Turner, T.E., Macasek, M.A., Nuzzo-Jones, G., Heffernan, N..T, Koedinger, K. (2005). The Assistment Builder: A Rapid Development Tool for ITS. In C.K. Looi, G. McCalla, B. Bredeweg, & J. Breuker (Eds.) *Proceedings of the 12th Artificial Intelligence In Education*, 929-931. Amsterdam: ISO Press.

The Assistment Builder: A Rapid Development Tool for ITS

Terrence E. TURNER, Michael A. MACASEK, Goss NUZZO-JONES, Neil T. HEFFERNAN
Worcester Polytechnic Institute
100 Institute Rd, Worcester, MA 01609
(508) 831-5569
tinylox@wpi.edu, macasek@wpi.edu, goss@wpi.edu, nth@wpi.edu
Ken KOEDINGER
Human-Computer Interaction Institute
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. Intelligent Tutoring Systems are notoriously costly to construct [1], and require PhD level experience in cognitive science and rule based programming. The goal of this research was to ease the development process for building pseudo-tutors [6], which are ITS constructs that mimic cognitive tutors but are limited in that they only work for a single problem. The Assistment Builder is a system designed to rapidly develop, test, and deploy simple pseudo-tutors. These tutors provide a simple cognitive model based upon a state graph tailored to a specific problem. These tutors offer many of the features of rule-based tutors, but without the expensive creation time. The system simplifies the process of tutor construction to allow users with little or no ITS experience to develop content. The system provides a web-based interface as a means to build and store these simple tutors we have called *Assistments*. This paper describes our attempt to make the process of developing content easy for teachers. We present some evidence to suggest that these novice users can develop a tutor for a problem in under thirty minutes.

Keywords: Intelligent Tutoring Systems, Cognitive Tutor, Authoring Tools, Web Deployment, Pseudo-Tutors

The Assistment Builder: A Rapid Development Tool for ITS

Terrence E. TURNER, Michael A. MACASEK, Goss NUZZO-JONES, Neil T. HEFFERNAN
Worcester Polytechnic Institute
100 Institute Rd, Worcester, MA 01609
(508) 831-5569
tinylox@wpi.edu, macasek@wpi.edu, goss@wpi.edu, nth@wpi.edu
Ken KOEDINGER
Human-Computer Interaction Institute
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. Intelligent Tutoring Systems are notoriously costly to construct [1], and require PhD level experience in cognitive science and rule based programming. The goal of this research was to ease the development process for building pseudo-tutors [6], which are ITS constructs that mimic cognitive tutors but are limited in that they only work for a single problem. The Assistment Builder is a system designed to rapidly develop, test, and deploy simple pseudo-tutors. These tutors provide a simple cognitive model based upon a state graph tailored to a specific problem. These tutors offer many of the features of rule-based tutors, but without the expensive creation time. The system simplifies the process of tutor construction to allow users with little or no ITS experience to develop content. The system provides a web-based interface as a means to build and store these simple tutors we have called *Assistments*. This paper describes our attempt to make the process of developing content easy for teachers. We present some evidence to suggest that these novice users can develop a tutor for a problem in under thirty minutes.

1.0 Introduction

This research aims to develop tools for the rapid development and deployment of Intelligent Tutoring Systems (ITS). Specifically, this research focused on so-called "pseudo-tutors" that are a simplification of cognitive rule-based tutors [6]. Model tracing rule-based tutors [1] have been shown to be effective [7], but development time on them is highly prohibitive, from 100-1000 hours of development time per hour of content [8][1]. Development also requires a very specialized knowledge set. Tutor developers are required to be expert system programmers, in addition to developing the cognitive model, to say nothing of being a content expert. Another aim of this research was to make our tools accessible to novices, with no programming experience, and less than an hour of training.

A pseudo-tutor is a simplified cognitive model based on a state graph. State graphs are finite graphs with each arc representing a student action, and each node representing a state of the problem interface [4][11]. Student actions trigger transitions in the graph, and the current state of the problem is stored by the graph. Pseudo-tutors have nearly identical behavior to a rule-based tutor, but suffer from having no ability to generalize to different problems [5]. This pseudo-tutor approach allows for predicted behaviors and provides feedback based on those behaviors. We also combined this state graph with a conceptually broader branching structure referred to as scaffolding. Scaffolding provides sub-problems to the initial question, often designed to address specific concepts within the initial question. Both initial and scaffold questions can branch to different scaffolding questions depending on a student's actions. This allows for a higher-level of predicted actions to be handled.

