Extensible Problem Specific Tutor (xPST):

Easy authoring of intelligent tutoring systems

by

Sateesh Kumar Kodavali

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Computer Science;

Human Computer Interaction

Program of Study Committee:
Vasant Honavar, Co-major Professor
Stephen Gilbert, Co-major Professor
Alexander Stoytchev

Iowa State University

Ames, Iowa

2010

DEDICATION

I would like to dedicate this thesis to Krishna, my beloved friend of all time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This would not have been possible without the support of so many individuals who helped me shape this thesis into it's final form. This thesis also reflects the relationships with many generous and inspiring people I have met and worked with since the beginning of my master's program at Iowa State University. I would like to express my deep gratitude to my program of study committee, Dr. Stephen Gilbert, Dr. Vasant Honavar and Dr. Alexander Stoytchev for being very kind and generous towards me. I would like to specially thank my advisor Dr. Stephen Gilbert for guiding me through all the phases from the beginning to the end of my Master's program, ensuring my academic success, involving me in fruitful research discussions and providing me with financial support.

I would like to thank Dr. Steve Blessing, assistant professor of psychology at the University of Tampa for being instrumental in steering research ideas in the right direction. The development on xPST would not have been possible without the foundational efforts by Steven Ourada, the senior architect of xPST. I would also like to acknowledge the help from Mike Oren, Ross Bohner, Shrenik Devasani, Jay Roltgen, Wutthigrai Boonsuk for helping me with conducting studies and providing advice on various technical issues.

I would like to extend my special thanks to Dr. Kasthurirangan Gopalakrishnan for his pure love and guidance all through out my master's program at Iowa State University.

I am grateful for the friendship of so many of the students and staff of the Human Computer Interaction program, the Virtual Reality Applications Center and the Computer Science Department. I would also like to thank Pamela Shill, Linda Dutton, Karen Koppenhaver, Jean Bessman, and Lynette Sherer for their tireless efforts to make

opportunities possible for me.

My friend Sindhu provided me with unwavering moral support all through out, which helped me grow and concentrate even more. I would also like to thank my parents and friends, who have always encouraged me in any pursuit.

ABSTRACT

An Intelligent Tutoring System (ITS) is an artificially intelligent educational software application that teaches a user skills by giving personalized feedback as the user completes tasks within a problem domain. Despite their popularity, authoring these systems is a labor-intensive process, requiring many different skill sets. A major component of an ITS is the cognitive model. Historically its implementation has required not only cognitive science knowledge, but also programming knowledge as well. To address this challenge, the Extensible Problem Specific Tutor (xPST) was developed for easy authoring of ITSs for existing software and websites. This work develops an xPST authoring tool to simplify the process of xPST authoring by the end user and to help conduct research experiments. It also evaluates the xPST system in terms of the time taken by the users to author successful models. This work also extends xPST framework to enable the creation of generalized tutors in addition to problem specific tutors. To help non-technical military trainers create xPST tutors in game scenarios, this work develops a Torque xPST Driver plugin to enable xPST authoring in Torque 3D game and evaluates authoring in spatial environment scenarios like 3D games using the authoring tool. Finally, this work compares xPST and Cognitive Tutor SDK (another authoring framework) using a fraction addition study and shows that the ratio of training development time to training experience time using xPST is approximately 50% less that that of using Cognitive Tutor SDK. This thesis also shows that there is no significant difference between the "beginner programmer" and "experienced programmer" groups in terms of the time taken to author the tasks using xPST.

CHAPTER 1. INTRODUCTION

"Learning by Doing" has proven to be a successful paradigm for educational training. An Intelligent Tutoring System (ITS) is a computer based training software system having an artificial intelligence (AI) component attached to it. This AI component can keep track of the students progress, give feedback and hints as required and can provide customized training to the student on a task. The key distinguishing factor of these systems comes from the fact that they are "intelligent" unlike the normal computer aided learning systems like audio lectures, presentations etc. Content models (also known as; cognitive models, knowledge bases, expert systems, or simulations) give ITSs depth so that students can "learn by doing" in realistic and meaningful contexts. Models allow for tutoring instruction to be generated in real time. Instructional models allow the computer tutor to more closely approach the benefits of individualized instruction by a competent tutor.

1.1 Anatomy of ITS

ITSs generally comprise of four modules: the interface module, the expert module, the student module and the tutor module. The student interacts with an ITS using the interface module, which is generally in the form of a GUI or a simulation. The expert module contains complete description of the knowledge or behaviors that represent expertise in the subject-matter domain the ITS is teaching. The student module contains description of the knowledge or behaviors of the student including his/her misconceptions

and knowledge gaps. The tutor module is responsible for detecting the knowledge gaps and providing appropriate feedback or hints as required. Figure 1.1 shows a pictorial representation of a general ITS.

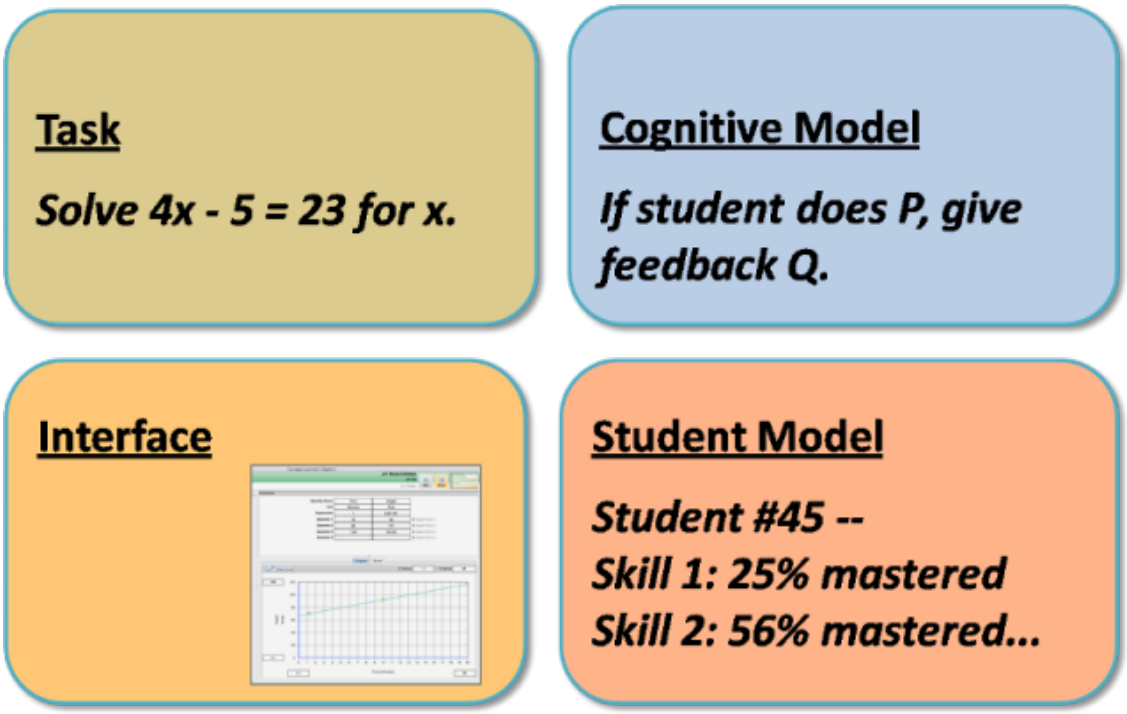


Figure 1.1 A Pictorial representation of an Intelligent Tutoring System.

1.2 Research Questions

The organization of this thesis is centered around answering the following research questions.

1. What learning curve, if any, exists when users use xPST to author tutors on existing web-interfaces?

2. What learning curve, if any, exists when users use xPST to author tutors on existing game interfaces?

3. Is there a difference between xPST and Cognitive Tutor SDK in terms of the training development to training experience time ratio?

4. Is there a significant difference between the "beginner programmer" and "experienced programmer" groups in terms of the time taken to author using xPST?

This thesis also studies how the extensions provided to xPST can support generalizable tutoring and how Torque xPST Driver can support xPST tutoring in game environments.

1.3 Thesis Organization

This chapter provided an overview of Intelligent Tutoring Systems, the various components of an ITS, and the research questions this thesis aims to answer. Chapter 2 goes through a careful literature review framing a background for answering the research questions. Chapter 3 gives the details of the xPST Authoring System, its architecture, the distinction between xPST and CTAT (a different ITS authoring system), the various components of an xPST file and the xPST Authoring Tool. Chapter 4 tries to answer Research Question 1 by giving the details of the xPST authoring study conducted to understand the usability of the xPST system to author tutors on existing web interfaces. Chapter 5 presents the Torque xPST Driver to enable xPST tutoring in game environments, its various components, and the Torque Game Engine, which is used as the simulation environment. It also describes the extensions provided to the xPST framework to support generalizable tutoring apart from problem specific tutoring and the additional functional checktypes added to the framework. Chapter 6 tries to answer Research Question 2 by presenting a study in which participants author models

in game environments using xPST. Chapter 7 attempts to answer Research Questions 3 and 4 by describing the results of the *fraction addition* study which compares xPST with Cognitive Tutor SDK in terms of the training development to training experience time ratio and also compares the time differences between "beginner programmer" and "experienced programmer" groups. Chapter 8 provides a summary of this thesis focusing on my specific individual contribution.

CHAPTER 2. REVIEW OF LITERATURE

The area of previous research that directly relate to this thesis is authoring tools for easy authoring of ITSs. More specifically this thesis explores two research areas. The first is easy authoring of ITSs on existing 2D interfaces. The second is easy authoring of ITSs in complex spatial environments like 3D games. There exists vast amount of literature on the main research topic and an extremely detailed analysis of the field is beyond the scope of this work. However, I will endeavor to summarize the main conclusions, and point out some particularly relevant studies.

The first research area deals with using an existing interface to build ITSs on top of it. Re-using an existing interface with a tutor reduces the time required to develop the tutor and any issues of learning transfer, a concern of past researchers [Corbett et al. (1997)]. If the ITS environment is the same as the non-ITS environment then such issues of transfer largely disappear. The second research area deals with how to easily author ITSs in complex spatial environments like 3D games. For learning to be effective it should be scaffolded or guided [Kirschner et al. (2006)]. During the last few decades, the very nature of teaching in modern universities has changed. Motivating students by setting challenges, goals and problems which are engaging is being seen as a key factor in the learning process [Laurillard (1993)]. Research has shown that students learn better and retain more when they actively engage in the learning process. Tutoring within games or tutoring using games provides these advantages compared to tutoring with traditional software. Pedagogy researchers have shown an increased interest in incorporating gaming principles into teaching and learning [Kirriemuir and

McFarlane (2004)]. Games manage to maintain the user's attention with a background story, high-end graphics and the feeling of immersion within a simulated environment. Games can encourage active learning and motivate participation by giving rewards when students complete a task. An emerging model of games suggests that they excel by providing learners with situated experiences of activities, whereby they develop new ways of thinking, knowing, and being in worlds [Shaffer et al. (2005)]. In recent years, the research in this area is growing fast due to the interest of military to use ITSs in 3D games to carry out military training.

2.1 Evolution of ITSs

The evolution of Intelligent Tutoring Systems is fascinating, originating in the Artificial Intelligence (AI) movement of the late 1950's and early 1960's. In 1958 Skinner reintroduced the idea of "Teaching Machines" [Skinner (1958)] based on the Verbal theory, highlighting the need and advantages of such machines for the future in comparison to the audio-visual aids. The first teaching machine was invented by Pressey in 1934. Teaching machines help the students in being active instead of passive receivers. The paper described and evaluated the working of few such types of machines to teach the students. This paper has been influential in orienting the research community to focus on such type of machines. "Programmed Instruction" [Kay (1968)], a method of presenting new subject matter to students in a graded sequence of controlled steps emerged making use of the teaching machines. Also the education system in United States during this period also provided encouraging support for the development of ITSs. In the 1960's, researchers created a number of Computer Assisted Instructional (CAI) systems that were generative [Uhr (1969)]. These systems were essentially automated flash card systems, designed to present the student with a problem, receive and record the student's response, and tabulate the student's overall performance on the task. These

programs generated sets of problems designed to enhance student performance in skill-based domains, primarily arithmetic and vocabulary recall. By the late 1960's and early 1970's, researchers had moved beyond merely presenting problems to learners while collecting and tabulating their responses, to considering the student a factor in the overall instructional system [Suppes (1967)]. Researchers developed systems that altered the presentation of new materials based on the history of a student's responses. During that period there was crisis in AI since the researchers came to understand that the problems of AI were slightly intractable than the relatively straightforward challenges of building faster computers.

In 1982, Sleeman and Brown reviewed the state of the art in computer aided instruction and first coined the term Intelligent Tutoring Systems (ITSs) to describe these evolving systems and distinguish them from the previous CAI systems [Sleeman and Brown (1982)]. During the 1980's, computer scientists specializing in AI continued to focus on the problems of natural language, student models, and deduction. However, the field also attracted researchers from outside the computer science disciplines. Anderson, who was working in cognitive science, developed the Adaptive Control of Thought (ACT*) theory of cognition [Anderson (1983)]. During mid 1980's ITSs met Educational Psychology forming a discipline combining the work of researchers from AI, cognitive science and education moving towards a more "cognitively oriented form of software engineering."

One of the biggest challenges for developing ITSs is that they are very hard to develop and require expertise in various areas. Ideally they require a team consisting of a software developer, domain experts, a teaching expert and usability engineers. To address this issue a number of authoring tools have been developed to date. In the last few years there has been significant progress in the development of ITS authoring tools and in the understanding of the key issues involved. The development efforts to date represent many diverse approaches, and it is still too early to get a sense for which

approaches will prove to be the most useful. A popular classification of authoring tools by category to better understand the state of art of the ITS authoring tools was given in [Murray (2003)].

At this point it would be useful to recall the research questions this thesis aims to answer. Out of the various ITS authoring tools available, xPST is the one that is discussed at length in this thesis. xPST, aimed to tutor on procedural tasks instead of conceptual tasks, tries to remove the need for expertise in various domains as stated above making it easier for the non-programmers to authors tutors by using the existing interfaces for tutoring.

2.2 Effectiveness of ITSs in training/learning

ITSs have been successfully used to tutor on a variety of domains such as mathematics, geometry, and economics [Aleven et al. (2006),Arruarte et al. (1997) and Koedinger et al. (2004)]. Within the different types of ITSs, model-tracing tutors have been particularly effective [Anderson et al. (1989),Koedinger et al. (1997),Ritter et al. (2007) and VanLehn et al. (2005)]. These tutors contain a cognitive model of the domain that the tutor uses to check most, if not all, student responses. For example, the Andes physics tutor by VanLehn [VanLehn et al. (2005)], and his colleagues contains a cognitive model of how to solve problems within physics. This model is referenced for each step in the problem solving process in order to make sure the student stays on path. The model can also be used to provide hints and other assistance to the student learning the material.

This type of intense interaction and feedback has led to impressive student learning gains. Students using the Andes physics tutors have demonstrated large gains in effect size over paper-and-pencil homework, on both standardized and more conceptual, experimenter-designed tests [VanLehn et al. (2005)]. Results from another model-tracing tutor that instructs students on how to program show that they can master the material

in a third less time [Corbett (2001)]. Carnegie Learning's Cognitive Tutors for math is in use by over 1000 school districts by hundreds of thousands of students. Typical results indicate a 30% improvement on standardized tests such as the SATs, and double that on more targeted instruments, as well as significant learning time reductions [Franklin and Graesser (1996)].

2.3 Parent Systems to xPST

This section describes the two important parent systems to the xPST architecture presented in this work. They are the CTAT (Cognitive Tutor Authoring Tools) and Cognitive Tutor SDK. It also describes the TutorLink architecture which enables Cognitive Tutor SDK to tutor on existing interfaces forming the basis for xPST system.

2.3.1 Cognitive Tutor Authoring Tools (CTAT)

CTAT [Aleven et al. (2006)] stands for Cognitive Tutor Authoring Tools which as of current day is the leading cognitive model authoring tool in the academic world. It is a collection of authoring tools developed at Carnegie Mellon University's Learn Lab. This system has been initially called TDK (Tutor Development Kit). CTAT supports development of two types of tutors, Cognitive Tutors and Example-Tracing Tutors, which represent different trade-offs in terms of ease of authoring and generality. Cognitive Tutors rely on a rule-base cognitive model and have been successful in improving students math proficiency in American high schools [Koedinger et al. (1997)]. They can tutor on many facets of algebra and can tutor on as many examples of a given *type* of problem as you want, e.g., hundreds of y=ax+b problems. Example-Tracing Tutors on the other hand can only tutor on one kind of problem, and sometimes don't even generalize as much as cognitive tutors. They do not require much programming compared to Cognitive Tutors [Koedinger et al. (2004)]. This is because they don't attempt to solve the classic

problem in AI of generalizing from a knowledge structure that's been manually encoded.

The Cognitive Tutor Authoring Tools comprise three separate applications: an external GUI Builder (typically, NetBeans or Adobe Flash), a set of core tools for demonstration-based task analysis and for testing and debugging cognitive models, and an external editor for cognitive models (typically Eclipse).

In CTAT, the authoring of Example-Tracing Tutors and Cognitive Tutors is organized around examples of demonstrated behavior. These examples include alternative strategies for solving problems and errors students are expected to make, and can be recorded conveniently with CTAT's Behavior Recorder tool. They can be used as the basis for Example-Tracing Tutors to provide guidance to students. The examples can also be used as planning cases and semi-automatic test cases for cognitive models, if an author is developing a Cognitive Tutor.

2.3.2 Cognitive Tutor SDK

Cognitive Tutor SDK [Blessing et al. (2006)] is a set of cognitive ITS authoring tools build on an architecture called Tutor Runtime Engine (TRE) [Ritter et al. (2003)], which was inspired by TDK. The main idea behind Cognitive Tutor SDK is to come up with GUI representations that enable the cognitive model designer to do their work without any programming and with a clarity not offered by previous systems. As per Cognitive Tutor SDK design the two main components of a model-tracing ITS cognitive model are the object model and the rule hierarchy. The object model represents the pieces of the domain to be tutored (e.g. table cells in a worksheet, components of a graph, equations to solve and the terms within those equations), and this object model is used by the rules to provide the tutoring to the student. The plan behind the SDK is to apply concepts that have been successful in other aspects of computer applications, such as tree views, hierarchies, and point-and-click interfaces, to the design of model tracing ITSs. Development of such representations so they are usable by non-programmers in

this context is difficult, but critical to lowering the bar in terms of both time and money in the creation of such systems. As noted in [Blessing et al. (2006)] the Cognitive Tutor SDK has been successful in terms of being able to be used by cognitive scientists who are not professional programmers, but it still requires significant computational background and it not easy enough for, say, a high-school teacher to use.

Figure 2.1 shows a screenshot of the Cognitive Tutor SDK. The Type Hierarchy Inspector can be seen in the upper left. The left pane of that window shows the current type hierarchy and the right pane shows information about the currently selected item in the left.

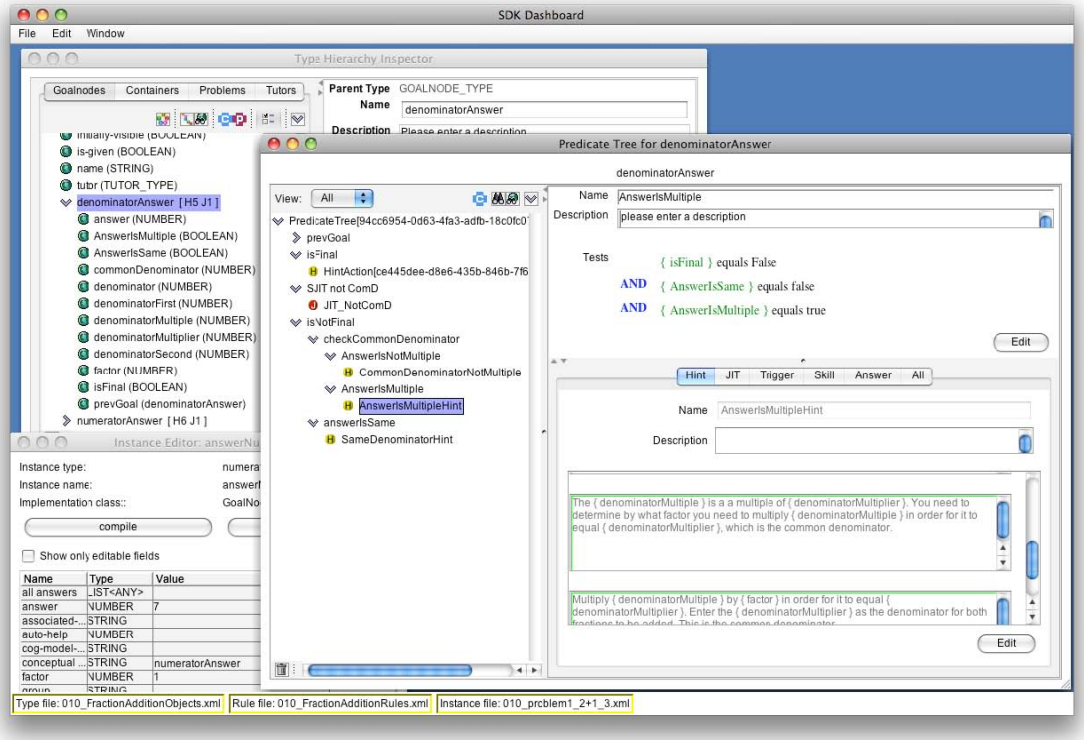


Figure 2.1 Screenshot of the Cognitive Tutor SDK showing the Type Hierarchy Inspector, Predicate Tree and the Instance Editor.

In the view of object model, the requirements for this particular tool are similar to other existing tools, in that the basic functionality is to display and edit objects consisting of attribute/value pairs. However, there are additional requirements for Cognitive Tutors that makes using any off-the-shelf or previously produced software problematic. In particular, pre-defined object types exist that have their own properties and behaviors. For example, there is a goalnode object type (representing a students subgoal in the problem), that has a set of predefined properties, and attached to these goalnode types is a predicate tree inspector. In addition to the object model, the other main piece of a cognitive model specifies the goal-state behaviors, such as right answers, hints, and just-in-time (JIT) messages forming the backbone of a model-tracing tutor. Cognitive Tutor SDK has a non-code based (GUI) representational scheme for the rules using tools like Rule Editor and Tutorscript editor.

