The Accidental Tutor: Overlaying an Intelligent Tutor on an Existing User Interface

Stephen B. Gilbert
Iowa State University
1620 Howe Hall
Ames, IA 50011-2274 USA
gilbert@iastate.edu

Stephen B. Blessing
University of Tampa
401 W. Kennedy Blvd. Box Q
Tampa, FL. 33606-1490 USA
sblessing@ut.edu

Liz Blankenship
University of Michigan
School of Information
1085 South University Ave.
304 West Hall
Ann Arbor, MI 48109-1107 USA
lizblank@umich.edu

Abstract
Intelligent Tutoring Systems (ITSs) have been shown to have dramatic impact on student learning [9]. However, these gains have been mostly in topics in which the interface has been designed with the intelligent tutor in mind. This research investigates the HCI challenges that result from creating two model-tracing ITSs for use with existing interfaces. We describe overlaying a tutor on an image-editing program and a web-based application. We highlight three main HCI challenges: 1) integrating a problem scenario in the context of the existing application, 2) providing learners with appropriate feedback during task performance, and 3) allowing learners to explore the interface while making sure they complete the task.

Keywords
Intelligent tutoring system, interaction design

ACM Classification Keywords
H.5.2. Graphical user interfaces; Interaction styles; Training, help, and documentation.

Introduction
In this research, intelligent tutoring systems (ITSs) are built on top of already existing third-party application interfaces for the purposes of software training. An ITS is a system which provides a learner with a series of problem scenarios to work through (tasks), provides feedback to the learner either during or after the learner's interaction with the system, and builds a model of the learner's mastered skills along the way.

The unique contributions of this research overall consist of 1) our technical approach to the under-investigated task of creating an ITS on top of existing third-party software applications that were not originally intended to enable tutoring, 2) validating our approach in two systems by increasing user competence more quickly [7,12], and 3) an analysis of the interaction design challenges that result from the "mashup" of an ITS and an original application, altering the original application's intended interaction patterns [6]. This paper focuses on the last component.

The cognitive model within an ITS can provide feedback after every learner interaction, tacit approval for positive progress, or for errors or explicit requests for help, the system may display a help message or highlight the mistake in some way. A decision in the interaction design of an ITS is how tightly to hold the learner to a successful path towards the goal vs. allowing exploration. Note that unlike "click-through" training videos, ITSs can allow the learner multiple correct paths to complete a task.

Model-tracing ITSs have been shown to be effective across a wide variety of learning domains (e.g., algebra, chemistry, physics and even English composition). Typical results indicate a 30% improvement on standardized tests such as the SATs and significant learning time reductions [4]. Many of these results, however, have used interfaces specifically designed with the tutor in mind.

Re-using the existing interface with a tutor reduces both the time required to develop the tutor and any issues of learning transfer. With some ITSs, researchers have had concerns about whether skills being learned in the ITS will transfer to the non-ITS environment [5]. 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 transfer largely disappear.

Related Systems: Help & User Assistance
Intelligent Tutoring Systems, interactive help and Computer-Assisted Instruction (CAI) programs have been built on top existing applications since the 1970s [13]. Unlike in many earlier systems, the tasks within xPST tend to be large in scope with multiple paths to successful completion, and the help and error messages are intended to provide more cognitively-based instruction, teaching the user an appropriate conceptual model of the software: not only what to click next, but why. xPST also has the potential to track a learner's skills across tasks, allowing for personalized instruction and monitoring of which skills are mastered.

Research Context
This research is based on two efforts to overlay intelligent tutoring systems on existing software interfaces that were not originally designed for tutoring. The first is Paint.NET, a standalone .NET application for

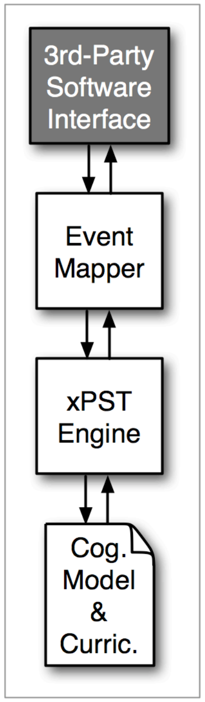


Figure 1. xPST system, an instantiation of the architecture of plug-in tutor agents described by Ritter & Koedinger in [12]. xPST is open source and available on Google Code.

