case of developing models for Cognitive Tutors, developers must have PhD-level expertise in cognitive task analysis and advanced symbolic artificial intelligence (AI) programming skills (e.g., LISP and rule-based production system skills). In general, behavioral modeling is hard -- challenging and time-consuming even for the few individuals today who have the rare combination of PhD level competence in both cognitive psychology and AI. Currently, it is largely out of reach for those in the applied and basic research areas who would like to apply modeling in their research or development. Modeling is hard for numerous reasons: 1) cognitive task analysis and knowledge acquisition, 2) advanced AI programming, 3) testing and debugging, and 4) extending, scaling up, and regression testing. Our goal is create an authoring architecture that addresses each of these difficulties. Our target is the modeling of high-level cognition and complex problem solving and domains with multiple alternative correct and incorrect strategies as is needed for example in Computer-Generated Forces (e.g., Chandrasekaran, & Josephson, 1999). Key to our approach is the use of analytic and empirical methods from the field of Human-Computer Interaction (HCI), not just to address issues of effective user-interface design but also to create and choose features that are most useful and most needed by modelers. We will have succeeded if we make modeling both *easier and faster* for current modelers and *possible* for researchers, trainers, and educators who are not experts in cognitive psychology or AI.

Opportunity

The current team is well positioned to tackle this problem. First, all co-PIs have extensive experience in the development of cognitive models. Ken Koedinger was a key architect of the cognitive models for the successful Cognitive Tutors mentioned above. Neil Heffernan developed an educational system based on two interacting cognitive models, one of the trainee and one of the trainer and showed that this combination increases the effectiveness of the system, as compared to working with a single model (Heffernan & Koedinger, 2002). Vincent Alevan

has been extensively involved in the cognitive modeling efforts for three of the six curriculum units of the Geometry Cognitive Tutor, which is being marketed nationally, and has also demonstrated improved effectiveness (Aleven & Koedinger, 2002). We have built cognitive models in a variety of AI languages and cognitive theories including Prolog, LISP, Frames (Koedinger & Anderson, 1990), LOOM (Aleven, Popescu, & Koedinger, 2001; Aleven & Ashley, 1997; Ashley & Aleven, 1994), OPS-5 (Striebel, Stewart, Koedinger, Collins, & Junck, 1987), Soar (Miller, Lehman, & Koedinger, 1999), and ACT-R (Koedinger et al, 1997; Koedinger & MacLaren, 2002). The ACT-R theory of cognition (Anderson & Lebiere, 1998) has been used successfully to model human behavior in hundreds of domains. Because we are targeting high level cognition and complex problem solving, the fine grain size of the current ACT-R 5.0 (<http://act.psy.cmu.edu>) is not optimal for our needs. Our current modeling is done in a language based on ACT-R that has both object-oriented as well as production system¹ features.

Second, the current team has knowledge of and experience with many HCI methods used to produce interactive software with high usability. The use of these methods is crucial to the current proposal, as we discuss further below.

Finally, we have already been engaged in rapid prototyping and testing of a set of cognitive modeling authoring tools in a 6 month pilot project being funded by ONR and ending in September. These prototype authoring tools were “alpha tested” recently (July 15-19) with 16 researchers from around the world at the weeklong CIRCLE Summer School on Intelligent Tutoring Systems. Half of these participants had little prior programming experience. Using our prototype tools, they were able to create initial cognitive models for a domain of their interest within three days. However, as should be expected at this stage in development, we uncovered a number of usability problems with the current tools and, more importantly, identified new features that modelers desire. This kind of intense interaction with users is a key part of the "contextual design" process (Beyer & Holtzblatt, 1998) that is central to our proposal. This process is much more likely than alternative approaches to lead to an efficient tool suite where

¹ Production rule systems(e.g., CLIPS or ACT-R) are used by cognitive modelers and expert systems creators . These systems are composed of if-then rules (or “productions”) that operator on entities (sometimes called “facts” or “working memory elements”). The production system we are using is based upon on ACT-R so we use theACT-R term “working memory element.”

highly desired and frequently used features are in the foreground and rarely needed features are in the background.

Solution

We propose to implement and evaluate Cognitive Modeling Authoring Tools (CMAT) that will support all phases of the process of creating models of complex human behavior: cognitive task analysis, model implementation, model testing, and scaling up. The problem of making behavior modeling affordable is as much or more so about getting the human-computer interaction (HCI) details of the authoring architecture right as it is about innovation in algorithms. Most programming environment design is primarily driven by the intuitions of the designers. Papers on many existing environments designed to support cognitive modeling, such as iGen/COGNET (Zachary, Le Mentec, & Ryder, 1996), OneSAF (Henderson & Rodriguez, 2002), and The ACT-R Environment (<http://act-r.psy.cmu.edu/software/#environment>) do not mention the use of HCI methods. Instead of relying on intuitions, we will employ HCI methods and analytic techniques, including Contextual Inquiry (Beyer & Holtzblatt, 1998), Heuristic Evaluation and Cognitive Walkthrough (Nielsen & Mack, 1994), Think Aloud User Studies (Ericsson & Simon, 1984), and Keystroke Level Modeling (Card, Moran, & Newell, 1983). In software system design, the devil is in the details and intuitive design inevitably leads to systems that "work" in some sense, but that are often frustrating to use, missing desirable features, and are not optimized for users most frequent and repeated needs. The HCI methods we will employ provide a more rationale and direct scientific means for designing complex systems that are effective and efficient to use.

PI Koedinger has been using or teaching these techniques for more than 10 years and has considerable experience in designing and developing software systems that demonstrably meet their objectives (e.g., Alevan & Koedinger, 2002; Koedinger & Anderson, 1990; Koedinger, et al, 1997; Koedinger & Anderson, 1998; Heffernan & Koedinger, 2002; Striebel, et al., 1987). This system design success includes the Cognitive Tutor courses described above. The objectives there were to create a system that would dramatically enhance student learning and that would be desirable and easy for teachers to adopt and use in their classroom. Both objectives were well achieved. A more directly relevant project (Mathan, Koedinger, Corbett, & Hyndman, 2000) involved redesigning RIDES, an authoring environment for simulation-based intelligent tutoring systems (ITS). The goal was to make the existing system more usable and to

dramatically reduce the time required of authors to create simulation-based ITSs. This project made use of the HCI methods described above, including Heuristic Evaluation, Cognitive Walkthrough, Think Aloud User Studies, and Keystroke Level Modeling. Through an iterative design process facilitated by use of these methods we were able to reduce programming time by a factor of 2.6. Mathan, Koedinger, Corbett, & Hyndman (2000) describe a study in which participants performed 3 authoring tasks (1 with a tutorial and 2 on their own). Whereas participants using the original system took 49.1 minutes on average to perform these tasks, with the redesign participants required only 18.7 minutes on average to perform these tasks. All participants using the redesign were faster than all participants using the original system. In a second study, we demonstrated that the techniques used were applicable to much more complex tasks than those used in the first study (Mathan et al., 2000). In other words, we were able to replicate the process of creating dramatic time savings through iterative design using empirical and analytical HCI methods.

In the current project, we propose to apply many of the same design principles that were responsible for the reduced programming time in the RIDES project. The principles are:

- *Bring the interface closer to the mental model of the user*—for any given task to be carried out with the system, the steps in the system’s user interface should be ordered so that they conform to the user’s mental model of the task. For example, steps that are done to prepare for other steps should not be done “up front”; as much as possible these steps should be carried out at the moment that their effect is needed.
- *Bring the user’s mental model closer to the interface*—when it is not feasible to bring the interface closer to the user’s mental model, or when the user may not have a mental model, the system should help the user construct a mental model of how the task is to be performed with the system. For example, the system can lead the user through the steps by means of an interface that communicates the steps, a “Wizard” in Microsoft terminology.
- *Type it once*—as much as possible, the user should have to type the same name only once. Afterwards, the name should be available in menus, by keystrokes, or by automatic completion of important words during text editing. Such methods are likely to be faster and less error-prone than recall from memory.