It is desirable to enable the construction of model-tracing ITSs around pre-existing software. This would eliminate or at least greatly diminish the time spent doing traditional interface programming that is to develop a custom interface on which learners will be tutored. By allowing authors to create an ITSs for off-the-shelf software, this will lower the bar for creating such systems. One can imagine tutoring not only math or statistics using Microsoft Excel as the interface (or some other spreadsheet), but one could also tutor on Microsoft Excel itself (or any other application requiring training) [Ritter and Koedinger (1996)]. Such a scenario would be a boon to corporate training environments. What is needed is a way for the Tutor Runtime Engine, the tool that uses the cognitive model created by the SDK, to communicate with third-party software (that is, software that the authors creating the ITSs did not program). Even though the interfaces for the current Cognitive Tutors were developed essentially in tandem with their cognitive models, the code for the interfaces was separate from the tutoring code, with the two pieces communicating to each other through a messaging protocol. This separation allows to use Cognitive Tutor SDK described in the previous section, and

also allows for the possibility of third-party applications to be the student interface. TutorLink [Blessing et al. (2007)] came as the solution to this problem. TutorLink acts as a communication link between TRE and the off-the-shelf software. Figure 2.2 shows architecture of TutorLink.

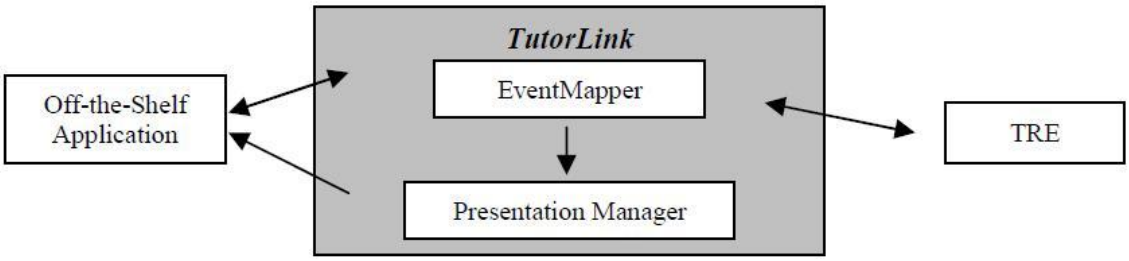


Figure 2.2 TutorLink Architecture.

Once the user interacts with the interface, that interaction needs to be noted by TutorLink and sent to the TRE for the appropriate tutoring action. The part of TutorLink that receives the interaction message from the interface is called the EventMapper. As its name implies, this part takes the incoming message and maps the event into something the TRE can understand (that is, a goalnode within the cognitive model). In the case, where the interface was built using tutorable widgets, this mapping is straightforward. In other cases, there needs to be a more structured mapping table that maps between interface widgets and goalnodes. This mapping has to be supplied by the author. Such a mapping application using AppleEvents has been described [Ritter and Blessing (1998)]. In general terms, the author starts a listening application, then begins to interact with the widgets in the interface in order to identify their names, and then maps those to goalnodes.

2.3.3 Comparison of the Parent Systems with xPST

The comparison between xPST and its parent systems shows a relationship between the "Ease of Use" vs "Power". They seem to be inversely proportional to each other. The main idea behind CTAT is also to enable non-programmers or non-cognitive scientists to author ITSs. The key advantage of CTAT to the existed ITS authoring systems is that it requires no AI programming for example-tracing tutors (for Cognitive Tutors, one need to program rules in JESS). However, it is confined to a specific set of interfaces using CTAT-friendly widget libraries (e.g. certain Java beans or Flash actionscript libraries). It also uses quite complex tools like "GUI Builder," "Behavior Recorder," "Working Memory Editor," "Conflict Tree," etc which demand some level of programming knowledge to understand and use them effectively. The complex GUI interface might seem difficult to interact with for non-cognitive scientists. This way CTAT has still a deep learning curve. But it is very powerful to create generalizable tutors for repetitive tasks (e.g. do 30 math problems of this form). The Cognitive Tutor SDK has similar underlying concepts from CTAT but had used TRE as its underlying engine. The Cognitive Tutor SDK can produce much generalizable tutors than CTAT. It can make a tutor for all algebra problems and thus rules that apply generally can apply to any problem (e.g. "forgot the negative sign" rule can apply to y=ax+b but also to y=sin(x)+cos(x) or any other problem that might be appropriate). Also Tutorscript allows much more power with customizable programming. It also still uses some complex tools like "Rule Editor", "TutorScript Editor" which again has some programming bent. TRE makes certain assumptions that are not helpful when tutoring on software (e.g. it assumes that the tutor author can fully control and track the state of the interface, graying out buttons as needed, etc). And like CTAT, Cognitive Tutor SDK also does not enable authoring of ITSs on existing interfaces. The xPST system has arisen to overcome these problems. xPST based its idea on completely removing such complex GUI's as existed

in its parent systems and provides a simplex text file interface. This way the author breaks the task into a sequence of steps and writes them in a file. This made xPST more usable by non-programmers or non-cognitive scientists to author ITSs with very less effort. It also uses TutorLink architecture to be able to tutor on existing interfaces removing the need to create tutoring specific interfaces. However, xPST is more limited in power. It could only author problem-specific tasks (and later game-based tasks) where repetition is not required. This works well for software-based learning and game based training where training is majorly task specific. Table 2.1 shows a comparison of CTAT, Cognitive Tutor SDK and xPST in view of various factors.

Table 2.1 Comparison of CTAT, Cognitive Tutor SDK and xPST

	CTAT	**Cognitive Tutor SDK**	**xPST**
Ease of use	Low	Medium	High
Power	Medium	High	Low
Enable authoring on existing interfaces	No	No	Yes

Figure 2.3 shows the history or evolution of this research.

In view of these systems and the comparison between them [Table 2.1] and the results [Blessing et al. (2009)] evaluating the programming knowledge needed to use the Cognitive Tutor SDK, we can predict that it will be easier for non-programmers to use xPST if xPST removes some of the complex data representations of the SDK, e.g. predicate rule hierarchies.

The Cognitive Tutor SDK enables the creation of cognitive models that contain abstracted instruction over instances. However, it seemed to be difficult to use in other situations. In two of the tutors built using Cognitive Tutor SDK, for example, the authors created a lot of declarative and procedural representations that ultimately received very little use. For example, [Hategekimana et al. (2008)] created a tutor for Paint.NET, software similar to Adobe Photoshop. One exercise taught users how to resize and scale an image. While one could imagine using this instruction in multiple

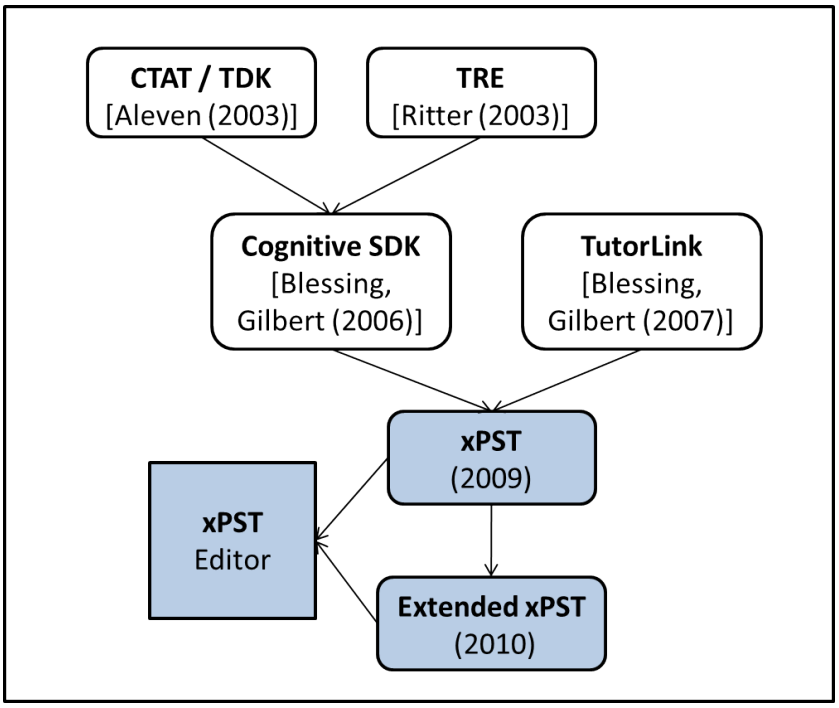


Figure 2.3 Evolution of this work showing the relationship between xPST and its parent systems.

image-manipulation instances, in the actual tutor it was used in only a couple of exercises. The power of having a model-tracing tutor, in which the instruction could be abstracted over multiple instances, was lost. However, the author still spent much time creating the representations that contained the instruction.

Figure 2.4 shows a screenshot of ITS on the top of Paint.NET application showing a JIT message.

Likewise, the tutor for the CAPE web based Authoring Tool used at Vanderbilt University as part of the VaNTH ERC [Roselli et al. (2008)], the authoring process contained similar issues. Ultimately, the tutored instruction centered over a set of eight problems. Much work went into the declarative and procedural structures of the tutor, but their re-use was not nearly as great as what one would see in a Carnegie Learning math tutor.

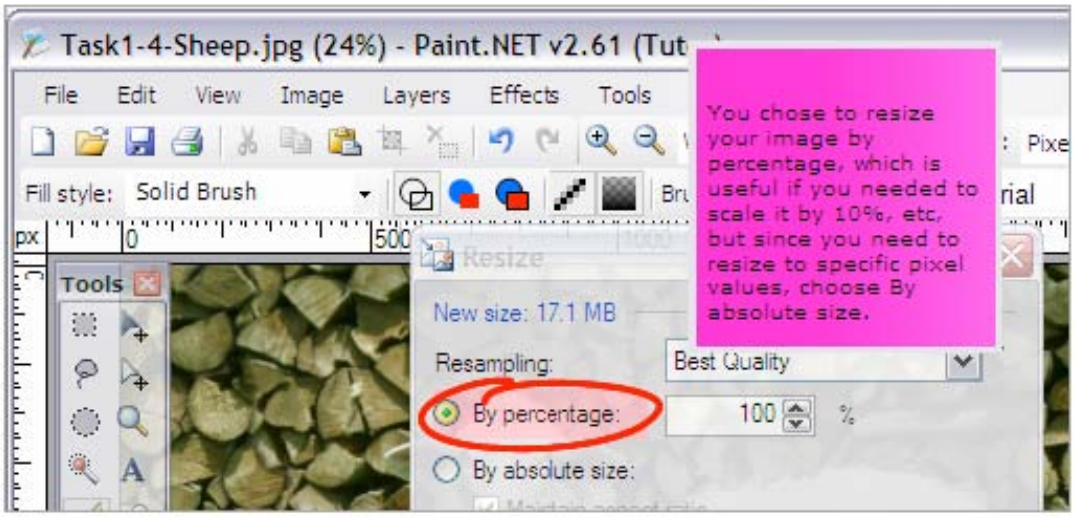


Figure 2.4 ITS on the top of Paint.NET application showing a JIT message.

Ultimately, the effort spent developing those representations seemed disproportionate to their usefulness in the completed model. What was desired for these situations was a more streamlined system where the tutoring could be developed without the need for as much underlying structure typical of model-tracing tutors. The Extensible Problem Specific Tutor (xPST) authoring system was designed to eliminate these problems.

Figure 2.5 shows a screenshot of the ITS prototype for the VANTH Web-based Authoring Tool.

2.4 Emergence of Games in Tutoring

ITS researchers have begun exploring how games and features that are found in games (e.g., embodied agents) can be used in intelligent tutors. For example, McQuiggan, et al. [McQuiggan et al. (2008)] have examined how topics in middle school science

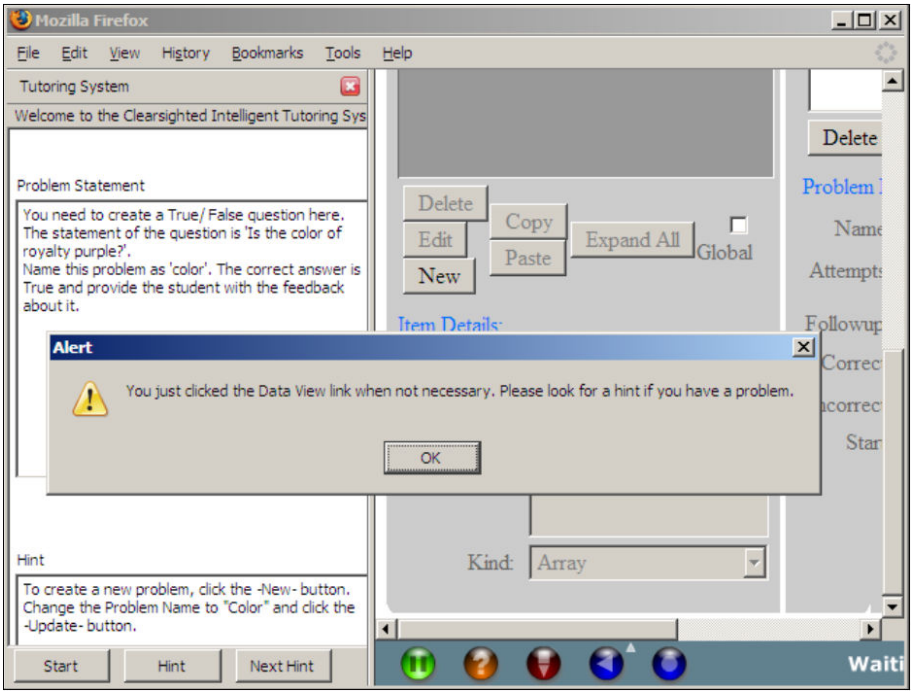


Figure 2.5 A screenshot of the ITS prototype for the VANTH Web-based Authoring Tool. The tutor on the left side panel shows the initial problem statement to address and a hint. A JIT (just-in-time message) gives feedback on top of a partial screenshot of the Web-based Authoring Tool on the right.

could be taught using a tutor built on a commercial 3D-game engine. Students search an island science post to find clues to solve a mystery. While interacting with non-player characters and making observations in the virtual world, students learn scientific principles. [Johnson (2009)] describes a tutor in which users learn cultural issues while interacting in a serious game (a game designed for a primary purpose other than pure entertainment). Gomez-Martin, et al. [Gomez-Martin et al. (2004)] have developed a system called *JV2M* which borrows ideas from games to teach programmers with Java knowledge the internal workings of the Java Virtual Machine. In some knowledge

domains, games may be the only possible means of simulating and practicing real world problems. In the military for instance, simulations have been used for teaching pilots to fly as well as for training of combat scenarios that would otherwise be too deadly or expensive to train in the field [Stottler (2000)]. [Livak et al. (2004)] presents a method to unify the computer generated forces community and the online training community, seemingly disparate areas, by using a single cognitive model to provide both tutoring and computer generated forces capability. While previous ITSs have been most effective for learning conceptual tasks like teaching physics [VanLehn et al. (2005)] and procedural tasks like teaching algebra [Aleven et al. (2006)], but little has been done to explore tutoring on procedural tasks in spatial environment scenarios like 3D games.

For some learning domains, games are a more natural way to learn than traditional classrooms. Crawford (1984) suggests that games are "the most ancient and time-honored vehicle for education. They are the original educational technology, the natural one, having received the seal of approval of natural selection. We don't see mother lions lecturing cubs at the chalkboard; we don't see senior lions writing their memories for posterity. Game-playing is a vital educational function for any creature capable of learning." The optimal learning state is that of being in "flow" [Csikszentmihalyi (1990)]. It refers to a mental state of immersion and clarity. Athletes call it "being in the zone", and the term has made its way into a number of fields including video games research.

It is important to understand the difference between various terms like educational simulations, virtual worlds and serious games as stated in [Aldrich (2009)]. *Educational Simulations* are structured environments, abstracted from some specific real-life activity, with stated levels and goals. They allow participants to practice real-world skills with appropriate feedback but without affecting real processes or people. *Virtual Worlds* are 3D environments where participants from different locations can meet with each other at the same time. These environments can capture and convey enough social cues, such as

body language, interactive props, and the look and feel of "real" surroundings to convince some part of the participant's brains that they are physically in other world. Increasingly important, some virtual worlds also enable participants to build and otherwise change the environment. *Second Life* is a best-known example of a virtual world. *Serious games* are interactive experiences that are easy and fun to engage while building awareness of the real world context. Serious games usually require no coaching outside help and even spread through word of mouth, promoted by people who enjoy playing them.

CHAPTER 3. xPST AUTHORING SYSTEM

Re-using an existing interface with a tutor reduces the time required to develop the tutor and any issues of learning transfer. With past ITSs, researchers have had concerns about whether skills being learned in the ITS will transfer to the non-ITS environment [Corbett et al. (1997)]. If the ITS environment is the same as the non-ITS environment (e.g., learning how to edit images in the context of Adobe Photoshop itself, rather than alternating between a tutorial video and the software), then such issues of learning largely disappear.

3.1 xPST vs. CTAT

xPST [Blessing et al. (2009)] stands for Extensible Problem Specific Tutor and CTAT stands for Cognitive Tutor Authoring Tools. xPST[1] was developed with software training as the target application. Traditional software training often uses videos based on screen-recordings (e.g., from Adobe Captivate or TechSmith Camtasia Studio), but this passive technique to learning has been shown to be less effective than an ITS [Hategekimana et al. (2008)]. In form, xPST is similar to the CTAT [Aleven et al. (2006)]. CTAT and xPST both allow the author to quickly create a model for a particular problem instance by writing hints and other tutoring aspects while the author manipulates the interface. xPST differs from CTAT in that CTAT requires the authors to use either a Java or Flash interface built using specific CTAT widgets and xPST does not.

[1]The xPST Authoring System is open source and it is available on Google code repository at http://code.google.com/p/xpst/.

3.2 xPST Architecture

The xPST architecture (see Figure 3.1) is an instantiation of the architecture of plug-in tutor agents described in [Ritter and Koedinger (1996)]. The xPST file, which contains information that allows for instruction akin to a model-tracing tutor, includes information describing the objects within the learning domain and rules that determine which feedback the student will receive at any given moment. Every interface element of the application for which one needs learning instruction is mapped to an object and has one or more rules associated with it. The rules contain the instructional feedback. The Plugin or Listener (Firefox Plugin for Web interfaces) eavesdrops on user actions in the third-party software and sends them to the xPST Tutoring Engine, which checks them with the xPST file. Feedback is mapped back to the client UI control and displayed on the interface, e.g., via coachmark-style graphics [Hewes et al. (1994)]. Note that while ITSs for academic topics like math typically require a more complex cognitive model, so that learners can receive high-quality personalized feedback across a large number of similar math problems, software training does not require such repetitive tasks, and the cognitive models are typically simpler and thus problem specific.

The third-party software can be a stand-alone application or a website. If the task is tutoring on a stand-alone application, then the system can listen for user events in three ways by using: 1) widgets that automatically send the needed events (the method used in [Ritter and Koedinger (1996)] with AppleEvents); 2) accessibility hooks built into the software (used frequently by screen readers like JAWS and software like Adobe Captivate); and 3) low-level OS events. xPST enables tutoring on any website viewable in Firefox that can be monitored via the Document Object Model (DOM) or on any stand-alone application in which you can insert a "listener" function to eavesdrop on user events. The xPST Engine runs on its own server or locally and communicates with the other components via TCP/IP, allowing the tutored application and the tutor to

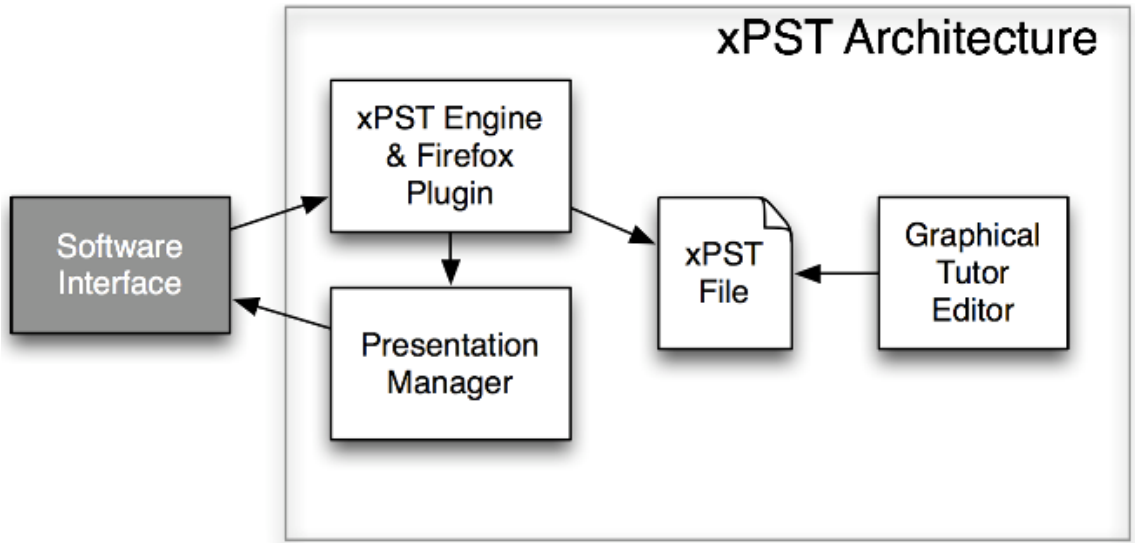


Figure 3.1 The architecture of xPST. The Firefox plugin "eavesdrops" on the software interface or website that needs tutoring. The Presentation Manager gives visual feedback using the software interface. The xPST file provides the feedback and goal structure needed for each task within the tutor. The Graphical Tutor Editor enables teachers to create the xPST file without programming skills.

run on different servers.