1.1 Overview of the Assistments Project

These pseudo-tutors are being developed and deployed as part of the Assistments Project [10]. The Assistments architecture is an interactive content delivery system designed to deploy both pseudo-tutors and full cognitive model tutors over the web [9] with centralized database logging of student actions [3]. The architecture is highly extensible and provides for many different tutoring strategies, including the aforementioned scaffolding strategy described above. The foundation of this architecture is the content representation, an XML (eXensible Markup Language) schema that defines each problem. A problem consists of an interface definition and behavior definition. The interface definition provides a collection of simple widgets to be displayed to the student. The behavior definition is a representation of the state graph and its transitions, or a cognitive model (e.g. JESS rules). Many types of behaviors are possible within the representation and architecture. These two parts of the representation are consumed by the runtime Assistment architecture, and presented to the student over the web. Student actions are then fed back to the representation, and compared with the state graph or used to model trace.

1.2 Purpose of the Assistment Builder

The XML representation of content provides a base for which we can rapidly create specific pseudo-tutors. We sought to create a tool that would provide a simple web-based interface for creating these pseudo-tutors. Upon content creation, we could rapidly deploy the tutor across the web, and if errors were found with the tutor, bug-fixing or correction would be quick and simple. Finally, the tool had to be usable by someone with no programming experience and no ITS background. This applied directly to our project of creating tutors for the mathematics section of the Massachusetts Department of Education (MCAS) test [10]. We wanted the teachers in the public school system to be able to build pseudo-tutors. These pseudo-tutors are often referred to as *Assistments*, but the term is not limited to pseudo-tutors.

A secondary purpose of the Assistment Builder was to aid the construction of a Transfer Model. A Transfer Model is a cognitive model construct divorced from specific tutors. The Transfer Model is a directed graph of *knowledge components* representing specific concepts that a student could learn. These *knowledge components* are then associated with a specific tutor (or even sub-question within that tutor) so that the tutor is associated with a number of *knowledge component* arcs associated with it. This allows us to maintain a complex cognitive model of the student without necessarily involving a production rule system. It also allows analysis of student performance in the context of the Transfer Model, resulting in knowledge tracing [2] and other useful methods. The simplest way to "mark" tutors in a Transfer Model is to associate the tutors (or their sub-questions) with *knowledge components* when the tutors are created. At present, the Transfer Model being used to classify 8th grade mathematics items has 174 *knowledge components*. Over the six months since the inception of the Assistment Builder, nearly 1000 individual problems have thus far been associated with these 174 *knowledge components*.

1.2.1 Assistments

The basic structure of an *Assistment* is a top-level question that can then branch to scaffolding questions based on student input. The Assistment Builder interface uses only a subset of the full content XML representation, with the goal of producing simple pseudo-tutors. Instead of allowing arbitrary construction of question interfaces there are only five widget choices available to a content creator. These are radio-buttons, pull-down menus, checkboxes, text-fields, and algebra text fields that automatically evaluate mathematical expressions. The state graphs for each question are limited to two possible states. An arc occurs between the two states when the end-user answers a question properly. The student will remain in the initial state until the question is answered properly or skipped programmatically.

The scaffolding questions mentioned above are all queued as soon as a user gets the top-level question incorrect, or requests help in the form of a hint (for either event, the top-level question is skipped). Upon successfully completing the displayed scaffolding question the next is displayed until the queue is empty. Once the queue is empty, the problem is considered complete. This type of linear *Assistment* can be easily made with our tool, by first creating the main item and then the subsequent scaffolding questions. When building an *Assistment* a user may also add questions that will appear when a specific incorrect answer is received. This allows branches to be built that tutor along a "line of reasoning" in a problem, which adds more generality than a simple linear *Assistment*. Many *Assistment* authors also use text feedback on certain incorrect answers. These feedback messages are called bug messages. Bug messages address the specific error made, and match common or expected mistakes.