Windows much like Adobe Photoshop. The Paint.NET tutor produced significantly better task performance than learners with "click-through" training videos [7]. The second is a web-application called the CAPE Web-Based Authoring Tool, created as part of the VaNTH Engineering Research Center, which we will call the CAPE tutor below. The web-based CAPE tutor reduced the time to complete a training task by an average of 14% and reduced user frustration [12]. While these experimental studies are discussed in detail elsewhere, this work focuses on the interaction design involved.

Intelligent Tutoring Architecture: xPST

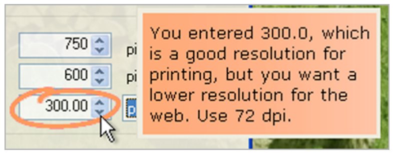
Ritter and Koedinger proposed an architecture for building a tutor for an existing interface and demonstrated two examples of using it in [11]. Our previous work [1, 7, 12] has been inspired by this approach and investigated the technical feasibility of instantiating an architecture that can accomplish similar goals more generally and achieve the results they foresaw with web-based tutoring. The overall architecture of xPST is illustrated in Figure 1.

The Cognitive Model 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 we need learning instruction is mapped to an object and has one or more rules associated with it. The rules contain the instructional feedback. The curriculum contains a set of tasks to complete. The Event Mapper eavesdrops on user actions and sends them to the xPST tutoring Engine, which checks them with the Cognitive Model. Relevant feedback is mapped back to the client UI control and displayed in situ. Note that while ITSs for academic topics like physics typically require a more complex cognitive model, so that learners can receive high-quality personalized feedback across a large number of similar physics problems, software training does not require such repetitive tasks, and the cognitive models are typically simpler and thus "problem specific."

The third-party software could be a stand-alone application or a website. If tutoring on a stand-alone application, the system can listen for user events in three ways: by using 1) widgets that automatically send the needed events (the method used in [11] with AppleEvents); 2) accessibility hooks built into the software (used frequently by screen readers 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 run on different servers.

Interface Design Challenges

The instructional design of an ITS is usually based on principles summarized by [2] that emphasize "learning by doing." The ideal learning environment includes doing a task that is relevant to the learner with scaffolding from a human tutor. A good human tutor accomplishes several interaction design feats that challenge a software-based tutor: human tutors 1) distinguish themselves from the software to be learned, 2) balance learner exploration with interruption for guidance, and 3) offer feedback with an appropriate balance of what to do next (procedural guidance) and why (conceptual guidance). The software-based tutor, the ITS, must accomplish these feats while being easy to use, effectively supporting the learning goals, and making up for any shortcomings in the usability of the underlying interface.



You entered 300.0, which is a good resolution for printing, but you want a lower resolution for the web. Use 72 dpi.

Figure 2. A Just-In-Time error message (JIT). The tutor circles the error and gives the message upon mouse-over.

The first concern, distinguishing the tutor from the software being tutored is necessary when the tutor is present only during a training context and absent during regular use (as with xPST), rather then deeply integrated and present at all times. A tutor for training that chooses the learner's goals is far easier to construct than a full-time integrated tutor, which requires inference of those goals during use of the software. To distinguish itself, the xPST tutor has a more casual look and feel than most software, e.g., using bright colors throughout and hand-drawn-looking coachmarks (see Figure 2). When tutoring on websites, the xPST task scenario appears in a separated sidebar.

The second concern is more complex: knowing when to interrupt and whether to take control. This issue exists with all ITSs, but when tutoring on existing software that is designed to be highly powerful with many menus and subdialogs, the issue becomes more complex; users often like to learn by exploring. Click-through training videos based on screencasts (e.g., Camtasia, Captivate) resolve this issue at one extreme: the learner has no control; she or he is restricted to follow a path dictated by the creator of the tutorial. A system that offers the learner full control but no help whatsoever is the other extreme.

