Chapter 25 Lowering the Technical Skill Requirements for Building Intelligent Tutors: A Review of Authoring Tools

H. Chad Lane[1], Mark G. Core[2], Benjamin S. Goldberg[3]
[1]University of Illinois, Urbana-Champaign, [2]University of Southern California, [3]US Army Research Laboratory

Introduction

Educational technologies have come to play an important role in advancing the science of learning. By consistently applying a set of pedagogical policies (and not getting tired while doing so), educational technologies can be used to address precise questions about how people learn and how to best help them. The resulting findings often answer important questions about human learning, which, in turn, can positively influence the design of future educational technologies or even possibly educational practices. A second way learning science researchers seek to have impact is by getting the technology in the hands of as many learners as possible. Unfortunately, with more users come more requirements, and therefore, additional questions educational software designers need to address. For example, can a system be tailored to the specific needs of a class, teacher, or individual learner? Can it be used in a new task domain? Is it possible to reorganize or create new course content? Can the pedagogical approach and/or content embedded in the system be adjusted or even replaced? Sadly, but understandably, software that is created for lab studies or specific end-user needs do not often address these questions. If the aim is to "go big," then it is no longer feasible to create one system suited for all needs tools for configuring and creating content are a requirement.

In this chapter, we focus on intelligent tutoring systems (ITSs), an instance of educational technology that is often criticized for not reaching its full potential (Nye, 2013). Researchers have debated why, given such strong empirical evidence in their favor (Anderson, Corbett, Koedinger & Pelletier, 1995; D'Mello & Graesser, 2012; VanLehn et al., 2005; Woolf, 2009), intelligent tutors are not in every classroom, on every device, providing educators with fine-grained assessment information about their students. Although many factors contribute to a lack of adoption (Nye, 2014), one widely agreed upon reason behind slow adoption and poor scalability of ITSs is that the engineering demands are simply too great. This is no surprise given that the effectiveness of ITSs is often attributable to the use of rich knowledge representations and cognitively plausible models of domain knowledge (Mark & Greer, 1995; Valerie J. Shute & Psotka, 1996; VanLehn, 2006; Woolf, 2009), which are inherently burdensome to build. To put it another way: the features that tend to make ITSs effective are also the hardest to build. The heavy reliance on cognitive scientists and artificial intelligence (AI) software engineers seems to be a bottleneck.

This issue has led to decades of research geared toward reducing both the skills and time to build intelligent tutors. The resulting ITS *authoring tools* serve different educational goals, but generally seek to enable creating, editing, revising, and configuring the content and interfaces of ITSs (Murray, Blessing & Ainsworth, 2003). A significant challenge lies in the accurate capture of the domain and pedagogical expertise required by an ITS, and many authoring tools focus on eliciting this knowledge. Unfortunately, as ITS technology has evolved, the authoring burden has increased rather than decreased, and so the tension between usability of an authoring tool and the sophistication of the target ITS knowledge representation is substantial.

In this chapter, we focus on the problem of *reducing* the technical knowledge required to build ITSs (only one of many possible goals for authoring tools). We review the important historical attempts that most directly sought to open up the creation of ITSs to nonprogrammers (like educators) as well as more recent work that addresses the same goal. We review popular approaches, such as programming by

demonstration, visualization tools, and what you see is what you get (WYSIWYG) authoring, and summarize the limited experimental evidence validating these approaches. The central questions driving this review are (1) In what ways have researchers sought to make authoring more intuitive and accessible to nonprogrammers? (2) For what purposes have these tools been developed? (3) What components of an ITS have they addressed? and (4) How have researchers evaluated these approaches and what have they learned about intuitive authoring? The chapter ends with suggestions for future research, including identifying ways to empirically understand the sophistication vs. ease-of-authoring tradeoff, leveraging more findings from the human-computer interaction (HCI) community, and addressing the glaring gap in authoring research as it relates to learning management systems (LMSs).

The Problem

What makes ITSs so difficult to create? ITSs provide the fine-grained support necessary for deep learning unlike most traditional computer-aided instructional systems (VanLehn, 2011), but this ability requires greater complexity (VanLehn, 2006). We use the Generalized Intelligent Framework for Tutoring (GIFT) architecture, shown in Figure 1, as a general model of the complex system of interconnected components typically present in an ITS, and the end product of ITS authoring. The specific requirements, roles, and complexities of each component are described elsewhere (Sottilare, 2012), but some of the more prominent components in terms of authoring include the following:

- The **learner module** tracks the learner's state over the course of a session. The learner module updates performance measures based upon assessments of learner activities, and may estimate changes in understanding, acquisition of skills and learner emotions.

- The **pedagogical module** makes the instructional decisions that drive the tutor's behavior. These decisions can vary in scope from topic and problem selection to deciding whether to give feedback and the content and style of this feedback.

- The **domain module** handles domain-specific responsibilities such as assessing learner problem solving and providing help. This module may use general information about the target task and any associated simulation as well as rely upon problem-specific data.

- The **tutor-user interface** provides the communication channel(s) between learner and tutor, such as speech, text, visualizations, etc. It defines the scope of potential tutoring interactions.