Content creators can also use the Assistment Builder to add hint messages to problems, providing the student with hints attached to a specific scaffolding question. This combination of hints, buggy messages, and branched scaffolding questions allow even the simple state diagrams described above to assume a useful complexity. *Assistments* constructed with the Assistment Builder can provide a tree of scaffolding questions branched from a main question. Each question consists of a customized interface, hint messages and bug messages, along with possible further branches.

1.2.2 Web Deployment

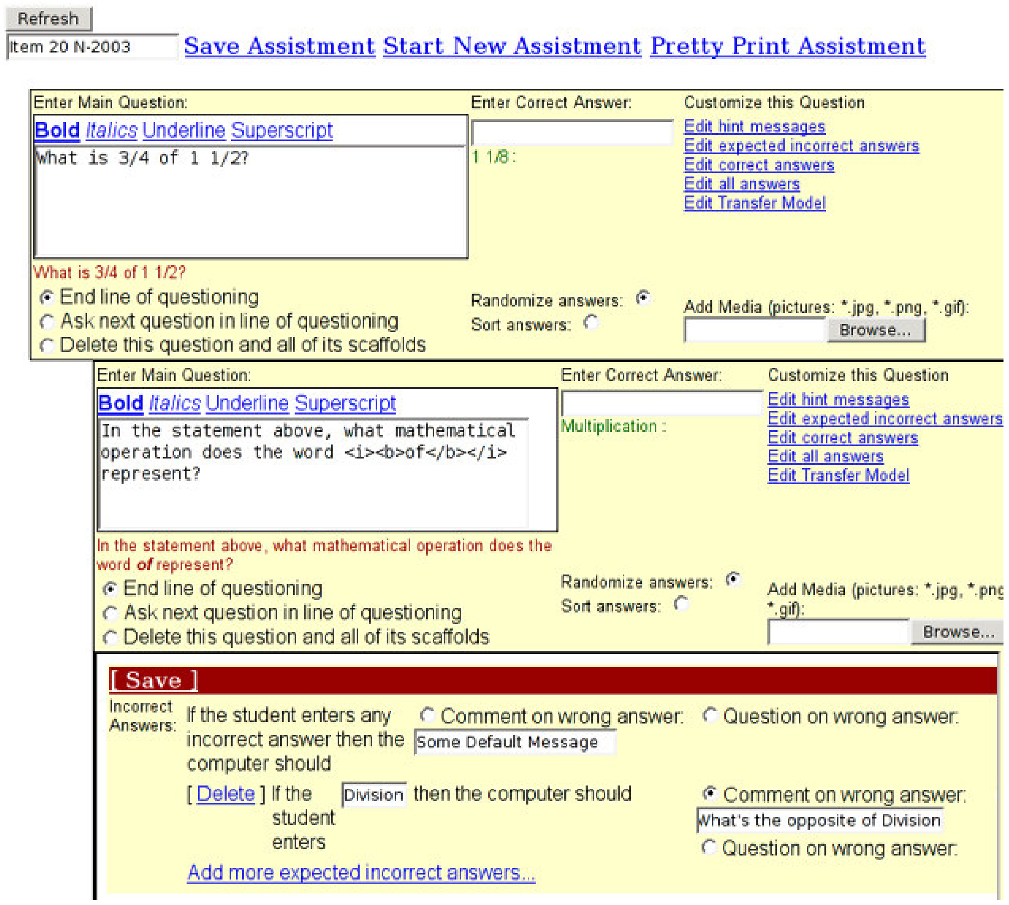
We constructed the Assistment Builder as a web application for accessibility and ease of use purposes, a teacher or content creator can create, test, and deploy an *Assistment* without installing any additional software. A user can design and test his *Assistment* and then instantly deploy it. If further changes or editing are needed the *Assistment* can be loaded into the builder, modified, and then redeployed across all the curriculums that make use of the tutor. By making the *Assistment* Builder available over the web, if a new feature is added users do not need to update any software. They reap the benefits of any changes made to the system as soon as they log on. By storing created *Assistments* locally on our servers, allowing end-users to easily create a curriculum and assign it to a class for use by students is a simple task.