In addition, the proposed authoring tools incorporate the following three design principles, which did not figure very prominently in the RIDES evaluation:

- *User-guided generalization*—The proposed tools embody an approach to constructing production rule models that we call user-guided generalization. This approach is related to "programming by demonstration" techniques (Blessing, 1997; Cypher et al, 1993; Lieberman, 2001), however, it implies a different division of work between the user and the tools that takes better advantage of the strengths of each. In both approaches, the process of building a model starts with the user demonstrating and recording the details of a few concrete problem scenarios. The process of generalizing from these instances, however, in order to create a general model that works for a whole class of problems, is different. In programming by demonstration, the system does most or all of the generalizing. With few examples from which to generalize and with little background knowledge, this process is error-prone. In the user-guided generalization approach, by contrast, the user is in charge of the generalization step. The system provides much assistance. First, it generates elements of a "concrete" production rule model that are likely to work on the demonstrated problem scenarios but are not guaranteed to be general. Second, the system provides tools designed specifically to support the process of generalizing a concrete model. This approach requires more work by the human author in the generalization step but also offers far greater opportunity for the user to guide the process based on knowledge about the application domain or about production systems. This division of labor is likely to result in fewer errors and overall in a better trade-off than that offered by programming by demonstration systems.
- *Hide syntax details*—as much as possible, the tools should make it unnecessary for the user to remember syntax details or even shield the user from such details altogether. For example, structured editor techniques and templates are likely to help both beginning and expert users.
- *Provide flexible levels of structure*— for common subtasks, the system should provide structure in ways described under other principles. The system should also enable users (primarily expert users) to structure tasks for themselves, for example, by giving them access to lower-level tools or the individual steps of Wizards. Also, the system should make it easy to switch between "more structured" and "less structured" modes of working.

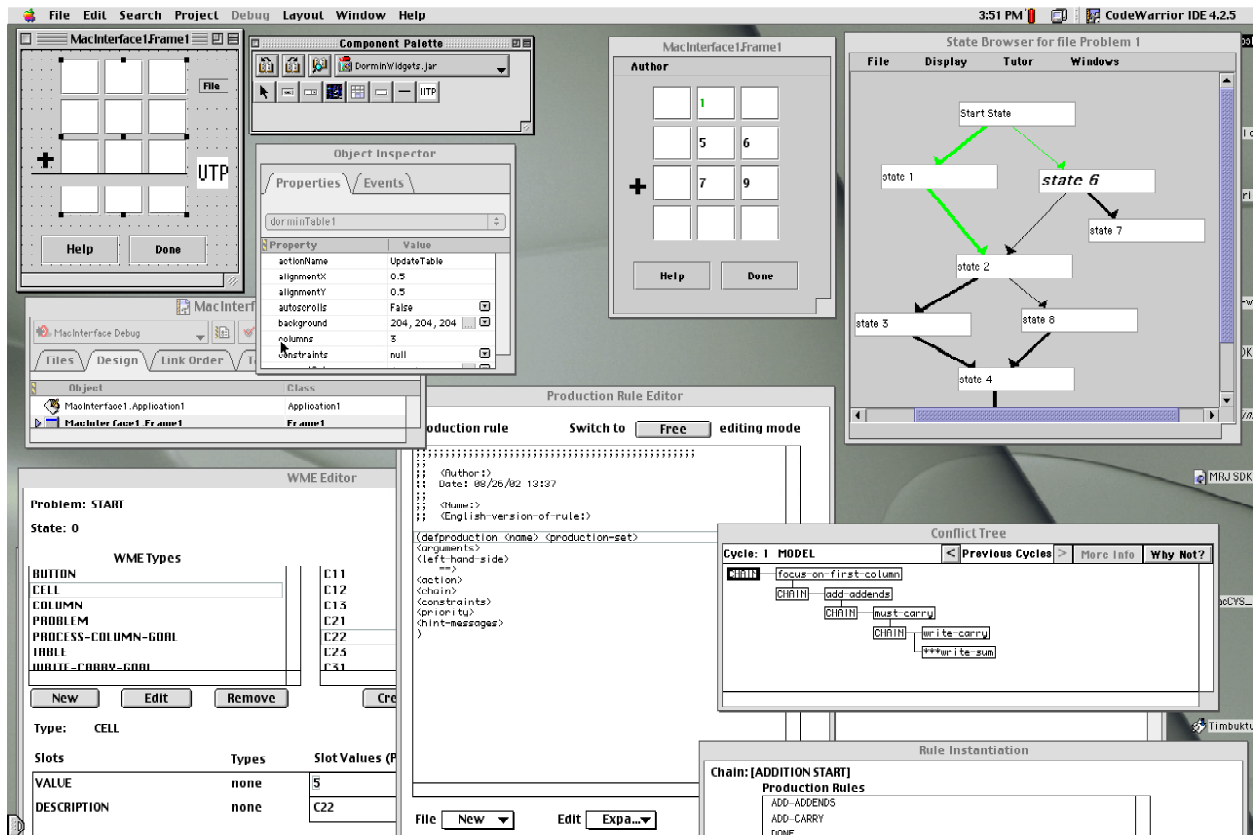


Figure 1: The prototype Cognitive Modeling Authoring Tools

The six design principles described above have been used (and will continue to be used throughout the project) to design an initial suite of tools which will be described next. These principles are applicable to the construction of other development environments. Part of our project output will be to further refine and test these principles, which work well, which do not, and under what conditions.

The Cognitive Modeling Authoring Tools Suite

The tool suite that we are building, illustrated in Figure 1, currently consists of the following tools:

- *An Intelligent GUI Builder*, whose windows are shown in the top-left quadrant of Figure 1, can be used to create a graphical user interface (GUI) in which the modeler can then demonstrate how to carry out the task to be modeled. A simple example of a running interface created with this tool is shown in the top middle of Figure 1 (i.e., “MacInterface1.Frame1”). Use of this tool automatically creates parts of the behavior model.

- A *Behavior Recorder* (i.e., “State Browser for file Problem 1,” top right) records alternative paths through a given problem scenario, as the modeler demonstrates these paths.
- A *WME Editor* and a *Production Rule Editor*, dedicated editors used to implement the production rules that model the demonstrated paths (see Figure 1, bottom part of the screen, left and middle).
- A debugging tool called the *Cognitive Model Visualizer* has two windows shown in the bottom right (i.e., “Conflict Tree” and “Rule Instantiation”).

1. Interface Development and Cognitive Task Analysis

The first step in the process of building a computational model of human behavior is to study humans performing tasks in the domain of interest. Our long-standing experience in cognitive task analysis and cognitive model development indicates that it is crucial in this step to attend to the interface or environment in which the task is performed (Anderson, Corbett, Koedinger, & Pelletier, 1995). Here we construe the term “interface” broadly to include not only computer interfaces, but also paper, the physical environment, and interactions with other people, whether it is the real target interface (e.g., airplane cockpit and communications) or a training interface (e.g., a simulator). Our proposed authoring architecture will support the rapid prototyping of new training interfaces as well as connect with existing simulations that meet communication protocol standards (Koedinger, Suthers, & Forbus, 1999; Ritter & Koedinger, 1996). A key feature of our authoring architecture is an Intelligent GUI (Graphical User Interface) Builder that will allow modelers to create, in a day or less, a GUI in which the task can be performed (cf., Munro, 1994). Our GUI Builder extends a fully featured commercial GUI builder (i.e., Code-Warrior) by adding the functionality needed to communicate with the other tools. Using the GUI Builder, an author can create a GUI by dragging intelligent GUI widgets from the “Component Palette” (top, second window from the left) and dropping them onto the GUI being constructed (top-left window entitled “MacInterface1.Frame1”).