3.2.1 Components of the xPST File

The xPST file (a sample is provided in Appendix C) is a text file written using a syntax that is designed to be easy to read and write for an inexperienced cognitive modeler. The file contains three sections: Sequence, Mappings, and Feedback. The sequence identifies a path of steps the user takes through the problem space to achieve the goal specified in the task. The mappings section maps interface identifiers to the steps noted in the sequence that the user takes. Finally, the feedback section provides

hints and error messages for each step within the sequence. Because of this relatively simple syntax, the authoring tool for xPST can be a text editor. An online text editor for creating web-based tutors where authors can immediately jump to their target website and test the current xPST file has been created.

For example, one tutor built teaches AP Biology students in high school how to use the NCBI Bioinformatics tools that are available online. For one task, learning how to search for a DNA sequence using the BLAST tool, the sequence contains steps like Click-Nucleotide-BLAST-Link, then Enter-DNA-Sequence, etc. In sequences, steps can be separated by *then, and, or, and until*. Until is used when there is a set of UI controls that are all submitted to the system at once, typically by a button like Search or Go or Submit or OK in a dialog panel. These conjunctions can be used in any combination and grouping to allow for much flexibility in how the tutoring can progress, much more so than the typical screen-capture movie. The Mappings section for the above steps contains mappings that match the less understandable labels that NCBI's site uses to name that link internally, e.g. "homeBlastn:click" to the more friendly step name in the sequence. Whenever that link is clicked, the xPST tutor passes along the message to that particular tutor step.

The Feedback section specifies the desired answer and any hints or JITs (just-in-time error messages). JIT error syntax allows several variations, e.g. the tutor author can decide whether to allow a mistaken keystroke or click to reach the target software (and presumably lead to unwanted to results) or to block the learner's action so that the error message can appear without allowing the potentially damaging action to take place. Also, JITs can occur based on a variety of tests such as "If learner's input is less than 5, give this JIT." Finally, as in other tutor APIs, xPST supports variable replacement within hints and JITs, so that if an incorrect answer were assigned to the variable "almost", an error message might say: "You chose {almost}, which is close, but its {answer}."

It has been confirmed that the xPST approach can be used to develop real tutors rapidly; one of the most extensive efforts is described in [Roselli et al. (2008)], in which the tutor taught university faculty how to use a complex web-based homework authoring tool. Other smaller efforts include tutoring on the NCBI site (see Figure 3.2) and on Slashdot Journals.

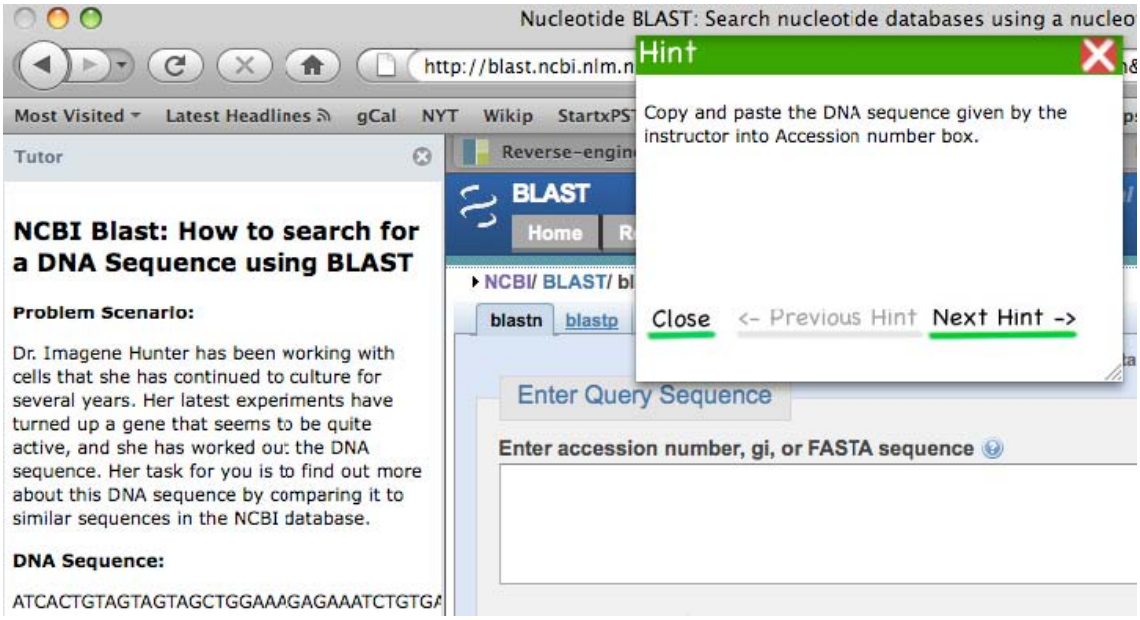


Figure 3.2 Tutor on NCBI WebSite.

3.3 xPST Authoring Tool

The xPST Authoring Tool is an online web-based tool to author xPST files. It is designed to serve two purposes: 1) To provide an easy to use graphical user interface (GUI) to author xPST files without installing any software on the client computer 2) To provide a tool to log the amount of time spent by the author in each of the Mappings,

Feedback and Sequence sections of the xPST file, which helps in conducting research experiments involving xPST authoring. The logs and the xPST files generated by the author are automatically stored on the server of the researcher without any work required from the author.

The xPST Authoring Tool provides an Integrated Development Environment (IDE) for authoring xPST files. It accomplishes all the background work like creating the properties file, the scenario file, the linking html file and appropriately ties them up with the xpst file taking off a bunch of housekeeping work from the author allowing him to focus on the cognitive model alone. It provides the feature of syntax checking and informs the author of any errors as he is editing the file. This feature greatly helps the non-programmers in easing the authoring process as it has been found that most of the errors encountered by the author initially during the xPST authoring process are simple syntax errors. It also provides a starting standard template for the authors and the authors can add their cognitive model into the standard template seamlessly. In-place help is provided for each section of the file. The xPST Authoring Tool also provides auto save functionality similar to Google docs. Figure 3.3 shows the xPST Authoring Tool with its standard template.

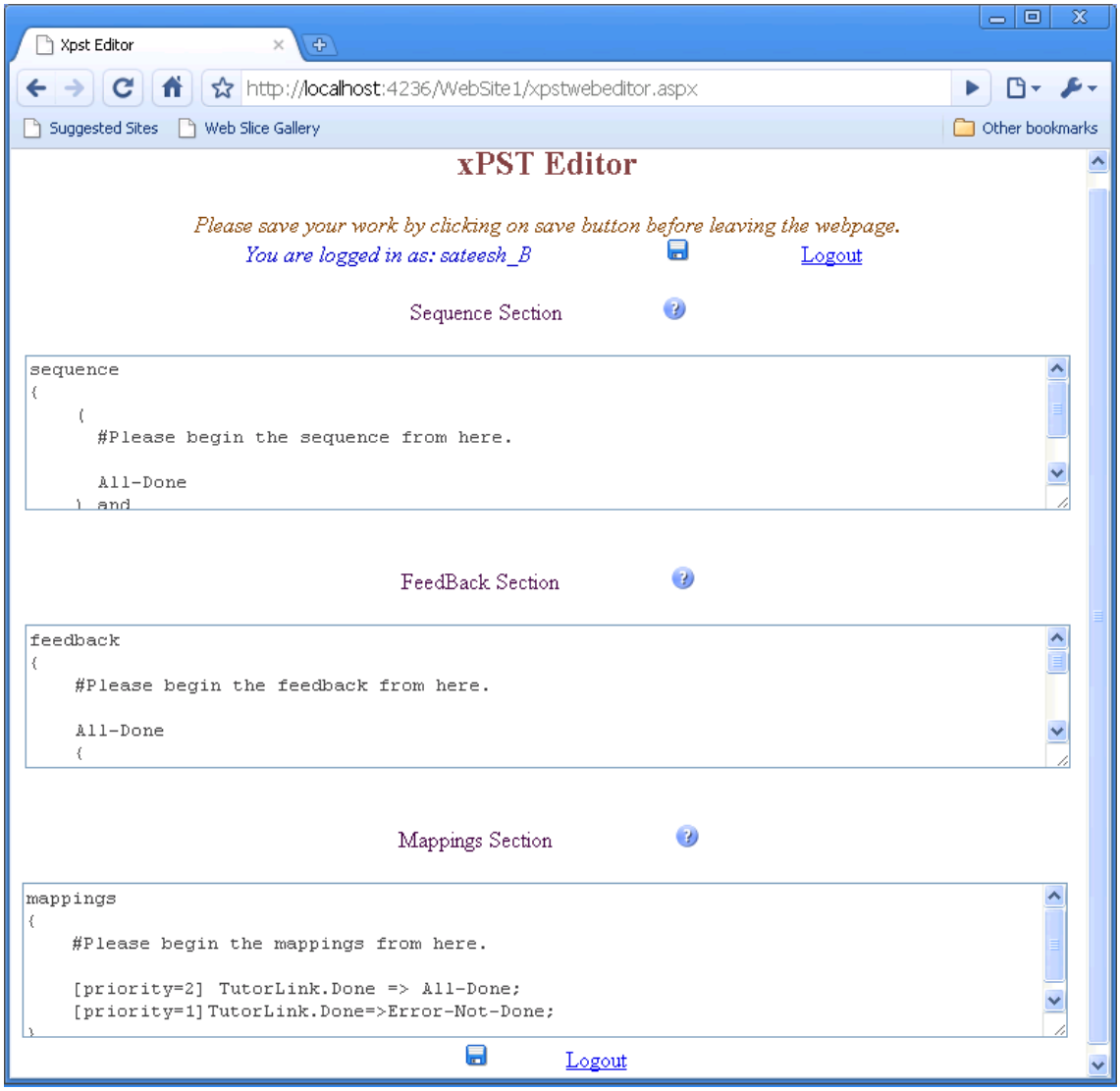


Figure 3.3 xPST Authoring Tool with its standard template.

CHAPTER 4. xPST AUTHORING STUDY

We tested the ability of novice users of xPST to create models using the xPST Authoring Tool. Our approach is based upon a study conducted previously which examined another authoring tool described in Blessing and Gilbert (2008). Our purpose with this study[1] is to ensure that the xPST authoring approach is useful to people with different backgrounds and to examine the learning curve in the time course of learning to author.

4.1 Methods

4.1.1 Participants

We conducted the study during Fall 2008 in an introductory HCI graduate class at Iowa State University. The class had 48 first-year students in the HCI program from a variety of different departments. An extra credit assignment was presented to the class. Students could elect to do the assignment or not, and then they could elect to make their data available for analysis or not. Eighteen students made some attempt at doing the assignment, and 10 agreed to participate in the study.

4.1.2 Materials

The assignment was composed of three main parts: 1) information about intelligent tutoring systems in general; 2) a worked example of creating a model using the xPST

[1]The research described in this chapter has been published in the Proceedings of the 14th International Conference on Artificial Intelligence in Education [Gilbert et al. (2009)].

authoring system; and 3) the assignment itself. The information concerning intelligent tutoring in general was very brief: a web page (http://www.hci.iastate.edu/HCI521/bin/view/Main/CogModelActivity)that contained roughly two pages of text that discussed tutoring at a very high level (e.g., needing to think about a student's goals, what help messages to provide, etc.), and a 5-minute screen movie which demonstrated the VaNTH xPST tutor, designed for a Vanderbilt University web application [Roselli et al. (2008)].

The worked example was much more in-depth, and composed of four parts: 1) a 44-minute screen movie showing someone creating the model while annotating their actions; 2) the resulting commented xPST file; 3) a six-page web document containing the technical information portrayed in the video; and 4) a website where the student could try the example portrayed in the movie for him or herself. This worked example involved a web page (http://xpst.vrac.iastate.edu/WebxPST/employmentappmenu.html) one might use to apply for a job, which asked for such information as name, education history, and major. The page used a variety of different entry widgets in order to illustrate the various features of xPST-based models. Finally, the assignment consisted of designing an xPST tutor for three different tasks (as described in Table 4.1 below), all centered on the issue of searching a particular library database (the ACM Portal, http://portal.acm.org/dl.cfm?coll=portal&dl=ACM).

Each task specified different parameters that needed to be used in the search and ended with the user sorting the found results in a particular manner or showing the reference in ACM reference notation. For example, Task A asked the user to find a full-text article about intelligent tutors (but not a math one) by someone from MIT. The student had to find the paper with the most downloads that met those criteria. These tasks were chosen because they are real-world tasks in which students not experienced with such database searches might benefit from having a tutor. All three tasks were similar in scope. They were not intended to vary in difficulty, but that was not calibrated

Table 4.1 Three tasks to be done along with their descriptions

Task Name	**Task Description**
Task A: Search by University	Use the Advanced Search to find articles from authors at MIT on intelligent tutoring that are not about math and where you have full text available. Sort the list of hits to see which one has had the most downloads over the past 6 weeks. How many articles are there, and what's the name of the paper with the most downloads?
Task B: Search Proceedings	Do a search to find articles from the CHI conference proceedings (but not the extended abstracts) on multitouch that have full-text available in ACM Portal. How many total CHI articles on multitouch are there? Which one has the most citations? Note that the official name of the CHI conference is the Conference on Human Factors in Computing Systems.
Task C: Find Reference	You can remember an important article by Ritter and one other person from 1996. Find the article and the exact ACM Reference Citation format for it.

separately. Task A and Task C has 6 goalnodes (refer Chapter 2 for definition). Task B has 7 goalnodes and requires some good thought from the participant to sequence them correctly. For more elaborate description of the tasks see "Your deliverables section of the study" in Appendix A.

In order to complete the assignment, the students had available to them the on-line xPST Authoring Tool as described above. The xPST Authoring Tool allowed us to track progress in creating the models, including the ability to know how long and how often students worked on each of the three major sections of an xPST file (mappings, sequence and feedback). Lastly, participants filled out an exit questionnaire that asked for demographic information and the participants reflections on using the xPST Authoring Tool.

4.1.3 Procedures

The class instructor briefly presented the assignment to the students. If they elected to do the assignment, then the students had three weeks to complete it. All materials were available via a web page, and students were free to work on the assignment at any time or place until its due date. The materials suggested that it will take 5 - 10 hours to do the assignment.

4.2 Results

We divide the results into three parts: 1) qualitative and quantitative measurements of the models produced; 2) timing data concerning model creation; and 3) analysis of the end-of-task questionnaire.

4.2.1 Model Analysis

As stated above, 10 students gave permission to analyze their models. As a group, these 10 participants produced 26 models: one participant produced only one task model, two participants produced two task models, and the other seven participants completed all three task models. When it is sensible we will use all data in the analyses, but at other times we will use data only from the seven participants that completed all the assignments. We will make clear which set of data we are using.

Blessing and Gilbert (2008) classified the models produced by the participants in the SDK Authoring Tool into one of five categories (other researchers have used similar scales [Martin et al. (2007)]). We did the same thing with these models produced with the xPST Authoring Tool. The scores were based on actually running the model. The scores are reflective of the models behavior. Due to the direct mapping between how an xPST model behaves to how the model looks, the scores would be similar if not identical if based on the model's structure. As can be seen in Table 4.2, all the

participants performed quite adequately at producing models, with only one model not being sufficient, and majority of participants (18 out of 26, 69%) received a score of either 4 or 5.

Table 4.2 How the cognitive models were scored

Score	Description	Models
5	A model that produces behaviors close to an ideal model, in terms of hints and just-in-time messages	6
4	A very good model that is beyond just being sufficient	12
3	A sufficient model where the student can complete the task	7
2	Model provides hints, but does not provide enough guidance for a novice	1
1	Model runs but produces nonsensical help	0

The three participants who did not complete all three tasks created the poorer cognitive models. Of the five models produced by these three people, the model that scored a 2 came from that group. One model scored a 4, but the other three models scored 3s. Of the seven people who completed all three tasks, one person scored all 3s, another scored one 3 and two 4s, and the remaining participants and models scored a mixture of 4s and 5s.

4.2.2 Timing Data

The xPST Authoring Tool kept track of how long participants spent working on the three parts of an xPST model and how many edits were made to each part. Table 4.3 displays the average time participants spent on various parts of the models, split by the three different tasks. Table 4.4 displays number of editing sessions. Note that these data represent the participants that completed all 3 tasks and for which we have valid times (7 participants completed all 3 tasks).

The standard deviation of the timing data (in hours) for Tasks A, B and C was 1.82,

Table 4.3 Time to complete model actions within a task (times in hours, with percent of total in parentheses). 21 models considered

	Task A	**Task B**	**Task C**
Sequence	1.30 (35.0%)	1.02 (40.3%)	0.58 (33.5%)
Mappings	1.06 (28.6%)	0.58 (22.9%)	0.40 (23.1%)
Feedback	1.35 (36.4%)	0.93 (36.8%)	0.75 (43.4%)
Total	3.71 (100%)	2.53 (100%)	1.73 (100%)

Table 4.4 Editing sessions to complete model actions within a task (percent of total in parentheses). 21 models considered

	Task A	**Task B**	**Task C**
Sequence	37.67 (24.5%)	16.83 (26.0%)	12.67 (19.5%)
Mappings	46.33 (30.1%)	22.67 (35.1%)	24.17 (37.3%)
Feedback	70.00 (45.5%)	25.17 (38.9%)	28.00 (43.2%)
Total	154.00 (100%)	64.67 (100%)	64.84 (100%)

1.70 and 1.00 respectively. Figure 4.1 shows the histogram of timing data by tasks and participants for the 7 participants who completed all the three tasks. We can observe the learning curve as these participants move from task to task. Participants were not required to work on the tasks in order, but all did so, as evident from their log files. As noted above, the 3 tasks are similar in scope. Participants got more proficient in using the tool and creating these models. The histogram shows that all the participants have the expected learning curve.

Figure 4.2 shows what the average participant was doing while working through the task model. We analyzed the data for each participant by dividing progress in writing the model into quintiles. Within each quintile we calculated what percentage of the time was spent on sequence actions, mapping actions, and feedback actions. This particular graph is based on all available data (26 task models), though all graphs are very similar to one another regardless of how the data are sliced. Much of the sequencing work was done first, and the mapping and feedback work was then done in tandem. This was the way the worked example movie showed during training went through the process of

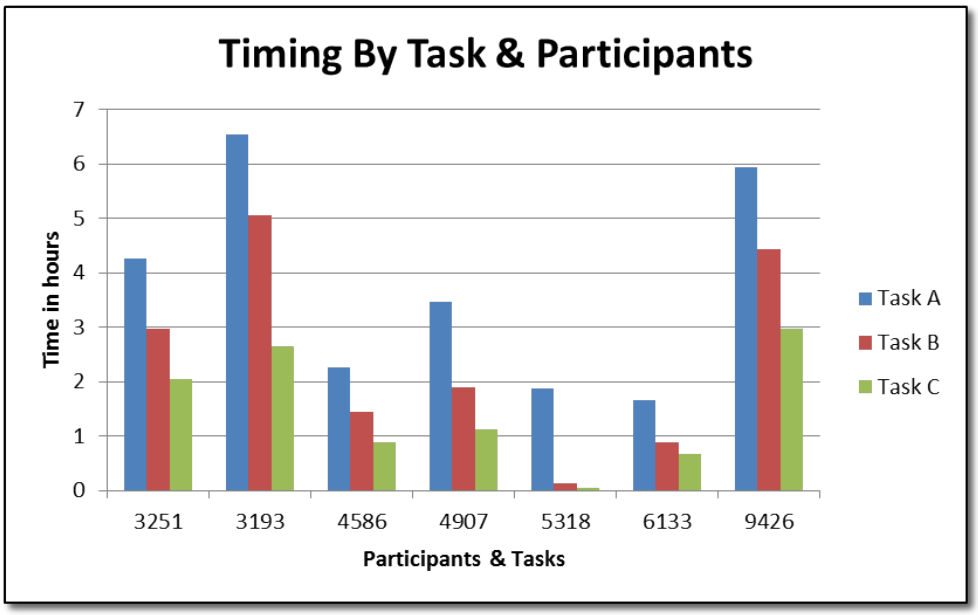


Figure 4.1 Histogram showing Timing By Task and Participants.

model creation, and it appears that most of the participants adopted that path.

4.2.3 Exit Questionnaire Data

Participants completed a short questionnaire after they were done with the assignment (see Figure A.1 in Appendix A). The questionnaire asked for demographic information (sex, home department, number of undergraduate programming courses and self-rated "techie" score) and contained four questions asking them to reflect on their experiences.

Of the 10 participants, all but one were male, and five of them came from engineering departments (two were undecided, one computer science, and one veterinary medicine major). All but two had taken more than five computer science courses as an undergraduate; the remaining two had taken only one course. They self-rated themselves as

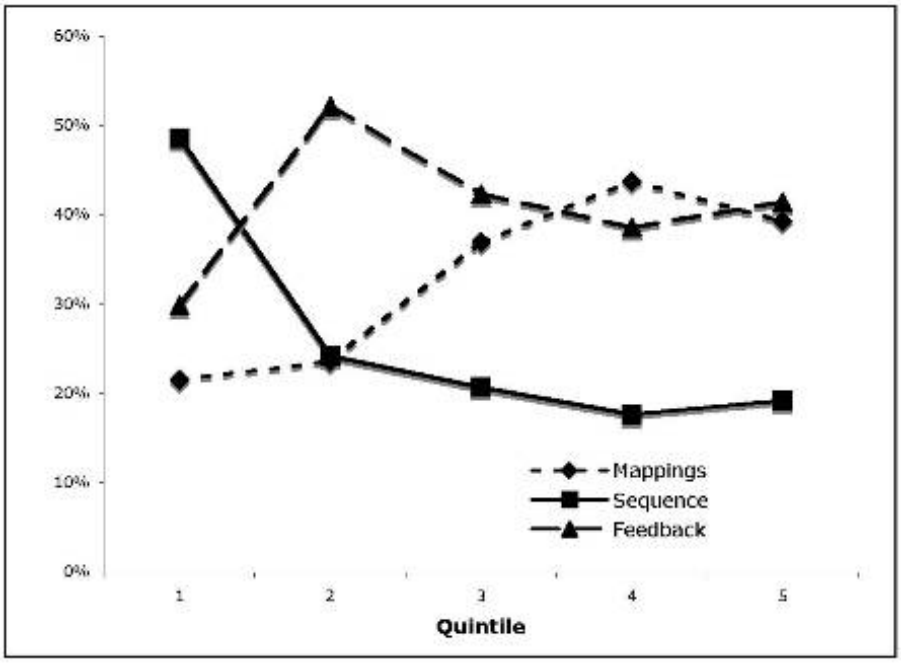


Figure 4.2 Activity Graph of xPST Authoring Study - I

at least moderately technologically sophisticated, with two 5s, three 4s, and the rest 3s. Given the small sample size and somewhat homogeneous group, it is hard to make conclusions concerning who may have produced better models based on demographic information. A previous study [Blessing and Gilbert (2008)] found that the number of computer courses was a good predictor of cognitive model success. Examining the two people who had only taken one programming course apiece, one of them wrote uninspired models that scored a 3, but the other wrote among the best models (two 5s and a 4). One of the two people who had taken more than 10 courses also wrote uninspired models (all 3s). The other person who had taken more than 10 courses wrote excellent models (a 5 and two 4s). In our small sample we have examples of all possible combinations of programming courses and model rating.