1.2.3 Features

Though there are many significant improvements to be made to the Assistment Builder's user interface, it is usable and reasonably easy to learn. When a user first begins to use the Assistment Builder they will be greeted by the standard blank skeleton question. The initial blank skeleton question will be used to create the root question. The user will enter the question text, images, answers, and hint messages to complete the root question. After these steps the appropriate scaffolding is added. The question layout is separated into several views the *Main View*, *All Answer View*, *Correct Answer View*, *Incorrect Answer View*, *Hints View*, and *Transfer Model View*. Together these views allow a user to highly customize their question and its subsequent scaffolding.

Initially the user is presented with the *Main View* (see Figure 1). In this view question text, correct answers, and images can be added to the question. Additionally the user can add new scaffolding off the current question, and specify if they would like the answers to be in a sorted or random order. The *Main View* is designed to gather the absolute minimum information needed to generate a question.

Upon completion of the items in the *Main View* the user then has the option to move to other views in order to further refine the question. Typically the next view to complete is the *All Answer View*. This can be done by selecting "Edit all answers" from the list of links below Customize this Question from the *Main View*. In the *All Answers View* the user has the option to add additional correct answers as well as expected incorrect answers. The expected incorrect answers serve two purposes. First, they allow a teacher to specify the answers students will most likely expect as the correct answer and provide feed back in the form of a message or scaffolding. Second, the user has the option to select the question input type, certain input types present a list of answers for the student to select from, expected incorrect answers are useful here to populate this list; additional tutoring by means of message or scaffold is also possible with these incorrect answers.


Refresh
Item 20 N-2003
Save Assistment Start New Assistment Pretty Print Assistment

Enter Main Question:
Bold Italics Underline Superscript
```
What is 3/4 of 1 1/2?
```
What is 3/4 of 1 1/2?
[✓ End line of questioning]
Ask next question in line of questioning
Delete this question and all of its scaffolds

Enter Correct Answer:
1 1/8 :
Randomize answers: [✓]
Sort answers:

Customize this Question
Edit hint messages
Edit expected incorrect answers
Edit correct answers
Edit all answers
Edit Transfer Model
Add Media (pictures: *.jpg, *.png, *.gif):
Browse...

Enter Main Question:
Bold Italics Underline Superscript
```
In the statement above, what mathematical
operation does the word <i><b>of</b></i>
represent?
```
In the statement above, what mathematical operation does the word ***of*** represent?
[✓ End line of questioning]
Ask next question in line of questioning
Delete this question and all of its scaffolds

Enter Correct Answer:
Multiplication :
Randomize answers: [✓]
Sort answers:

Customize this Question
Edit hint messages
Edit expected incorrect answers
Edit correct answers
Edit all answers
Edit Transfer Model
Add Media (pictures: *.jpg, *.png *.gif):
Browse...

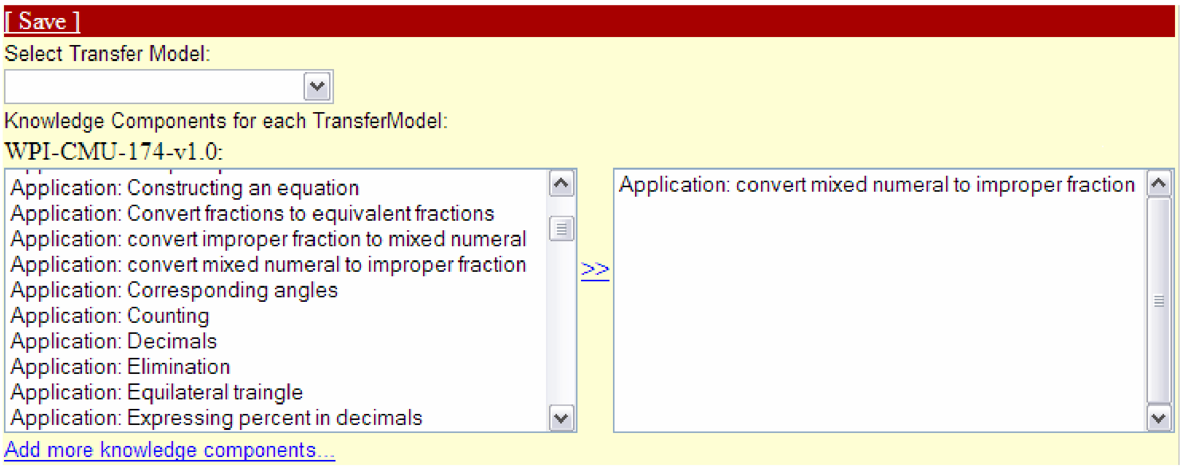
[Save]
Incorrect Answers: If the student enters any incorrect answer then the computer should — Comment on wrong answer: Some Default Message — Question on wrong answer:
[Delete] If the student enters Division then the computer should — [✓ Comment on wrong answer:] What's the opposite of Division — Question on wrong answer:
Add more expected incorrect answers...