Tests of an early version of this GUI Builder during the 2002 CIRCLE Summer School on Intelligent Tutoring Systems suggest promise. For instance, a NavAir trainer who is a non-programmer, built a prototype GUI for aspects of a gunnery task using the drag and drop widgets provided by the GUI Builder. The interactions with other Summer School participants working in military domains helped us to become aware that some features that are important to their

needs were missing from our design: while some modelers want to create their own graphical user interfaces (i.e., worlds in which models and people behave) from scratch, some participants wanted to be able to plug the modeling tools into an existing simulation (e.g., simulated war games). Thus, we extended our prior plans to not only include interface building tools, but also a plug-in capability to add existing simulations (cf., Ritter & Koedinger, 1997; Koedinger, Suthers, & Forbus, 1999). We will specify an application program interface (API) for inspectability and recordability that components (i.e., user interfaces, agent models, or simulations) need to meet in order to interact with our architecture. This design change inspired by Summer School attendees illustrates our commitment to perform informal and formal user studies of modelers and modelers-to-be to understand what features are most needed.

Once an author has created or plugged in an interface, the interface can be used for cognitive task analysis and knowledge acquisition, particularly to collect think-aloud protocols (Ericsson & Simon, 1984). Think-aloud protocols from domain experts help in identifying alternative solution strategies and the required knowledge behind them.

Our tools will reduce the cost of collecting protocol data. As subjects demonstrate a solution of a given problem scenario, a tool called the *Behavior Recorder* automatically records their actions (as well as their verbalizations) in a diagram. In Figure 1, the Behavior Recorder is shown in the window entitled “State Browser for file Problem 1” (top right). The nodes (i.e., square boxes) in the graph represent the states of the interface or simulation. The arcs (i.e., arrows) represent state changes, due either to actions taken by the user, or to external events caused by a simulator, or to actions of other real or simulated agents. For instance, the Behavior Recorder can display the actions of four different tank operators as well as actions performed by enemy units. In the behavior diagram recorded in Figure 1, there are multiple different paths through the problem scenario, which the cognitive model being constructed will need to cover.

In addition to recording the actions, the Behavior Recorder will store an audio or video recording of the user while solving the problem. It will automatically attach the relevant sections of the recording to the corresponding link(s) in the Behavior Recorder diagram, thus providing easy access to the verbal protocol data that are relevant at any specific point during the execution of the problem scenario. This feature will often make it unnecessary to get a complete transcription of the video or audio recording, thus reducing a major cost factor in protocol analysis. Also, the Behavior Recorder allows the author to roll back a problem scenario or

simulation to an earlier state and consider what alternative actions might have been taken at that point. This allows the author and/or the trainee to do a “what if I had done X at this point ...” analysis. Alternative courses of action or sequences of events are rendered as alternative paths through the diagram. For instance, our NavAir trainer can use his interface (or an existing simulation) to collect think aloud protocols from gunnery experts. The think aloud data provides a guide for the knowledge acquisition needed to create a cognitive model of gunnery experts. Furthermore, as described in the next section, the experts' actions are used to automatically create an initial code base for the cognitive model. Once created, the model can be used as a simulated agent within the simulation and/or as the basis for a tutor for gunnery training.

2. Behavior Model Development, Implementation and Debugging

In addition to reducing the cost of protocol analysis, the Behavior Recorder and other proposed tools will help reduce the cost of model development. The Behavior Recorder will reduce the cost of model creation since it will do a sizable part of the programming job that is currently done by hand. Given a problem scenario and a demonstrated solution trace to this problem scenario in the form of an Behavior Recorder diagram, the tools will be able to generate elements of a “concrete” production rule model including working memory specification and concrete versions of production rules. The author can use the WME Editor (see Figure 1, middle, bottom) to inspect the state of working memory. Most of the working memory elements shown in Figure 1 were constructed automatically by the tools based on the GUI that was constructed.

The tools will provide needed assistance when writing production rules. When the author is ready to write a rule or rules, he or she can start by indicating in the Behavior Recorder diagram which action (or state transition) the rule should implement. This piece of information enables the authoring system to assist the author in two ways. First, because the system has a clear (although possibly incomplete) notion of what the working memory elements should look like in the states before and after the rule fires, the system can make a reasonably accurate suggestion of what an *instantiated* production rule would look like that would cause the given state transition. According to the principle of user-guided generalization, it is then up to the human author to perform the non-obvious generalizations from these concrete rule instances. He or she can do so by editing the instantiated rule to create a generalized version. A dedicated tool called the *Production Rule Editor* will help with this generalization process and is shown in its initial state

in Figure 1. It combines features of a structured editor with wizards designed to support the various subtasks of the rule generalization process.

The Production Rule Editor will make implementing code easier, faster, and less prone to error. However, Fred Brooks, in his now classic *The Mythical Man-Month* (1975), found that programmers spent only 1/6 of their time actually coding, but spent over half of their time testing and debugging, with the remaining one third of their time planning². The proposed tools will address these time-consuming tasks as well. Because the Behavior Recorder diagram is essentially a specification³ of how the production rule model should behave, it significantly aids in the debugging process. It can be used to make sure that a given production rule does indeed implement the desired action (i.e., causes the intended state transition in the diagram). Normally, programming environments cannot provide assistance of this type, because they do not know what the program under construction is supposed to do. Empirical studies by Vessey (1985) have shown the most difficult part of debugging is *error localization* (i.e., finding where an error has occurred). The Behavior Recorder can assist modelers in error localization by running the model up to the last point(s) in the solution space record where the model is performing correctly. As mentioned, the Behavior Recorder is able to offer this support because the modeler has already demonstrated his or her intent earlier in the process.

A tool called the *Cognitive Model Visualizer* will further provide support for error localization. This tool provides a dynamic view of a cognitive model as it is run on a given problem. That is, the tool shows a tree diagram of alternative production rule paths that were explored by the production rule interpreter as it considered what to do next. For example, the “Conflict Tree” window in Figure 1 (bottom right) shows two different paths of productions that could have just fired. Further, the Cognitive Model Visualizer allows the author to do “what if” and “why not” analysis to understand what the model can do and why it is not doing what is expected. The “Rule Instantiation” window (Figure 1, bottom right) is used in “why not?”

² See also Collofello & Woodfield (1989) for more recent empirical estimates of how programmers spend their time.

³ The Behavior Recorder’s diagram is a partial specification of the how the system is supposed to behave. It is only *partial* because the author does not have to demonstrate every possible sequence of actions. A complete specification of a system is usually very expensive to produce. We believe that we can get a great deal of leverage out of these partial specifications, which do not impose any additional cost because the specifications will be needed at testing time.

analyses related to truncated or abandoned production rule paths. This tool will greatly facilitate debugging and will help the modeler in creating production rules that fire exactly when needed, a notoriously difficult task when creating production rule models.

3. Testing, Validation & Verification: Behavior Recorder as Partial Specification

For all the advantages that production rule models have (e.g., Anderson & Lebiere, 1998; Newell, 1990), it can be difficult to understand the behavior of a production rule model just by looking at the code (Barr & Feigenbaum, 1989). As a result, validation and verification of models tends to be time consuming and costly. Sometimes changes made in a production rule model to make it work on a new problem cause it to fail on problems on which it worked previously. Thus, in order to prevent this regression, the modeler must frequently run the model on a set of representative test scenarios. The Behavior Recorder addresses this issue and makes this form of regression testing far less time-consuming than if it were done by hand. Since Behavior Recorder diagrams are specifications of how the model should behave on any given problem, they can be used for automated testing. Behavior diagrams created with the Behavior Recorder specify all actions to be modeled. They may also include paths (marked as such by the modeler) that represent wrong or sub-optimal behavior that should either not be covered by the model, or should be “acknowledged” by the model as representing sub-optimal behavior. The latter (modeling "buggy" behavior) is particularly useful in training situations to comment on errors or to simulate a more realistic companion agent. The former (tagging incorrect model behavior) is generally useful in model testing since a correct model must both produce correct behavior and not produce incorrect behavior.