In the free response questions (there were four total, asking them to reflect on this particular experience, the task in general, what was challenging, and other uses of xPST), five participants explicitly said that they liked the approach that xPST took with regards to providing training on existing websites. Three people were neutral, one person thought the process was too complex, and another made the point that if the website were designed correctly, no training would be needed. Two people pointed out that the usefulness of such an approach increased with the site's complexity. Four users took the current implementation to task, citing weaknesses such as the training materials, the limited functionality of the editor, and the inability to circle or highlight items in the interface as the system provides feedback (not implemented within xPST for this particular study).

4.3 Discussion

There are two main items we would like to note from the study. First, people were able to use the xPST Authoring Tool. Everyone who attempted to start the task produced at least one working model. The instruction provided was minimal, about an hour, but with that instruction a mix of people created a model of a particular task. The second item to notice is the the presence of a learning curve as the participants moved from task to task. There was a spread in terms of time to create the models, but all users reduced their times upon successive models. By the third model most participants were able to complete their model in just over 1.5 hours. After another model or two, that time would be close to 1 hour (the developers of xPST could probably produce such models in 30-45 minutes). That 1 hour would provide around 10 minutes worth of instruction. So this study tries to answer our first research question that there exists a learning curve when users use xPST to author tutors on existing web-interfaces, but we cannot prove its significance since the study does not have enough number of participants. A

better study with more participants is conducted later, which is described in Chapter 7. The small up-front cost of training coupled with a small time in producing the training could make the xPST approach attractive to those who want to provide model-tracing feedback to existing interfaces.

Figure 4.3 shows a screenshot of a tutor in action on the ACM portal showing a Hint message for the user. Figure 4.4 shows a screenshot of a tutor in action on the ACM portal showing a JIT message since the user did not enter the exact phrase *intelligent tutoring* in the query box.

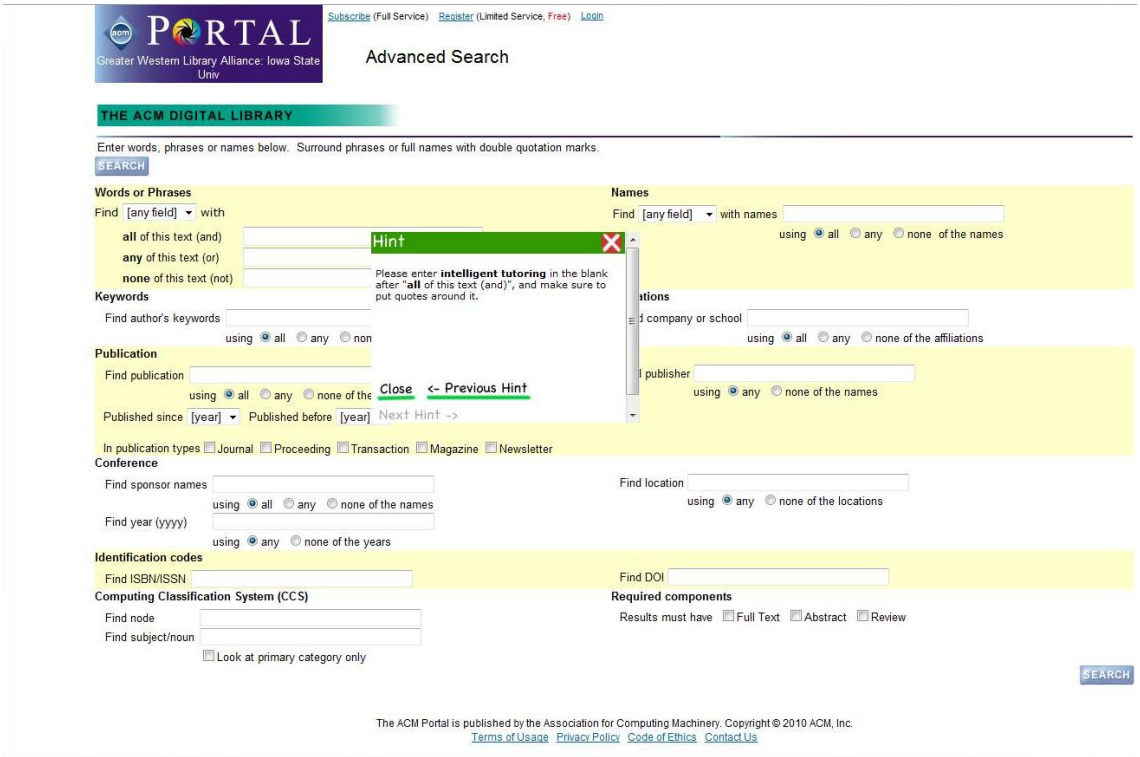


Figure 4.3 A screenshot of a tutor in action on the ACM portal showing a Hint message for the user.

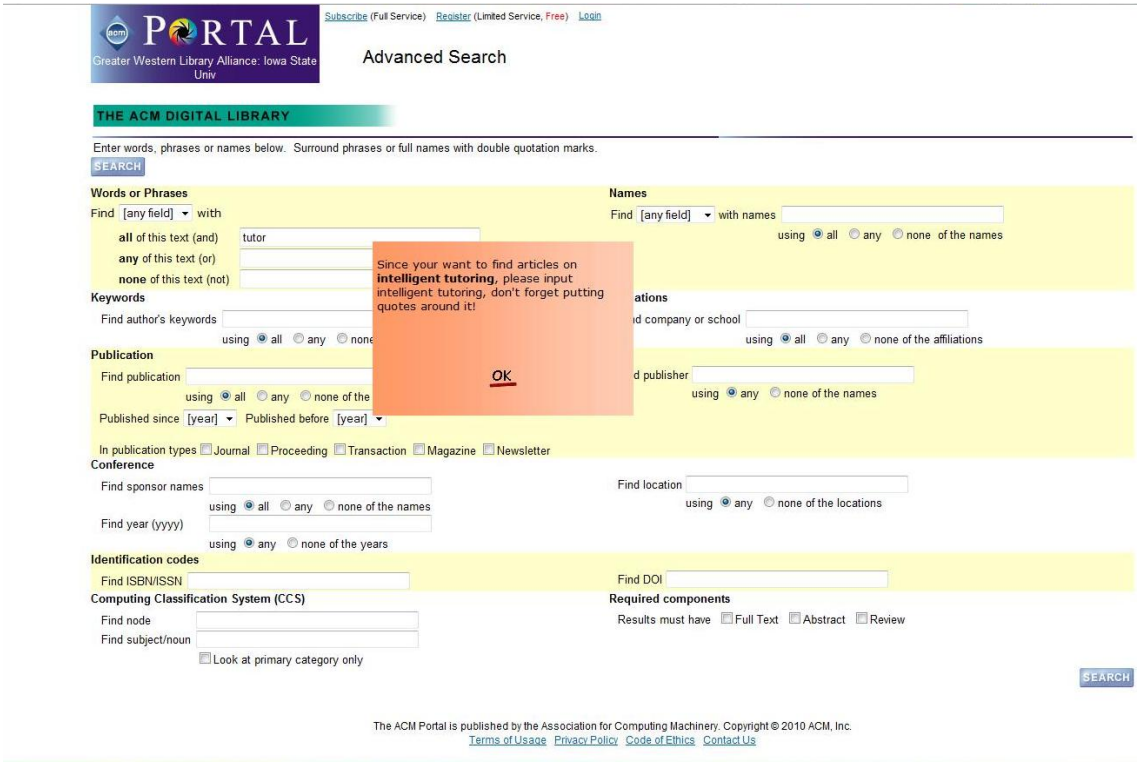


Figure 4.4 A screenshot of a tutor in action on the ACM portal showing a JIT message since the user did not enter the exact phrase *intelligent tutoring* in the query box.

CHAPTER 5. TORQUE xPST DRIVER AND EXTENSIONS TO xPST

This chapter describes the Torque xPST driver and the extensions provided to the xPST framework. Torque xPST driver acts as a bridge between the Torque game engine and the xPST engine to enable xPST tutoring in games. The extensions provided to the xPST framework allow for the creation of generalizable tutors and to give proactive feedback. Additional functional checktypes are also added to the xPST framework to enable tutoring on math problems.

5.1 Torque xPST Driver

Torque xPST driver serves the job of the Firefox plugin (see Chapter 3) when tutoring on web based interfaces. The driver eavesdrops on the events happening in the game and sends them to the xPST engine. It also receives the appropriate feedback from the xPST engine and sends it to the game.

We have participated in a series of direct interactions with the military trainers at the Camp Dodge-Iowa National Guard and took their feedback on authoring tutors in games. Based on that, we understood that the authoring of tutors in games seemed hard for the non-technical military trainers due to the inherent domain complexity of the 3D environment and the lack of programming knowledge. In the hope of giving them a tool to author problem specific military tutors we developed this driver so that they can use the easy to use xPST framework to develop tutors in games.

5.1.1 Torque Game Engine Advanced (TGEA) and TorqueScript

Before describing the Torque xPST driver, it is worth mentioning the simulation engine we have used. We have used Torque Game Engine Advanced (TGEA) as our simulation engine. It is a commercial off-the-shelf game engine from GarageGames. It provides various core functionalities required for game development like the rendering engine, physics engine, 3D graphs, collision detection etc. Instead of starting from scratch, using an off-the-shelf game engine drastically reduces the game development time and helps the author concentrate more on the tutoring task.

TGEA supports scripting using TorqueScript. TorqueScript is similar in syntax to JavaScript and allows the developer to create modifications (mods) of the existing games. The Torque xPST driver is written completely in TorqueScript.

5.1.2 Components of Torque xPST Driver

The Torque xPST driver comprises of two components: the "Listener module" and the "Presentation module". The Listener module listens to the events happening in the game and sends them to the xPST engine over the network. It also receives the feedback from the xPST engine and sends it to the Presentation module. The Presentation module is responsible for presenting the received feedback to the user. The xPST Torque driver communicates with the xPST engine and the game engine using a message format called "Dormin message." It is essentially a long string in a specific format containing the various attributes informing the current state of the task, the message to be communicated and the action verb which determines what to do with the message.

Figure 5.1 shows the pictorial representation of the Torque xPST Driver.

The xPST Torque driver also provides an interface to communicate between various entities in the game and to register location based events in the game.

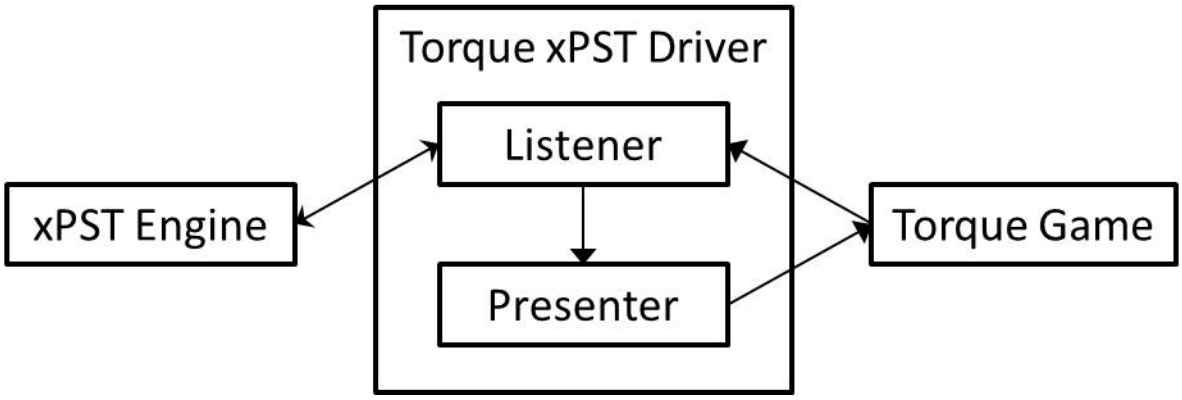


Figure 5.1 Torque xPST Driver Architecture.

5.1.2.1 Communication Events

Unlike traditional GUI software or websites, many of the tasks in games require the player to be able to communicate with other entities in the game. A special goalnode *starttalk* has been introduced to provide feedback related to invoking the communication with other entities. The driver also provides an interface (see Figure 5.2) that allows the player to choose the entity with which to communicate and the message that will be communicated. This approach facilitates tutoring on the protocol of communication and on the type of the communication messages that should be used, a common training task in the military, where communication is frequently highly-structured.

For example, if the player is supposed to choose the *Evacuate* command for the task, but he chooses a different command, say, the Fire command, a JIT can be launched saying "*You used Fire command on this occupant. That's not something you need to do right now.*" This multiple-choice user interface for tutoring on communication is designed to tutor on communication protocol and procedure: what to say and when and

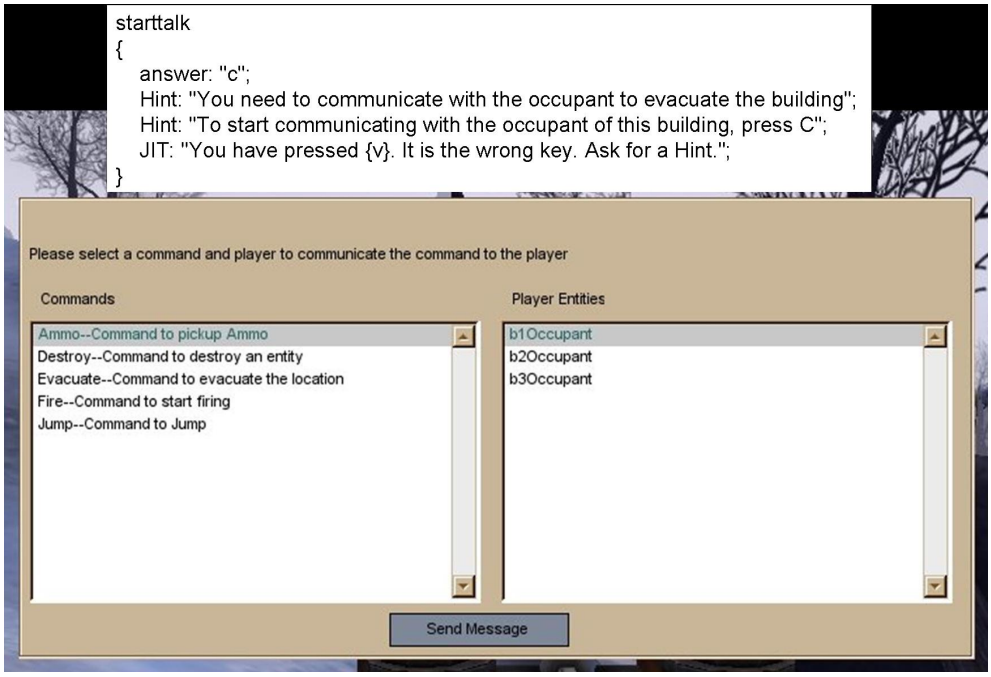


Figure 5.2 Torque xPST driver interface for communication events.

how to say it. Future research will evaluate its effectiveness within military scenarios.

5.1.2.2 Location Events

Location events facilitate tutoring on the navigational aspects of the player's performance. Unlike the traditional GUI software or websites, almost every task in a game requires the player to move within the virtual environment. To register a location based event the author places a trigger at the appropriate location and assigns it an identifier (entity-id). The author can use the *entityid-enter* goalnode to tutor on when the player enters a particular designated location in the game.

For example, the goalnode *b1-enter* is triggered when the player enters building1 (b1). Figure 5.5 shows the *b1-enter* goalnode along with the appropriate feedback given

to the user in the game.

Figure 5.3 shows the comprehensive picture of the xPST architecture along with the Torque xPST Driver.

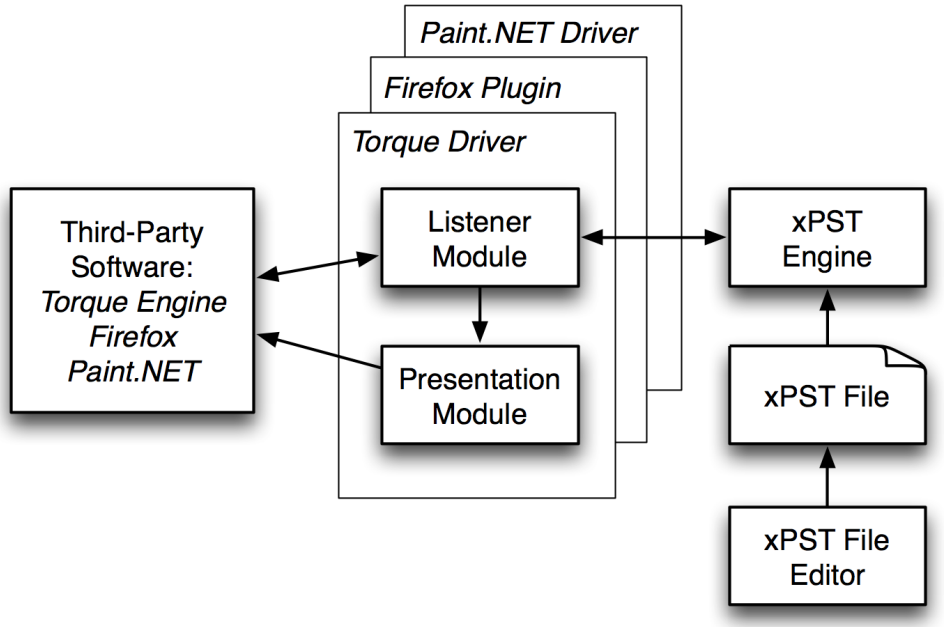


Figure 5.3 The xPST architecture along with the Torque Driver.

The framework of the xPST driver can be leveraged to various other game engines by making the syntactical script changes required to be able to suit with that particular game engine. The Torque driver enables tutoring in 3D games created with Torque Game Engine. As shown in Figure 5.3, the Torque driver is one of the many possible interfaces to xPST and one of the several we have built. The xPST Firefox plugin is an interface which is used to tutor on websites. The Paint.NET Driver is another interface that is used to tutor on Paint.NET, an image editing application. Likewise, many interfaces could be built to tutor with xPST on different existing software applications.

In general, we see that the xPST Torque driver provides a mechanism to author xPST tutors in a complex domain like Torque 3D game. The four important steps required to create a game tutor from scratch include: 1) Create the tutoring scenario, which in effect consists of building the game map. 2) Give unique identifiers to each entity in the game on which you plan to tutor. 3) Make a list of the events corresponding to the entities chosen to tutor on and give appropriate mapping names in the mappings section of the xPST file. The mappings associate game-based events with the goalnodes that need to evaluate those events within the tutor. 4) Complete the feedback and sequence sections of the xPST file, listing the appropriate feedback for each goalnode and the sequence(s), in which the goalnodes may to be accomplished. Step 1 is perhaps the most difficult step, and is required of the scenario author even in the absence of a tutor. Past research [Gilbert et al. (2009)] (see Chapter 4) has demonstrated that novice authors can accomplish steps 2-4 with little training in a simpler software setting, that of using a website to search an online database. It would be of interest to explore whether military trainers could use a modification of those previous tools to overlay an xPST tutor on existing virtual training scenarios effectively. A related study is described in Chapter 6.

5.2 Extensions To The xPST Framework

xPST language can model tutor for any task which contain a set of steps to be performed in a given order and each step has an answer which denotes the correct answer for that step to be successfully completed. A limitation of xPST is that there is no way to carry out generalizable tutoring since there is no way that xPST can remember the state of the previous steps and retrieve it appropriately in the order tree. So the current step cannot make use of the states of the previous steps for tutoring purposes. This restricts xPST to enable building only problem-specific tutors and not generic tutors.

State spaces and domains that are complicated (e.g., games) require feedback to be given to the user proactively to either notify about the current step or to provide information regarding the next steps. xPST supported only two kinds of feedback *Hints*, JITs but none of them is proactive.

The xPST reference documentation and the xPST JavaDoc documentation describing the expressiveness of the language prior to adding these extensions are given at http://xpst.googlecode.com/svn/trunk/xPSTLib/doc/syntax/index.html and http://xpst.googlecode.com/svn/site/javadoc/index.html respectively. The instructions on how to create a cognitive model using xPST is given at http://code.google.com/p/xpst/wiki/CreatingACognitiveModel.

We have added extensions to xPST framework to eliminate these limitations. This section gives an overview of these extensions and how they can be used to eliminate the previous xPST limitations. Each subsection contains appropriate URLs to the code corresponding to those extensions.

5.2.1 Generalizable Tutoring

xPST has been basically designed to tutor on problem specific tasks. Each task contains a set of steps to be performed in some order. Previously xPST had no capability to remember the state of the previous steps and hence could not tutor on them once they were completed. To drive the point home, we use a simple addition task example. The web page has three text boxes, the first two are to enter two numbers to be added and the third text box is to enter the sum of the two numbers. Once the user entered the first number and moved to the second step (goalnode) of entering the second number, the state of the first step is lost, and hence tutoring on addition cannot be done for any generic numbers that the user enters. Though this example seems to be simple this turns out be a very useful feature as it can cater to creation of generalizable tutors (one tutor for many problems) and can enable conditional JITs depending on the previous

state (e.g., in the case of a game environment, if the player picks up a crossbow in a previous step give feedback1 else give feedback2).

