

# Development and evaluation of a lesson authoring tool for AutoTutor

Suresh C. Susarla

*Department of Psychology, University of Memphis*  
ssusarla@memphis.edu

Amy B. Adcock

*College of Education, University of Memphis*  
aadcock@memphis.edu

Richard N. Van Eck

*College of Education, University of Memphis*  
rvaneck@memphis.edu

Kristen N. Moreno

*Department of Psychology, University of Memphis*  
kmoreno@memphis.edu

Art Graesser

*Department of Psychology, University of Memphis*  
a-graesser@memphis.edu

**Abstract:** This paper describes the process of developing an Electronic Performance Support System (EPSS) for AutoTutor 3D. The new architecture of AutoTutor 3D has four models: Domain model, Student model, Tutor model and Interface model. To date, the complexity of authoring the scripts used by AutoTutor has presented a significant challenge. Creation of a tool to simplify this process gives us the ability to disseminate AutoTutor across many different domains. The tool was created using a rapid prototyping approach and incorporates real world case based scenarios based on actual teacher experience with the tool, and a point-and-query help system. This tool and the model for its design may inform the development of similar EPSSs in the future.

## 1. Introduction

AutoTutor is a computerized tutor developed by the Tutoring Research Group (TRG) at the Institute for Intelligent Systems at the University of Memphis. AutoTutor serves as a learning scaffold that assists students in expressing verbal content through discourse processing acts such as pumps, hints, prompts, etc. It simulates the discourse patterns and pedagogical strategies of a human tutor [1]. Currently AutoTutor has working versions in two different subject domains: computer literacy and physics. While many of the technical challenges to implementing agent systems like AutoTutor have been solved, it remains a significant challenge to generate the content which is used to deliver the instruction. This content is represented in AutoTutor via the curriculum scripts. Generating these scripts has

until now required not only domain knowledge of the content to be taught, but also knowledge of the architecture of AutoTutor. For example, script authors not only had to construct instructional materials but also had to include any necessary codes required for the system. If systems like AutoTutor are to be disseminated widely across environments and domains, it is necessary to find ways to support developers of content in the process of generating curriculum scripts. This paper describes the developmental process of an automated system to create domain general curriculum scripts for use in the next version of AutoTutor (AutoTutor 3D).

## **2. Architecture of AutoTutor 3D**

The current version of AutoTutor 3D differs significantly from previous versions in architecture, although the basic modules and functionality remain the same. AutoTutor 3D utilizes a hub and spoke infrastructure similar to the galaxy communicator infrastructure developed by DARPA [2]. A central hub acts as messenger to the connected modules (see Figure 1).

There are five modules connected to the central hub 1) Client module 2) Speech Act Classifier (SAC) module 3) Assessments module 4) Dialog module and 5) Log module. In addition to these modules, there are four supporting utilities: the Latent Semantic Analysis, Parser, Question Answering, and the Curriculum Script Utilities.

Modules and utilities differ from each other in that while modules interact with each other via the hub, utilities provide things needed for different modules during that communication process. Utilities are available to all the modules, but not all modules use all the utilities. For instance, the LSA utility is primarily used by the SAC module and the Assessments module.

### *3. Communication between modules*

There is a state object that is a common tablet that every module will write on. The information that is written in the state object will be shared among the modules to use when needed. The Hub controls this flow. The process is triggered by the Client module. After initializing a tutoring session, the client module will write the student's response to the state object, and this is then passed over to the SAC. The SAC then parses and classifies the input and puts the parse, question content, and classification into the state table. This is then passed to the Assessments module. The Assessment module calculates the new information, repeated information and average contribution. For local assessments it calculates the verbose length and expectation coverage. These global variables are then put into the state table and passed over to the hub. The dialog module then collects this state object and calculates the next topic, subtopic from the assessments. It also keeps track of the dialogue history and feedback data. Finally, it puts the next dialog move into the state object. At the same time log modules collect the state object and send the copy of the object to the log database. After all the modules have made their contribution for a student turn, the client module sees the state table and continues with the appropriate dialog move (from the state table).

AutoTutor uses this modular approach to take in and assess input from the student to create a tutoring session tailored to each student's individual needs. The architecture of AutoTutor 3D is primarily static; the only element that needs to be altered to extend the usability of the system is the curriculum script utility. This utility uses scripts written by the instructor to deliver pedagogically appropriate information to the learner. These scripts

must adhere to a pre-defined structure and syntax in order for the curriculum script utility to communicate with the different modules during the tutoring session. This structure and syntax has been driven by the needs of AutoTutor rather than the needs of individual script authors. As a result, these scripts remain difficult for those unfamiliar with AutoTutor to develop.