CMAT will provide feedback like “Due to the latest production rule changes, five more steps are now modeled correctly. However, two steps are no longer modeled correctly.” Messages such as this should be enormously appreciated by any production rule writer. Furthermore, the Behavior Recorder will be able to show the programmer in a graphical manner where the problems are. The Behavior Recorder indicates which actions are modeled correctly by the system and which ones are not, by running the model and marking the arrows in the diagram in red or green. In this way, the author will be able to see how changes in the production rules effect the overall performance of the model. When we demonstrated a prototype of this feature

at our summer school, the programmers that have had experience developing production rule models were very excited by the prospect of testing their models automatically.

In proposing affordability enhancements to a programming environment, one must avoid playing a shell game, reducing time in one area of program design, development or testing, but unwittingly increasing time in another. Although use of the Behavior Recorder may appear to be adding steps that are not necessary in other approaches to behavior modeling, this is not the case. To test a behavior model, the programmer must test whether it behaves appropriately in situations it is intended to model. Putting the model through its paces in testing involves the same kinds of steps required in the creation of the Behavior Recorder's diagram. In essence what we have done is to move aspects of testing before coding rather than after. This move has three positive consequences. First, it encourages the modeler to more carefully plan and document model objectives prior to coding and thus should reduce errors and blind allies. Second, it provides the system with data on model objectives that can be used to automate creation of aspects of the model. It does so without adding any new steps (just moving them) and other steps are eliminated. Third, the test cases are being automatically recorded so that they can be used in automated testing.

Affordability

In this section, we summarize the reasons why CMAT will make behavioral modeling more affordable and present preliminary analytic and empirical evidence of enhanced affordability.

Summary of affordability arguments

We summarize the main affordability arguments presented in previous sections of the proposal. These arguments address all phases of cognitive modeling.

Cognitive task analysis: The GUI Builder and Behavior Recorder will facilitate the recording and analysis of think-aloud protocols. They will also make it easier to create detailed specifications of how a model-to-be-constructed should behave on given problems, in the form of behavior diagrams.

Implementation: The Behavior Recorder will generate some model elements automatically. The Production Rule Editor and WME Editor will streamline the editing of model elements, by

applying the 3 following principles: “user-guided generalization”, “type it once”, and “flexible levels of structure”.

Testing and debugging: The Behavior Recorder provides guidance in testing, by evaluating and displaying to what extent a model exhibits the behavior specified in a given behavior diagram. The Behavior Recorder helps in localizing errors, by letting the modeler put the model in any state in the behavior diagram. The Cognitive Model Visualizer makes it easy for the modeler to follow what happens during model execution and makes error localization easier by supporting Why Not? analyses.

Scaling up, regression testing, and maintenance: The Behavior Recorder supports automated re-testing of a model, by running it on a test suite of problems after changes in a model.

Intangibles

- Using CMAT, a modeler creates diagrams depicting the desired behavior of the model early on, which may lead to better planning and design of the main elements of the model.
- Using CMAT, a modeler is likely to make fewer coding errors, which may substantially reduce the time needed for debugging and testing.
- Using CMAT, the modeler is freed from having to attend to many low-level details and may therefore be better able to attend to the overall design of the model.
- CMAT better integrates cognitive task analysis, knowledge acquisition, and model building to yield efficiencies. Overlapping aspects of these activities can be done once rather than repeated by both a psychologist and a model programmer. While the psychologist is using the Behavior Recorder in task analysis, he or she is already creating elements of the model. The programmer can more efficiently pick up from where the psychologist left off because they are using the same integrated tools.
- More generally, aspects of the modeling task, like building interfaces and demonstrating correct solutions, can be taken on by less expensive staff than behavior modelers.

Preliminary Analytic and Empirical Evidence for Affordability

To get a sense of the progress we have already achieved with our prior ONR funding and to test the feasibility of our proposed design methods, we performed preliminary analytic and empirical analyses of the affordability of our current and planned designs. We compared our existing modeling tools, an environment called TDK (Anderson & Pelletier, 1991), to 1) the preliminary

CMAT tools as they stand after 4 months of ONR funding, and 2) a preliminary redesign of the tools that has not yet been implemented. It should be noted that the control condition is no strawman, because TDK has been used for over a decade to create many large-scale cognitive models that have been successfully applied in the real world.

Preliminary Analytic Evidence for Affordability: A Key Stroke Level Analysis

In this section, we present preliminary analytic evidence that CMAT will lead to significant savings in the time it takes to develop cognitive models. We illustrate the use of an analysis method called the Keystroke Level Model (KLM) methodology, which was introduced by HCI pioneers Card, Moran, and Newell (1983). Using this method, we perform a fine-grained analysis focused on the ease with which a specific model-building task can be carried out, both with the “old” TDK, and with CMAT. The analysis focuses on the performance of expert modelers, who have overcome conceptual difficulties and have become proficient with the tools. We plan to use the KLM methodology frequently over the course of the project, to evaluate alternative designs or planned changes to the tools before actually implementing them, as is evident in the plan of work shown below.

The KLM methodology is a way of estimating the time required for error-free expert performance on routine tasks in a given computer interface. In order to provide such an estimate, one creates a detailed specification of the task at the level of keystrokes, pointing and clicking with the mouse, homing the hands on the mouse or keyboard, and the mental preparation for such actions. The total time required to complete the task is then estimated as the sum of the number of operations in each of those four categories, multiplied by an estimated average time for the given type of operations. For example, average typists type one key per .2 seconds. This number includes the correction of errors caught immediately. The average mouse pointer move takes 1.1 seconds. Time estimates derived from KLM models have been shown to correlate well with actual expert performance times (Card, Moran, & Newell, 1983).

We focus on three commonly-occurring modeling tasks related to three different stages of model building, namely (1) creating the initial configuration in working memory for a problem scenario, (2) writing a production rule of medium complexity, and (3) diagnosing why a rule that

was expected to fire did not (i.e., a small debugging task⁴). We created somewhat simplified KLM encodings for the steps involved in this subtask, carried out in three different environments: (1) the existing modeling environment, TDK, (2) CMAT Tools in their current state, and (3) the CMAT Tools as we foresee that they will be, when we implement our current ideas for how to improve CMAT. Given the limited time available, we did not create a model of using the future CMAT tools for the first of the three tasks. Readers interested in seeing more detail about the models can download an Excel file with the models at URL <http://www.cs.cmu.edu/~aleven/CMAT-KLM.xls>

The time estimates derived from the models are shown in Table 1. The reported time estimates are somewhat lower than the actual times one would expect an expert to take, since, for the sake of simplicity, the models do not account for the “mental operations” involved in preparing to take the next action in the interface. We will however include such mental operators in future KLM models created in the course of the project.

