A Historical Perspective on Authoring and ITS: Reviewing Some Lessons Learned

Benjamin D. Nye[1] and Xiangen Hu[1,2]
[1]University of Memphis
[2]China Central Normal University
{bdnye,xhu}@memphis.edu

Introduction

This section discusses the practices and lessons learned from authoring tools that have been applied and revised through repeated use by researchers, content authors, and/or instructors. All of the tools noted in this section represent relatively mature applications that can be used to build and configure educationally-effective content. Each tool has been tailored to address both the tutoring content and the expected authors who will be using the tool. As such, even tools which support similar tutoring strategies may use very different interfaces to represent equivalent domain knowledge. In some cases, authoring tools even represent offshoots where different authoring goals led to divergent evolution of both the authoring tools and the intelligent tutoring systems (ITSs) from a common lineage. Understanding how these systems adapted their tools to their particular authoring challenges gives concrete examples of the tradeoffs involved for different types of authoring. By reviewing the successes and challenges of the past, these chapters provide lessons learned for the development of future systems.

Authoring Tools for Adaptive and Data-Driven Systems

In general, for ITS authoring tools, discussion often centers on tools for creating content, such as new problems or new dialogs that interactively help the learner step-by-step. While these are a key part of the authoring process, mature authoring tools tend to cover a wider array of authoring and configuration options. These activities range from small activities like selecting HTML pages to larger tasks such as manually selecting or sequencing curriculum topics. In other cases, the problem is not so much authoring as versioning: maintaining and updating content in a reliable way. Within this section, all of these activities will be considered as facets of the larger authoring lifecycle.

This lifecycle typically includes the following steps: 1) Creating initial content module (e.g., a problem), 2) Interacting with module like a student, 3) Revising the module, 4) Selecting and composing modules for inclusion in a given curriculum, 5) Collecting data on student interaction, and 6) Revising module based on collected data. From the standpoint of content quality, each of these steps contributes to development of effective tutoring and learning. Efficient tools for certain stages of this lifecycle may be less effective for other stages. For example, while a series of simple may be efficient for entering the initial content, that same interface would not necessarily make it easy to find and correct a specific field during the revision step. As such, all systems must make choices about the authoring activities that receive the most support, often based on the types of expected authors. With this in mind, the chapters in this section describe a variety of approaches to authoring.

In Chapter 3, Blessing, Aleven, Gilbert, Heffernan, Matsuda, & Mitrovic discuss different approaches to "Authoring Example-based Tutors for Procedural Tasks." This paper discusses the convergence of multiple lines of authoring tools for step-based problem solving tutors toward example-based authoring. Example-based authoring, also sometimes called instance-based authoring, provides an interface where the author builds tutoring content and student support (e.g., hints) for an individual example or limited class of parameterized examples. By comparison, traditional authoring techniques often required implementing a full set of explicit domain rules. A number of advantages for such tutors are provided, which are evident in the authoring tools presented. For some systems, such as ASSISTments and Cognitive Tutor Authoring Tools (CTAT), this approach was chosen to lower barriers to authoring so that instructors could develop ITS content. For other systems, such as xPST (Extensible Problem-Solving Tutor), the approach allows tightly integrating tutoring with a wide variety of content, ranging from 3D games to web pages. Finally, in systems such as ASPIRE and SimStudent, algorithms are used to generalize domain rules and constraints that enable the ITS to tutor a wider variety of problems than were explicitly authored. Particularly since domain content experts are much more likely to be able to author examples than create formal representations of their rules, this approach is appealing for well-defined procedural tasks.

In Chapter 4, Matuk, Linn, and Gerard describe the authoring capabilities of the Web-based Inquiry Science Environment (WISE) system. While WISE is does not currently focus on adaptive elements, the system has a strong focus on both theory-based (the Knowledge-Integration framework) and data-driven development and revision of content. This system demonstrates the potential reach of a well-designed system designed around teachers, with over 10,000 teachers registered to use WISE. Their main principles are to provide tools accommodate a range of abilities, allow users to reuse, revise, and extend what others have made, reporting student data as evidence to inform revision, and allowing flexibility for authors to repurpose the system for their goals. Compared to many authoring systems, WISE strongly supports later parts of the authoring lifecycle (i.e., selecting content and data-driven revision).

In Chapter 5, Jacovina, Snow, Dai, and McNamara describe the authoring tools for iSTART-2 and Writing Pal. These systems use natural language processing techniques to support reading comprehension strategies and essay-writing skills, respectively. Authoring tools within these systems are novel in a few ways. First, the tools explicitly contain distinct features that are intended for researchers (e.g., randomizing the use of a certain feedback strategy) versus for teachers (e.g., modifying or selecting content). In general, authoring in these systems attempts to mirror the student experience with the system but with buttons to edit content or behavior. Second, the tools are being designed to allow authoring behavior that is associated with stealth assessments, such as feedback or experimental activities. Compared to other systems in this section, this work explores the potential for collecting and applying rich metrics on student behavior (e.g., the narrativity of a student's essays).