Figure 1 - Assistment Builder – Initial Question, One Scaffold, and Incorrect Answer View

Once the user has finished with this view either the *Correct Answer View* or the *Incorrect Answer View* (see Figure 1) is the next step. In the *Correct Answer View* the user can now provide a specific message to be displayed on a correct answer as well as add additional correct answers. From the *Incorrect Answer View* further information can be provided as to the action that will be taken on a specific incorrect answer; new incorrect answers can also be added here. The user now has the option to specify a message to be displayed for an incorrect answer or the option to scaffold. If the scaffolding option is chosen a new question block will appear indented below the current question.

In the *Hints View* messages can be created that will be presented to the student when a hint is requested. Hints can consist of text, an image, or both. Multiple hint messages can be entered; one message will appear per hint request in the order that they are listed in this view. The final view is the *Transfer Model View* (see Figure 2). In this view the user has the option of specify one or more skills that are associated with this particular question. These skills are then used to establish learning accomplished on this problem.

Important to note is that each view, except the *Main View*, requires the user to save before leaving the view. This ensures that all entered data is stored. At any point the user can preview the problem by clicking on the preview link to view the current state of the question.


[Save]
Select Transfer Model:
Knowledge Components for each TransferModel:
WPI-CMU-174-v1.0:

Application: Constructing an equation
Application: Convert fractions to equivalent fractions
Application: convert improper fraction to mixed numeral
Application: convert mixed numeral to improper fraction
Application: Corresponding angles
Application: Counting
Application: Decimals
Application: Elimination
Application: Equilateral traingle
Application: Expressing percent in decimals

>>

Application: convert mixed numeral to improper fraction

Add more knowledge components...

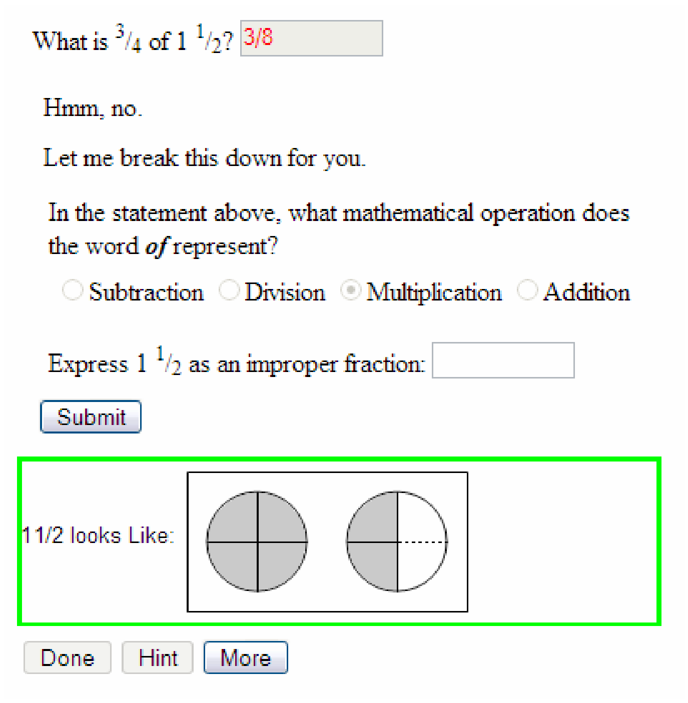
Figure 2 - Transfer Model View

As mentioned above there are two methods of providing scaffolding question; either by selecting Ask Next Line of Questioning from the *Main View* or specify scaffolding on a specific incorrect answer. In utilizing either of these methods a new skeleton question will be inserted into the correct position below the current question. Creating a scaffolding question is done in the exact manner as the root question. Once a user is satisfied with the root question and its scaffolding they provide a name for the question and save the *Assistment*. After saving the *Assistment* the tutor is ready to be used. An *Assistment* can be modified at anytime by loading it into the Assistment Builder and changing its properties accordingly. A completed running *Assistment* can be seen in Figure 3.