Table 1: Results of a preliminary evaluation of CMAT using Keystroke Level Models: estimates of the time (in seconds) spent at the keystroke level for three commonly-occurring modeling tasks

	TDK	Current CMAT	Future CMAT
Initialization	209.3	90.6	
Writing rule	203.1	207.3	130.1
Debugging	39.3	30.8	12.5

The KLM analysis predicts that the current CMAT Tools will reduce the time needed to create an initial working memory configuration by a factor of 2.3. This reduction is impressive especially if one considers that the time reported for CMAT includes not only the creation of initial structure in working memory but also a GUI and a demonstrated solution. These items are

⁴ Debugging tends to get harder as the number of rules in the system grows. Using a rule base with only five productions, if anything, caused us to underestimate the role of debugging and the time savings afforded by CMAT.

likely to lead to further savings in other steps. The current CMAT Tools led to more modest predicted savings in the debugging task and no savings in the task of writing a production rule. However, the results indicate also that we can expect significant savings in the latter two tasks when we implement the planned design changes. The anticipated time reduction in the debugging task is especially interesting, given that as mentioned above, programmers spend much of their time testing and debugging. In the next section we show empirical results that test the predictions based on the KLM models.

The savings afforded by the new authoring tools may well be greater than those indicated by the current analysis. First, the analysis of the future CMAT Tools was based on a design that was created rather quickly, in order to be able to report the analysis in the current proposal. We expect further savings when we compare time estimates based on more carefully crafted designs informed by user studies. Second, the analysis does not reflect savings due to the fact that according to the “type it once” principle mentioned above, an author using CMAT can often rely on recognition of relevant information (e.g., in menus) rather than having to recall it from memory, which is error-prone. Third, the analysis does not reflect savings in debugging time that will occur because the tools are likely to help modelers write models with fewer errors.

More important than these very preliminary results are the following points related to our planned use of the KLM methodology: First, the KLM analysis helps in identifying parts of the design that could be improved. For example, looking at the KLM model one can readily see where there is a high density of mouse pointer moves, which are the most costly operations in terms of time. Redesign efforts can then focus on those parts. Second, KLM is a useful tool for evaluating designs before actually implementing them and provides a more scientific basis for choosing one design over another. Finally, the KLM analysis illustrates that, as we argued above, the “devil is in the details.” The significant savings afforded by the future CMAT Tools, as compared to the current version (see Table 1), are due to an accumulation of savings from many relatively small changes in the design, rather than to a single crucial feature.

We plan to judiciously use this approach during the development of CMAT, focusing on a broader range of modeling tasks than is possible within the scope of proposal writing.

Preliminary Empirical Evidence for Affordability: A User Case Study

In addition to the KLM analysis presented above, we conducted a preliminary *empirical* analysis comparing the amount of time it took to complete a modeling task with the existing TDK and the current preliminary version of CMAT. We focused on modeling a portion of a gunnery domain, taken from the specification created by one of our Summer School participants. Specifically, the task was to implement, test, and debug two elements of a production rule model for this domain, namely, the cognitive structure representing the initial state of the problem scenario and a single production rule. One of the proposal's authors completed this task in 50 minutes using the TDK. He then did the same task using CMAT, this time taking only 15 minutes to complete it. Finally, to deal with the confound of order, the same author re-did the task using the TDK tools. This time he needed 30 minutes, which was still twice as long as it took using CMAT.

The data in Table 2 are consistent with Brooks' claim that testing and debugging takes about half a programmer's time. Table 2 show that the majority of time savings occurred at the testing and debugging stage. Consistent with the results of the KLM analysis presented in the previous section, the time savings were not due to faster creation of production rules, although as mentioned, we do anticipate savings in this area in the future. We attribute the cost reductions at the debugging stage to three causes. First, the Behavior Recorder made it easier to see when the model was working and when it was not. Second, while the time needed to implement a rule using CMAT was the same as the time needed using the old TDK tools, the errors in the rule written were easier to find. Finally, the Cognitive Model Visualizer is a superior debugging tool than the older debugging function.

Table 2: Results of a preliminary empirical evaluation of CMAT. The number of minutes spent on the various sub-tasks for each trial.

	Trial 1 TDK	Trial 2 CMAT	Trail 3 Redo TDK
A: Initialization	10	5	8
B: Writing Rule	12	5	5
Implementation (A+B)	22	10	13
Testing & Debugging	28	5	17
Total	50	15	30

We anticipate that we will be able to show further savings as we continue to develop the tools and scale up to larger evaluation studies where we anticipate the automatic testing features will be more useful.

In summary, we presented some preliminary evidence that suggest that the CMAT Tools, even in their early stage of development, may substantially reduce the time needed to build and test models of human behavior. More detailed disaggregation of our analytic and empirical results will provide specific guidance for where redesign efforts are most likely to yield further cost-savings.

Planned Evaluation Activities

One way CMAT will be successful is if we demonstrate how it can substantially reduce the development time for cognitive models while maintaining or increasing system quality. Current estimates are that it takes 100-1000 hours of development for one hour of a useful cognitive model (Murray, 1999; Anderson, 1993, p. 254.) Our goal is to reduce this time by at least a factor of three. A further criterion for success is that the tools make the cognitive model development feasible for a much larger range of developers and also a much broader range of training problems.

We will carry out a number of evaluation activities to assess whether we have attained these goals. First, as much as we can we will place the tools in the hands of novice model builders. This will give us a chance to observe whether the tools are easy to learn, to do the usability studies needed to get the HCI right, and to build a community of users. A particularly good opportunity is offered by the yearly summer schools on intelligent tutoring systems that we offer through the NSF-sponsored CIRCLE Center. Participants of these summer schools will use the tools to create a substantial model within a week, with guidance from us. After the summer schools, we will make the tools available to those participants interested in continuing the development of more complete systems, thus planting the seeds for a user community. We plan to be actively involved in this community as their feedback will be invaluable. Ultimately, the success of our planned efforts to build a community of users will be a strong test of whether our authoring architecture advances the state of the art of creating models of complex human problem-solving and learning behavior.

Second, we plan to conduct experiments to document the reduction in development time afforded by the tools, for both expert and novice modelers. We will compare what was achieved in past summer schools and/or courses on tutor development with what will be achieved after these tools are developed. More formally, we plan to do controlled experiments to compare development time with the tools to development time without them or with alternative subsets of them. These experiments will be performed in the context of university courses and the summer schools. Whereas these summer schools involve novices, we also plan to study experts in model development, comparing various measures of model development efficiency. These experts will be recruited from modelers at CMU and our spin-off company Carnegie Learning, from people who have worked with us and from the emerging community of users.

Third, we will continue to use analytic techniques to guide our design like the keystroke level model method (Card, Moran, & Newell, 1983) illustrated above. This fine-grained analysis will focus on the ease with which specific model building and testing tasks can be carried out once conceptual difficulties have been overcome. Finally, we plan to show that the tools help even when building a cognitive model of substantial scale.

Conclusion

Behavioral modeling can be made significantly more affordable, both easier to learn for novices and faster for experienced modelers, by developing good authoring tools. The tools must support all phases of modeling in an integrated manner. Such tools can be developed by careful investigation of what modelers need, careful design, guided by relevant design principles, and by cycles of evaluation of prototypes, initially through informal user studies and eventually through more formal evaluation studies. All phases require application of HCI methods. The current team has the modeling experience as well the HCI expertise to take on this task, as evidenced by the fact that a prototype tool suite developed in this manner in a mere 4 months already shows substantial savings in common modeling tasks. We anticipate savings in development time of a factor three as we continue to develop the tools and scale up our evaluation studies.

References

Aleven, V.A.W.M.M., & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science*, 26(2), 147-179.