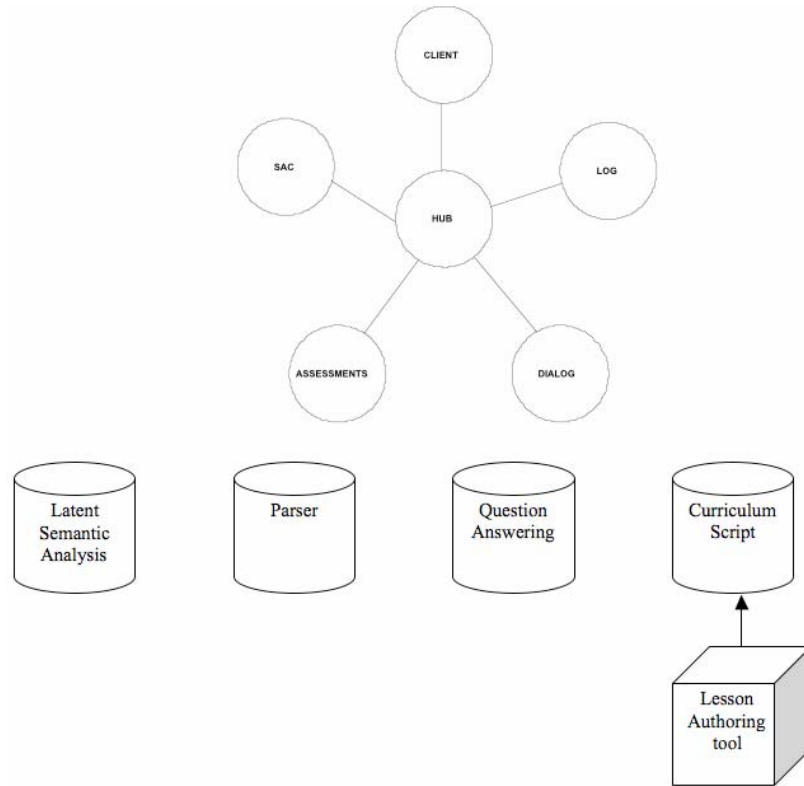


Figure 1. Hub and Spoke modular infrastructure in AutoTutor 3D.

Scripts are composed of thirteen main components. These components (e.g., the question, summary, hints, prompts, and assertions) are used by AutoTutor to deliver content and serve as the basis of judgment of the corresponding student responses. For example, the problem first presented to the student is referred to as the question. There is also an ideal answer, which is what AutoTutor is hoping the student will create in response to the problem. It is unlikely that the student will generate this answer immediately, however, so AutoTutor (via the script) looks for different pieces of the ideal answer (expectations and concepts) and delivers a series of hints, prompts, and assertions for each of these main ideas, depending on student contributions and responses.

These scripts have traditionally been written as text documents, with each component wrapped in syntactic tags that allow AutoTutor to identify the different elements of the script. For instance, the question might be identified using the tag `\question-1`, the ideal answer identified with the tag `\ideal-1`, and the hints, prompts, and assertions identified with `phint-1`, `pprompt-1`, and `pelab-1`, respectively. An example of a complete script is included in Appendix A. AutoTutor 3D now stores the components of these scripts in a database rather than in a text file, but each must still be created and stored correctly. Until now, this has required familiarity with the AutoTutor architecture and script syntax.

This paper will describe the development and validation of a tool for the automated creation of scripts to be used by the curriculum script utility in AutoTutor.

#### 4. Statement of the problem

Because of the improved functionality associated with the new modular architecture, it is now feasible to distribute AutoTutor to a wider audience. This distribution will allow educators to create a customized tutoring interaction suitable for deep level questions in any domain, and for any learner. But if tutoring systems like AutoTutor are to be successful, they must not only be accessible to everyone, they must also be easy to use. While the majority of AutoTutor is automatic and takes place in the background, the curriculum scripts must still be generated by subject matter experts (SMEs), most of whom it can be assumed will have little knowledge (and little desire to acquire more) of AutoTutor and the script syntax. Further, as evidenced by the current iteration of the AutoTutor architecture, it is possible that new functionality and improvements will require changes in the nature of these scripts (e.g., moving from text-based scripts to a database). Even if SMEs were to learn the structure and syntax for scripts today, they may have to learn new formats the next time around.

What is needed is a tool that, on the front end, speaks the language that SMEs are likely to understand (pedagogy and instructional strategies) and, on the back end, the language that the architecture of the system requires. In this sense, we need a translation tool to help SMEs communicate with AutoTutor (or whatever system we are working with). Not only does this make it possible for more people to develop curriculum scripts, but also ensure that when and if things change, the tool developers need only modify the back end of the tool, and the SME never sees any change.