2.0 Methods

To analyze the effectiveness of the Assistment Builder, we developed a system to log the actions of an author. While authors have been constructing items for nearly six months, only very recently has the Assistment Builder had the capability to log actions.

Each action is recorded with associated meta-data, including author, timestamps, the specific series of problems being worked on, and data specific to each action. These actions were recorded for a number of Assistment authors over several days. The authors were asked to build original items and keep track of roughly how much time spent on each item for corroboration. The authors were also asked to create "morphs," a term used to indicate a new problem that had a very similar setup to an existing problem. "Morphs" are usually constructed by loading the existing problem into the Assistment Builder, altering it, and saving it with a different name. This allows rapid content development for testing transfer between problems. We wanted to compare the development time for original items to that of "morphs" [10].



What is $^3/_4$ of 1 $^1/_2$? 3/8

Hmm, no.

Let me break this down for you.

In the statement above, what mathematical operation does the word *of* represent?

Subtraction Division [✓ Multiplication] Addition

Express 1 $^1/_2$ as an improper fraction:

Figure 3 - Assistment Running

Another trial of the Assistment Builder with less rigorous methodology was testing how authors with little experience would react to the software. To test the usability of the Assistment Builder, we were able to provide the software to two high-school teachers in the Worcester, Massachusetts area. These teachers were computer literate, but had no previous experience with intelligent tutoring systems, or creating mathematics educational software. Our tutorial consisted of demonstrating the creation of a problem using the Assistment Builder, then allowing the teacher to create their own with an experienced observer to answer questions. Finally, we hope to allow them to author *Assistments* on their own, without assistance.

3.0 Results & Analysis

Prior to the implementation of logging within the Assistment Builder, we obtained encouraging anecdotal results of the software's use. A high-school mathematics teacher was able to create 15 items and morph each one, resulting in 30 *Assistments* over several months. Her training consisted of approximately four hours spread over two days in which she created 5 original *Assistments* under supervision. While there is unfortunately no log data to strengthen this result, it is nonetheless encouraging.

The logging data obtained suggests that the average time to build an entirely new *Assistment* is approximately 25 minutes. Entirely new *Assistments* are those that are built using new content and not based on existing material. This data was acquired by examining the time that elapsed between the initialization of a new problem and the problem save time. Creation times for *Assistments* with more scaffolds naturally took longer than those with fewer scaffolds. Experience with the system also decreases *Assistment* creation time, as end-users who are more comfortable with the Assistment

Builder are able work faster. Nonetheless, even users who were just learning the system were able to create *Assistments* in reasonable time. For instance, Users 2, 3, and 4 (see Table 1) provide examples of end-users who have little experience using the Assistment Builder. In fact, some of them are using the system for the first time in the examples provided.

Username	**Number of Scaffolds**	**Time Elapsed (min)**
User 1	10	35
User 1	2	23
User 2	3	45
User 2	2	31
User 2	0	8
User 3	2	21
User 4	3	37
User 4	0	15
User 5	4	30
User 5	2	8
User 5	4	13
User 5	4	35
User 5	3	31
User 5	2	24
		Average: 25.4 minutes

Table 1 - Full Item Creation

We were also able to collect useful data on morph creation time and Assistment editing time. On average morphing an *Assistment* takes approximately 10-20 minutes depending on the number of scaffolds in an Assistment and the nature of the morph. More complex Assistment morphs require more time because larger parts of an *Assistment* must be changed. Editing tasks usually involve minor changes to an *Assistments* wording or interface. These usually take less than a minute to locate and fix.