- Aleven, V., & Ashley, K. D. (1997). Teaching Case-Based Argumentation Through a Model and Examples: Empirical Evaluation of an Intelligent Learning Environment. In B. du Boulay & R. Mizoguchi (Eds.), *Artificial Intelligence in Education, Proceedings of AI-ED 97 World Conference* (pp. 87-94). Amsterdam: IOS Press.
- Aleven, V., Popescu, O., & Koedinger, K. R. (2001). A Tutorial Dialogue System with Knowledge-Based Understanding and Classification of Student Explanations. In the Working Notes of the 2nd IJCAI Workshop on Knowledge And Reasoning In Practical Dialogue Systems, August 5, 2001, Seattle.
- Ashley, K. D., & Aleven, V. (1994). A Logical Representation for Relevance Criteria. In S. Wess, K. D. Althoff, & M. M. Richter (Eds.), *Topics in Case-Based Reasoning; Selected Papers from the First European Workshop, EWCBR-93* (pp. 338-352). Lecture Notes in Artificial Intelligence, 837. Berlin: Springer-Verlag.
- Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Erlbaum.
- Anderson, J. R., & Lebiere, C. (1998). *The Atomic Components of Thought*. Hillsdale, NJ: Erlbaum.
- Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4 (2), 167-207.
- Anderson, J.R., & Pelletier, R. (1991). A development system for model-tracing tutors. In *Proceedings of the International Conference of the Learning Sciences*, 1-8.
- Ball, T. & Eick, S. G. (1996) Software evaluation in the large. *Computer*, 29(4):33-43.
- Barr, A., & Feigenbaum, E.A. (Eds.). (1989). *The Handbook of Artificial Intelligence, Vol 1*. Reprint. Reading, MA: Addison-Wesley.
- Beyer, H. & Holtzblatt, K. (1998). *Contextual Design: Defining Customer-Centered Systems*. San Francisco, CA: Morgan Kaufman.
- Blessing, S. (1997) A Programming by Demonstration Authoring Tool for Model-Tracing Tutors. *International Journal of Artificial Intelligence in Education*, 8, 233-261.
- Brooks, F. P.(1975) *The Mythical Man-Month*. Addison-Wesley, Reading MA.
- Card, S.K., Moran, T.P., & Newell, A (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Chandrasekaran, B. & Josephson, J. R. Cognitive Modeling For Simulation Goals: A Research Strategy for Computer-Generated Forces. In *Proceedings of the 8th Conference on Computer Generated Forces and Behavioral Representations*. 11-13 May 1999, Orlando, FL. Defense Modeling and Simulation Office & DARPA, pp. 117-126.
- Collofello, J.S., & Woodfield, S. N. (1989) Evalting the effectiveness of realiability assuance techniques. *Journal of Systems and Software*, 9(3):191-195.
- Corbett, A. T., Koedinger, K. R., & Hadley, W. H. (2001). Cognitive Tutors: From the research classroom to all classrooms. In Goodman, P. S. (Ed.) *Technology Enhanced Learning: Opportunities for Change*. Mahwah, NJ: Lawrence Erlbaum Associates.
- Cypher, A., Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B. A. and Turransky, A. (1993) eds. *Watch What I Do: Programming by Demonstration*. Cambridge, MA: The MIT Press.

- Ericsson, K. A., & Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data*. Cambridge, MA: The MIT Press.
- Heffernan, N. T., & Koedinger, K. R. (2002) An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor. In the *Proceedings of the Sixth International Conference on Intelligent Tutoring System 2002*. Biarritz, France.
- Henderson, C. & Rodriguez, A. (2002). Modeling in OneSAF. In *Proceedings of the 11th Conference on Computer Generated Forces and Behavioral Representation*. May 7-9, 2002. Orlando, FL.
- Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8, 30-43.
- Koedinger, K. R., & Anderson, J. R. (1998). Illustrating principled design: The early evolution of a cognitive tutor for algebra symbolization. *Interactive Learning Environments*, 5, 161-180.
- Koedinger, K.R., & MacLaren, B. A. (2002). Developing a pedagogical domain theory of early algebra problem solving. CMU-HCII Tech Report 02-100. Accessible via <http://reports-archive.adm.cs.cmu.edu/hcii.html>.
- Koedinger, K. R., Suthers, D. D., & Forbus, K. D. (1999). Component-based construction of a science learning space. *International Journal of Artificial Intelligence in Education*, 10.
- Lieberman, H. (Ed.) (2001). *Your wish is my command: programming by example*. San Francisco: Morgan Kaufmann Publishers.
- Mathan, Koedinger, Corbett, & Hyndman (2000) Effective Strategies for Bridging Gulfs Between Users and Computer Systems. Published in *Proceedings of HCI-Aero 2000: International Conference on Human Computer Interaction in Aeronautics*. Toulouse, France September 27th- 29th 2000. pp 197 - 202
- Miller, C. S., Lehman, J. F., & Koedinger, K. R. (1999). Goals and learning in microworlds. *Cognitive Science*, 23, (3), 305-336.
- Munro, A. (1994). RIDES Authoring Reference. Behavioral Technology Laboratories, USC (<http://btl.usc.edu/rides/>)
- Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10, pp. 98-129.
- Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.
- Nielsen J. & Mack, R.L. (1994). *Usability Inspection Methods*. New York: John Wiley & Sons, 1994.
- Ritter, S. & Koedinger, K. R. (1996). An architecture for plug-in tutoring agents. In *Journal of Artificial Intelligence in Education*, 7 (3/4), 315-347. Charlottesville, VA: Association for the Advancement of Computing in Education.
- Streibel, M.J., Stewart, J., Koedinger, K.R., Collins, A., & Junck, J.R. (1987). MENDEL: An intelligent tutoring system for genetics problem solving, conjecturing, and understanding. *Machine-Mediated Learning*, 2, 129-160.
- Vessey, I. (1985) Expertise in debugging computer programs. *International Journal of Man-Machine Studies: A process analysis*, 23(5):459-494.

Zachary, W., Le Mentec, J.C., & Ryder, J. (1996). Interface Agents in Complex Systems. In C.A. Ntuen & E.H. Park (Eds.), *Human Interaction with Complex Systems: Conceptual Principles and Design Practice* (pp.35-52). Boston: Kluwer.

Overall Statement of Work for both CMU and WPI

We propose to design, build and test the authoring tools (Behavior Recorder, Production Rule Editor, Intelligent GUI Editor, and Cognitive Model Visualizer) outlined in the proposal. Furthermore, we plan to evaluate the cost saving provided by these tools. A hallmark of good programming system development is iterative design, user testing, and redesign, which we will employ throughout this project. While system development and evaluation will continue throughout the project, each year will have a different focus. The first year of the project will focus on developing base functionality in all of the tools. We will pay particular attention to the functionality of the Behavior Recorder as well as the GUI Builder. The second year will focus on Cognitive Model Visualizer and adding the functionality to the Behavior Recorder that supports the early stages of model development like collecting think aloud protocols. The third year will focus on the later stages of programming including program maintenance. In this year we will focus on the functionality of the Behavior Recorder to do large scale regression testing. Evaluation activities will emphasize formative evaluation in the first two years. We will move on to summative evaluation in the third year to demonstrate the impact of the tools.

We will continue to organize a summer school each year, which we foresee will attract a group of 15+ researchers and subject matter experts to come to learn to build tutors. These summer schools will be opportunities to collect data we can use to evaluate if the tools are making building models easier and faster. In addition to the summer schools we will have students in Heffernan's class at WPI and Koedinger's class at CMU use the tools. We plan a close collaboration between the researchers at both WPI and CMU. Below is our plan for how the above work will be divided up between WPI and CMU.

Table 3 lists specific tasks to be performed, by whom, and estimates of the length of time to complete each task (measured in months). Please note the iterative design and re-design anticipated as we apply good HCI methods to create useable software.

CMU Specific Statement of Work

The CMU team (Koedinger & Alevan, with the assistance of researcher Mike Schneider⁵) will begin with extensive storyboarding of the tools. Alevan and Koedinger will also supervise the user testing. The first year implementation tasks focus on the GUI Builder and the Behavior Recorder. Alevan will be responsible for the Key Stroke Level Modeling of the tools built in the first year. In the second year the CMU team will focus on adding functionality to the Behavior Recorder to support the collection of think aloud protocols. How tasks are assigned to individuals in the third year will depend upon the previous rounds of iterative redesign.