Intelligent tutoring system authoring tools have been in existence for some time (e.g., Murray, 1998 [3]; Macias & Castells, 2001 [4]; Toole & Heift, 2002 [5]). These tools commonly take advantage of a fully developed expert module and provide maximum flexibility and choice for the script designer (Murray, 1999 [6]). This scenario is not unique to the needs of pedagogical agent systems. The corporate world has long made use of tools like this to help people make decisions, judgments, and diagnoses that would otherwise be difficult or impossible to make. These tools are called, collectively, Electronic Performance Support Systems (EPSSs) (Gery 1991 [7]; Raybould 1990 [8]).

EPSSs most often appear as coaches or wizards that ask the user a series of questions in a language they understand, and use their responses to generate decisions (business) or diagnoses (health). It is possible to use the same approach to help novices develop products (in this case, curriculum scripts), that they would not otherwise be able to generate. Because we know what the curriculum scripts should look like, we can design an EPSS that speaks to SMEs in their own language and use their responses to generate the scripts in the language AutoTutor (or whatever tool) expects and understands.

The weak link in this process lies in our ability to develop a tool that SMEs in all domains can interact with successfully, both in terms of ease of use and accuracy of the resultant script. While we understand the requirements of the system, knowing what makes sense to the SME is not so easily determined without significant input from different SMEs. A good tool will use the language, examples, and approach that is most familiar to the SME, and will strike a good balance between the amount of help (coaching) that is presented automatically vs. under the SMEs control. EPSSs incorporate what we traditionally think of as 'help' into the tool itself. But if the tool is to be used by experts and novices alike, the *amount* of help the SME requires up front (i.e., that is built in) will vary. A good EPSS should present just enough help for an expert user, but provide access to additional help for more novice users. The key lies in determining where to draw this line. One way to do this lies in the use of rapid prototyping.

Rapid prototyping is widely used in engineering and in the development of software. In this model, iterative prototypes of the final product are developed, beginning with low

fidelity prototypes (e.g., mock-up models, text-based outlines and descriptions of functionality) and progressing to the final product. Along the way, user feedback is incorporated in a controlled, systematic evaluation process. This prevents major (i.e., expensive) changes late in the development cycle.

Rapid prototyping has also been proposed for use in developing instructional products (Tripp & Bichelmeyer, 1990 [9]). In addition to the advantages mentioned above, rapid prototyping of instructional products can be useful when the designers do not have the domain expertise needed to develop the tool up front. Because an EPSS of the nature we have described here could be seen to straddle the worlds of both instruction and software, this model was adapted for the development of the ASAT. Creating low fidelity prototypes and putting them before SMEs who represent different aspects of the target audience allowed us to refine the content and find a good balance between the help that is embedded in the tool and that which is under the SMEs control. It also allowed us to capture useful information (examples, analogies, etc.) which could then be incorporated into the help systems we included in the tool. In the ASAT, there are two basic types of help under the SMEs control (i.e., not automatically presented as part of the coaching, interview technique): A case study of a hypothetical script author and A series of contextual help questions linked to their corresponding questions, called point & query (Graesser et al, 1992 [10]). Terminology definitions are also provided through hot-linked text that brings up definitions throughout the tool.

The process began with the creation of paper-based scripts representing the content of the proposed authoring system, including examples and directions for interacting with the interface. Figure 2 shows an excerpt of the paper-based scripts used in the tool. These were delivered as paper-based documents that were read one page at a time by a SME with no teaching experience. Think aloud protocol and interview were used to collect data, which was then used to revise the content and to generate the help systems content.

The modified scripts were then integrated into the EPSS using Macromedia *Authorware*. An additional one-to-one evaluation was then conducted with an expert high school teacher with 20 years of teaching experience. Data from this second stage were used not only to revise the content, but also to generate additional content for the case study and built-in point-and-query help system. The tool was modified again based on this information, and a version was created for additional testing and evaluation with approximately 20 SMEs with varying teaching experience.

- Now that you have generated a hint and a prompt for this sentence, the system needs to know what to say to the student if they still don't get it. This is called an assertion. A good assertion is simply the sentence restated in a more conversational style.
- For example, one expectation for the pumpkin problem we have been using as an example is "The runner and the pumpkin are moving with constant horizontal velocity." Our assertion for this (after the student has not gotten the hint and the prompt correct), might be "The man and the pumpkin have the same horizontal velocity before and after the pumpkin is released."
- Type your assertion in the box below.

Figure 2. Excerpt from paper based script.