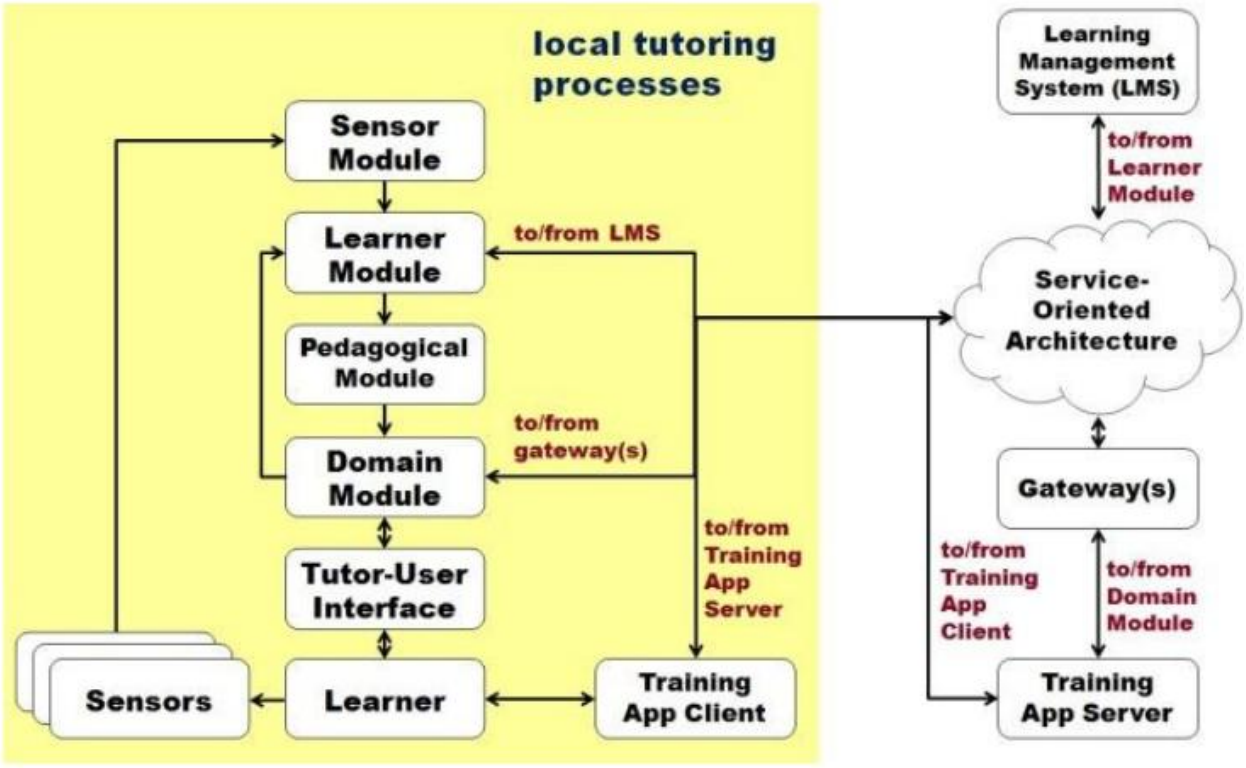

Figure 1. Overview of the GIFT architecture (Sottilare, 2012).

It is typical for authoring tools to focus on specific components that are most important or relevant for the target ITS. This approach also makes the process more viable as authors focus their attention on a few components while the rest may remain unchanged from problem to problem, or domain to domain. By starting with the GIFT framework, which seeks to maximize generality, rather than a specific system, the full range of possible targets for an ITS authoring tool is more apparent.

ITS Authoring Tools: Goals And Tradeoffs

In two broad and thorough reviews of the field (Murray, 1999, 2003), two (of the many) take-away messages are that authoring tools (1) have been developed with a wide variety of goals in mind and for many different categories of users, and (2) present a huge space of tradeoffs both in terms of their own implementation and those that must be addressed by authors using the system. Authoring success stories, such as Cognitive Tutor Authoring Tools (CTAT) (Aleven, McLaren, Sewall & Koedinger, 2006) and REDEEM (Ainsworth et al., 2003), always impose a reasonable level of constraints on their authors and make assumptions so that tasks can stay manageable.

Murray (2003) identifies five typical goals for authoring tools, roughly in order of importance or predominance (p. 509):

(1) Decrease the effort required to build an ITS (e.g., time, cost).

(2) Decrease the "skill threshold" for building ITSs.

(3) Support authors in the articulation of domain or pedagogical knowledge.

(4) Enable rapid prototyping of ITSs.

(5) Rapidly modify and/or prototype with the aim of evaluating different approaches.

Systems in category (1) can include those built for cognitive scientists and programmers. For example, CTAT (Aleven, et al., 2006) includes two primary methods for tutor creation, the first of which involves creating and debugging production rule-based cognitive models directly. CTAT includes a variety of tools for organizing, testing, watching, and editing cognitive models, all designed for users with high levels of technical skill. The second type of authoring uses *example-tracing*, which is described below and more directly addresses Murray's second goal.

Category (1) stands in direct contrast to (2), which involves lowering the bar on what authors need to know or be able to do. Typically, authoring tools that seek to do this have the aim of allowing teachers, subject-matter experts, and other educators to create ITSs to address their own needs. Category (2) is the focus of this chapter how have researchers attempted to "simplify," or at least remove some of the more technically onerous aspects of building ITSs? Goals in categories (3) through (5) are in many ways orthogonal to (1) and (2). Articulating domain and pedagogical knowledge (3) is a requirement for ITSs and can be accomplished with no specialized tools, tools for technically skilled authors, or tools for nonprogrammers. Similarly, the rapid creation of ITSs and variations on existing ITSs for the purposes of testing or running experiments can also be accomplished regardless of the tools used.

Of course, whatever purposes an authoring tool is intended to serve will directly impact the design. In turn, the design of any content-creation tool (e.g., word processors, presentation software) involves tradeoffs. In the case of ITS authoring tools, a variety of tradeoffs are apparent. One tension is that complexity in one area of authoring can introduce difficulties in other areas. For example, a tool could allow authors to create a custom graphical user interface (GUI) for their ITS. However, the underlying ITS has no idea what the controls (e.g., text boxes and menus) of the GUI mean, Thus, either the author has to develop a knowledge representation and link it to the GUI, or develop a model in terms of GUI actions (e.g., example-tracing in CTAT). A second tension that arises is between the complexity of an authoring tool and ease of use. For example, the full power of cognitive modeling is available in CTAT, but requires programming skills and the resulting cognitive models can be onerous. In general, the greater the expressive power provided by an authoring system, for any module, the more complicated other components become to build. Therefore, we find a distinct tradeoff between this expressive power and the ease at which authoring can be accomplished.

In the remainder of this chapter, we focus our attention on techniques researchers have used to reduce the skills needed in order to build ITSs (category (2) from the list above). A tradeoff made in most of these examples is increased authorability but reduced sophistication of the underlying tutors that are built. In other words, the assumptions made and steps taken to "simplify" the authoring process have generally led to simpler models. This is not necessarily true in all of the cases below, however. SimStudent (Matsuda et al., 2013) and Authoring Software Platform for Intelligent Resources in Education (ASPIRE) (Mitrovic et al., 2009), for example, use machine learning techniques to infer more than what the authoring activities provide on the surface. Another common tradeoff is simplifying the process by limiting what the author can create or change. For example, REDEEM (Ainsworth, et al., 2003) uses a "courseware catalogue" as a starting point, and SitPed (Lane, Core, et al., in-press) starts with pre-constructed scenarios.

