The *ASSISTment Builder*: Towards an Analysis of Cost Effectiveness of ITS Creation

Neil T. Heffernan, Terrence E. Turner, Abraao L. N. Lourenco, Michael A. Macasek, Goss Nuzzo-Jones, Kenneth R. Koedigner*

Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609
1-508-831-5569

Carnegie Mellon University
5000 Forbs Avenue
Pittsburg, PA 15213
421-268-2000

nth@wpi.edu, terrence.turner@gmail.com, aln@wpi.edu, macasek@wpi.edu, goss@wpi.edu, koedinger@cmu.edu*

Abstract

Intelligent Tutoring Systems, while effective at producing student learning [2,7], are notoriously costly to construct [1,9], and require PhD level experience in cognitive science and rule based programming. The literature suggests [1,9] that it takes at least 200 hours of work to build 1 hour on ITS content. We have been engaged in building tools to reduce the development time, by allowing authors with no programming experience to build "pseudo-tutors" [6]. Pseudo-tutors are ITS constructs that mimic cognitive tutors but are limited in that they only apply to a single problem. The *ASSISTment Builder* is a tool designed to rapidly create, test, and deploy a very simple type of pseudo-tutors called *ASSISTments*. These tutors provide a simplified cognitive model based upon a state graph designed for a specific problem. These tutors offer many of the features of rule-based tutors, but with shorter creation time. The system simplifies the process of tutor creation 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, testing, and deploying content easy for teachers. We present data to suggest with the *ASSISTments Builder* we have reduced the costs of building pseudo-tutors by as much as a factor of four. We have achieved this time reduction, while at the same time making tools that eliminated the need for AI rule-based programming. We conclude with some discussion of the limitations and trade-off that have been made.

Introduction

This research seeks to address the high development time of cognitive rule-based tutors in Intelligent Tutoring Systems (ITS). Despite the effectiveness of model-tracing rule based tutors [7], it has been shown that development time can be between 200-1000 hours per hour of content created [1,9]. Creating cognitive tutors also requires high level computer science and cognitive psychology domain knowledge; typically PhD level experience in Artificial intelligence rule-based programming.

The Office of Naval Research funded Carnegie Mellon University and Worcester Polytechnic Institute to create tools to reduce the cost of making intelligent tutoring systems. There are two ways to reduce these costs. One is to make tools that are faster to use. The other is to make them easier to use, thus removing the need for PhD level Artificial Intelligence rule-based programmers and cognitive scientists. The goal was to provide a tool to allow rapid content creation to users with little computer science or cognitive psychology background. Koedinger, Aleven, Heffernan, McLaren & Hockenberry created the Cognitive Tutor Authoring Tools (CTAT) that allowed the creation of what were termed "pseudo-tutors" [6]. Pseudo-tutors represent a simplified cognitive model that is comprised of a state graph. This graph is finite, and each node representing a possible state of the problem. User actions are represented by arcs in the graph, with specific user actions triggering state transitions [12]. A user's location in the graph represents the problem's current state, and student actions correspond to possible transitions from that state. Despite having similar behavior to rule-based tutors, pseudo-tutors lack the ability to generalize over similar problems [5]. However, they can be designed to predict certain behaviors and respond accordingly. CTAT allowed all this but suffered a few limitations. First, even though CTAT requires no programming, it still requires an author to download, and set up, an Integrated Development Environment called NetBeans to build the interface that the students will use. We instead chose to allow these pseudo-tutors to be built and accessed via a web-site. The web-site hosts the Builder Application as well as the service that allows students to access that content. A second limitation of CTAT was that it was not easy to carry on a "dialog"; so ASSISTment added this feature by combining the state graph with a branching problem structure we call "scaffolding". Scaffolds are sub-problems usually designed to address a specific skill needed to solve the initial problem. Scaffolding questions in turn contain their own state graphs, and depending upon student actions, scaffolds can branch into other scaffolds. The *ASSISTment Builder* was designed as a tool to create these types of scaffolding pseudo-tutors and is the basis of our research.

The next several sections will describe the ASSISTment system and builder, before we report on 1) the usability of the system by teachers, 2) the time it takes to build content, and 3) the time it takes to tag content with knowledge components. We will conclude with a discussion of the limitations of this work.

