From SDK to xPST: A New Way to Overlay a Tutor on Existing Software

Stephen B. Blessing[1], Stephen B. Gilbert[2], Liz A. Blankenship[3], Bhavesh Sanghvi[2]

[1]University of Tampa, [2]Iowa State University, [3]University of Michigan
[1]401 W. Kennedy Blvd., Tampa, FL 33606; [2] 1620 Howe Hall, Ames, IA 50011
[3] 1085 South University Ave., 304 West Hall, Ann Arbor, MI 48109
sblessing@ut.edu, gilbert@iastate.edu, lizblank@umich.edu, bsanghvi@cs.iastate.edu

Abstract

Our past work has investigated the use of the Cognitive Model Software Development Kit (SDK) for creating the cognitive models that underlie model-tracing Cognitive Tutors. Though successful at increasing the number of people who could author such a cognitive model, for certain kinds of situations the Cognitive Model SDK proved cumbersome. The present work discusses a new authoring system, xPST, that allows an example-based tutor to be built on top of existing software. xPST-based tutors have been built for two real-world systems that had existing interfaces.

Starting Point: The Cognitive Model SDK

We have met with success at developing a SDK for cognitive models (Blessing & Gilbert, 2008). The Cognitive Model SDK allowed authors to develop the representations necessary for the cognitive models for model-tracing intelligent tutors. Carnegie Learning, Inc. uses this SDK to create their Cognitive Tutors for math, a commercially successful intelligent tutoring system.

The Cognitive Model SDK enables the creation of cognitive models that contain abstracted instruction over instances. However, we have struggled with its use in other situations. In two of the tutors we had built, for example, the authors created a lot of declarative and procedural representations that ultimately received very little use. For example, Hategekimana, Gilbert, and Blessing (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 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.

This material is based upon work supported by the National Science Foundation under Grant No. EEC-9876363.

Likewise, when we created 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 8 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. Ultimately, the effort spent developing those representations seemed disproportionate to their usefulness in the completed model. What we 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 result has been the Extensible Problem Specific Tutor (xPST) authoring system.

The xPST Authoring System

The xPST Authoring System shares some in common with another authoring tool, the Cognitive Tutor Authoring Tool (CTAT; Aleven et al., in press). CTAT allows authors to create Example-based Tutors. These tutors are specific to a certain problem or a narrow class of problems. Likewise, an xPST tutor is specific to a certain problem scenario. The intent of both the CTAT and xPST authoring tools is to allow the author to quickly create a model for a particular problem instance by creating hints and other tutoring aspects while the author manipulates the interface itself.

The other goal of xPST is to allow tutoring on any interface. Whereas it is possible to do so with CTAT in some circumstances, typical tutors are built with CTAT-specific widgets. The power of xPST's approach should be apparent. It will enable the creation of a model-tracing-like tutor to be created for any piece of software, opening up possibilities in both academic and commercial settings.

The architecture used by xPST to communicate with third-party software is similar to that used by the Cognitive Tutor SDK to communicate with off-the-shelf software. This architecture, called TutorLink, is shown in Figure 1.

TutorLink serves as the intermediary between the third-party application and the Tutoring Agent (xPST in the current case). It knows how to map actions in the interface

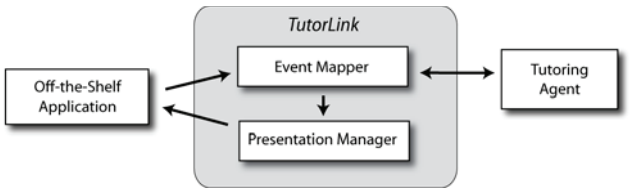

Figure 1. TutorLink Architecture.

to the proper pieces in the tutor model and how to display hints and other tutoring information within the application. TutorLink is what makes xPST extensible. Agents can be written for TutorLink that allow it to communicate with different kinds of off-the-shelf applications. When we did the Paint.NET tutor, a piece was written that allowed it to communicate with the .NET framework. When we did the CAPE tutor, a Firefox plugin was written that allowed it to communicate to most webpages. Writing an agent takes expertise, but once it is done it will open up new interfaces. We have had initial success with an agent that communicates with any software on Windows using the default MFC widget set by using the accessibility hooks.