Approaches to Building Intuitive Authoring Tools

The AI-heavy components of an ITS (domain module, pedagogical module, learner module) generally require detailed information from authors. This type of ITS authoring is similar to the task of *knowledge engineering*, which was widely recognized as onerous and a possible impediment to the growth of expert systems. The general challenge of capturing or eliciting knowledge from subject matter experts was recognized early, leading to a great deal of research focused on the knowledge elicitation problem (Hoffman, Shadbolt, Burton & Klein, 1995). ITSs share this problem, but with the additional burden of needing to address issues related to pedagogy that is, how to present information, assess learners' knowledge, deliver feedback, and so on. Thankfully, ITSs often do *not* require such heavy-duty models as fully elaborated expert systems, and solutions that go beyond basic computer-aided instruction but not as far as a fully-fledged ITS still have a great deal of value. For example, the ASSISTments approach provides teachers with authoring tools to create, share, and deploy step-by-step example problems that approximate ITS behaviors by providing step-level support for a learner without the need for traditional student modeling, expert modeling, and so on (Heffernan & Heffernan, 2014).[7]

Although many authoring tools indirectly lower the skill threshold needed to build a tutor, either through hiding implementation details or automating steps, the systems included in this review are systems that prioritize usability. A requirement for inclusion is that a system be explicitly built for and tested with nonprogrammers who are either instructors or subject-matter experts in an educational role. A second requirement is that the authoring tool supports the population of ITS-like components (see Figure 1) via interaction with the author. The end result of the authoring process is a learning system that performs at least some of the behaviors from the standard definition of a tutoring system (VanLehn, 2006).

A final dimension we highlight is Murray's (2003) distinction between *performance* and *presentation* roles that an ITS can play. An ITS that focuses on performance typically assesses the learner during problem solving (or other form of practicing a skill) and provides feedback and scaffolding. This is perhaps the most common role of an ITS given that they often help learners during homework. An authoring tool that addresses the presentation of content is geared toward a more direct instructional role, such as that played by educational videos in massive open online courses (MOOCs) or flipped classrooms. The presentation of content can be made highly adaptive based on learner behaviors and embedded assessments, and so there are many opportunities to go beyond what simple videos or reading can achieve. Historically, many ITSs sought to play both roles and use shared models between the two to increase the level of personalization (Brusilovsky, 2012). The distinction between authoring for performance or presentation (or both) is used in the discussion that follows.

Authoring for Content Delivery

REDEEM (Ainsworth, et al., 2003) represents one of earliest and most successful attempts to put authoring tools in the hands of teachers. Using intuitive interfaces, REDEEM walks authors through a workflow that operates primarily on existing content so that a generated ITS can later present it adaptively to learners. In addition, authors can increase interactivity in the resulting lessons by creating questions and identifying "reflection points" that allow the system to know when students should spend more time processing the material. REDEEM most directly supports non-technical authors in the following ways:

1. REDEEM provides a well-defined *workflow* with integrated stages that are all clearly defined for authors so that they understand the overall process and end result.

[7] Although ASSISTments does include elaborate records of student performance used in service of helping instructors assess their students' learning.

2. It adopts a slider-based approach to allow authors to specify parameters such as suitability of resources for learners, and amount of student choice.

3. Instructional strategies are expressed in ways that are familiar to authors (who are instructors).

The REDEEM workflow takes authors through three distinct phases: (1) describe the course material by organizing and marking content with structural annotations, (2) define the kinds of learners who will use the resulting ITS, and (3) describe how the system should teach those learners with the content articulated during step 1. REDEEM assumes the availability of a "courseware catalogue," which consists primarily of reading content. For this content, authors are asked to identify sections, label the content in sections along various dimensions (e.g., how difficult or familiar it will be to students), and finally describe relations between sections (e.g., section B is an application of the concept described in section A). These relationships help the system build a semantic representation of the course content, which is then used by the resulting ITS to adapt instruction. In addition, authors also create interactive content such as multiple choice and fill-in-the-blank questions. The second step for REDEEM authors is to create a base of learner types (based on the author's discretion), while the third is the "glue" that brings these steps together.

To complete their ITS, authors must define a body of tutoring *strategies* that ultimately tell a REDEEM ITS how to use the annotated content. A slider metaphor is used to configure the details of the tutoring strategies. For example, if a teacher wants to allow the ITS to engage in practice with the learner, they can define this strategy by using a series of sliders to set the parameters appropriately (Figure 2). The strategy includes information about when to use it, how to deploy it, and how to provide help. Although there is a cap of 20,000 different strategies, authors tend to create about 7 per tutor (Ainsworth, et al., 2003, p. 213).