The ASSISTments Project Framework

The *ASSISTment Project* is research project by Worcester Polytechnic Institute and Carnegie Mellon University and funded by grants from the Department of Education, the National Science Foundation. The mission of the *ASSISTment Project* is to provide cognitively based assessment of students while tutoring them. This mission is supported by three goals [11]. The first goal is to provide tutoring content to students. The second goal is to provide useful and up-to-date reports on students to teachers. The final goal is to provide the tools to allow teachers to create their own tutoring content.

The *ASSISTment* system provides assessment through student reports to teachers. The reports are updated in real time, even as students are using the system. The system provides different types of reports to teachers based on statistical analysis. Some of the most important reports that we provide are the predicted MCAS score for a student, student effort score, the predicted student performance based on skills mapped to previous questions.

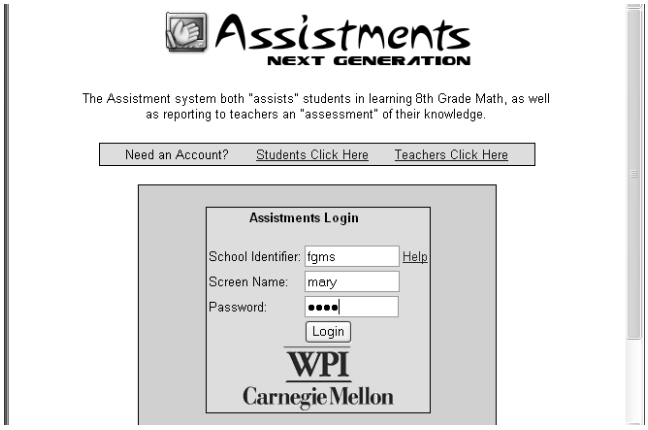
The final goal of the *ASSISTment Project* is to provide teachers with tools to allow them to easily create content for their own classes. The research involving the *ASSISTment Builder* is in support of this final goal. We have created a web based tool that allows teachers to create content online at their own leisure, using whichever platform they have available. We make claims regarding the ease of development for the *ASSISTment Builder* and present data regarding the performance of its users.

Builder Interaction with the CTOP

At the core of the *ASSISTment Project* is the Common Tutor Object Platform (CTOP), a lightweight component framework for creating and deploying all applications in the *ASSISTment Project* [10]. The CTOP was designed with extensibility in mind it consists of a core object model and a data layer [10]. The core object model contains components considered to be universally applicable to ITS software [10]. The *ASSISTment Builder* uses the problem component and its subcomponents, the interface and the behavior. The interface subcomponent is made up of high-level widgets which are interpreted by the runtime application for viewing and interacting with the user [10]. The behavior subcomponent defines the result of an action on the interface; i.e. whether a specific answer corresponds to a transition to a new state in the state graph representing the tutor [10].

The *ASSISTment Builder* allows a user to specify the high level widgets to be used for an interface as well as the properties associated with that interface. It does this by using the Interface component API to provide a form based GUI that exposes the configurable parts of the interface in an easy to modify manner. Similarly, the *ASSISTment Builder* uses the Behavior component API to display the state graph linking states and strategies in form based GUI that is easy to update. Strategies currently supported include message strategies (messages that are displayed when the user enters a specific answer or requests help), and scaffolding questions, which are represented in a nested list structure not dissimilar from a hierarchical tree. The *ASSISTment Builder* also updates the interface and behavior as each one is changed.

Figure 1: The *ASSITment.org web-site*.



The *ASSISTment Builder*

The main goals of the *ASSISTment Builder* are ease of use and accessibility during content creation. The initial prototype of the *ASSISTment Builder* was developed without the CTOP and suffered from maintenance and stability problems. To address these issues our research focused on pseudo-tutors and used the CTOP component framework for ease of development and maintainability. The web was chosen as the delivery medium to make the tool immediately available to users. The only requirement to use the tool is registration on our website; no software needs to be obtained or installed. Our primary users are middle school and high school teachers in the state of Massachusetts who are teaching the curriculum of the Massachusetts Comprehensive Assessment System; thus, the *ASSISTment Builder* was designed with an interface simple enough for users with little or no computer science and cognitive psychology background. The *ASSISTment Builder* also includes other tools to allow teacher themselves to create content and organize it into curriculums and assigned to classes, all of which can be done by the teachers themselves. This provides teachers with a total web-based solution for content management and deployment.