Several design challenges arise from this control issue. Should the tutor disable certain features of the interface, making it easier to focus on relevant UI features that need to be learned (ala Training Wheels in [3]), or leave all features enabled, so that the learner can explore in an ad hoc fashion? If the latter, how much exploration should be allowed? Microsoft Windows Guided Help, for example, grays out all controls except the control that must be clicked next.

A second control-based issue arises from the existence of multiple correct paths to the goal, e.g. remove red-eye from an image and then reorient it, or reorient it first. The design challenge is whether to teach the learner a variety of methods or remain silent while the learner progresses to the goal, even on a suboptimal path. The xPST architecture encourages the latter approach, initially preparing hints towards the optimal path, but then queuing up hints along a sub-optimal but successful path if the learner has started along it.

Design Principles

Our ITS designs address the second and third challenges posed by the human tutor above by following several basic principles: A) leave features of the software intact (i.e., do not disable components); B) interrupt the learner as little as possible; C) give feedback in many small doses, so that an expert can be satisfied with a little but a beginner can find more detail; and D) only stop the learner if he or she is about to take a step that would lead down an irreparable path (e.g., deleting a key element). These principles still leave several options for reacting to errors. For example, the ITS could block a learner's action and say, "Sorry! This step would lead to…." That approach is highly invasive, however, and a preferable approach when possible is to allow the step but give the message, "Note that you have just… A better way is to…" This method allows the learner to fail (and thus learn), but informs why it is incorrect. Research from ACT-based model-tracing ITSs suggests that it is better to have the tutor intervene much sooner rather than later, and not have the student explore longer fruitless paths [5].

Specific Interaction Challenges

We now discuss the three specific design challenges mentioned previously: 1) integrating a problem scenario into the context of the existing application, 2) providing learners with appropriate feedback during task performance, and 3) allowing learners to explore the interface while making sure they complete the task.

First, in the software to be learned, the learner needs a way to initiate the tutor and choose a task. In both ITSs we designed so far, "Tutor" was added to a menu item to load a task list from the curriculum. The scenario and requirements of the selected task were then displayed in a sidebar in Firefox or in a floating task panel in Paint.NET. These sidebars contain not only a description of the task, but also some conceptual knowledge relating to the task.

Learners are given feedback when they take incorrect steps in the form of Just-In-Time error messages (JITs). JITs may be presented either in a modal state, forcing the user to click or make a key press to dismiss the message, or they may be non-modal, not requiring interaction before the user proceeds. In the case of Paint.NET, actions producing an incorrect state were often allowed to affect the image canvas, but a modal JIT forcing the user to undo would occur on top of the canvas (Figure 3). This approach allowed the user to make a mistake and see the consequences but still be guided back on path.

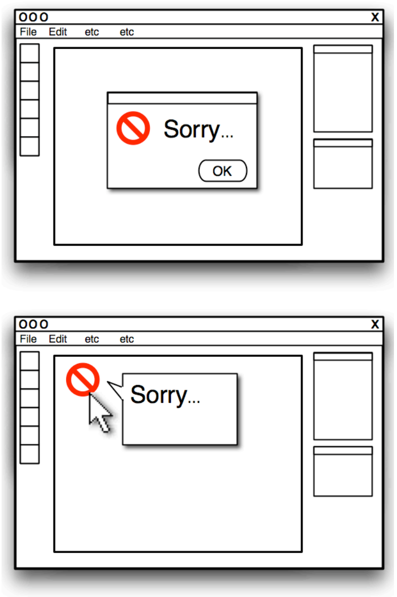


Figure 3. Just-In-Time error messages (JITs) can be modal (on top) if they require action or ambient, with feedback upon hover if not.

When building a tutor on top of an existing interface, much more than when the interface is designed with the tutor, the tutor must instruct the learner concerning the interface itself in addition to the domain knowledge. The cognitive model must therefore provide appropriate guidance to help users overcome any usability flaws in the underlying program, which the tutor author has no control over. For instance, with the CAPE tutor, several buttons with the same label "New" are often visible on the screen at the same time. If learners click the wrong one, the tutor blocks the action and appropriately directs them to the correct one.