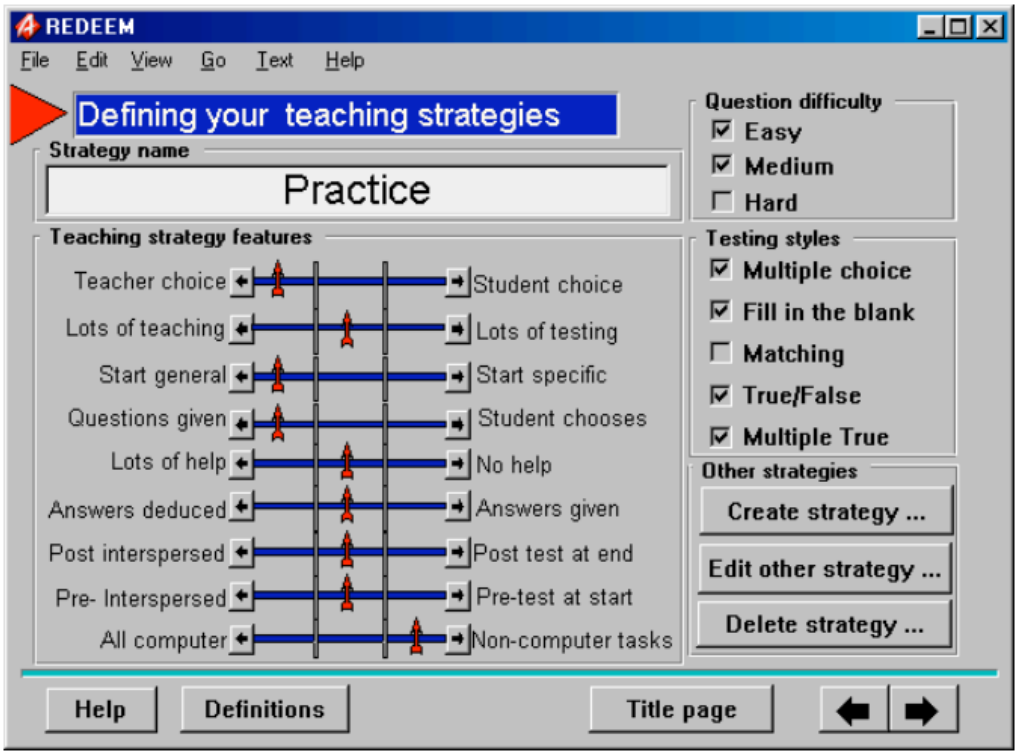

Figure 2. Creating a teaching strategy in REDEEM (Ainsworth, et al., 2003).

REDEEM has undergone multiple evaluations showing that teachers can use the system to create tutors, and that they find it easy to use. In addition, tutors created with REDEEM have been compared against computer-aided instruction (CAI) counterparts. These studies have demonstrated a significant difference in learning outcomes with effect sizes as large as 0.76 for REDEEM vs. 0.36 for CAI systems with the same content (Ainsworth & Fleming, 2006).

Generalizing from Examples

One important skill instructors and subject matter experts share is that they can solve problems in the domain in which they teach or work. This observation was made in ITS authoring as well as in the areas of expert systems and programming (Cypher & Halbert, 1993). Authoring by demonstration seeks to leverage this observation by allowing instructors and experts to simply *show* the authoring system how students should solve problems instead of forcing instructors and experts to explicitly encode domain knowledge. Given a sufficient number of examples, the authoring tool will develop a generalized model of the skill that can be used in an ITS.

Building on the work in expert systems and programming-by-demonstration, Blessing (2003) was one of the pioneers in applying this idea to building an ITS. His Demonstr8 authoring system could generate an ACT-style tutor for arithmetic based on authors solving problems. ACT Tutors have expert models consisting of production rules capable of solving problems, and learner actions are continually compared against this model through a process called model tracing. Demonstr8 creates such an expert model by attempting to determine the rationale behind steps in solved examples and create rules that apply across examples. However, an author must first create an underlying knowledge representation such that Demonstr8 can generalize from the author's behavior. For example, authors need to define the concept of a column of numbers as well as basic arithmetic operations such as subtracting two digits. It is also the case that the underlying model cannot be hidden from the author who may need to adjust production rules as well as specify details such as goals and subgoals.

It is worth noting a similar approach adopted in DISCIPLE (Tecuci & Keeling, 1998). Here, the domain is history and the task is to determine whether a source is relevant to a specific research assignment, and why or why not. Only the GUI and the specialized problem solver are specific to the target domain/task. To define the target task, an educator with help from a knowledge engineer first builds an ontology. The next step is developing a set of problem-solving rules; in this case, the rules determine whether a source is relevant based on properties of the source and the research assignment. The general approach is for the educator to teach the authoring system through demonstration (i.e., providing a correct answer), limited explanation (e.g., pointing out a relevant feature), and feedback. In the case of feedback, the system generates new examples and the educator helps debug the rules when incorrect results appear. The educator must then extend the set of natural language generation templates allowing the system to generate tutoring guidance from the underlying knowledge representation. The resulting history ITS was highly rated in surveys from an experiment with students and teachers, and the domain module was judged highly accurate by an external expert.

ASPIRE (Mitrovic, et al., 2009) is an authoring tool developed at the University of Canterbury for the construction of tutors that use constraint-based modeling as the primary method for knowledge representation. Constraints are fundamentally different from production rules in that they encode properties that must hold in solutions rather than generate problem solving steps. Constraints, when violated by a student's solution, capture teachable moments in which constraint-based tutors can help. Authors are required to create constraints, feedback messages, problems, and if necessary, a user interface. Although different from production rules, constraints are still a form of knowledge representation and building a constraint base for an ITS requires a certain level of technical skill.

ASPIRE's predecessor, Web-Enabled Tutor Authoring System (WETAS) (Mitrovic, Martin & Suraweera, 2007), supported users with some technical expertise while ASPIRE falls into the category of systems built for nonprogrammers seeking to automate some of the more complex tasks of building ITSs with machine learning.

To build a constraint-based tutor with ASPIRE, authors must perform three steps, none of which require programming skills: (1) design a (text-based) interface, (2) build a domain ontology (Figure 3), (3) create problems and solutions in the interface. ASPIRE can use the ontology to automatically generate syntactic constraints corresponding to domain requirements. For example, an instructor might require students to specify a lowest common denominator (e.g., the student must type a number into the relevant location in the GUI). Semantic constraints model the correctness of the answer. Authors are required to specify alternate solutions, and ASPIRE automatically generates semantic constraints accommodating these variations (Mitrovic, et al., 2009). Perhaps the most burdensome step in creating an ASPIRE tutor lies in the creation of a domain ontology, which has the potential to become overly complex. However, as can be seen in Figure 3, the process is simplified in that only basic hierarchical relationships are needed (such as specialization).