In Chapter 6, Charlie Ragusa outlines the design principles of the GIFT authoring tools, which are currently being used by multiple groups to integrate tutoring into environments as varied as 3D worlds and PowerPoint presentations. A major focus of this chapter is the need and development of collaborative authoring tools: frameworks that allow multiple authors with complementary expertise to contribute effectively. These processes are essential, since the knowledge needed to author an ITS tends to be spread across multiple experts.

Finally, in Chapter 7, Steve Ritter describes practices related to authoring and refining ITS content across the lifecycle of a commercial product, based on practices used by the widely-used Cognitive Tutor system. This chapter focuses significantly on methods to leverage student data to improve an ITS over time. The discussion revolves around the types of changes that are often necessary (e.g., parameters, design of the tasks, content) and methods to determine the changes (e.g., manually, automatically calculated, crowdsourced). Versioning issues are noted with data-driven models, such as data becoming less-applicable if the design of the task has changed. Also, suggestions are made for which types of changes are best-suited for certain methods (e.g., certain parameter changes can be automatically rolled out). These issues reflect the realities of balancing data-driven design with a regularly-used product that must also behave reliably for users on a day-to-day basis.

Themes and Lessons Learned

Across these chapters, some common themes emerged for systems that have matured to reach wider user bases. Strong themes included:

1. User-Centric Design: Authoring tools that are tailored for the specific authors who are intended to use them. In some cases, building multiple tools that serve qualitatively different types of authors. Both systems with wide user bases of authors (ASSISTments and WISE, both with >1k teachers) strongly focused on serving the common needs of teachers, which include being able to modify and add content. This was also a significant theme for multiple other systems (e.g., iSTART-2).
2. Workflows: In some cases, multiple tools and qualitatively different approaches are used to build, refine, and enhance different parts of a system. The GIFT discussion focuses extensively on collaborative authoring. The Cognitive Tutor product lifecycle discussion also describes a multi-faceted authoring process.
3. Constraints: Authoring tools constrain the author (by design). For each of the systems with large student user bases (Cognitive Tutor, ASSISTments, and WISE, all with > 75k students), authoring and configuration was often significantly constrained. In many cases, this was to simplify the authoring process. However, systems may also attempt to limit certain types of configurations or authoring that are not pedagogically sound within the system. This raises the issue that sometimes the options that are not given for authoring can be as important as those that are.
4. Content vs. Adaptivity: Different authoring tools and processes emphasize different parts of the content authoring cycle, with systems for teachers tending to support simple content creation revision (WISE, iSTART-2 for teachers, ASSISTments) and systems with stronger use by the research community providing more tools for training step-based adaptivity (CTAT, SimStudent, GIFT, ASPIRE).
5. WYSIWYG (What You See Is What You Get): Nearly all of the systems in this chapter describe methods to quickly view the content after it is authored, incrementally and iteratively (CTAT, SimStudent, xPST, ASSISTments, iSTART-2, WISE, ASPIRE). By allowing authors to see what they are creating in real-time, these tools enable a more direct authoring process.
6. Generalization Algorithms: While some of these systems use complex formal representations (e.g., ontologies, production rules), the field has taken steps toward authoring using examples. As

such, research on methods to identify general principles or rules from examples has become an important topic (SimStudent, ASPIRE).

7. Versioning and Maintaining Content: For systems with large user bases, these chapters touched on the complexities and advantages of maintaining a large system, such as supporting modified content, tracking its evolution, and retaining only content with signs of effectiveness evident in the student data (Cognitive Tutor and WISE).

Based on these lessons learned, a few areas of focus emerge. First, support for example-based authoring and other WYSIWYG approaches is probably essential to help instructors author new ITS-tutored activities. Second, collecting and presenting centralized data about an existing repository of tutoring modules (such as GIFT's Domain Knowledge Files) could significantly improve the ability and confidence of authors trying to select tutoring for an activity. This data could also be used for versioning that tracks, maintains, and prunes the set of recommended tutoring modules over time (an issue that is explored in Chapter 6). Finally, this work implies that multiple authoring interfaces are needed to support the research community versus instructors. With these shifts, GIFT could expand its user base and also increase the effectiveness of content over time. More generally, these are lessons that authoring tools for ITS and other learning technologies should follow to ensure that their systems are easier to author, effective for learners, and can be revised and maintained over time.