WPI Specific Statement of Work

The first year, the WPI team (consisting of Heffernan with the assistance of a graduate student) will implement the Production Rule Editor within a third party Java-based production rule system (such as CLIPS or Jess). In addition, Heffernan will develop an “action interface” that will allow the production system chosen to be hooked up to the Behavior Recorder. Also, in the first year, Heffernan will instrument the tools to collect log file of user interactions with the tools. In the second year, Heffernan will design and implement the application program interface ("API") that will allow CMAT to communicate with third party simulators. In the second year Heffernan will also be focused on design and implementation of the Cognitive Model Visualizer debugging tool. How tasks are assigned in the third year will depend upon the previous rounds of iterative redesign.

⁵ Mike Schneider is a usability specialist with an M.Sc. degree in HCI. He is also a programmer and has implemented part of the CMAT tools.

Table 3: Tasks, length of time, types of tasks, and personnel assigned to each task.

Task description	Cost in Months			Personnel
	Test	(Re)design	Implement	
First year (Oct 1, 2002 to Oct 1, 2003)				
Keystroke level models of expert modelers	1			Aleven
Development of story boards for the GUI Builder, Behavior Recorder, and Production Rule Editor		2		Aleven, Heffernan, Koedinger
Further Implementation of GUI Builder, Behavior Recorder, and Production Rule Editor			3	CMU Programmer
Implement logging of tool use			1	CMU Programmer
User testing with expert modelers in lab	2			CMU Programmer
Redesign of tools		1		Aleven, Heffernan, Koedinger
Further Implementation of GUI Builder, Behavior Recorder, and Production Rule Editor			2	CMU Programmer
User testing with novices in Koedinger's classroom spring semester	2			CMU Programmer
Integrate JESS as modeling language (production system) and provide "action interface" to the simulation environment (get JESS to interact with EXE)			4	WPI Graduate Student
Implement log file collection in Java			2	WPI Graduate Student
Summer School evaluation in 2003. Target 1st year tools.	2			CMU Programmer
Implement Production Rule Editor with JESS			6	WPI Graduate Student
Redesign of 1st year's tools based on Summer School evaluation		1		Aleven, Heffernan, Koedinger
<i>First years subtotals for each different type of activity</i>	7	4	18	
Second Year (Oct 1, 2003 to Oct 1, 2004)				
Implementation for 1st year CMAT changes			2	CMU Programmer
Development of story boards for the Cognitive Model Visualizer, and for use of the Behavior Recorder in collecting think alouds		2		Aleven, Heffernan, Koedinger
Keystroke level analysis of 2nd year tools	1			CMU Programmer
Implementation for 2nd year CMAT tools			4	WPI Grad + CMU Programmer
Implement Java version of Cognitive Model Visualizer			3	WPI Graduate Student
User testing with novice modelers in Heffernan's and Koedinger's classrooms in Fall semester	2			WPI Grad + CMU Programmer
User testing with expert modelers in lab	2			CMU Programmer
Redesign based on classroom and lab tests		1		Aleven, Heffernan, Koedinger
Implementation of changes			4	CMU Programmer
Summer School evaluation in 2004 with formal comparison of use by experts v. by novices. Target 2nd year tools	2			CMU Programmer
Design API for alternative simulation plug-ins		1		Aleven, Heffernan, Koedinger
Implement API for alternative simulation plug-ins			3	WPI Graduate Student
Test the simulation API with an externally provided simulation	3			WPI Graduate Student
Redesign of 2nd year's tools based on Summer School evaluation		1		Aleven, Heffernan, Koedinger
<i>Second year's subtotals for each different type of activity</i>	10	5	14	
Third Year (Oct 1, 2003 to Oct 1, 2004)				
Implementation for 2nd year CMAT changes			3	WPI Grad + CMU Programmer
Development of story boards for the web deployment and the regression testing component of the Behavior Recorder		2		Aleven, Heffernan, Koedinger
Keystroke level analysis of 3rd year tools	1			CMU Programmer
Implementation for the above features 3rd year tools			6	WPI Grad + CMU Programmer

User testing with novice modelers in Heffernan's and Koedinger's classrooms in Fall semester	2		WPI Grad + CMU Programmer
Final controlled experiment with expert users	2		WPI Grad + CMU Programmer
Redesign based on classroom and lab tests		1	Aleven, Heffernan, Koedinger
Implementation of changes		3	WPI Grad + CMU Programmer
Summer School evaluation in 2005 with formal comparison of use by experts v. by novices. Target 3rd year tools	4		WPI Grad + CMU Programmer
Redesign of 3rd year's tools based on Summer School evaluation		1	Aleven
Implementation for 3rd year CMAT changes		3	WPI Grad + CMU Programmer
Final report		1	Aleven, Heffernan, Koedinger
<i>Third year's subtotals for each different type of activity</i>	<i>10</i>	<i>4</i>	<i>15</i>
<i>Cumulative totals for each type of activity</i>	<i>27</i>	<i>13</i>	<i>47</i>

KENNETH R. KOEDINGER

A. Professional Preparation

Ph. D. Cognitive Psychology. Carnegie Mellon University. December, 1990.

M.S. Computer Science. University of Wisconsin, Madison. May, 1986.

B.S. Math and Computer Science. University of Wisconsin, Madison. With distinction. May, 1984.

B. Recent Appointments

Associate Professor with Tenure. Human-Computer Interaction Institute. School of Computer Science, Carnegie Mellon University. 2001 to present.

Senior Research Scientist. Faculty position equivalent to Associate Professor. Human-Computer Interaction Institute. School of Computer Science, Carnegie Mellon University. 1999 to 2001.

Co-Founder, Board of Directors, and Consultant. Carnegie Learning, Inc. 1998 to present.

C. Career Summary

My multi-disciplinary preparation has been critical to my research goal of creating educational technologies that dramatically increase student achievement. Toward this goal, I create "cognitive models", computer simulations of student thinking and learning, that are used to guide the design of educational materials, practices and technologies. These cognitive models provide the basis for an approach to educational technology called "Cognitive Tutors" in which we create rich problem solving environments for students to work in and provide just-in-time learning assistance much like a good human tutor does. I have developed Cognitive Tutors for mathematics and science and have tested them in the laboratory and the classroom. In a whole-year classroom study with our Algebra Cognitive Tutor, I have shown that students in our experimental classrooms outperformed students in control classes by 50-100% on targeted real world problem solving skills and by 10-25% on standardized tests. My research has contributed new principles and techniques for the design of educational software and has produced basic cognitive science research results on the nature of human thinking and learning. I have authored 47 peer-reviewed publications, 2 textbooks, 6 book chapters, and 41 other papers and have been a Project Investigator on 15 major grants. I currently codirect the Pittsburgh Advanced Cognitive Tutor Center and am managing teams of cognitive scientists, programmers, and teachers to create integrated learning solutions that include text materials, teacher training and Cognitive Tutors. I am a co-founder of Carnegie Learning Inc., a company marketing these technology-enhanced learning solutions to schools and colleges across the country.

D. Pertinent Publications

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4 (2), 167-207.

Aleven, V.A.W.M.M., & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science*, 26(2).

Corbett, A. T., Koedinger, K. R., & Hadley, W. H. (2001). Cognitive Tutors: From the research classroom to all classrooms. In Goodman, P. S. (Ed.) *Technology Enhanced Learning: Opportunities for Change*. Mahwah, NJ: Lawrence Erlbaum Associates.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8, 30-43.