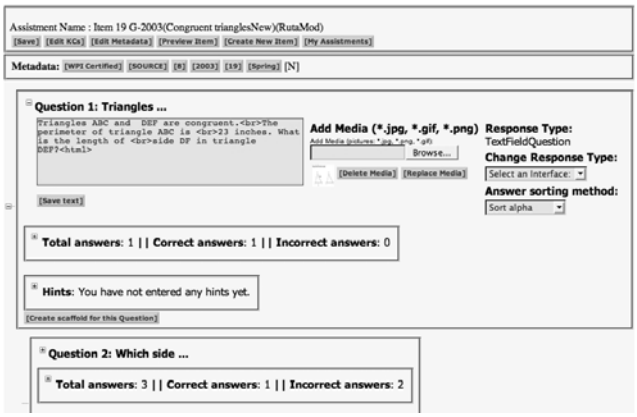


Figure 2: The *ASSISTment Builder*.

ASSISTments

The pseudo-tutors created by the *ASSISTment Builder* are a subset of the tutors possible under the CTOP. These tutors and pseudo-tutors are referred to as *ASSISTments* throughout this paper.

An example of a basic *ASSISTment* is a top-level question that branches into scaffolding problems depending on the student's actions. To simplify content creation there are only five choices of high level widgets for the interface available to content creators: radio-buttons, pull-down menus, checkboxes, text-fields, and algebra text fields. The *ASSISTment Builder* also allows users to add images to a problem's interface. A problem's state graph consists of only two states. The student will remain in the initial state until they answer the problem correctly, or they are programmatically moved forward. Other incorrect student actions will keep them in the initial state, but may be mapped to specific tutoring strategies. These strategies include branching into scaffolding problems, or specific textual and/or visual feedback called buggy messages that address common student errors.

Scaffolding problems are queued immediately after the behavior consumes an interface action that results in a transition to a state containing scaffolds. One or more scaffolding problems can be mapped to a specified user action. In the *ASSISTment Builder* an incorrect answer to the top-level problem or a request for hints on the top-level problem will immediately queue a list of scaffolding problems specified by the content creator. Upon answering a scaffolding problem correctly the student is presented with the next one in the queue until it is empty. When an *ASSISTment* has no more problems in queue it is considered to be finished.

Aside from buggy messages and scaffolds, a problem can also contain hint messages. Hint messages provide insights into methods to solve the given problem. Combining hints, buggy messages, and scaffolds together provides a means to create *ASSISTments* that are simple but can address complex behavior. Content creators can create complex tree structures of problems each with their own specific buggy messages, hints, and possibly sub-scaffolds.

ASSISTment Builder Structure

We constructed the *ASSISTment Builder* as a web application for accessibility and ease of use purposes. A content creator can build, test, and deploy an *ASSISTment* without installing any additional software. It is a simple task to design and test an *ASSISTment* and release it to students. If further changes or editing are needed the *ASSISTment* can be loaded into the *ASSISTment Builder*, modified, and saved; all changes will immediately be available in all curriculums that contain the *ASSISTment*. By making the *ASSISTment Builder* available over the web, new features are instantly made available to users without any software update. The central storage of *ASSISTments* on our servers makes a library of content available to teachers which they can easily combine with their own created content and release to their classes organized in curriculums.

Another goal was to redesign the *ASSISTment Builder* to make use of the CTOP component framework. To do this the Apache Struts Framework was used in conjunction with the CTOP to maintain a strict MVC architecture. By following a strong Model 2 Model View Controller (MVC) design pattern extending the *ASSISTment Builder* is also easy. The CTOP is designed to be extendable with new types of tutors, widgets, and user interfaces. The *ASSISTment Builder* is only concerned with a specific portion of the CTOP, but whenever new widgets or functionality is added all that needs to be done is adding new controllers and views. Sharing code between the *ASSISTment Builder* and CTOP means less code to write as well as swift benefit from improvements to the CTOP. The decoupled nature of the *ASSISTment Builder* also makes it easy to change or update the web forms that are presented to users.