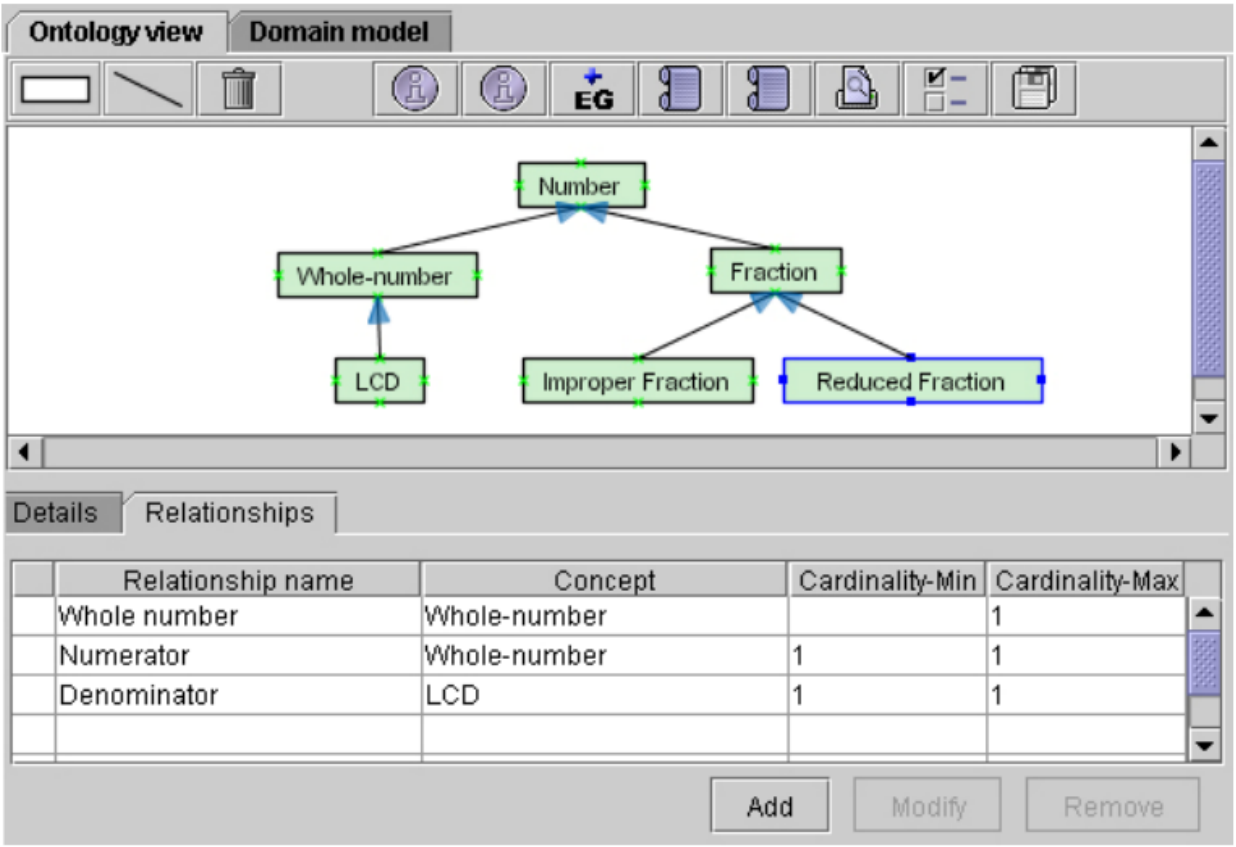

Figure 3. The ASPIRE ontology editor, used in the automated generation of constraints (Mitrovic, et al., 2009)

When compared to a hand-authored domain model, ASPIRE was found to generate all of the same syntactic constraints and covered 85% of the semantic. A larger pilot evaluation of a tutoring system generated by ASPIRE with a subject-matter expert author (and nonprogrammer), showed that it produced significant learning gains and that learners followed expected learning curves (Mitrovic et al., 2008).

Authoring by Constructing Elaborated Examples

One of the most prominent efforts to provide authoring tools for non-experts is CTAT at Carnegie Mellon University. CTAT provides tools for building two kinds of tutoring systems: *example-tracing* tutors, which involve problem-specific authoring but no programming (Aleven, McLaren, Sewall & Koedinger, 2009), and *cognitive tutors*, which require AI programming and the development of a problem-independent cognitive model of the target skill(s) (Anderson, et al., 1995). The work on Cognitive Tutors shows there is room for improvement in authoring tools for ITS programmers. For Cognitive Tutors, CTAT has been shown to reduce development time by a factor of 1.4 to 2 (Aleven, et al., 2006).

In keeping with the goals of our chapter, we focus on non-programmers and example-tracing tutors. For example-tracing tutors, early evaluations of efficiency gains using CTAT are also impressive: a reduction in development costs by a factor of 4 to 8 over traditional estimates on ITS development (Aleven, et al., 2009). Example-tracing tutors are created by demonstration. The appeal, therefore, is that an author can create solution models by simply solving problems in ways that learners might. This is accomplished in CTAT by defining a specific problem, solving it, and then expanding the resulting solution in ways so that other common problem solving actions can be recognized, such as alternate solutions and common misconceptions. These different problem-solving steps form a *behavior graph* such as the one on the right side of Figure 4. Learners take actions by manipulating a GUI (left side of Figure 4) and the tutoring system matches these actions to nodes in the behavior graph. The graph branches as authors specify a variety of correct and incorrect approaches to solving the problem.

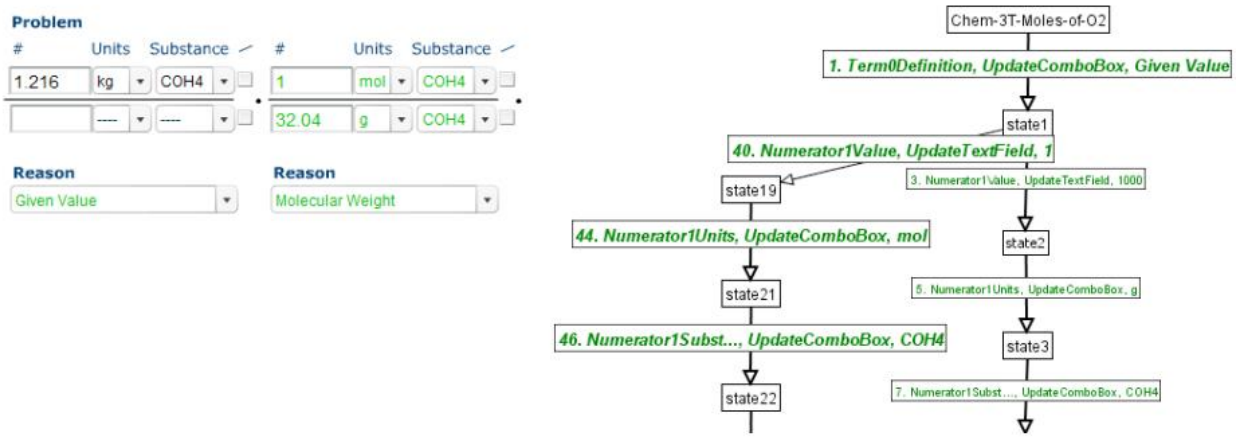

Figure 4. An authored example in CTAT for stoichiometry (Aleven, et al., 2009).