The answer to a goalnode is referred to as *checktype* in xPST language. These checktypes can also be functions which take in parameters and return an answer for the goalnode. They are specifically called *functional checktypes*. In order to carry out generalizable tutoring a new generic functional checktype "Ans" has been added to the framework. The "Ans" checktype accepts a single parameter which is the goalnode name and it returns the answer to that goalnode even though the step has been completed. The answer returned can be any of the answer types supported by xPST and this allows the author to fire conditional JITS. http://xpst.vrac.iastate.edu/extensionscode/functionalchecktypes.zip contains the zip file with the code for the "Ans" generic functional checktype along with other functional checktypes provided.

Let's look back at the addition example. We will add a constraint that the two addends should be equal to see the "Ans" checktype in action. So now the answer to the second goalnode will be *Ans("GN1")* to check if both the numbers are equal or not. An appropriate JIT can be fired if that is not the case. Figure 5.4 shows the example xPST snippet depicting this.

```
SetNumber2
{
    answer: Ans("SetNumber1");
    Hint: "Enter the second addend";
    Hint: "The first and second addends need to be equal";
    JIT: "Sorry. You need to have equal addends for this problem";
}
```

Figure 5.4 xPST snippet showing "Ans" checktype in action.

5.2.2 Proactive Feedback

xPST supported providing feedback (Hints and JITs) when the user asks for it. State spaces and domains which are complicated (eg., games) require feedback to be given to the user proactively to either notify that the current step is completed or to provide information regarding the next steps. Apart from the Hints and JITs a new feedback type "OnComplete" has been added to the xPST framework. This is used to provide proactive feedback regarding the current step completion or as reminder about the next steps. This feedback fires as the user completes the current step in the sequence. `http://xpst.vrac.iastate.edu/extensionscode/emscript.g` contains the grammar file of xPST after the addition of the new "OnComplete" feedback type.

For example, in the case of a game environment, Figure 5.5 shows the scenario using the *OnComplete* feedback to inform the user about the current step completion that he or she has entered Building 1.

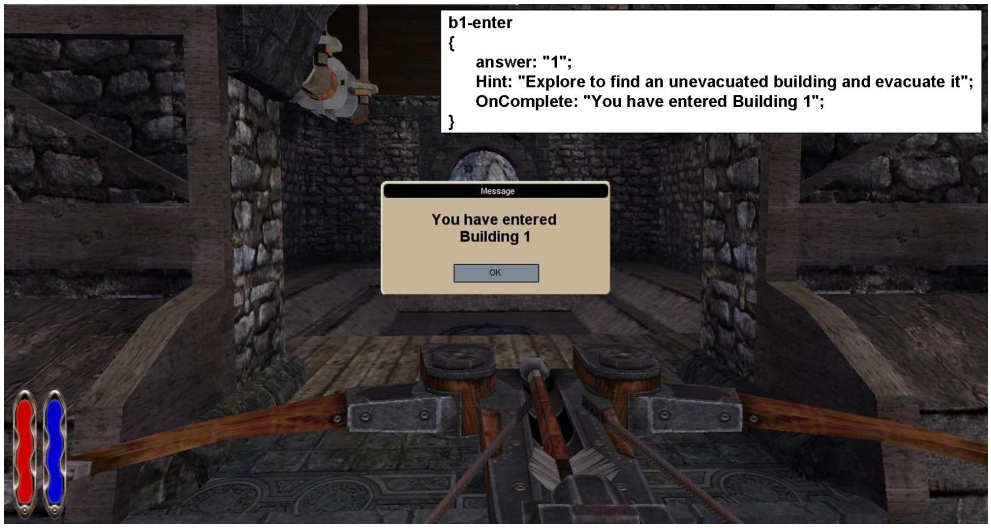


Figure 5.5 Example showing the proactive "OnComplete" feedback type.

5.2.3 Additional Functional Checktypes

We have also incorporated additional functional checktypes into the xPST framework to be able to tutor on math problems. They include the basic arithmetic functional checktypes which are "Sum", "Subtract", "Multiply" and "Divide". Each of these checktypes accepts the two required parameters for the operation to be performed. Also to tutor on fraction addition task (see chapter 7) we have also incorporated the following checktypes into the framework. `http://xpst.vrac.iastate.edu/extensionscode/functionalchecktypes.zip` contains the code for the additional functional checktypes. They are illustrated in Table 5.1.

Table 5.1 Fraction task related functional checktypes

Functional Checktype	**Description**
Lcm("step1","step2")	Function returns True if the user's answer is the lowest common multiple (LCM) of the answer of step1 and answer of step2
EqNumerator("num1","denom1","lcm")	Function returns the equivalent numerator if converting the fraction num1/denom1 to a fraction with a new denominator (lcm)
IsMultiple("step1")	Function returns True if the user's answer is a multiple of the answer of step1

Figure 5.6 shows the example xPST snippet using the "Lcm" functional checktype and also the conditional JITs.

These checktypes are created to support very basic math tutoring. In the future, we hope to create libraries of functional checktypes to be able to tutor on various problem domains.

```
setLCM
{
    answer: Lcm("SetDenom1","SetDenom2");
    Hint: "Enter the LCM of the denominators";
    JIT{v == Sum("SetDenom1","SetDenom2")}: "Fraction denominator
    sum is computed by taking the LCM of the denominators and not
    just by adding them";
    JIT: "The LCM of the denominators is incorrect. Please check
    again.";
}
```

Figure 5.6 xPST snippet showing "Lcm" functional checktype and the conditional JITs.

CHAPTER 6. TORQUE xPST AUTHORING STUDY

The motivation behind conducting this study was to ensure that the Torque xPST driver can support xPST authoring in game environments. This study tests if there exists any learning curve when the novice xPST users use the Torque xPST driver to create xPST game tutors.

The methods used in this study were similar to those used in our earlier study described in Chapter 4.

6.1 Methods

6.1.1 Participants

Participants were contacted through email advertisement (see Figure B.1 in Appendix B). There were 21 interested participants. As a first step, the interested participants took a pre-survey (see Figure B.2). Based on the pre-survey 14 of them were selected to do the complete study. Out of them, two participants did not author any models and expressed their sadness for dropping right in the beginning of the study because they were in their finals week of the semester. We did not hear back from two other participants who took the pre-survey but did not author any models. So we effectively had the data of 10 participants who participated in the study.

6.1.2 Materials

The materials for the study were provided via the web page (http://xpst.vrac.iastate.edu/TorqueTutor/cogstudy.html) similar to the previous study (see Chapter 4). The web page has an embedded 15-minute video which goes through the process of creating a Demo Tutor using the xPST Torque driver. The web page also has links to the commented xPST file and the sample Demo Tutor game application. The Demo Tutor task requires the user to shoot once at the enemy, called "Kork," and then pick up the crossbows present near the fireplace. Finally, the study consisted of designing an xPST tutor for two different tasks.

The first task in the study was *Target Acquisition*. The task teached soldiers how to locate an enemy target so that an assisting aircraft can destroy it. The scenario consisted of a Target (tower). The player moved in the scenario, entered the proximity region of the target, started communication, issued a *report location* command to the Base, again started communication and then issued a *Fire* command to the Base. The second task in the study was *Evacuate*. The task aimed at teaching soldiers how to evacuate cottages in a threatened village environment. There were three cottages in the game scenario with an occupant in each one. The player was supposed to enter a cottage, start communication, and issue the Evacuate command to the occupant of that cottage. The evacuation of cottages could be done in any order. Once all three cottages were evacuated, the task was complete. The necessary mappings for authoring tutors for the two tasks were provided in Table B.1 in Appendix B. The Evacuate task was little more complicated than the Target Acquisition task in terms of the number of goalnodes required to be authored and the linearity level of the sequence section. Since the cottages can be evacuated in any order, the sequence section of the Evacuate task was more non-linear compared to Target Acquisition. The Evacuate task required a minimum of 7 goalnodes, and Target Acquisition task required a minimum of 3 goalnodes

apart from the "Off-path" goalnodes (steps that could be done in the interface that were not relevant to the current task) for successful completion.

The participants were asked to use the on-line xPST Authoring Tool, which was used for the study described in Chapter 4 to author their xPST files . The participants also filled out an exit questionnaire [Figure B.3 in Appendix B] giving feedback on their usage of the xPST Torque Driver and the xPST Authoring Tool.

6.1.3 Procedures

The interested participants initially took the pre-survey, which tested whether they had the *minimum programming experience* (simple HTML editing or SPSS scripting or editing Outlook filter rules or similar kind of minor programming). The participants who had met the required criteria as determined by the pre-survey were then moved on to the complete study. Any under qualified or over qualified participants were excluded at this stage of the study and were provided a compensation of $3 in cash for taking the pre-survey.

The participants who were selected had two weeks to do the study. They were promised a compensation of $40 in cash for taking the pre-survey, successfully authoring tutors for two tasks in the study and completing the exit survey (see Figure B.3 in Appendix B). The task was said to be successfully completed when: 1) there was a cognitive model that worked, meaning it ran in the game and the system provided hints, and 2) there was a tutor that guided the learner through the steps of task as described. The successful participants were also entered into a random lottery for $149 cash prize (the cost of a 5th Gen iPod Nano). All materials were available via a web page (`http://xpst.vrac.iastate.edu/TorqueTutor/cogstudy.html`), and students were free to work on the study at any time or place until its due date. The materials suggested 6 - 12 hours to do the tasks. During task development, technical support was provided by the author in a consistent manner. Support for interface issues was provided

(e.g., not knowing how to install or launch the game) but participants were not given guidance on how to write an xPST file other than pointing to the online documentation.

6.2 Results

We conducted model analysis to qualitatively evaluate the data and also conducted the quantitative timing and editing sessions data analysis. We also report the feedback given by the participants in the exit survey questionnaire.

6.2.1 Model Analysis

10 participants took the complete study. These 10 participants produced 20 models. In Gilbert et al. (2009) we classified the models produced by the participants in the xPST Authoring Study - I into one of five categories (other researchers have used similar scales [Martin et al. (2007)]). We did the same thing with these models produced with the xPST Authoring Tool in this study. The categories were modified to add a new category: "Model has the required hints but does not run due to syntactical errors." The scores were based on actually running the model. Table 6.1 shows that all participants performed well at producing models, with only two models not being sufficient, and with the majority (16 out of 20, 80%) receiving a score of either 4 or 5.

Table 6.1 How the cognitive models were scored

Score	Description	Models
5	A very good model that is beyond just being sufficient	8
4	A sufficient model where the student can complete the task	8
3	Model provides hints, but does not provide enough guidance for a novice	2
2	Model runs but produces nonsensical help	0
1	Model has the required hints but does not run due to syntactical errors	2

The one participant who made syntactical mistakes while authoring the models could not get the two models running, and these models scored 1. One participant had their two models scored 3 and the rest of the participants had their models scored a mixture of 4s and 5s.

6.2.2 Timing Data

The timing and editing sessions data were collected from the xPST Authoring Tool. Table 6.2 displays the average time participants spent on various parts of the models, split by the three different tasks. Table 6.3 displays number of editing sessions.

Table 6.2 Time to complete model actions within a task (times in minutes, with percent of total in parentheses)

	Task A	**Task B**
Sequence	3.49 (17.68%)	2.63 (19.04%)
Mappings	5.87 (29.74%)	4.61 (33.38%)
Feedback	10.38 (52.58%)	6.57 (47.58%)
Total	19.74 (100.0%)	13.81 (100.0%)

Table 6.3 Editing sessions to complete model actions within a task (percent of total in parentheses)

	Task A	**Task B**
Sequence	8.1 (18.6%)	10.0 (26.9%)
Mappings	15.8 (36.2%)	12.2 (32.9%)
Feedback	19.7 (45.2%)	14.9 (40.2%)
Total	43.6 (100%)	37.1 (100%)

We noted two interesting aspects of this data. The first item to notice was the learning curve as these participants moved from task to task. Participants were not required to work on the tasks in order, but all did so, as evident from their log files. As mentioned above, the Task B (Evacuate task) was more complicated than Task A (Target Acquisition task), but more interestingly the time spent and the number of steps required in Task B was less than Task A. The second item of note was that the time

spent and the number of steps (see Table 6.3) taken in the sequence section of Task B was more compared to sequence section of Task A though the total time and the total steps was reduced. But as mentioned, Task B had more complicated sequence than Task A. Though there was no proof of significance due to the small number of participants, it seemed participants were able to complete more modeling in less time after a quick learning curve. To overcome the number of participants limitation of this study we conducted another study described in Chapter 7.

The standard deviation of timing data (in minutes) were 9.16 and 6.24 for Tasks A and B respectively. Figure 6.1 shows the histogram of timing data by tasks and participants of the 9 participants whose all the models ran successfully. The histogram shows that all the participants except one have the expected learning curve.

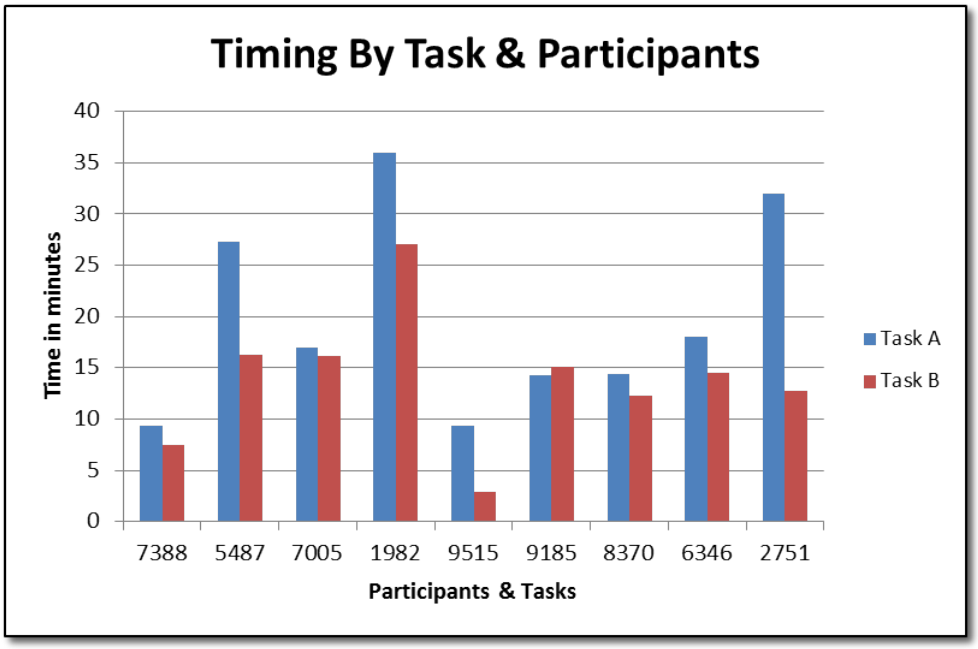


Figure 6.1 Histogram showing Timing By Task and Participants.

Figure 6.2 gives an idea about the trend the average participant was following while authoring the task. Similar to the previous study (see Chapter 4), we calculated the average quintile data for the 18 task models from the nine participants (the one participant whose models scored 1 and did not run was excluded in this since those models did not have the necessary step data to be plotted). We see that the same result was replicated in authoring game tutors too.

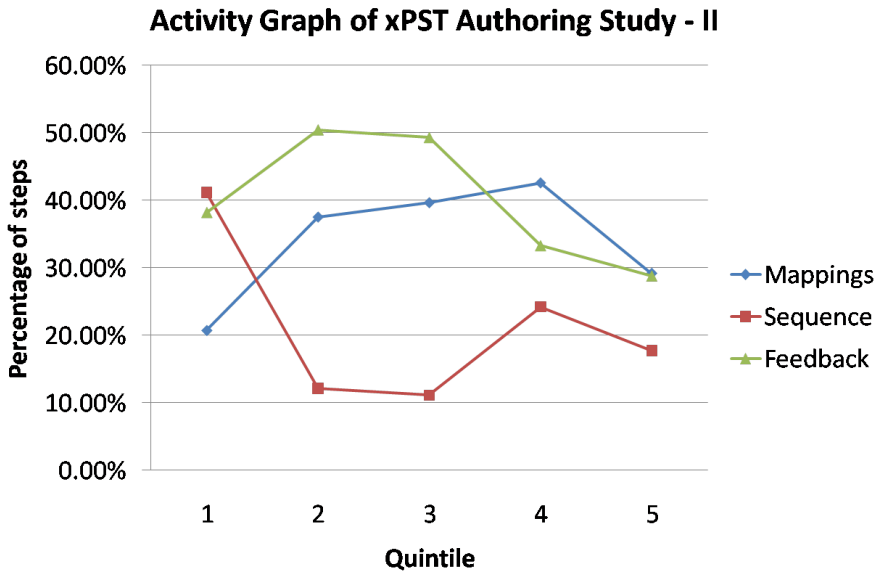


Figure 6.2 Activity Graph of xPST Authoring Study - II

Table 6.4 shows the average quintile data (on which the graph is generated) corresponding to the 18 models.

Table 6.4 Average Quintile data of the 18 models

	Quintile 1	**Quintile 2**	**Quintile 3**	**Quintile 4**	**Quintile 5**
Mappings	20.68%	37.47%	39.60%	42.53%	29.15%
Sequence	41.14%	12.11%	11.12%	24.21%	17.74%
Feedback	38.18%	50.42%	49.28%	33.25%	28.76%

We also found that the average number of contiguous edit steps for Task A and Task B were 10.33 and 8.88 respectively. This also strengthened the learning curve observation

as the amount of switching between sections to fix or amend a previous edit had been reduced. It should also be noted that the significance of the result cannot be proved since we did not have enough data.

6.2.3 Exit Questionnaire Data

The exit questionnaire (see Figure B.3 in Appendix B) asked for their undergraduate major, number of undergraduate programming courses taken and self-rated "techie" score similar to previous study (see Chapter 4). There were also two free response questions: 1) "Do you think this approach makes the tutoring in 3D games relatively easy?", 2) "Do you have any suggestions for the authoring tool or authoring framework?". Five people explicitly said that the approach seemed relatively easy to set up a tutor to guide someone through the game tasks. Two of them said they were not sure, one person thought the documentation and the little help provided when he/she was stuck was helpful, one person felt the idea was great, and the other person (who scored a 1 on both models) felt that it was too complex to understand. This participant also mentioned that he could not spend much time due to his graduation. Most of the participants suggested improving the authoring tool to be more user friendly and have more error checking. There were also suggestions to have more documentation and code samples.

6.3 Discussion

The Torque xPST Driver enables xPST authoring in Torque 3D game. Although the domain is complex, people are able to author tutors with progressively smaller times after a little instruction time. This study tries to answer our second research question of whether there is a learning curve as the participants move from task to task even though the tasks get complex. This study stands as a proof of concept to show that the

Torque xPST driver enables xPST authoring in game environments. But again we note that these results are not significant due to the lower number of participants and further research is required to prove the significance.

CHAPTER 7. FRACTION ADDITION AUTHORING STUDY

The idea behind conducting this study was to examine and compare the extended xPST framework with the Cognitive Tutor SDK in terms of the ratio of training development time to training experience time for authoring similar tasks and to evaluate the differences between "beginner programmers" and "experienced programmers" (described below) in terms of the time they take to author models using xPST and the model scores.

The materials and the procedures used for this study were mostly similar to those used in the previous study (see Chapter 6). So they will be discussed here briefly by giving reference to Chapter 6 wherever required.

7.1 Methods

7.1.1 Participants

The number of participants in this study were increased from those in our previous studies (see Chapters 4 and 6) to analyze the significance of our results. They were contacted through email advertisement (see Figure D.3 in Appendix D). Similar to the previous study they also took a pre-survey (see Figure D.1) but here the participants were not eliminated based on the pre-survey results. We had got 28 participants out of which 14 were categorized as "beginner programmers" and the other 14 were categorized

as "experienced programmers" depending on the number of undergraduate programming courses they have taken. "Beginner programmers" took fewer than three programming courses and the "experienced programmers" took three or more programming courses. From now on we use BP and EP to represent "beginner programmers" and "experienced programmers" respectively.

7.1.2 Materials

Similar to the study described in Chapter 6, the web page (see Figures D.4, D.5, D.6 in Appendix D) hosted all the materials required for the study. It had a 22-minute video showing the creation of a Demo Addition Tutor using the extended xPST Authoring System. The Demo Addition Tutor task (see Figure 7.1) was built on a web page which had two text box controls where the user could enter two values to be added, and the tutor guided the user in the process of addition of two numbers, giving all the required Hints and JITs. The study required the participants to design an xPST tutor for three different tasks using the on-line xPST Authoring Tool (see Chapter 4) and take the exit questionnaire (see Figure D.2 in Appendix D).

Task A gave tutoring on fraction addition where the denominators of both the fractions are same (eg., 1/5 + 2/5). The user interface had the text boxes to enter input fractions and the text boxes to enter the fraction sum. Task B gave tutoring on fraction addition where the denominator of the second fraction is a multiple of the denominator of the first fraction (eg., 1/3 + 1/6). Task C gave tutoring on fraction addition where the denominators of both the fractions are neither same nor the denominator of the second fraction is a multiple of the denominator of the first fraction (eg., 1/3 + 2/5). For Task B and Task C the user interface had the text boxes to enter input fractions and the text boxes to enter the LCM and the equivalent numerators and the text boxes to enter the fraction sum. The task complexity increased as the user moves through Task A to Task B to Task C in terms of the generality of fraction addition and also in terms

of the numbers of steps required for authoring due to increase in the number of UI text boxes. Figure 7.1 shows the user interface of the Task C.