Features

The initial view presented to users of the *ASSISTment Builder* is a top level problem. The view has been redesigned based on user input. At the very top of the screen are several links to help manage *ASSISTments*. The problem is blank and users can enter answers, buggy messages, question text and/or images as well as selecting the interface widget they wish. A content creator can also add hints. However, hints and scaffolds are mutually exclusive in the top level problem, and a user must select either one for the top level problem. Each section in the problem view is collapsible to allow users to conserve screen space.

The question section is the first section that content creators will usually use. This section allows a user to specify a problems question text using html and/or images as well as select the interface widget they wish to use and the ordering method used to sort the answers. There are currently three ways to order answers: random, alphabetic, or numeric. This interface is shown in figures 3 and 4.

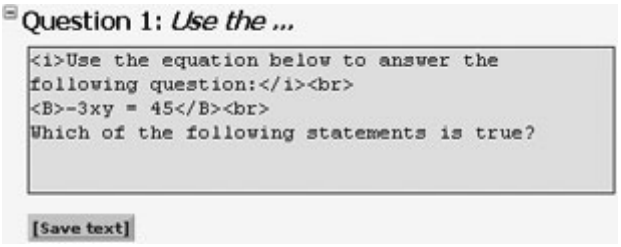


Figure 3: Text from a scaffolding question.

Figure 4: Adding media to a scaffolding question.

The answer section of the problem view allows a content creator to add correct answers and expected incorrect answers. Users can map buggy messages to a specific incorrect answer. Users can also edit answers or toggle their correct or incorrect status. The answer section is shown in figure 5.

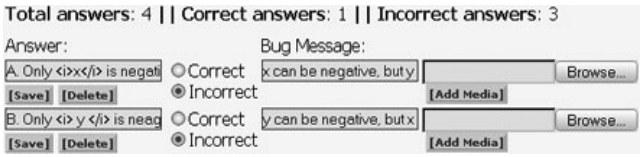


Figure 5: Adding answers to a scaffolding question.

The hint section allows users to enter a series of hints to the applicable problem. Hints can be reordered. This section contains an option to create a bottom out hint for the user that just presents the student with the solution to the problem. This is shown in figure 6.

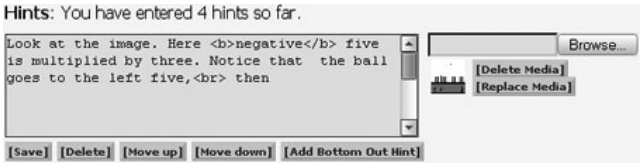


Figure 6: Adding a hint to a scaffolding question.

A typical *ASSISTment* will contain scaffolds and after a user is finished creating the top level problem they will proceed with adding scaffolds. The view for a scaffolding problem is exactly the same as that for the top level problem, only slightly indented to mark it as a scaffold.

Knowledge Component Tagging

The *ASSISTment Builder* supports others applications besides content creation. One of these applications is the mapping of *knowledge components*, which are organized into sets known as *transfer models*. Knowledge components are a means to map certain *skills* to specific problems to specify that a problem involves knowledge of that skill. This mapping between skills and problems allows the reporting system to track student knowledge over time using longitudinal data analysis techniques [3]. In a separate paper accepted to WWW2006, we report on the ability to track the learning of individual skills using a coarse-grained model provided by that state of Massachusetts that classifies each 8th MCAS math item in one of five categories (i.e. knowledge components in our project): Algebra, Measurement, Geometry, Number Sense, and Data Analysis [3].

The current system has more than twenty transfer models available, each with up to three hundred knowledge components. In order to more efficiently manage transfer models, the *ASSISTment Builder* makes use of the preference architecture, allowing users to specify the transfer models they will use. Once those are specified, the user is allowed to browse the knowledge components within each transfer model and to map the ones they select to the problem.