Example-tracing does not require a machine-readable ontology defining the domain and there is no machine learning that must be debugged by the author. However, this knowledge-light approach means that authors must annotate steps in the behavior graph with hint and feedback messages as well as links to the learner module (e.g., taking this step provides evidence that the learner understands a certain domain concept). This annotated behavior graph contains all the information needed for the ITS to assess learner actions and provide step-level feedback.

The Situated Pedagogical Authoring (SitPed) project at the University of Southern California (Lane, Core, et al., in-press) focuses on problem-solving through conversation (e.g., a therapist talking to a client) using simulated role players. Using SitPed, authors create an ITS by doing the following:

- Specifying paths through the problem space by simultaneously *solving* problems (either correctly or intentionally incorrectly) and indicating the relevant skills and misconceptions.

- Pausing during problem solving to create hints and feedback messages associated with the current situation.

Like the case of example-tracing tutors, authors work in the same learning environment that learners use. Thus, SitPed falls roughly into the category of WYSIWYG authoring tools (Murray, 2003) because authors are constantly reminded of what the resulting learning experience will be like. In the case of SitPed, demonstration is not simply a technique to hide technical details. Simulated conversations and simulations in general allow learners to explore a wide space of possibilities, and it can be difficult for authors to visualize the learner's perspective unless they are also working through examples in the same simulation.

One of the difficulties in building an ITS for a simulated role play is that simulated role players can be implemented in a variety of ways. The initial version of SitPed targets branching conversations. At each step in the conversation, learners are selecting utterances from a menu and the virtual role player consults a tree to lookup its response and the next set of menu items. This tree simply contains the lines of the conversation as well as the associated animations corresponding to performance of the role player lines. In branching conversations, it is necessary for the author to play through all branches of the tree and link each possible learner choice to the skills and misconceptions of the domain. This process is illustrated in Figure 5. Although the goal is to recreate the learner experience as much as possible, authors need to be able to see relevant context (e.g., the dialogue history in the middle) and make annotations corresponding to the skills and common mistakes of the domain which we refer to as tasks (via the menu labeled "Task List"). This exhaustive exploration of the possibilities is necessary because of the difficulty of automatically understanding the dialogue well enough to identify skills such as active listening. For simulated role players with models of dialogue and emotion, more scope for generalization may be possible based on expert conversations.

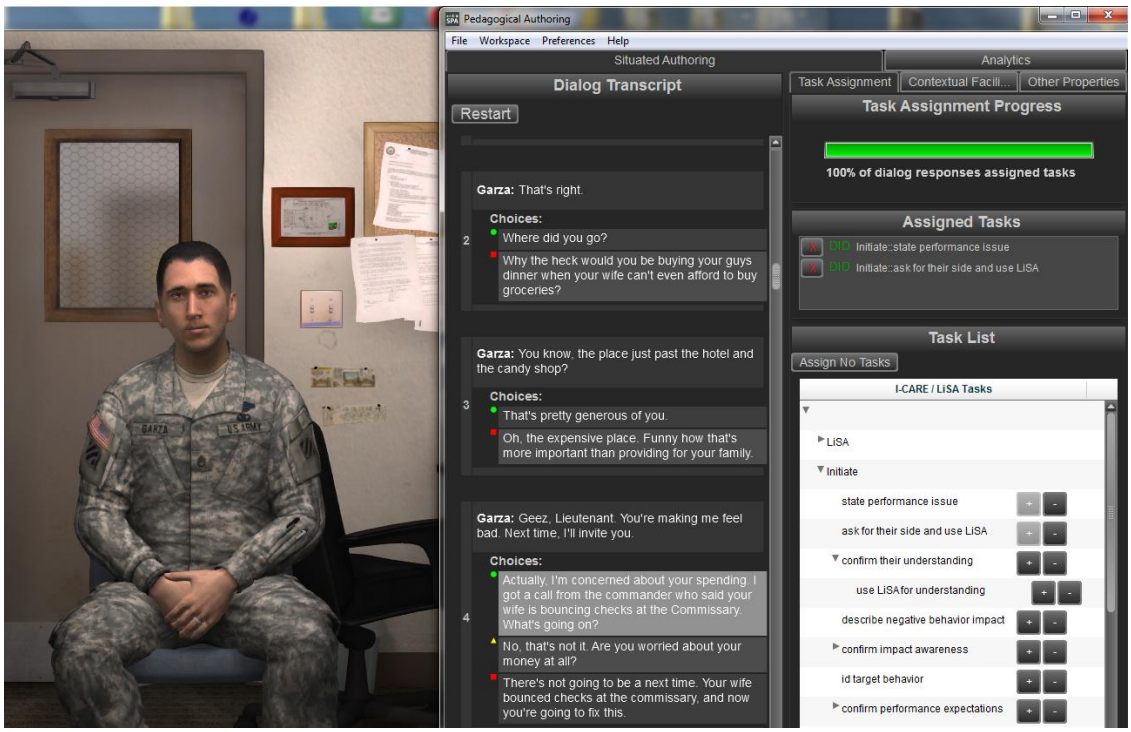

Figure 5. The SitPed Authoring domain tagging screen (Lane, Core, et al., in-press).

SitPed also provides a tool to create simple, hierarchal task models (similar to those created in GIFT), which form the basis for the linking screen shown in Figure 5. The full workflow requires an author to (1) create a task list; (2) load a scenario file (these are authored in a commercial product called Chat Mapper, an editor designed specifically for tree-based branching stories; (3) run through the scenario as many times as needed to annotate the possible student choices; (4) create hints and feedback content during those runs; and (5) test the final product out in a "student" view that recreates the actual learning experience (with no additional tools visible). In step 5, authors also have the option to activate the assessment engine and see how the authored model classifies each action (i.e., positive, negative, or mixed) with respect to the task list.