Koedinger, K. R., Suthers, D. D., & Forbus, K. D. (1999). Component-based construction of a science learning space. *International Journal of Artificial Intelligence in Education*, 10.

Ritter, S. & Koedinger, K. R. (1996). An architecture for plug-in tutoring agents. In *Journal of Artificial Intelligence in Education*, 7 (3/4), 315-347. Charlottesville, VA: Association for the Advancement of Computing in Education.

VINCENT A.W.M.M. ALEVEN

A. Professional Preparation

Ph. D. Intelligent Systems, University of Pittsburgh, 1997

M. Sc. Intelligent Systems, University of Pittsburgh, 1992

M.Sc. Computer Science, Delft University of Technology, 1988

B. Recent Appointments

Systems Scientist. Human-Computer Interaction Institute. School of Computer Science, Carnegie Mellon University. 2000-present.

Post-Doctoral Fellow. Human-Computer Interaction Institute. School of Computer Science, Carnegie Mellon University. 1997-2000.

C. Career Summary

Dr. Vincent Aleven is a Systems Scientist in the Human-Computer Interaction Institute at Carnegie Mellon University. His research interests include learning and the design of intelligent instructional software. Currently, his research focuses on two themes. First, he investigates how existing “2nd-generation” computer-based tutors can be made more effective by supporting metacognitive processes such as self-explanation and help seeking. This research takes place under the umbrella of the CIRCLE Center, a large NSF-sponsored research collaboration between groups at Carnegie Mellon and the University of Pittsburgh, with the goal of applying lessons learned from investigations of humans tutoring to the design and implementation of computer tutors. Vincent participated in an NSF/DFG-sponsored series of Early Career Workshops for American and German researchers in the field of instructional technology. In this context he initiated a collaborative effort involving colleagues at three German universities and one American university with the purpose of investigating students’ help-seeking behavior in different contexts using different technologies. Second, Vincent is involved in a research project to develop authoring tools for the development of Cognitive Tutors. In this role, he helped design the Tutor Agent Development Kit and has managed a development team consisting of three research programmers and two undergraduates. Vincent was a member of the program committee for the 10th International Conference on Artificial Intelligence in Education, AI-ED 2001 and of the Program Committees for the 7th, 8th, and 9th International Conferences on Artificial Intelligence and Law (ICAIL). He was involved in organizing (among others) the following workshops: the ITS-2002 Workshop on Empirical Methods for Tutorial Dialogue Systems (co-chair), the AIED-2001 Workshop on Tutorial Dialogue Systems (chair), the AIED-2001 Workshop on Help Provision and Help Seeking in Interactive Learning Environments, and the AAAI 2000 Fall Symposium on Building Dialogue Systems for Tutorial Applications.

D. Pertinent Publications

- Aleven, V., and K. R. Koedinger. An Effective Meta-cognitive Strategy: Learning by Doing and Explaining with a Computer-Based Cognitive Tutor. *Cognitive Science*, 26(2), 147-179, 2002.
- Aleven V., O. Popescu, and K. R. Koedinger. Pilot-Testing a Tutorial Dialogue Systems that Supports Self-Explanation. In the *Proceedings of Sixth International Conference on Intelligent Tutoring Systems, ITS 2002*.
- Aleven V., O. Popescu, and K. R. Koedinger Towards Tutorial Dialog to Support Self-Explanation: Adding Natural Language Understanding to a Cognitive Tutor. In *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future, Proceedings of AI-ED 2001*, edited by J. D. Moore, C. L. Redfield, and W. L. Johnson, 246-255. Amsterdam, IOS Press, 2001.
- Aleven, V., and K. R. Koedinger. Limitations of Student Control: Do Student Know when they need help? In *Proceedings of the 5th International Conference on Intelligent Tutoring Systems, ITS 2000*, edited by G. Gauthier, C. Frasson, and K. VanLehn, 292-303. Berlin: Springer Verlag, 2000. *Best Paper Award ITS 2000*.
- Aleven, V., and K. D. Ashley. Teaching Case-Based Argumentation Through a Model and Examples: Empirical Evaluation of an Intelligent Learning Environment. In *Artificial Intelligence in Education, Proceedings of AI-ED 97 World Conference*, edited by B. du Boulay and R. Mizoguchi, 87-94. Amsterdam: IOS Press, 1997.
- Ashley, K. D., and V. Aleven. Reasoning Symbolically about Partially Matched Cases. In *Proceedings IJCAI 97, the Fifteenth International Joint Conference on Artificial Intelligence*, edited by M. Pollack, 335-341. San Francisco, CA: Morgan Kaufmann, 1997.

NEIL T. HEFFERNAN

A. Professional Preparation

Ph. D. Computer Science. Carnegie Mellon University. 2001

M.S. Computer Science. Carnegie Mellon University. 1997

B.A. Computer Science and History. Amherst College. *Summa cum laude*. 1993.

B. Recent Appointments

Assistant Professor. Worcester Polytechnic Institute. Department of Computer Science. July, 2002 to present.

Post-Doctoral Researcher. Human-Computer Interaction Institute. School of Computer Science, Carnegie Mellon University. 2001 to 2002.

C. Career Summary

Dr. Neil Heffernan graduated *summa cum laude* from Amherst College in History and Computer Science. Neil taught mathematics to eighth grade students in Baltimore City as part of *Teach for America*, a program that selectively recruits top candidates to teach in inner-city schools., Neil attended Carnegie Mellon University's Computer Science department to do research in creating educational software that leads to higher student achievement. For his dissertation, he built the first *intelligent tutoring system* that incorporated a model of tutorial dialog. This system has been shown to lead to higher student learning, by getting students to think more deeply about problems. It is based upon detailed studies of student learning as well as studies of experienced human teachers. The system (free at www.AlgebraTutor.org) has been used by thousands of students and teachers and has been awarded many educational awards. The system is currently in the process of being licensed to a spin-off company and has a patent pending. As a post-doc, Neil managed a team of four programmers and PhDs to creating the initial authoring tools used successfully by researchers from universities, commercial companies and military labs, from around the world in the 2002 Summer School for building intelligent tutoring agents. Neil is a Spencer Foundation / National Academy of Education Postdoctoral Research Fellow. Neil is now a assistant professor at Worcester Polytechnic Institute, where one of his projects is organizing "The Learning Open" (www.LearningOpen.org), an interdisciplinary collaboration between educational software researchers and classroom teachers to study what are the benefits of different instructional approaches.

D. Pertinent Publications

- Heffernan, N. T., & Koedinger, K. R. (2002) An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor. In the *Proceedings of the Sixth International Conference on Intelligent Tutoring System 2002*. Biarritz, France.
- Heffernan, N. T., & Koedinger, K. R. (2001) Results from a Web-Based Tutor for Writing Algebra Expressions for Word-Problems. *Sciences et Techniques Educatives*. A French journal named "Educational Sciences and Technology."
- Heffernan, N. T. (2001) Intelligent Tutoring Systems are Forgotten the Tutor: Adding a Cognitive Model of Human Tutors. Dissertation. Computer Science Department, School of Computer Science, Carnegie Mellon University. Technical Report CMU-CS-01-127.
- Heffernan, N. T., & Koedinger, K. R. (2000) Intelligent Tutoring Systems are Missing the Tutor: Building a More Strategic Dialog-Based Tutor. *AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications*.
- Heffernan, N. T. & Koedinger, K. R. (1998). A developmental model for algebra symbolization: The results of a difficulty factors assessment. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, (pp. 484-489). Hillsdale, NJ: Erlbaum
- Heffernan, N. T. & Koedinger, K.R. (1997). The composition effect in symbolizing: The role of symbol production vs. text comprehension. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, (pp. 307-312). Hillsdale, NJ: Erlbaum. [The Marr Prize winner for best student paper.]