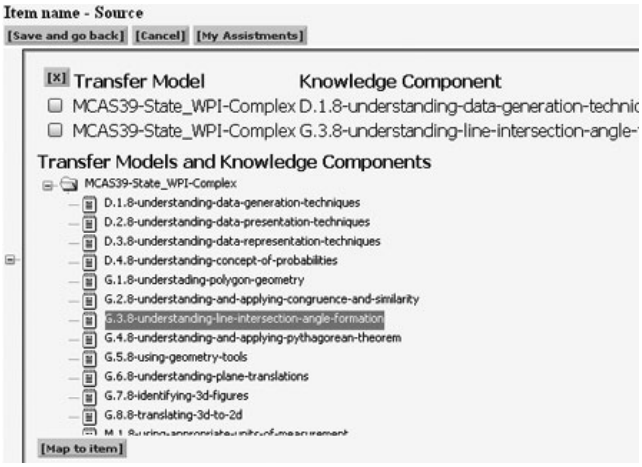


Figure 7: Tagging an *ASSISTment* with skills

Evaluation Methods

We present two types of results. First we investigated the usability of the Builder by non-programmers. Secondly, we investigated the amount of time it takes to build these types of tutors. To capture the time it takes to build these types of tutors, we need to capture the time it takes to create the content (i.e., write scaffolding questions, hint messages and bug messages) as well as the time it takes to tag items with knowledge components that can be used to do intelligent problem selection as well as reporting to teachers (described in the Knowledge Component Tagging Section above.) Because tagging the knowledge components should come before writing the content, we first discuss that. In the ASSISTment system, we built our content based upon a group of 280 released items from the state of Massachusetts Department of Education test. Two subject matter experts spent 6 hours

tagging each of the 280 items with up to three skills. At the end of the 6 hours, the subject matter experts had created 93 skills and tagged all 280 items with, at most, 3 skills per question. It then took another of 12 hours of data entry to put the results in the computer. So the total time spent tagging the items and putting the result in the computer was 24 hours, or about 5 minutes per item. But how much time does it take to create the content? In 2004-2005, we created ASSISTment for these 280 items, and in [11] we report results that showed that these 280 ASSISTment led to real student learning.

Unfortunately, when these 280 items were built the *ASSISTment Builder* was not logging the time it was used, so we asked had to rely on self-reports. Our 4 most prolific authors estimated an average time of between 1 and 2 hours [which means that the 5 minutes to tag an item is not a very significant piece of the time required so to build a tutor]. We wanted to get more accurate results so we engineered the *ASSISTment Builder* log user actions while building ASSISTments, and will report on the latest usage of the *ASSISTment Builder* below. Each log message contained the action logged (e.g. editing a hint, adding an incorrect answer, uploading an image, etc.) the user who performed the action, as well as a timestamp. We logged the creation and editing of various types of *ASSISTments*. Some *ASSISTments* were simply a single MCAS problem entered into the system with no scaffolds, hints, or bug messages. Others were more typical *ASSISTments* that contained multiple scaffolds so report the number of scaffolds. Some were already built *ASSISTments* that were now being modified with different numbers, otherwise known as *morphs*. Given that a significant portion of user time is spent outside of the *ASSISTment Builder* planning out content and creating images we performed a survey with content creators and asked them to estimate how much time they spent building specific items in the logs. They were asked to break down the times according to time spent on each task.

Results

Before reporting out timing results, we pause to report on the results relating the reducing the cost by making it possible for non-programmers to use the tool. A university class at an education school with nine teachers was able to use the *ASSISTment Builder* as part of a University course. These teachers received about 4 hours of training by the first author. Various user-interface bugs were discovered, but at the end of the session, these teachers were creating content. At least two of these teachers are still making content for 6 months after the end course. One of these teachers surprised us by using the builder to make items for a French course. In another University setting, we had two WPI students that were secondary math teachers in local public schools, create content. In one 1.5 hours section, we observed in our lab the teacher creating 3 ASSISTments. In the past 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. No logging was implemented at the time so we don't know how long she spent to build the rest of the 30 ASSISTment. Nevertheless, these anecdotal reports suggest that we have achieved the main goal of making a tool that non-programmers can use to create content.