A study of SitPed was conducted in 2014 that consisted of two phases. In the first phase, a set of 11 domain experts were split across three authoring conditions: full SitPed (as described), SitPed Lite (hypertext-only with no graphics or sound), and a specialized spreadsheet. Authors were given scenario data and asked to annotate it both with appropriate tasks (as well as known or likely misconceptions) and tutor messages. In the second phase, the data sets generated from each condition were used to create three separate tutoring systems (randomly using one of the data sets from each corresponding group). Initial results from phase 1 suggest that authors in the SitPed conditions generally wrote *longer* feedback messages and created *more* links to the task list in their authored models, but covered far less of the scenario space. Although the results of phase 2 are still being analyzed at the time of this writing, initial results suggest that learners in all three conditions demonstrated learning gains with trends in favor of SitPed (Lane, Core, et al., in-press).

Our final example of an authoring tool that leverages examples (or equivalently, demonstrations) is actually a more recent addition to CTAT. *SimStudent* (Matsuda, Cohen & Koedinger, 2014; Matsuda, et al., 2013) extends work started with Demonstr8 to pursue the holy grail of authoring: deriving cognitive models (or more generally, expert models with rich representations) based on demonstration of the skill. SimStudent uses inductive logic programming to infer production rules interactively with an author. That is, as an author interacts with SimStudent, the author is tutoring the system an author can simply show SimStudent what to do at any point or it can let SimStudent perform problem solving steps. In the latter case, the author gives feedback to confirm or correct the emerging model. SimStudent creators have demonstrated that the interactive approach produces higher levels of accuracy in the resulting models (Matsuda, et al., 2014). Similarly, the use of induction allowed the resulting model to go beyond the examples used to train it (MacLellan, Koedinger & Matsuda, 2014). By adding the element of teaching a model (versus simply demonstrating and letting the system observe), there seems to be a payoff in terms of how close the resulting models can get to a hand-authored cognitive model.

Authoring by Demonstration in Simulation-Based Learning Environments

Other key ITS authoring efforts have leveraged ideas from programming by demonstration and authoring with examples. In the area of simulation-based ITS authoring tools, RIDES was a pioneering effort that, among a wide variety of other capabilities, included demonstration as a method for extracting tutoring content (Munro, 2003; Munro et al., 1997; Munro, Pizzini, Johnson, Walker & Surmon, 2006). RIDES provided authors the ability to build their own interfaces and show how to use them in a specialized *demonstration* mode. In addition, RIDES had a rich language for modeling physical systems, identifying expert pathways through the system, and authoring pedagogical feedback. The system incorporates simulation along with other instructional materials to support procedural learning. RIDES has been used

to develop tutors for a variety of domains, including diagnosis of faulty electronic systems and shipboard radar tracking.

A more recent instance of authoring by demonstration, being used in a simulated environment, is the Extensible Problem Specific Tutor (xPST) (S. Gilbert, Devasani, Kodavali & Blessing, 2011). By seeking to capture programming by demonstration in simulated, 3D environments, xPST is unique in its use of freely explorable 3D environments combined with authoring. The system allows authors to link tutoring behaviors to events in a 3D environment to enable the detection of key events (such as recognizing that a step in a procedure has been taken) and then tutoring at those times by delivering hints and feedback. While xPST does not involve authoring directly inside the 3D environment, a web interface is available that has been tested with nonprogrammers who have been able to successfully create tutors for specific skills (S. B. Gilbert, Blessing & Kodavali, 2009).

Themes

In this review of authoring systems that have specifically built to be accessible by nonprogrammers, a few key themes emerge that both highlight the limitations imposed by addressing this audience, and the key areas for future research that we address in our conclusions. Although the systems discussed in this chapter have seen some limited successes, they remain largely in the research and prototype categories with much to be desired in terms of ease of use and accessibility. This seems to be one stable, and undesirable, status of the field (Devasani, 2011; Murray, et al., 2003), although authoring ITS-like interactions have seen dramatic adoption rates (Heffernan & Heffernan, 2014). In nearly every system reviewed here, authors are not spared the inherent complexity of ITS creation (with SimStudent and ASPIRE standing out as possible exceptions). Authors must organize and annotate the vast space of possible learner actions given a specific problem (e.g., example-tracing tutors and SitPed), or alternatively create the complex pedagogical policies needed for ITS decision-making (e.g., REDEEM and xPST). It could be argued that this inherent complexity is unavoidable. In the rest of this section, we summarize these issues by highlighting three themes that emerge from the review.

Theme One: Leveraging of Intuition and Existing Skillsets

All of the authoring tools in this review seek, to varying degrees, to leverage intuitive and simplified elicitation methods such as providing examples or using basic ontology creation tools. The two primary categories of approaches reviewed are (1) intuitive interfaces for capturing domain and pedagogical knowledge and (2) leveraging of an author's existing skillsets and knowledge, such as solving problems in the targeted task domain. Developing GUIs (both for the learner by an author and for the author by a software engineer) are general challenges for authoring systems (Murray, 2003); however, the specific challenge of developing an interface for nonprogrammers is at its core a HCI problem. Interfaces matter, and they matter to a very large degree when system designers ask non-technical audiences to create highly technical content. The second approach, and one that shows up in several different forms in the reviewed systems, is to ask authors to do what they already know how to do: solve problems. Whether it is creating examples, demonstrating a task, or interactively walking through a problem space emulating a learner, this path to building tutors is showing significant promise. Evaluations cited in this review suggest that nonprogrammers can do this, and that they can produce models with reasonably good accuracy.

Theme Two: Tradeoff Between ITS Sophistication and Methods of Elicitation

A closely related theme is the fundamental tradeoff that occurs when a system limits the expressive power of the author (which is a subset of the full space of tradeoffs outlined by Murray (2003, p. 518)). Of

course, limiting expressive power is necessary to simplifying knowledge elicitation (barring a profound discovery in the knowledge elicitation field). For example, REDEEM provides a wide array of easy-to-understand sliders and a logical workflow while allowing authors to create "simple" tutors. Similarly, SitPed supports authors in navigating a large problem space by working in the learner's environment, but does not produce an expert representation with the richness of most traditional ITSs. But emerging research that leverages machine learning techniques is beginning to close this gap. Both SimStudent and ASPIRE go one step further by attempting to build cognitive models and constraint bases from simplified interactions with an author. These are important advances in the field that directly address known shortcomings. The primary limitation is that these approaches are currently limited to fairly simple task domains that involve symbol manipulation (although important ones, including algebra and arithmetic). This is a typical progression in AI and so we suspect future work will push this research into new task domains and uncharted spaces.