4.0 Conclusions

The Assistment Builder is a conceptual extension of the CTAT project [6]. However, the pseudo-tutors constructed by the Assistment Builder are generally dialog based, rather than workspace centric. The Assistment Builder's goal was to limit interface options as a trade-off for development speed, while the CTAT tools are more complex and powerful in terms of interface (and thus have a higher learning curve). The educational impact of this trade-off is presently unknown and needs exploration.

The Assistment Builder has been in use over six months by a variety of users involved in the Assistments project. Teachers, developers, and others have used it to develop pseudo-tutor *Assistments*. The end result has been over a thousand individual pseudo-tutors deployed on the web. The breadth of users who developed these *Assistments* and the number created would not have been possible without the Assistment Builder.

4.1 Future Work

In our continuing efforts to provide a tool that is accessible to even the most novice users we are currently working on two significant enhancements to the Assistment Builder. The first enhancement is a simplified interface that is both user-friendly and still provides a means to create powerful scaffolding pseudo-tutors. The most significant change to the current interface is the addition of a tab system that will allow the user to clearly navigate the different components of a question. The use of tabs allows us to present the user with only the information related to the current view, reducing the confusion that sometimes takes place in current interface. Additionally, with respect to the saving of inputted data we are taking steps to ensure that input is constantly saved without the need for users to specify that they wish to save. This will allow the users to focus on building the questions and creating rich tutors without worrying if their data will be lost.

The second significant enhancement is a new question type. This question type will allow a user to create a question with multiple inputs of varying type. The user will also be able to include images and Macromedia Flash™ movies. Aside from allowing multiple answers in a single question, the new question type allows a much more customizable interface for the question. Users can add, in any order, a text component, a media component, or an answer component. The ability to place a component in any position in the question will allow for a more "fill in the blank" feel for the question and provide a more natural layout. This new flexibility will no longer force questions into the text, image, answer format that is currently used.

Other planned enhancements are to provide an interface for creating more complex types of tutors with richer state graphs, and eventually rule-based tutors. Such improvements might include support for learning rule-based cognitive models from examples [5]. Other options could include a wizard interface for simple scripting support (similar to the scripting wizards in Microsoft Excel™).

References

1. Anderson, J. R. (1993). Rules of the mind. Hillsdale, NJ: Erlbaum.
2. 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.
3. Feng, Mingyu, Heffernan, N.T. (2005). Informing Teachers Live about Student Learning: Reporting in the Assistment System. *Submitted to the 12th Annual Conference on Artificial Intelligence in Education 2005, Amsterdam*
4. Jackson, G.T., Person, N.K., and Graesser, A.C. (2004) Adaptive Tutorial Dialogue in AutoTutor. *Proceedings of the workshop on Dialog-based Intelligent Tutoring Systems at the 7th International conference on Intelligent Tutoring Systems. Universidade Federal de Alagoas, Brazil, 9-13.*
5. Jarvis, M., Nuzzo-Jones, G. & Heffernan. N. T. (2004) Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems. Proceedings of 7th Annual Intelligent Tutoring Systems Conference, Maceio, Brazil. Pages 541-553
6. Koedinger, K. R., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M. (2004) Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. *Proceedings of 7th Annual Intelligent Tutoring Systems Conference, Maceio, Brazil.* Page 162-173
7. 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.
8. 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.
9. Nuzzo-Jones, G., Walonoski, J.A., Heffernan, N.T., Livak, T. (2005). The eXtensible Tutor Architecture: A New Foundation for ITS. *Submitted to the 12th Annual Conference on Artificial Intelligence in Education 2005, Amsterdam*
10. Razzaq, L., Feng, M., Nuzzo-Jones, G., Heffernan, N.T., Aniszczyk, C., Choksey, S., Livak, T., Mercado, E., Turner, T.E., Upalekar. R, Walonoski, J.A., Macasek. M.A., Rasmussen, K.P. (2005) The Assistment Project: Blending Assessment and Assisting. *Submitted to the 12th Annual Conference on Artificial Intelligence in Education 2005, Amsterdam*
11. Rose, C. P. Gaydos, , A., Hall, B. S., Roque, A., K. VanLehn, (2003), *Overcoming the Knowledge Engineering Bottleneck for Understanding Student Language Input , Proceedings of AI in Education.*