In Spring 2008 we developed a tutor to assist instructors learning how to create online assignments using VaNTH's web-based authoring tool based on CAPE technology. This tool allows instructors to author rich online learning exercises. It is entirely web-based. A training workshop had been developed in order to assist people in learning how to use the tool. However, the tool's creators wanted something to reduce the learning curve that still existed in learning how to use it. To that end, we partnered with them to create an ITS on top of their existing training exercises.

The exercises consisted of 8 scenarios and extensions that walked the instructor through various aspects of creating online content with the web-based authoring tool. In a workshop prior to the release of our ITS, only 2 of 8 students completed the training exercises, with many people reporting that the tool was difficult to use.

We created an ITS with the SDK that provided tutoring for each step in these exercises. Results from a workshop using this ITS were encouraging, with participants completing more exercises. The two developers were both first year graduate students, one who had SDK experience and the other did not. Together they created all instruction. The fact that these people (i.e., largely ITS author neophytes, though one did assist on the Paint.NET tutor) were able to create such rich instruction is testament to the power of the SDK. As mentioned, though, we were still dissatisfied regarding their progress and began to think of alternative ways to author these kinds of tutors (i.e., tutors based around a small set of problem scenarios, with not a lot of model re-use between the problems).

The resulting authoring tool is xPST, a streamlined authoring process for specific problem scenarios (http://code.google.com/p/xpst/). Like in CTAT the result is a tutor containing specific hints and just-in-time messages attached to particular interface elements at various points in the problem's solution. Unlike typical CTAT tutors, the interface elements are not widgets created specifically for the authoring tool.

There are three steps in creating a xPST-based tutor, all of which can be done in an online authoring tool. First, the author maps interface elements to the subgoals needed to solve the problem. For example, one step in solving a particular problem may be to indicate an answer via a radio button. The author can map the name the interface has for that radio button to a name more indicative of the student's goals. Second, those problem subgoals can be sequenced in the order needed to solve the problem. The xPST syntax allows for different means of ordering these subgoals (e.g., goals that need to be solved in sequence, goals that are unordered, and choices between various goal sequences). Lastly, hints and just-in-time messages, along with the correct answers are attached to these learning subgoals.

One of the original graduate students transitioned the VaNTH tutor to xPST. While it is hard to draw conclusions on the comparison, the original development took around 250 hours, and the transition took 25 hours. The model development was reportedly more straightforward and streamlined, with debugging much easier. The two tutors are functionally identical.

Conclusions

xPST allows for the fast creation of a model-tracing-like tutor for a particular problem scenario. The architecture can support an xPST tutor communicating with third-party interfaces. We transitioned one SDK tutor to an xPST tutor relatively quickly and with only a small effort. A current study is examining how novice xPST authors, even those with little to no programming experience, can learn to create these kinds of tutors.

References

Aleven, V., McLaren, B.M., Sewall, J., & Koedinger, K.R. (in press). Example-tracing tutors: A new paradigm for intelligent tutoring systems. Special Issue on Authoring Systems for ITSs *International Journal of Artificial Intelligence in Education*.

Blessing, S. B., & Gilbert, S. (2008). Evaluating an authoring tool for model-tracing intelligent tutoring systems. In *Proceedings of the ITS 2008 Conference* (pp. 204-215), Montreal, Quebec. Berlin: Springer-Verlag.

Hategekimana, C., Gilbert, S. & Blessing, S. (2008). Effectiveness of using an intelligent tutoring system to train users on off-the-shelf software. In K. McFerrin et al. (Eds.), *Proceedings of Society for Information Technology and Teacher Education International Conference 2008* (pp. 414-419), Las Vegas, NV. Chesapeake, VA: AACE.

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 2008 Conference.