## 5. Integrated help systems

### 6. Case-based help

The case-based help system is essentially a case study replicating the process that a teacher would go through to create a curriculum script using the tool. The scenario was created through an analysis of think aloud protocols with actual teachers during the evaluation process. Problems and solutions with the terminology, interface, or concepts were used to generate the case study components, which were then incorporated into an overall composite scenario. This scenario is accessible at any time during the authoring process.

Ms. Smith now needs to identify the important words in the outlined statements from the previous section. She thinks of the words she might give as a vocabulary list if she were teaching the entire class. These are the important words the system needs to make sure the student has the correct content knowledge. The scripting tool gives her the opportunity to review the statements and pick out important words.

Figure 3. Excerpt from the case study.

### 7. Point and Query

The Point and Query (P&Q) (Graesser et al, 1992 [10]) is a list of question-answer units that can be accessible from any part of the tool. These are context sensitive Frequently Asked Questions (FAQ's). In the tool the P&Q list of questions is available through a help button. When the user clicks the button, a list of questions linked to the page they are on appears. Users can click on one of the question and get answers to that specific one. Figure 4 shows an excerpt from the P&Q system.

Q1: What are the important words?  
A1: The important words are the words included in the ideal answer points that are directly related to the content. They are very specific in terms of the subject area.

Q2: How do I identify the important words?  
A2: Important words are equivalent to a vocabulary list given at the beginning of a lesson.

Q3: How many important words are there for each main point?  
A3: That is dependent on how specific the main points are. There could be one important word or all the words in the main point could be important.  
Don't include conjunctions, articles, etc.

Figure 4. Excerpt from the P&Q.

## 8. Glossary

The glossary is designed to provide precise definitions for terminology in the script authoring process. In the authoring tool, certain terms are hyperlinked to a window that gives the definition of the term.

## 9. Current and future plans

### 10. Evaluation

Version 1 of the tool is set for a series of one-to-one evaluations during June of 2003. Twenty teachers will generate simple scripts on the electoral process using the tool. Responses from these evaluations will be used to refine the language and content of the tool and to generate additional content for the help systems. After revisions from each of these evaluations are completed, the tool will be used by several SMEs with different experience and from different domains to verify that the tool will be suitable for anyone who wants to develop a script for use with AutoTutor.

### 11. Functionality

In order for this, or any similar tool to be truly effective, it must allow for refinement of the scripts based on teacher and student feedback. During this and all future uses of the tool in script generation, data from the application of those scripts within the AutoTutor architecture will be tracked and presented to the script author. In particular, the system will track common answers and concepts made by students during the tutoring dialog. Based on criteria such as frequency, the more common responses will be presented to the script author during future sessions for their possible inclusion in the script.

Ultimately, the goal is to automate as many of these processes as possible. Future versions may include automatic hint and prompt generation, and alternative teaching or instructional strategies. While this tool is designed primarily for the creation of scripts to be used with AutoTutor, it could easily be adapted for use with any tutoring system that makes use of hints and prompts. Additionally, if any tutoring system is to be successful (i.e., widely used), it must be accessible and simple enough for anyone to use. This tool, and the approach used during its development, may serve as a good model for future developers.

## References

- [1] Graesser, A.C., Wiemer-Hastings, K., Wiemer-Hastings, P., Kreuz, R., & TRG (1999). AutoTutor: A simulation of a human tutor. *Journal of Cognitive Systems Research*, 1, 35-51.
- [2] DARPA Communicator (1998). Retrieved April 2003 from the World Wide Web: <http://fofoca.mitre.org/>
- [3] Murray, T. (1998). Authoring knowledge based tutors: Tools for content, instructional strategy, student model, and interface design. *Journal of the Learning Sciences*, 7(1), 5-64.
- [4] Macias, J.A. & Castells, P. (2001). An authoring tool for building adaptive learning guidance systems on the web. Unpublished manuscript.
- [5] Toole, J. & Heift, T. (2002). The Tutor Assistant: An authoring system for a web-based intelligent language tutor. *Computer Assisted Language Learning*, 15(4), 373-386.
- [6] Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10, 98-129.
- [7] Gery, G. (1991). *Electronic performance support systems: How and why to remake the workplace through the strategic application of technology*. Boston, MA: Weingarten Publications.
- [8] Raybould, B. (1990). Solving human performance problems with computers. *Performance & Instruction*, 29(11), 4-14.
- [9] Tripp, S., & Bichelmeyer, B. (1990). Rapid prototyping: An alternative instructional design strategy. *Educational Technology Research & Development*, 38(1), 31-44.
- [10] Graesser, A.C., Langston, M.C., & Lang, K.L., (1992) Designing educational software around questioning *Journal of Artificial Intelligence in Education* 3, 235-241

## Acknowledgements

This research was supported by grants from the National Science Foundation (REC 0106965) and the Department of Defense Multidisciplinary University Research Initiative (MURI) administered by the Office of Naval Research under grant N00014-00-1-0600. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of ONR or NSF.