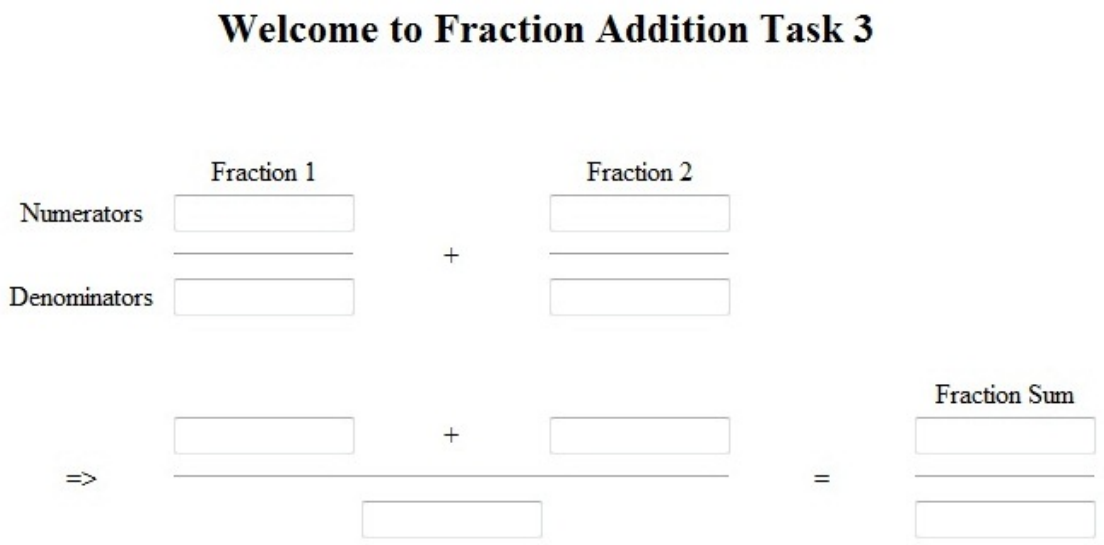


Figure 7.1 Fraction Addition User Interface for Task C.

7.1.3 Procedures

Participants who took the pre-survey were categorized into BP and EP groups depending on the number of undergraduate programming courses they had taken. The time limit to complete authoring the three tasks was 1 week. The compensation structure remained the same as it was in the previous study (see Chapter 6).

7.2 Results

We carried out the model analysis and the timing data analysis. The analysis in this study is little different from the previous study (see Chapter 6) because of the classification of the participants into two groups BP and EP. Since we have the required

number of participants in this study, we also carried out the statistical significance tests. We also mention the user feedback from the exit questionnaire.

7.2.1 Model Analysis

The 28 participants produced 84 models. Similar to the classification in Gilbert et al. (2009), we classified the models produced by the participants into one of the five categories (see Table 7.1). The table shows that the majority (64 out of 84, 76%) received a score of either 4 or 5.

Table 7.1 How the cognitive models were scored

Score	Description	Models
5	A model that produces behaviors close to an ideal model, in terms of hints and just-in-time messages	35
4	A very good model that is beyond just being sufficient	29
3	A sufficient model where the student can complete the task	8
2	Model provides hints, but does not provide enough guidance for a novice	8
1	Model runs but produces nonsensical help	4

Since the data is ordinal and naturally skewed due to more 4's and 5's, we used the Wilcoxon rank sum test, a non-parametric statistical significance test, to test if the BP and EP groups have equally high model scores. We have *H0: There is no difference in the means of the model scores between BP and EP* and *Ha: There is a difference in the means of the model scores between BP and EP*. Table 7.2 shows the mean rank and the sum of the ranks of the two groups.

Table 7.2 Ranks data from the Wilcoxon rank sum test

Groups	N	Mean Rank	Sum of Ranks
BP	14	14.29	200.00
EP	14	14.71	206.00

The ranks indicate that EP group has a little higher scores than BP group but it should be noted that the difference is very little. The test statistic (U = 95) and the p-value is 0.880 which is greater than 0.05. So we accept the null hypothesis. This shows that there is no significant difference in the means of the model scores between BP and EP.

7.2.2 Timing Data

Similar to the previous study (see Chapter 6), the timing data was obtained from the xPST Authoring Tool log files. Table 7.3 displays the average time participants spent in each of the three sections across the tasks A, B and C.

Table 7.3 Time to complete model actions within a task (times in minutes, with percent of total in parentheses)

	Task A	**Task B**	**Task C**
Sequence	11.56 (13.53%)	11.23 (12.47%)	2.04 (3.15%)
Mappings	10.01 (11.71%)	8.43 (9.36%)	3.52 (5.44%)
Feedback	63.88 (74.76%)	70.38 (78.17%)	59.10 (91.41%)
Total	85.45 (100.0%)	90.04 (100.0%)	64.66 (100.0%)

Table 7.3 shows the learning curve as the user progresses from task to task. We observe that there is an increase in the total time from Task A to Task B but it should be noted that this increase is due to increase in the time spent in the feedback section. This is because Task A has 6 goalnodes for which feedback needs to be authored where as Tasks B and C have 10 goalnodes because the denominators of the fractions are not same and they require intermediate reduction steps. But from Task B to Task C again the time decreases.

Figure 7.2 shows the histogram of timing data by tasks and participants. Out of 28 participants, 17 of them (8 people from BP group and 9 people from EP group) seem to have the expected learning curve. It is hard to tell the exact reason why other participants do not have the expected learning curve. We think one reason might be

the random fluctuations that one sees in real world data. Also, despite the instructions given, perhaps the participants might have left the authoring tool running even when they are not actually working on the task.

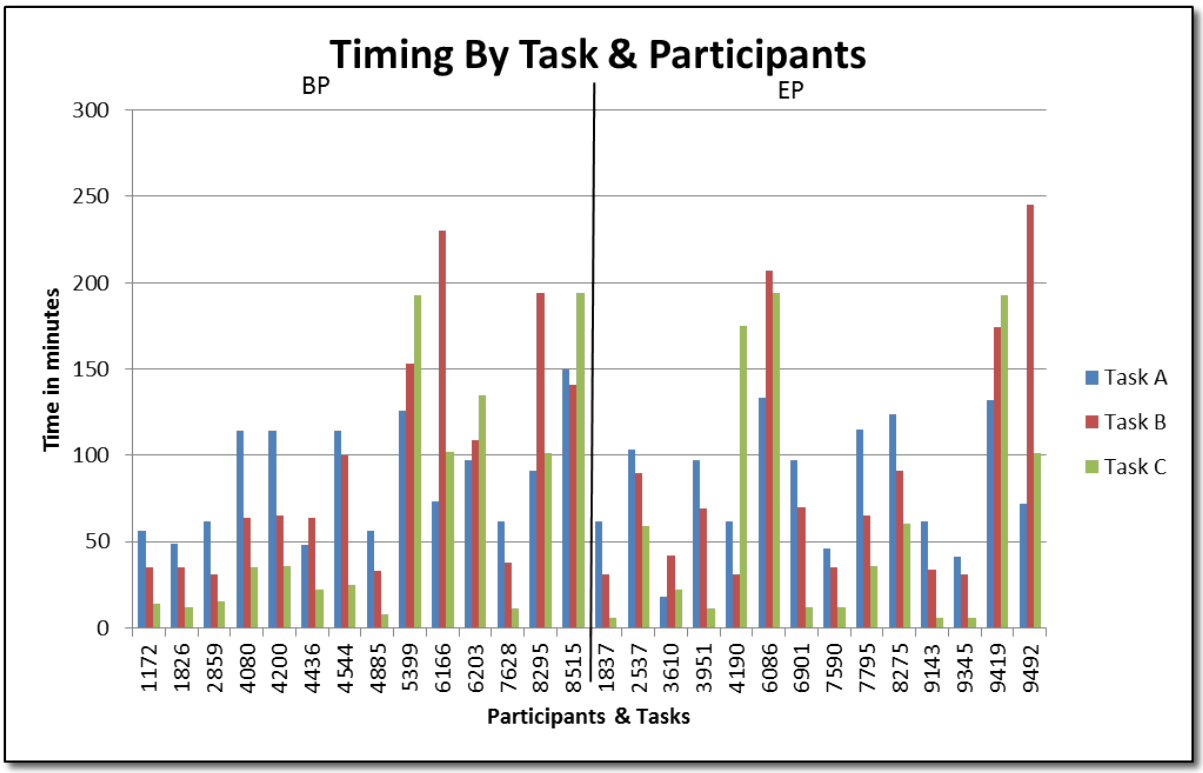


Figure 7.2 Histogram showing Timing By Task and Participants.

We conducted a 2x3 mixed factor ANOVA (a 3 level dependent factor of task number, and a 2 level independent factor of programming experience) with time taken as the dependent variable to study the between-subjects (BP and EP groups) effects in relation to time taken. This gave Type III Sum of Square = 221.780, df = 1, Mean Square = 221.780, F = 0.029 and p-value of .865 clearly showing that we can accept the null

hypothesis that there is no significant difference between the BP and EP groups in terms of the time taken to complete the tasks. Figure 7.3 plots a graph between the Estimated Marginal Means (the mean times) and the Programming Experience groups for all the three tasks. The plot shows that BP have taken more time than EP for each task but the difference between these times decreased gradually and for Task C there is little difference between times taken by BP(65.03) and EP(64.28).

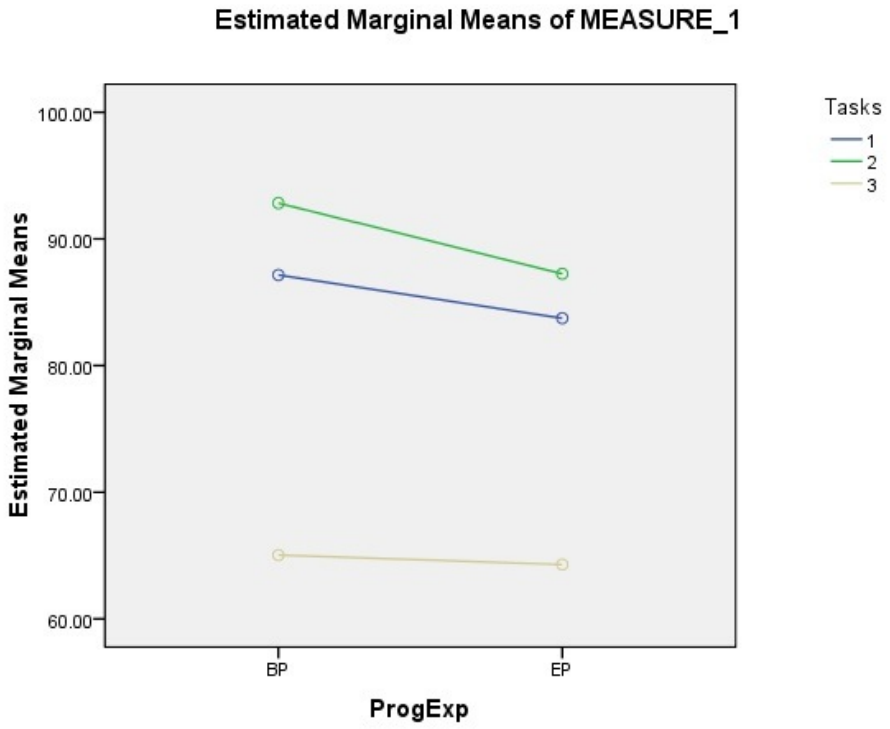


Figure 7.3 Estimated Marginal Means vs BP and EP for three tasks.

It would be interesting to compare Cognitive Tutor SDK and xPST in terms of the ratio of training development time to training experience time. This ratio is estimated to be 10:1 in the case of Cognitive Tutor SDK for authoring the three similar fraction addition tasks [Blessing and Gilbert (2008)]. The exit questionnaire asked the participants to give an estimate on the number of hours they needed to go through the instruction to do the tasks. From this data we have a total average instruction time of 150 minutes.

The total average development time for the three tasks is 240.15 minutes. The average total instruction experience time is estimated to be 75 minutes (15 minutes for Task A and 30 minutes each for Task B and Task C). This gives us the ratio of 5.2:1 which is approximately 50% less than that of Cognitive Tutor SDK.

7.2.3 Exit Questionnaire Data

Participants completed a short exit questionnaire (see Figure D.2 in Appendix D) after they were done with authoring the tasks. The questionnaire asked for the estimate of the time they needed to go through the instruction provided, what was challenging about cognitive modeling, other arenas where this kind of tutoring would be helpful and what do they think of this approach.

Participants mentioned that understanding the functions to use and thinking in lines of xPST was tough in the beginning. But as they progressed, they were able to grasp them. There were suggestions to use xPST in teaching mathematics, computer games like chess, language courses, biology, statistics and how to use email, chat, the Iowa State AccessPlus accounts management system, etc. There was also a suggestion to use xPST to teach Sudoku for deaf and dumb people. A majority of the participants felt that the approach is flexible and has the potential to teach novice users on problem specific tasks. There were also suggestions for improving the authoring tool to be more user friendly. We would like to incorporate this feedback in planning future xPST research.

7.3 Discussion

We see that the core extensions provided to xPST are able to support tutoring in cases where the previous state needs to be remembered. There is a learning curve as the participants move from task to task. As an answer to our third research question, we see that the ratio of training development time to training experience time is approximately

50% less than that of Cognitive Tutor SDK. And as an answer to our fourth research question, we see that there is no significant difference in the means of the model scores and the time taken to complete the tasks between BP and EP, which indicates that programming experience did not effect participants' ability to use the xPST tool. This result is a notable achievement in the effort to empower non-programmers to create intelligent tutoring systems.

CHAPTER 8. SUMMARY AND FUTURE WORK

This chapter summarizes the research challenges this thesis aimed to answer, the solutions proposed and my specific contribution. We have started over with a brief introduction to Intelligent Tutoring Systems and their anatomy. Then we presented a brief literature review on the evolution of ITSs, the effectiveness of ITSs in training/learning, the parent systems (CTAT, Cognitive Tutor SDK) to xPST, comparison of xPST with its parent systems, the emergence of games in tutoring and the xPST Authoring System in detail.

The first research question of interest is "What learning curve, if any, exists when users use xPST to author tutors on existing web-interfaces?". In the direction of answering this question we have conducted a study (described in Chapter 4) to test the ability of novice users of xPST to create cognitive models using the xPST Authoring System. The participants authored tutors for three tasks on ACM portal using the xPST Authoring Tool. We noticed two main points from the study. First, people were able to use the xPST Authoring Tool. Everyone who attempted to start the task produced at least one working model. The instruction provided was minimal, about an hour, but with that instruction a mix of people created a model of a particular task. The second item to notice is the the presence of a learning curve as the participants moved from task to task. There was a spread in terms of time to create the models, but all users reduced their times upon successive models. By the third model most participants were able to complete their model in just over 1.5 hours. After another model or two, that time would be close to 1 hour (the developers of xPST could probably produce such models

in 30-45 minutes). That 1 hour would provide around 10 minutes worth of instruction. So this study tries to answer our first research question that there exists a learning curve when users use xPST to author tutors on existing web-interfaces but we cannot prove its significance since the study does not have enough number of participants. The small up-front cost of training coupled with a small time in producing the training could make the xPST approach attractive to those who want to provide model-tracing feedback to existing interfaces. My specific contribution in this study involved the design and development of the xPST web based Authoring Tool along with the logging functionality, few bug fixes in the xPST Firefox Plugin needed for the study and monitoring the study from start to end.

The second research question of interest is "What learning curve, if any, exists when users use xPST to author tutors on existing game interfaces?". In the direction of answering this question we have conducted a study (described in Chapter 6) to test the ability of novice users of xPST to create tutors in 3D games using the extended xPST Authoring System. The participants authored tutors for two tasks in the Torque Game Engine Advanced using the xPST Authoring Tool. We see that the Torque xPST Driver enables xPST authoring in Torque 3D game. Given that the domain is complex, people are able to author tutors with progressively smaller times after a little instruction time though this result is not significant. There is a learning curve as the participants move from task to task even though the tasks get complex. My specific contribution in this study is to develop the Torque xPST Driver, design and conduct the study from start to end and analyze the results.

The third and fourth research questions of interest are "Is there a difference between xPST and Cognitive Tutor SDK in terms of the training development to training experience time ratio?" and "Is there a significant difference between the "beginner programmer" and "experienced programmer" groups in terms of time taken to author using xPST?". These research questions are answered in Chapter 7 by conducting a

study on the fraction addition problem which Cognitive Tutor SDK has used in the past. As in the previous studies we see a learning curve as the participants move from task to task. The ratio of training development time to training experience time using xPST is approximately 50% less than that of Cognitive Tutor SDK. We see that there is no significant difference between the BP and EP groups in terms of the time taken to complete the tasks. We also see that there is no significant difference in the means of the model scores between BP and EP. My specific contribution in this study involved developing the generalizable tutoring extensions to xPST, developing other additional functional checktypes and design and conduct the study from start to end and analyze the results.

Future research can explore to add more functionalities to Torque xPST Driver to support events like communication between teams in addition to communication between players, events on a category of entities instead of individual entities (eg: for tutoring on all tanker objects apart from just a particular tanker), events specific to military domains like navy, air force etc. The xPST framework can also be experimented to be embedded in applications on various hardware devices like mobile, multitouch etc. It is also worth exploring Natural Language Processing tutoring using xPST.

APPENDIX A. xPST AUTHORING STUDY - I

Your deliverables section of the study

Your user is someone who needs to learn how to use the ACM Portal online database. Use http://portal.acm.org/dl.cfm?coll=portal&dl=ACM to access it, or the link in the xPST editor. (Note that if you are off-campus, you won't be able to read the text of the articles you find unless you login to the ACM Portal via the ISU Library's proxy server, but you don't need to do that for this activity; this is just about search. If you do want to do this for some reason, use this link and login to the ISU Library proxy server with your ISU ID and library PIN.)

Your user needs to learn how to accomplish three tasks, so you will create three separate xPST files, one for each task. To do this, use the xPST Editor http://aphrodite.vrac.iastate.edu/WebSite1/xpstedit.aspx.

Note: This activity can be a little confusing because you are a student but you're also creating a tutor for someone who is a student. Below, for example, the italics text is what the learner, your target user, would see, but the normal text is for you to see.

Task A: Search by University

Create an xPST file that can tutor on the following task:

Use the Advanced Search to find articles from authors at MIT on intelligent tutoring that are not about math and where you have full text available. Sort the list of hits to see which one has had the most downloads over the past 6 weeks. How many articles are there, and what's the name of the paper with the most downloads?

Provide JITs as appropriate. Include hints for every step. You can decide whether you want to have multiple levels for your hints. Some people prefer to use the first level to remind the learner of his or her goal and the second level to give concrete directions about what is required, e.g. what to click or type. You could have further levels if desired. In your write up of the activity, be sure to comment on your rationale for your design decision.

You don't need to provide a way for the learners to input the answers to the questions like "How many articles are there?" Just assume they have pencil and paper to write that down on their own. These are mostly for your own reference while creating the cognitive model. The answers are 1) there are 9 articles from this search and 2) the one with the most downloads is "What would they think?: a computational model of attitudes."

Note that you need to put quotes around "intelligent tutoring" when you type it in the search field.

Task B: Search Proceedings

Create an xPST file that can tutor on the following task:

Do a search to find articles from the CHI conference proceedings (but not the extended abstracts) on multitouch that have full-text available in ACM Portal. How many total CHI articles on multitouch are there? Which one has the most citations? Note that the official name of the CHI conference is the Conference on Human Factors in Computing Systems.

You can use two approaches: Browsing the CHI proceedings and then searching for "multitouch," or use the Advanced Search.

Include both possible paths to the goal in your sequence (the browsing and the advanced search). The answer to total articles is 7. The one with the most citations is Smartskin by Rekimoto, with 72.

Task C: Find Reference

Create an xPST file that can tutor on the following task:

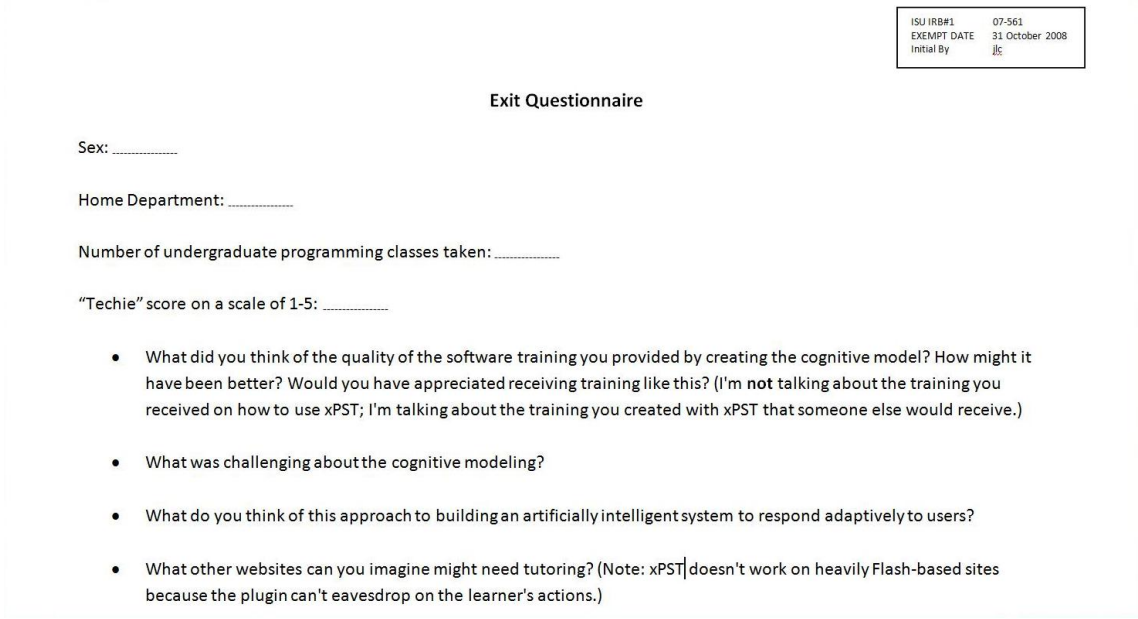
You can remember an important article by Ritter and one other person from 1996. Find the article and the exact ACM Reference Citation format for it.