This then bring up the next major questions, which is how long does it take to create this content, and is it faster that the 200:1 ratio suggested in the introduction of this paper? After we implemented logging by the builder, we obtained data for four authors who created a combined total of 25 ASSISTments that were deemed of sufficient quality, that Prof Heffernan allowed them to be released to students. Each of these users has a WPI student and had created several ASSISTments and was familiar with the system. These users self-reported timing data was also collected. The data is presented in table 1. The columns in the table are identified as follows: **S** is the number of scaffolds in the problem, **I** is the author estimated time spent creating images outside of the *ASSISTment Builder*, **P** is author estimated time spent planning the ASSISTment outside of the *ASSISTment Builder*, **B** is the time the author estimated **time** inside the *ASSISTment Builder* to create the item, and **L** is the time spent on the *ASSISTment Builder* according to the computer log records.

It can be seen from the table most users also spend a non-trivial amount of time outside of the *ASSISTment Builder* creating images and planning the structure of the ASSISTment. If we count only the time in the builder, they spend only about 20 minutes to build an item, but if we add on the self-reported planning and image creation time, we get an average time of about 1 hour to build an item. This is inline with self-reports from the authors that build the content for 280 ASSISTments reporting in [11]. To find the average time an ASSISTment provided content for, we looked at the 600+ students that used the ASSISTment System reported on in [11]. We found that an ASSISTment provided an average of 2 minutes of instruction. The ration of 60 minutes to build an ASSISTment to provide 2 minutes of content results in a ratio of 30:1 which compares very favorably to the 200:1 ration reporting in [1,9].

Table 1: Time spent on 25 *ASSISTments*

User	*ASSISTment*	S	I	P	B	L
C	1	5	3	10	30	60
A	2	3	3	0	45	18
A	3	5	3	0	25	19
C	4	3	3	0	30	33
A	5	4	3	0	35	37
A	6	3	3	60	10	17
A	7	3	3	0	45	14
A	8	4	3	0	30	36
A	9	3	3	60	10	7

A	10	3	3	0	25	17
A	11	4	3	60	10	16
A	12	3	3	60	10	8
B	13	3	40	5	15	17
B	14	3	40	20	10	7
B	15	3	0	7	5	27
B	16	3	50	15	15	13
B	17	3	30	10	10	11
B	18	6	150	40	30	25
B	19	4	60	15	10	14
B	20	5	40	15	10	6
B	21	4	60	20	15	10
D	22	11	0	5	50	40
D	23	1	0	10	10	15
D	24	3	0	10	40	30
D	25	8	0	10	30	20
Avg.		4	21	17	22	21

Conclusions

To discuss the limitation of our methods, we do not know if these ASSISTments it produces are as effective at increasing student learning as intelligent tutoring produced the more traditional approach. Our timing estimates could have been better with more complete computer logging data but given that it appears that using the builder is maybe a third of the average time it takes to build an item, we will still be left with accounting for the time outside of the tools. Another limitation to our approach is that will hundreds of small ASSISTments, we now have imposed upon ourselves more organizational overhead to be able to keep track of all these ASSISTments, and that additional time is not well accounted for in these analyses, but we think it's probably small.

This paper focused on reducing the costs to build intelligent tutoring systems. In this paper we describe our web-based system that we have used to create intelligent tutors that have been shown in lead to real learning [11]. We reported evidence to suggest that it took only about 5 minutes to tag ASSISTments with the needed knowledge-components, and only another 60 minutes or so to create the rest of the tutor. Using the average of 2 minutes of student use per ASSISTment gives us a very favorable speed up compared to the 200:1 ratio from the literature [1,9]. We also presented anecdotal data that normal teachers, not just rule-based AI programmers could create these tutors, thus "Opening the door to non-programmers'.

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, M., Heffernan, N.T, Koedinger, K.R., (2006) Addressing the Testing Challenge with a Web-Based E-Assessment System that Tutors as it Assesses, WWW2006, Edinburgh, Scotland.

[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] Macasek M.A., Heffernan, N.T., Towards Enabling Collaboration in Intelligent Tutoring Systems, Submitted to ICLS2006, Indiana, USA (2006).

[9] 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.

[10] Nuzzo-Jones., G. Macasek M.A., Walonoski, J., Rasmussen K. P., Heffernan, N.T., Common Tutor Object Platform, an e-Learning Software Development Strategy, Submitted to WWW2006, Edinburgh, Scotland (2006).

[11] 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.*