# Appendix A

## Sample Curriculum Script

-----note-----  
This script has been shortened for space considerations. One example of each main component is given. Notes such as this indicate where content has been removed.

-----note-----  
\ topic Pumpkin  
\topic\_abbrev HW  
\topic\_phase Early  
\topic\_normal\_time 1800

-----  
\didactic\_content-1 \easy

\info-1  
\question-1 <SpeakStyle PosNeutral>Suppose a runner is running in a straight line at constant speed, <pause 150> and the runner throws a pumpkin straight up. <pause 300> Where will the pumpkin land? <pause 300> Explain.

\ideal-1 The runner and the pumpkin are moving with constant horizontal velocity. When the runner throws the pumpkin upward only vertical forces are acting on the pumpkin. Because only vertical forces are acting on the pumpkin, there is no horizontal acceleration. The initial horizontal velocity of the pumpkin, which is the same as the runner, will not change. The pumpkin will travel up and down vertically and move at the same constant horizontal velocity as the runner, and as a result the pumpkin will land back in the runner's hands.

\concept-1 constant velocity  
\concept-1 same velocity  
\concept-2 constant velocity  
\concept-2 constant speed  
\concept-2 zero force

-----note-----  
Content shortened here for space considerations  
-----note-----

\concept-6 velocity  
\concept-6 velocities

\pgood-1-1 The pumpkin has the same horizontal velocity as the runner before it is released.

\pelab-1-1 The man and the pumpkin have the same horizontal velocity before and after the pumpkin is released.

\phint-1-1-1 <SpeakStyle PosNeutral>What can you say about the horizontal velocity of the runner and the pumpkin?  
\phintc-1-1-1 The horizontal velocities are the same.

-----note-----  
Content shortened here for space considerations  
-----note-----

\pprompt-1-1-1 <speakingStyle PosNeutral>The velocity of the pumpkin and the runner \emp\ are <pause 200> \pit=140\ the? \Rst\<clip Proclivity\_Slient><clip gaze\*>  
\ppromptc-1-1-1 The same.

-----note-----  
Content shortened here for space considerations  
-----note-----

\good-1 The horizontal velocities of the runner and the pumpkin are the same before and after the pumpkin is thrown.  
\good-1 The velocities of the runner and the pumpkin horizontally constant.  
\good-1 The horizontal velocities of the pumpkin and the runner are the same.

-----note-----  
Content shortened here for space considerations  
-----note-----

\bad-1-1 The horizontal velocities of the runner and the pumpkin are different.  
\bbad-1-1 The horizontal velocity of the runner is different from the horizontal velocity of the pumpkin.  
\splice-1-1 <SpeakStyle PosNeutral>The horizontal velocity of the runner and the pumpkin do not differ. <clip gaze\*>

-----note-----  
Content shortened here for space considerations  
-----note-----

\mconcept-1-1 Because no horizontal forces are acting on the pumpkin, the horizontal velocity will steadily decrease causing the pumpkin to land behind the runner.  
\mcorrect-1-1 no horizontal force is required to maintain a constant horizontal speed.  
\mverq-1-1 If the runner speeds up will the pumpkin land behind the runner's hand a little bit, next to the runner, or in front of the runner?  
\mverqc-1-1 The horizontal speed is not affected because horizontal velocity remains constant if zero horizontal force is applied to the object.  
\mhint-1-1-1 What is the horizontal force applied to the pumpkin after it leaves the runner's hands?  
\mhintc-1-1-1 Zero horizontal force is applied to the pumpkin after it is thrown.  
\mhint-1-1-2 Because there are no horizontal forces acting on the pumpkin to change the horizontal speed, what horizontal path will the pumpkin take in relation to the runner?  
\mhintc-1-1-2 It will be the same as the runner.

-----note-----  
Content shortened here for space considerations  
-----note-----

\summary-1 <SpeakStyle PosNeutral>The horizontal velocity of the pumpkin is the same as the runner. <pause 300> Zero force is needed to keep an object going with constant velocity. After it is thrown, only vertical forces are acting on the pumpkin. The horizontal velocity of the pumpkin is constant and is independent of the vertical forces acting on the pumpkin. <pause 300> Therefore, the pumpkin will land back in the hands of the runner. <pause 300><unload>