The last challenge relates to allowing the learner to explore the system while still solving the task at hand. Blocking learner interactions is not as common in tutors with custom interfaces, because the interface is designed to accommodate tutoring and usually does not have a dramatic number of possible user actions. Constructing a tutor on an existing interface requires consideration of all the application states that can be reached, and the different paths to those states (e.g., via menu selections or keyboard equivalents). There is a trade-off between the amount of flexibility that can be allowed in the learner's execution of the task and the difficulty in creating a cognitive model that identifies and guides the user through these many possible application states. As stated previously, we advocate a "middle-of-the-road" approach where learners are allowed to explore, but if they attempt to do an irreversible step, we block that attempt. Also, to facilitate the exploration of menus, a learner was allowed to browse them as much as desired, even when the menus were not related to a possible next correct step.

In our testing of the CAPE tutor [12], one user who had used the CAPE tool before commented on the restrictive nature of the tutor, suggesting that the tutor was better suited for beginners. Possibilities for future research include increasing the amount of flexibility and providing more accurate feedback based on monitoring the user's skill level.

Summary

We have created xPST to investigate ways to increase the learnability of software by overlaying a tutor on existing interfaces, getting positive results from two initial efforts. These two systems demonstrate the viability of how we overcame three HCI challenges in overlaying an intelligent help and guidance system on existing software. This research can assist designers in integrating an original interface with a tutor to increase learning transfer. Future work will extend xPST to game-based environments, which are faster-paced with more state variables than a productivity application. Separate but parallel work investigates the usability of xPST authoring tools by novices.

This work was supported in part by the National Science Foundation under OII-0548754 and EEC-9876363.

Citations

[1] Blessing, S. B., Gilbert, S, & Ritter, S. Developing an authoring system for cognitive models within commercial-quality ITSs. *Proc. of the 19th Int'l FLAIRS Conference*, AAAI Press (2006), 497-502.

[2] Bransford, J.D., Brown, A.L. & Cocking, R.R. *How people learn: Brain, mind, experience, and school*. Washington, DC: National Academy Press, 2000.

[3] Carroll, J. & Carrithers, C. (1984) Training wheels in a user interface. *Comm. ACM* 27(8), 800-806.

[4] Corbett, A.T. Cognitive computer tutors: Solving the two-sigma problem. *User Modeling: Proc. of the 8th Int'l Conf*, UM 2001, 137-147.

[5] Corbett, A., Koedinger, K., & Anderson, J. Intelligent tutoring systems. Helander, Landauer & Prabhu, (Eds.) *Handbook of Human-Computer Interaction (2nd),* Elsevier Science, (1997), 849-874.

[6] Hartmann, B., Doorley, S., & Klemmer, S.R. Hacking, Mashing, Gluing: Understanding Opportunistic Design. *Pervasive Computing*, 2008.

[7] Hategekimana, C., Gilbert, S. & Blessing, S. Effectiveness of using an intelligent tutoring system to train users on off-the-shelf software. In K. McFerrin et al. (Eds.), *Proc. SITE 2008*., AACE (2008), 414-419.

[8] Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. Intelligent tutoring goes to school in the big city. *Int'l J. of AI in Education*, (1997), 30-43.

[9] Livak, T., Heffernan, N. T., Moyer, D. (2004). Using cognitive models for computer generated forces and human tutoring. *13th Ann'l Conf. on Behavior Representation in Modeling and Simulation*.

[10] Murray, T. Authoring Intelligent tutoring systems: Analysis of the state of the art. *Int'l J. of AI in Education*, 1999, 98-129.

[11] Ritter, S., & Koedinger, K. (1996) An Architecture for Plug-in Tutor Agents, *J. of AIED*, 7(3-4) 315-347.

[12] 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. *Proc. ASEE 2008.*

[13] Shute, V. & Psotka, J. (1996) "Intelligent tutoring systems: past, present, and future," *Handbook of Research on Educational Communications and Technology* , Macmillan, New York, 570-600.