Because some learners may want to type "ritter" or "Ritter 1996" etc. in the general search field instead of using the Advanced Search, create a branch of your sequence to allow that, even though it's a bad idea (you get hundreds of hits). The normal sequence of steps would start with clicking Advanced Search, but this other branch would start with typing in the search field, getting the hits, but also a JIT explaining why it's a bad idea and that clicking Advanced Search next would be the right thing to do next. (Note that after doing a search, the Advanced Search link is located in the left navigation menu below the blue boxes.)

For that first step's answer to allow a large variety of combinations like "ritter," "Ritter," "ritter 96," "1996 Ritter" etc, set the answer in the xPST file equal to RegEx("[rR]itter(1996|96)?|(1996 |96)?[Rr]itter"); This formula parses regular expressions (a computer science term) and will accept all those combinations of Ritter and 1996 as correct while rejecting others. This may seem confusing now but will be more understandable after watching the training.

Note that when learners click into an article in the hit list, they can click either the title or the smaller "full citation" link. You need to have a step in your sequence for each of those. The "ACM Ref " link within the Ritter and Blessing article is what you want them to click as the final step. Note also that in the advanced search page, to set the publication date to 1996, you set "Published since" to 1996 and "Published before" also to 1996. This seems a little odd, as if the second date should really be called "published before or during." Also, because the ACM Portal website does some odd things with its drop down menus, use the following RegEx? formula as the answer for your steps about selecting 1996: RegEx("1996//s*");

Exit Questionnaire Data



ISU IRB#1 07-561
EXEMPT DATE 31 October 2008
Initial By jlc

Exit Questionnaire

Sex: .....

Home Department: .....

Number of undergraduate programming classes taken: .....

"Techie" score on a scale of 1-5: .....

- What did you think of the quality of the software training you provided by creating the cognitive model? How might it have been better? Would you have appreciated receiving training like this? (I'm **not** talking about the training you received on how to use xPST; I'm talking about the training you created with xPST that someone else would receive.)
- What was challenging about the cognitive modeling?
- What do you think of this approach to building an artificially intelligent system to respond adaptively to users?
- What other websites can you imagine might need tutoring? (Note: xPST doesn't work on heavily Flash-based sites because the plugin can't eavesdrop on the learner's actions.)

Figure A.1 Exit Questionnaire data sheet of xPST Authoring Study - I

APPENDIX B. xPST AUTHORING STUDY - II

Email Advertisement

Volunteers needed for Research Study

We are exploring methods of creating cognitive models for 3D first-person shooter video games. Research activities can be done at your own schedule within a two-week period on any Windows computer. Activities typically require between 6 and 12 hours total.

Compensation: $40 per volunteer, along with a chance to win an amount of $149, the price of an iPod Nano (5th Gen).

Your participation is completely voluntary. All of the information you provide will be kept strictly confidential and reported in summary form only. No individual will be identified, nor will your name be attached to any data. At the project's end, researchers will destroy any personal identifying information.

Limited programming experience required, e.g. editing HTML or doing SPSS scripting. You must an adult age 18 or older. To volunteer, please contact Sateesh Kodavali at skodaval@iastate.edu .

Figure B.1 Email Advertisement of xPST Authoring Study - II

Pre-Survey Questionnaire

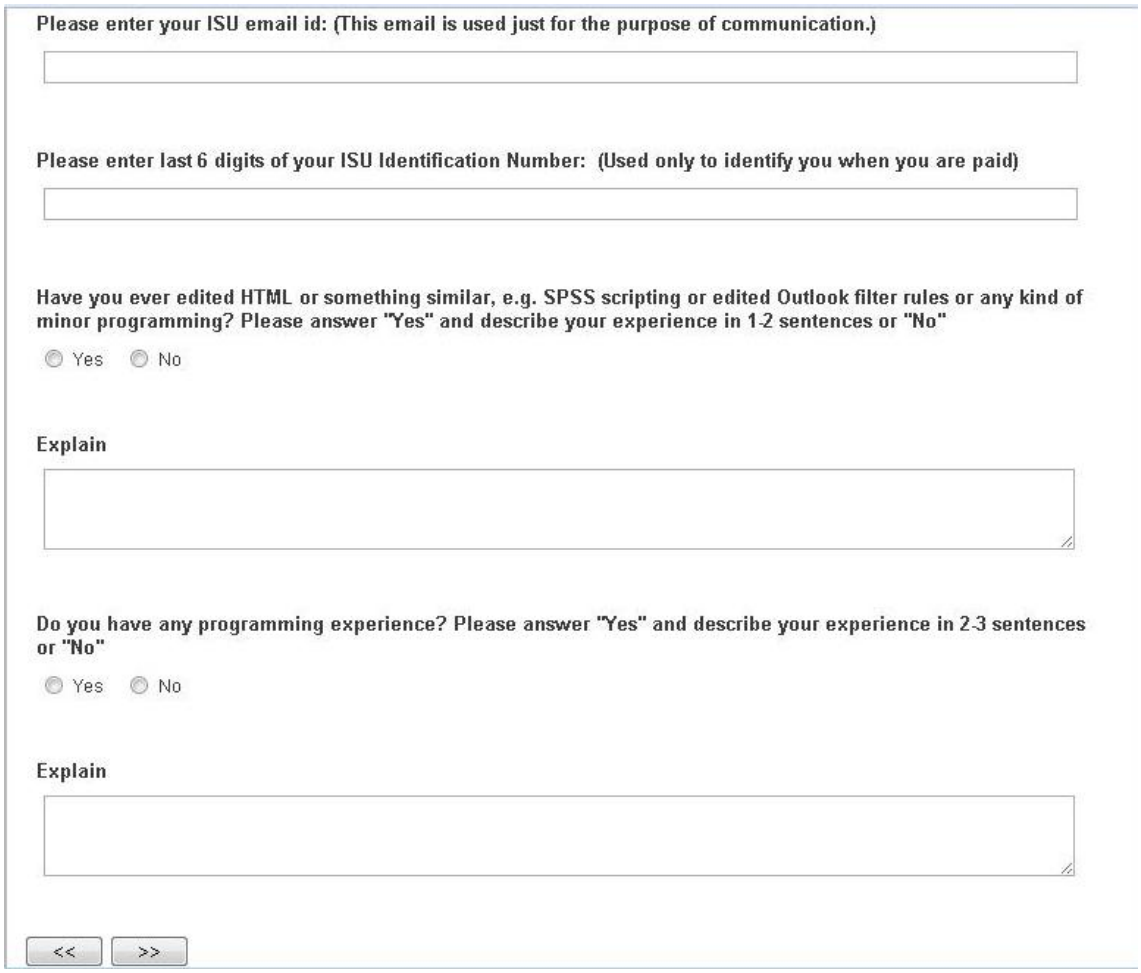


Figure B.2 Pre-Survey Questionnaire of xPST Authoring Study - II

Mappings Provided for the Two Tasks

The required mappings for authoring the two tasks in the study, their description and their correct answers are given in Table B.1.

Exit Survey Questionnaire

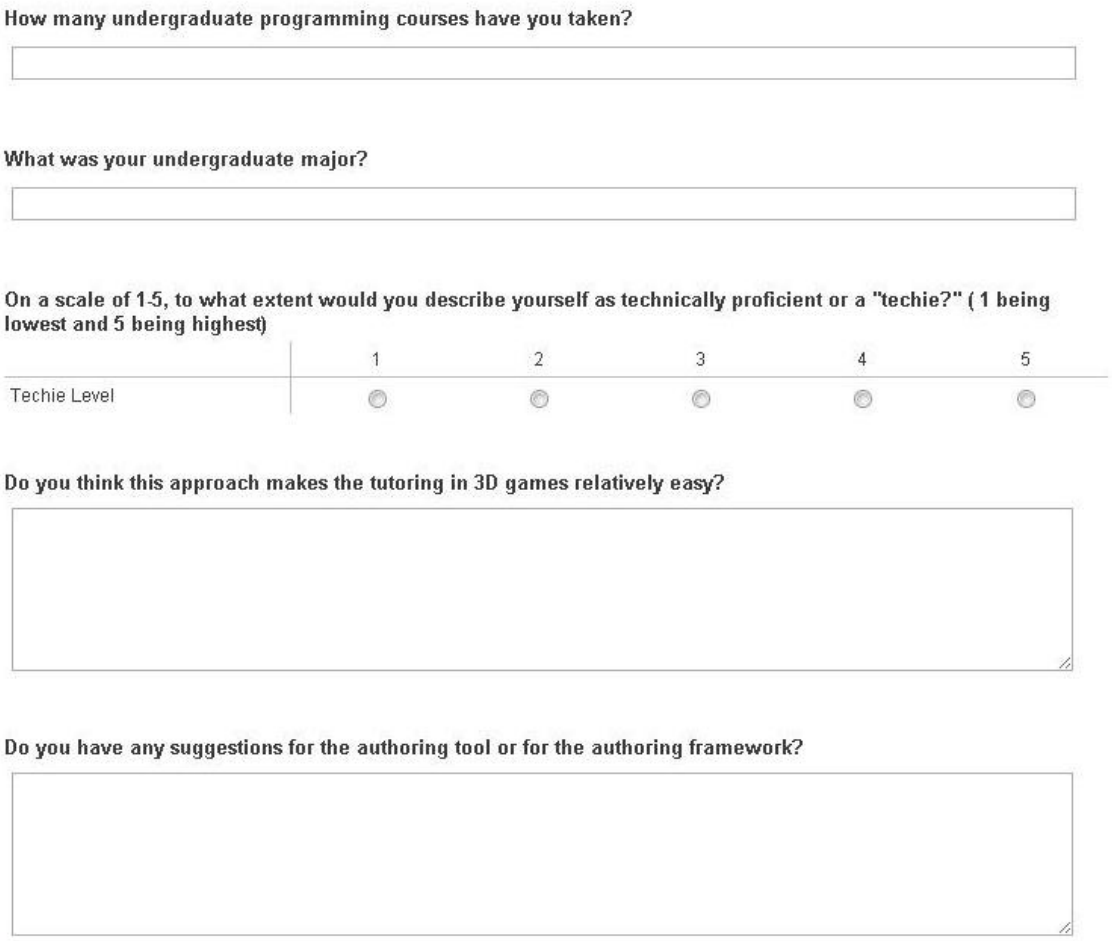


Figure B.3 Exit Survey Questionnaire of xPST Authoring Study - II

Table B.1 Mappings provided for the two tasks, their description and the correct answer

Mapping	**Description**	**Correct Answer**
Target:EnterProximity	Triggers when player enter the proximity region of the Target	"1"
base:Evacuate	Command asking the base to Evacuate	"Evacuate"
base:Ammo	Command asking the base use Ammo	"Ammo"
base:Fire	Command asking the base to Fire	"Fire"
base:Jump	Command asking the base to Jump	"Jump"
base:Destroy	Command asking the base to Destroy	"Destroy"
base:ReportLoc	Command to report the location of the target to base	"ReportLoc"
cottage1Trigger:Enter	Triggers when the player enters cottage 1	"1"
startcommunicate	Command to start communication	"c"
cottage1Occupant:Evacuate	Command asking the occupant in cottage 1 to Evacuate	"Evacuate"
cottage1Occupant:Ammo	Command asking the occupant in cottage 1 use Ammo	"Ammo"
cottage1Occupant:Fire	Command asking the occupant in cottage 1 to Fire	"Fire"
cottage1Occupant:Jump	Command asking the occupant in cottage 1 to Jump	"Jump"
cottage1Occupant:Destroy	Command asking the occupant in cottage 1 to Destroy	"Destroy"
cottage1Occupant:ReportLoc	Command to report the location of the target to base	"ReportLoc"

APPENDIX C. SAMPLE xPST FILE

Here is a sample xPST file of a DemoTask in a 3D game environment. The task requires the user to shoot once at the enemy, called 'Kork', and then pick up the crossbows, present near the fire place.

```
sequence
{
    (shoot-the-kork then pickup-the-crossbows) then
    All-Done and
    Error-Not-Done;
}

feedback
{
    shoot-the-kork
    {
        answer: "1";
        Hint: "Go find the Kork and shoot it once.";
        Hint: "A green horizontal bar appears below the crosshairs when you have correct aim at the Kork.";
        OnComplete: "You have successfully shot the kork.";
    }

    pickup-the-crossbows
    {
        answer: "1";
        Hint: "Go pick up Crossbows.";
        Hint: "A crossbow is a weapon consisting of a bow mounted on a stock that shoots projectiles.";
        Hint: "The corssbows are present near the fire place.";
        OnComplete: "You have picked up the crossbows.";
    }

    Error-Not-Done
    {
        answer: "0";
        JIT: "You are not done with this problem yet. Ask for a hint if you would like help.";
    }

    All-Done
    {
        answer: "1";
        Hint: "You have successfully completed the task.";
    }

}

mappings
{
    kork:shoot => shoot-the-kork;
    crossbow:pickup => pickup-the-crossbows;
    [priority=2] TutorLink.Done => All-Done;
    [priority=1] TutorLink.Done=>Error-Not-Done;
}
```

Figure C.1 Sample xPST file of a DemoTask in a 3D game environment.

APPENDIX D. xPST AUTHORING STUDY - III

Pre-Survey Questionnaire

Figure D.1 shows the pre-survey questionnarie of the xPST Authoring Study - III

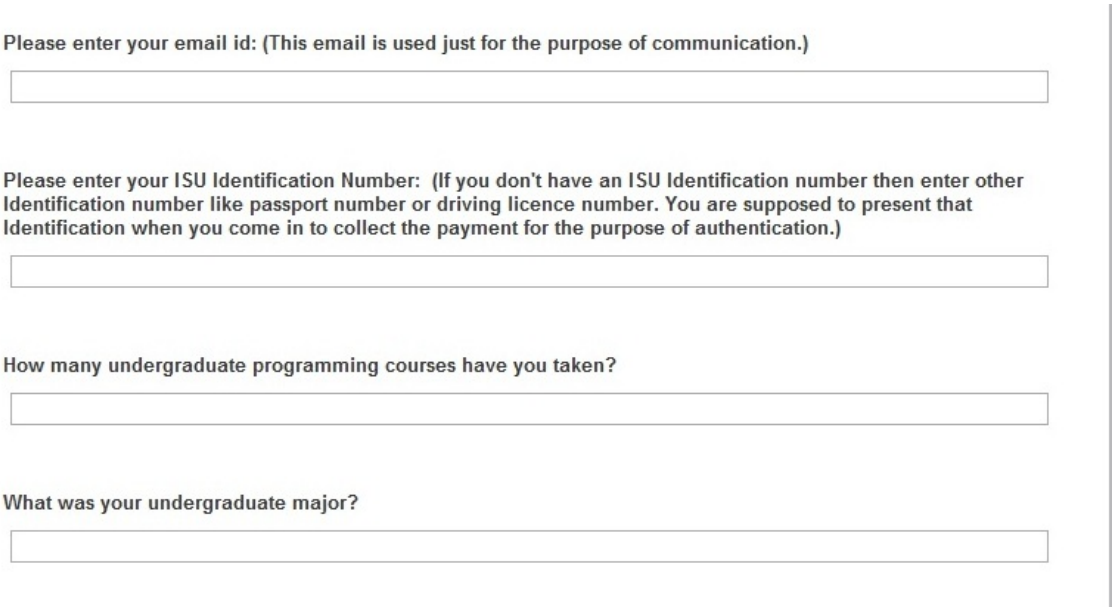


Figure D.1 Pre-Survey Questionnaire of xPST Authoring Study - III

Exit Survey Questionnaire

Figure D.1 shows the exit survey questionnarie of the xPST Authoring Study - III

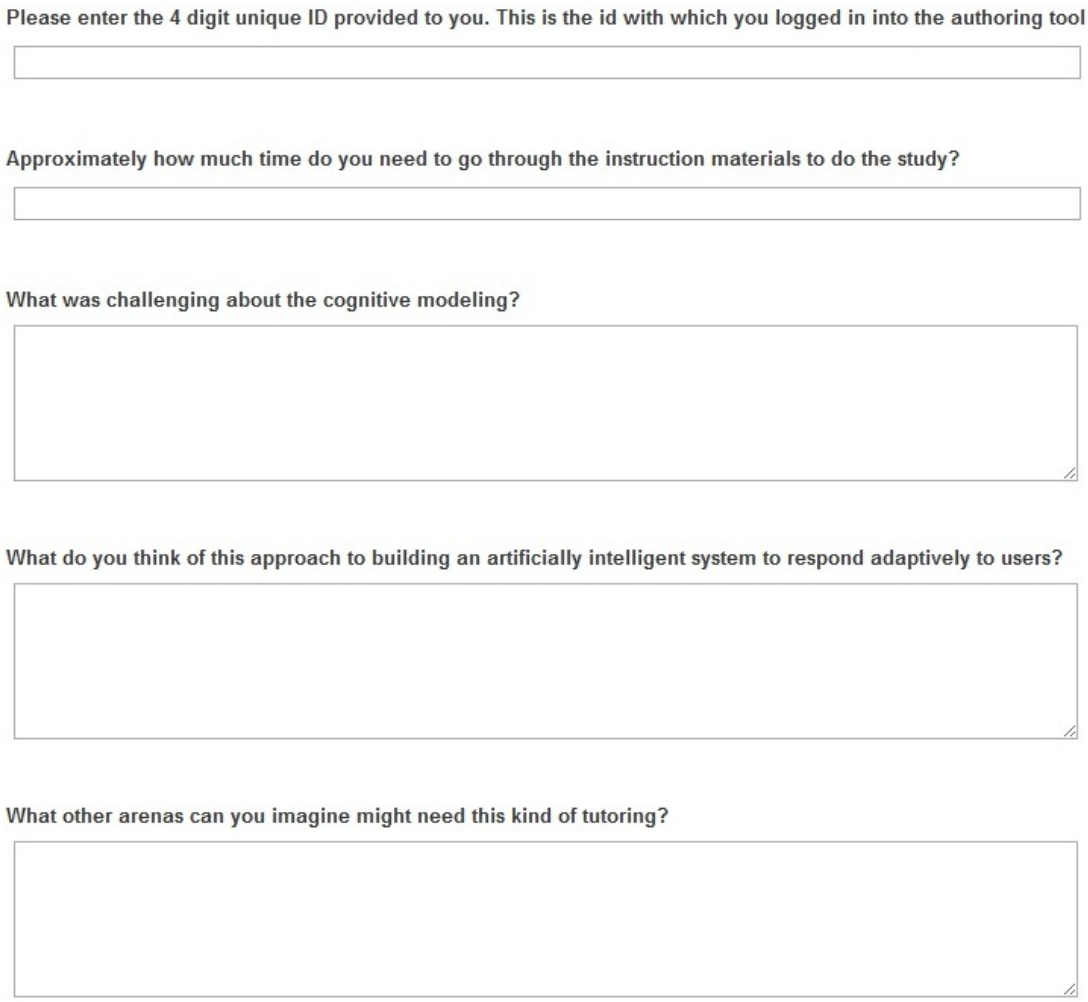


Figure D.2 Exit Survey Questionnaire of xPST Authoring Study - III

Email Advertisement and Screenshots of the study webpage

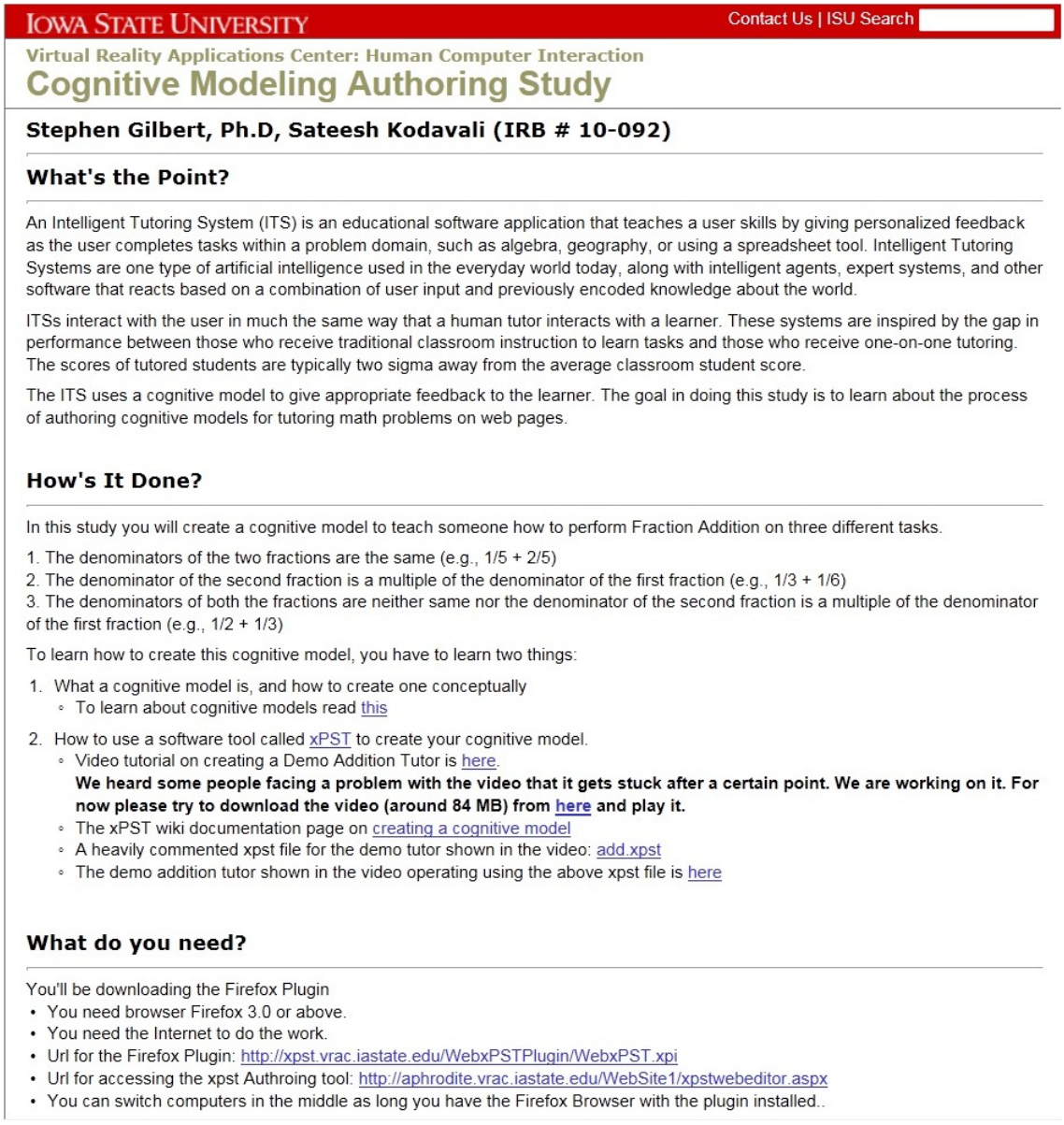
We are exploring methods of creating cognitive models (tutors) for math problems (fraction addition). Research activities can be done at your own schedule within a one-week period on any Windows computer. Activities typically require between 6 and 10 hours total.

Compensation: $40 per volunteer, along with a chance to win an amount of $149, the price of an iPod Nano (5th Gen).

Your participation is completely voluntary. All of the information you provide will be kept strictly confidential and reported in summary form only. No individual will be identified, nor will your name be attached to any data. At the project's end, researchers will destroy any personal identifying information.

You must an adult age 18 or older. To volunteer, please contact Sateesh Kodavali at skodaval@iastate.edu .

Figure D.3 Email Advertisement of xPST Authoring Study - III



IOWA STATE UNIVERSITY Contact Us | ISU Search

Virtual Reality Applications Center: Human Computer Interaction

Cognitive Modeling Authoring Study

Stephen Gilbert, Ph.D, Sateesh Kodavali (IRB # 10-092)

What's the Point?

An Intelligent Tutoring System (ITS) is an educational software application that teaches a user skills by giving personalized feedback as the user completes tasks within a problem domain, such as algebra, geography, or using a spreadsheet tool. Intelligent Tutoring Systems are one type of artificial intelligence used in the everyday world today, along with intelligent agents, expert systems, and other software that reacts based on a combination of user input and previously encoded knowledge about the world.

ITSs interact with the user in much the same way that a human tutor interacts with a learner. These systems are inspired by the gap in performance between those who receive traditional classroom instruction to learn tasks and those who receive one-on-one tutoring. The scores of tutored students are typically two sigma away from the average classroom student score.

The ITS uses a cognitive model to give appropriate feedback to the learner. The goal in doing this study is to learn about the process of authoring cognitive models for tutoring math problems on web pages.

How's It Done?

In this study you will create a cognitive model to teach someone how to perform Fraction Addition on three different tasks.

1. The denominators of the two fractions are the same (e.g., 1/5 + 2/5)
2. The denominator of the second fraction is a multiple of the denominator of the first fraction (e.g., 1/3 + 1/6)
3. The denominators of both the fractions are neither same nor the denominator of the second fraction is a multiple of the denominator of the first fraction (e.g., 1/2 + 1/3)

To learn how to create this cognitive model, you have to learn two things:

1. What a cognitive model is, and how to create one conceptually
   - To learn about cognitive models read this
2. How to use a software tool called xPST to create your cognitive model.
   - Video tutorial on creating a Demo Addition Tutor is here.
     We heard some people facing a problem with the video that it gets stuck after a certain point. We are working on it. For now please try to download the video (around 84 MB) from here and play it.
   - The xPST wiki documentation page on creating a cognitive model
   - A heavily commented xpst file for the demo tutor shown in the video: add.xpst
   - The demo addition tutor shown in the video operating using the above xpst file is here

What do you need?

You'll be downloading the Firefox Plugin

- You need browser Firefox 3.0 or above.
- You need the Internet to do the work.
- Url for the Firefox Plugin: http://xpst.vrac.iastate.edu/WebxPSTPlugin/WebxPST.xpi
- Url for accessing the xpst Authroing tool: http://aphrodite.vrac.iastate.edu/WebSite1/xpstwebeditor.aspx
- You can switch computers in the middle as long you have the Firefox Browser with the plugin installed..

Figure D.4 Screenshot 1 of xPST Authoring Study - III webpage

You Do It!

Plan about 6-10 hours for this activity.

Your login: You are already provided with a four digits id through email. Each xPST file will have a different login ID. If your user ID is 1234, then you'll use 1234_A for the Task A and 1234_B for the Task B and 1234_C for the Task C. If you would like to practice or try the Demo addition tutor described in the video use the login 1234_add_A.

Overview of General steps you will perform:

- Login to the xPST Authoring tool for the task which you would like to work on.
- Write the xPST sections of the Authoring Tool and Save it.
- Test it to see if you have what you want to see.
- Repeat the above two steps till you have your tutor as expected.
- Please logout when you are not working on the task so that we get correct estimates of the time you spent on the task.

Your Deliverables

Your user needs to learn how to accomplish three tasks, so you will create three separate xPST files, one for each task. To do this, use the xPST Editor linked above.

General requirements for all tasks:

A task is said to be completed when you 1) have a cognitive model that works, meaning it runs and the system provides hints, and 2) have a tutor that guides the learner through the all steps catering to the above mentioned conditions..

Provide JITS as appropriate. Include hints for every step. You can decide whether you want to have multiple levels for your hints. Some people prefer to use the first level to remind the learner of his or her goal and the second level to give concrete directions about what is required, e.g. what to type. You could have further levels if desired. Comment the xPST file extensively in order for the researchers to understand the rationale of your design decision.

Don't worry about rounding the fraction sum

Task A:

This task gives tutoring on fraction addition where the denominators of both the fractions are same. (eg: 1/5 + 2/5). The user interface has the text boxes to enter input fractions and the text boxes to enter the fraction sum.

For the example of adding the fractions 1/5 and 2/5 the correct answer is 3/5

Provide tutoring for the following:

- Check the input boxes are given only numbers and not text
- Check the numbers entered are not zero
- Check both the denominators of the two fractions are same
- Check the numerator sum entered is correct or not
- Check the denominator sum entered is correct or not

You are free to provide tutoring for other cases as well. Be creative on the mistakes that can be made.

Task B:

This task gives tutoring on fraction addition where the denominator of the second fraction is a multiple of the denominator of the first fraction. (eg. 1/3 + 1/6). The user interface has the text boxes to enter input fractions and the text boxes to enter the LCM (lowest common multiple) and the equivalent numerators and the text boxes to enter the fraction sum.

For the example of adding the fractions 1/3 and 1/6 the LCM of 3 and 6 is 6 and the equivalent numerator when converting the fraction 1/3 to the fraction with LCM as denominator is 2 and the equivalent numerator when converting the fraction 1/6 to the fraction with LCM as denominator is 1. And the correct fraction sum is 3/6

Figure D.5 Screenshot 2 of xPST Authoring Study - III webpage

Provide tutoring for the following:

- Check the input boxes are given only numbers and not text
- Check the numbers entered are not zero
- Check to see the denominator of the second fraction is a multiple of the denominator of the first fraction.
- Check if the LCM, the equivalent numerators are correct or not.
- Check the numerator sum entered is correct or not
- Check the denominator sum entered is correct or not

You are free to provide tutoring for other cases as well. Be creative on the mistakes that can be made.

Task C:

This task gives tutoring on fraction addition where the denominators of both the fractions are neither same nor the denominator of the second fraction is a multiple of the denominator of the first fraction. (eg. 1/3 + 2/5). The user interface has the text boxes to enter input fractions and the text boxes to enter the LCM and the equivalent numerators and the text boxes to enter the fraction sum.

For the example of adding the fractions 1/3 and 2/5 the LCM of 3 and 5 is 15 and the equivalent numerator when converting the fraction 1/3 to the fraction with LCM as denominator is 5 and the equivalent numerator when converting the fraction 2/5 to the fraction with LCM as denominator is 6. And the correct fraction sum is 11/15

Provide tutoring for the following:

- Check the input boxes are given only numbers and not text
- Check the numbers entered are not zero
- Check to see the denominators of both the fractions are neither same nor the denominator of the second fraction is a multiple of the denominator of the first fraction.
- Check if the LCM, the equivalent numerators are correct or not.
- Check the numerator sum entered is correct or not
- Check the denominator sum entered is correct or not

You are free to provide tutoring for other cases as well. Be creative on the mistakes that can be made.

Description of Useful Functions:

Check the video and the xpst file of the tutor in the video to see how these functions are being used.

Function	Description
RegEx("[1-9]\\d*")	Functions returns True if the user's answer is a number other than zero
Ans("step1")	Functions returns the answer of step1
Sum("step1","step2")	Function returns the sum of the answer of step1 and answer of step2
IsMultiple("step1")	Function returns True if the user's answer is a multiple of the answer of step1
Lcm("step1","step2")	Function returns True if the user's answer is the lowest common multiple (LCM) of the answer of step1 and answer of step2
EqNumerator ("num1","denom1","lcm")	Function returns the equivalent numerator if converting the fraction numerator(num1) / denominator (denom1) to a fraction with a new denominator (lcm)
IsNotMultiple("step1")	Function returns True if the user's answer is not a multiple of the answer of step1

Figure D.6 Screenshot 3 of xPST Authoring Study - III webpage

BIBLIOGRAPHY

Aleven, V., McLaren, B., Roll, I., Koedinger, K. R. (2006). Toward meta-cognitive Tutoring: A Model of Help-Seeking with a Cognitive Tutor. *International Journal of Artificial Intelligence in Education.*, *16*, 101–130.

Aleven, V., Sewall, J., McLaren, B. M., Koedinger, K. R. (2006). Rapid authoring of intelligent tutors for real-world and experimental use. *In Kinshuk, Koper, R., Kommers, P., Kirschner, P., Sampson, D. G., Didderen, W., (Eds.), Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006)*, 847–851.

Aleven, V., McLaren, B. M., Sewall, J., Koedinger, K. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. *In Ikeda, M., Ashley, K. D., Chan, T. W. (Eds.), Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)*, 61–70.

Aldrich, C. (2009). Learning Online with Games, Simulations, and Virtual Worlds: Strategies for Online Instruction. *John Wiley and Sons, 2009.*, 7–11.

Anderson, J. R. (1983). The Architecture of Cognition. *Cambridge, Massachusetts: Harvard University Press.*

Anderson, J. R., Conrad, F. G., Corbett, A. T. (1989). Skill acquisition and the LISP Tutor. *Cognitive Science.*, *13*, 467–506.

Arruarte, A., Fernandez-Castro, I. Greer, J. E. (1997). The IRIS Shell:How to Build ITSs from Pedagogical and Design Requisites. *International Journal of Artificial Intelligence in Education.*, *8*, 341–381.

Bell, B. (1998). Investigate and decide learning environments: Specializing task models for authoring tools design. *Journal of the Learning Sciences, Vol. 7. No. 1.*

Blessing, S., Gilbert, S. (2008). Evaluating an Authoring Tool for Model-Tracing Intelligent Tutoring Systems. *Proceedings of the 9th International Conf. on Intelligent Tutoring Systems*, 204–215.

Blessing, S. B. (1997). A programming by demonstration authoring tool for model tracing tutors. *Int. J. of Artificial Intelligence in Education. Vol. 8 , No. 3-4*, 233–261.

Blessing, S. B., Gilbert, S, Ritter, S. (2006). Developing an authoring system for cognitive models within commercial-quality ITSs. *In Proceedings of the Nineteenth International FLAIRS Conference.*

Blessing, S., Gilbert, S., Ourada, S., Ritter, S. (2007). Lowering the Bar for Creating Model-Tracing Intelligent Tutoring Systems. *In Proceedings of the 13th International Conference on Artificial Intelligence in Education.*

Blessing, S., Gilbert, S., Blankenship, L., Sanghvi, B. (2009). From SDK to xPST: A New Way to Overlay a Tutor on Existing Software. *Proceedings of the Twenty-Second International FLAIRS Conference.*

Blessing, S. B., Gilbert, S., Ourada, S., Ritter, S. (2009). Authoring model-tracing cognitive tutors. *The International Journal for Artificial Intelligence in Education, 19(2).*

Bruckman, A. (1997). Moose Crossing: Construction, community, and learning in a networked virtual world for kids. *PhD Dissertation, MIT Media Lab.*

Brusilovsky, P. (1998). Methods and Techniques of Adaptive Hypermedia. *In P. Brusilovsky, A. Kobsa, and J. Vassileva, editors, Adaptive Hypertext and Hypermedia, Chapter 1, Kluwer Academic Publishers, The Netherlands, 1998.*, 1–44.

Clancey, W., Joerger, K. (1988). A Practical Authoring Shell for Apprenticeship Learning. *Proceedings of ITS-88, Montreal*, 67–74.

Corbett, A. T. (2001). Cognitive computer tutors: Solving the two-sigma problem. *In the Proceedings of the Eighth International Conference of User Modeling.*

Corbett, A. T., Koedinger, K. R., Anderson, J. R. (1997). Intelligent tutoring systems. *In Helander, M. G., Landauer, T. K., Prabhu, P. (Eds.) Handbook of Human-Computer Interaction, 2nd edition Elsevier Science*, 849–874.

Crawford, C. (1984). The art of computer game design. *Berkeley, CA:Osborne/McGraw-Hill.*

Csikszentmihalyi, M. (1990). Flow: The psychology of optimum experience. *New York: Harper Perennial.*

Franklin, S., Graesser, A. (1996). Is it an agent, or just a program? A taxonomy for autonomous agents. *Proc. of the Third Intl Workshop on Agent Theories, Architectures, and Languages.*

Gilbert, S., Blessing, S. B., Kodavali, S. (2009). The Extensible Problem-Specific Tutor (xPST): Evaluation of an API for Tutoring on Existing Interfaces. *Proceedings of the 14th International Conference on Artificial Intelligence in Education.*

Gomez-Martin, M., Gomez-Martin, P., Gonzalez-Calero, P. (2004). Game-driven intelligent tutoring systems. *Proceedings of the Third International Conference on Entertainment Computing (ICEC).*, 108–113.

Hategekimana, C., Gilbert, S., Blessing, S. (2008). Effectiveness of using an intelligent tutoring system to train users on off-the-shelf software. *In McFerrin, K. et al. (Eds.), Proc. of Society for Info. Tech. and Teacher Education Intl Conf., AACE*, 414–419.

Hewes, J., Hills, M., Miyake, J., Sleeter, M. (1994). Creating interactive on-line instruction. *Professional Communication Conference, IPCC '94 Proceedings*, 446–451.

Hsieh, P., Halff, H., Redfield, C. (1999). Four easy pieces: Developing systems for knowledge-based generative instruction. *International Journal of Artificial Intelligence in Education.*

Johnson, W. L. (2009). A simulation-based approach to training operational cultural competence. *Proceedings of ModSIM, Virginia Beach, VA.*

Jong, T., VanJoolingen, W. R. (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research, Vol. 68 No. 2*, 179–201.

Kay, H. (1968). Teaching Machines and Programmed Instruction. *Penguin Books*

Kirriemuir, J., McFarlane, A. (2004). Literature reviews in games and learning. *Technical Report Report 8, Nesta FutureLab Series.*

Kirschner, P. A., Sweller, J., Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist, 41(2)*, 75–86.

Koedinger, K. R., Aleven, V., Heffern, McLaren, B. Hockenberry (2004). Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. *In the Proceedings of the Seventh International Conference of Intelligent Tutoring Systems, Maceio, Brazil.*

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.

Lajoie, S., Faremo, S., Wiseman, J. (2001). A knowledge-based approach to designing authoring tools: From tutor to author. *In Proc. of Artificial Intelligent in Education, J.D. Moore C. Redfield, L.W. Johnson (Eds). ISO Press.*, 77–86.

Laurillard, D. (1993). Rethinking university teaching: A framework for the effective use of educational technology. *Routledge.*

Livak, T., Heffernan, N. T., Moyer, D. (2004). Using Cognitive Models for Computer Generated Forces and Human Tutoring. *13th Annual Conference on (BRIMS) Behavior Representation in Modeling and Simulation. Simulation Interoperability Standards Organization. Arlington, VA.*

Major, N., Ainsworth, S., Wood, D. (1997). REDEEM: Exploiting symbiosis between psychology and authoring environments. *International J. of Artificial Intelligence in Education. Vol. 8 , No. 3-4.*, 317–340.

Major, N. P., Reichgelt, H (1992). COCA - A shell for intelligent tutoring systems. *In Frasson, C., Gauthier, G., McCalla, G.I. (Eds.) Procs. of Intelligent Tutoring Systems '92. New York: Springer-Verlag.*

Martin, B., Mitrovic, A., Suraweera, P. (2007). Domain modelling with ontology: A case study. *In Cristea, A., Carro, R.M., (Eds.) Proceedings of the 5th Int. Workshop on Authoring of Adaptive and Adaptable Hypermedia, User Modeling*, 4–11.

McQuiggan, S., Rowe, J., Lee, S., Lester, J. (2008). Story-based learning: The impact of narrative on learning experiences and outcomes. *Proceedings of the Ninth International Conference on Intelligent Tutoring Systems, Montreal, Canada.*, 530–539.

Merrill, M. D., ID2 Research Group (1998). ID Expert: A Second generation instructional development system. *Instructional Science, Vol. 26*, 243–262.

Munro, A., Johnson, M. C., Pizzini, Q. A., Surmon, D. S., Towne, D. M, Wogulis, J. L. (1997). Authoring simulation-centered tutors with RIDES. *International J. of Artificial Intelligence in Education. Vol. 8 , No. 3-4*, 284–316.

Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. *In Murray, T., Blessing, S., Ainsworth, S. E. (Eds.), Tools for Advanced Technology Learning Environments.*, 491–544.

Murray, T. (1997). Expanding the knowledge acquisition bottleneck for intelligent tutoring systems. *International J. of Artificial Intelligence in Education. Vol. 8 , No. 3-4*, 222–232.

Murray, T. (1998). Authoring knowledge-based tutors: Tools for content, instructional strategy, student model, and interface design. *Journal of the Learning Sciences, Vol. 7. No. 1.*, 5–64.

Murray, T. (1999). Authoring Intelligent Tutoring Systems: An analysis of the state of the art. *International Journal of AI in Education, 10.*, 98–129.

Parker, D. (2002). Show us a story: an overview of recent research and recourse development work at the British Film Institute. *English in Education, 36*, 38–45.

Ritter, S., Kulikowich, J., Lei, P., McGuire, C. L. Morgan, P. (2007). What evidence matters? A randomized field trial of Cognitive Tutor Algebra I. *In Hirashima, T., Hoppe, U., Young, S. S. (Eds.), Supporting Learning Flow through Integrative Technologies.*, *Vol. 162*, 13–20.

Ritter, S., Koedinger, K. (1996). An architecture for plug-in tutor agents. *Journal of AIED*, *7(3-4)*, 315–347.

Ritter, S., Blessing, S. B., Wheeler, L. (2003). User modeling and problem-space representation in the tutor runtime engine. *In P. Brusilovsky, A. T. Corbett , F. de Rosis (Eds.), User Modeling*, 333–336.

Ritter, S., Blessing, S. B. (1998). Authoring tools for component-based learning environments. *Journal of the Learning Sciences*, *7*(1), 107–131.

Roselli, R. J., Gilbert, S., Howard, L., Blessing, S. B., Raut, A., Pandian, P. (2008). Integration of an Intelligent Tutoring System with a Web-based Authoring System to Develop Online Homework Assignments with Formative Feedback. *American Society for Engineering Education Conference.*

Schewe, S., Reinhardt, B., Bestz, C. (1999). Experiences with a Knowledge Based Tutoring System for Student Education in Rheumatology. *In XPS-99: Knowledge Based Systems: Survey and Future Direction, 5th Biannual German Conference on Knowledge Based Systems, Lecture Notes in Artificial Intelligence 1570, Springer.*

Shaffer, D. W., Squire, K. D., Halverson, R., Gee J. P. (2005). Video games and the future of learning. *Phi Delta Kappan, 87(2)*, 104–111.

Skinner, B. F. Tecahing Machines. *Science, 128*, 969-977.

Sleeman, D., Brown, J. S. (1982). Introduction: Intelligent Tutoring Systems. *New York: Academic Press.*, 1–11.

Sparks, R., Dooley, S., Meiskey, L., Blumenthal, R. (1999). The LEAP authoring tool: supporting complex courseware authoring through reuse, rapid prototyping, and interactive visualizations. *International Journal of Artificial Intelligence in Education.*

Stottler, R. H. (2000). Tactical action officer intelligent tutoring system(tao its). *Proceedings of the Industry/Interservice, Training, Simulation and Education Conference (I/ITSEC).*

Suppes, P. (1967). Some theoretical models for mathematics learning. *Journal of Research and Development in Education*, *1*, 5–22.

Towne, D. M. (1997). Approximate reasoning techniques for intelligent diagnostic instruction. *International Journal of Artificial Intelligence in Education. Vol. 8 , No. 3-4*, 262–283

Uhr, L. (1969). Teaching machine programs that generate problems as a function of interaction with students. *Proceedings of the 24th National Confernece*, 125–134.

Van Marcke, K. (1998). GTE: An epistemological approach to instructional modeling. *Instructional Science, Vol. 26.*, 147–191.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Wintersgill, M. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence and Education.*, *15*,(3).

Virvou, M., Moundridou, M. (2001). Adding an instructor modeling component to the architecture of ITS authoring tools. *International Journal of Artificial Intelligence in Education 12(2).*, 185–211.

White, B., Frederiksen, J. (1995). Developing Metacognitive Knowledge and Processes: The Key to Making Scientific Inquiry and Modeling Accessible to All Students. *Technical Report No CM-95-04. Berkeley, CA: School of Education, University of California at Berkeley.*

Winne P. H. (1991). Project DOCENT: Design for a Teacher's Consultant. *In Goodyear (Ed.), Teaching Knowledge and Intelligent Tutoring. Norwood, NJ: Ablex.*