Theme Three: Focused Outputs from Authoring and Carefully Chosen Pedagogical Goals

Authoring tools naturally tend to narrow the problem in appropriate ways to help authors complete the task and produce a usable system. In other words, authoring tools for nonprogrammers may never allow a single author to create an end-to-end tutor for any task domain imaginable, but they can certainly operate in defined spaces and allow authors to tailor existing systems for their particular needs. For example, xPST provides tools for problem-specific tutoring (i.e., a new model for each new problem) and example-tracing is similar in that respect. In these cases, generalized ITSs are not the goal the goal is to offer step-level help and assess learner actions in specific situations. The norm for nonprogrammer authoring is populate only the ITS modules (see Figure 1) that matter most given specific pedagogical aims.

Conclusions

The overarching contribution of work on authoring tools for nonprogrammers thus far is that it provides proof-of-concepts that simplify authoring environments, and can be used to produce usable tutors. There continues to be limited studies on the *products* of authoring tools in terms of teaching and learning efficacy since most studies look at either time to create content, or the completeness of a produced model. Simple success at building a tutor is also an important, but limiting measure.

But, it is true that more studies are being conducted with authoring tools. REDEEM has been the focus of a long list of evaluations, even comparing performance against CAI counterparts (and outperforming them). Along those lines, SitPed's two-phase study model seeks to link authoring affordances to differences in learning. These are promising trends that will lead to better authoring tools that do not simply make authoring easier, but also nudge authors to make decisions that are in line with known principles of learning and effective pedagogy. This large and growing body of work supports the notion that serious attention is being paid to nonprogrammers and scalability issues and suggests that the field as a whole recognizes the problem with having effective ITSs in the world, but only seeing limited adoption.

There are still significant gaps in the research as a whole, however. Very few, if any of the systems reviewed here have treated the act of authoring as a cognitive skill. Although Murray (Murray, 2003) suggests that authoring tools could do a better job *teaching* authors how to author, the skill itself is largely treated as a black box. Highly relevant work on automated cognitive task analysis tools, such as DNA (Valerie J Shute & Torreano, 2003), has had an influence on ITS authoring tools, but there is still much room for improvement in modeling and supporting processes, decisions, and the creativity inherent in good authoring (which is a form of teaching, as evidenced by SimStudent).

With respect to the implications for the future of the GIFT architecture, it seems clear that a truly general system for non-technical audiences is not likely to bear fruit. Rather, given the nature of the systems built thus far and reviewed here, a more productive vision for GIFT would likely lie in creating specialized versions of GIFT for broad domain categories. For example, the history of cognitive tutors (Anderson, et al., 1995) and also with CTAT, many (but not all) of the systems focus on symbol manipulation tasks, such as geometry proofs, equation solving, stoichiometry, and so on. The commonality between a class of domains can be a powerful force in the world of authoring since it can support reuse of formalisms, pedagogy, and even interfaces. While GIFT is currently built around standards for formalizing assessment and pedagogy in any domain, a goal should be to provide tools that differentiate authoring across cognitive, affective, and psychomotor domains. The real challenge of authoring however is removing the required programming component to establish real-time assessment functions. As of now, all authoring is completed in GIFT's domain knowledge file (DKF) where a hierarchical representation of a domain is linked to specific condition classes authored for determining performance on an "at-", "above-", and "below-expectation" rating. Having a situated interface that enables an author to establish performance criteria while building out scenarios and problem sets is recommended.

A second, but related possible implication for GIFT from this work is the idea of building tools for building authoring tools (in the spirit of "compiler compilers" popular in the early years of computer science). In other words, designing a pipeline from engineers to end-users who face specific authoring tasks could further spread the work of building ITSs and expedite the production of ITSs that meet real needs. The problem would still exist for extracting rich knowledge from a non-programmer, but the engineer in the loop design could select the best methods given the task domain and the pedagogical goals, then produce tools that meet the need (e.g., authoring by example for practice support, or adaptive presentation of content using multiple-choice quizzes). The benefit is that GIFT already has many of the tools in place for programmers, so building customization tools and easily adjusted interfaces could imply an authoring tool generator is not far off. The open source nature of GIFT supports this approach, but buy-in is required from the ITS community as a whole so that the authoring tools, processes, and methods evolve as more systems are established using the GIFT architectural dependencies.

A final recommendation for GIFT that emerges from this review highlights the need for a longer-term vision for the future of authoring tools. GIFT is equipped with a number of important capabilities that address the needs of educators and researchers. This combination could play a major role in bringing together the creation of ITSs with authoring tools and evolving them to both improve (in terms of their ability to promote learning) and be shared within a community. In the spirit of the ASSISTments project that has brought together unprecedented numbers of teachers to create, share, and evaluate content (Heffernan & Heffernan, 2014), GIFT could provide the underlying tools that allow authors to create ITSs, collect important data such as pre- and post-test scores, physiological data, and usability data, to act as the hub for a living cycle of growth and improvement of user-generated ITSs. The vision of an instructor creating content then reviewing the aggregated results of days-worth of usage to then make adjustments and improvements to the content, is a powerful one for GIFT developers to consider. It is understood that future GIFT developments are focused on pushing authoring to a collaborative, web-based interface that uses customized tools to remove low-level programming for establishing the modules to deliver an ITS for a domain. The initial focus is on authoring environments to remove the burden of building out XML files that GIFT runs on. To make GIFT more extensible, a focus needs to address learning and training effectiveness tools to assess the effect of authored pedagogical approaches and their influence on performance outcomes. A sought after goal would be to establish probabilistic modeling approaches that can use reinforcement learning techniques to automatically adjust pedagogical policies based on outcomes across a large dataset of learners interacting with the system. This in turn introduces further complications when it comes to authoring tools, in that new techniques would be required to build out these policies when an individual is void of this knowledge and understanding.

In sum, research on authoring tools for nonprogrammers is making a great deal of progress and producing useful results. There is much room for continued improvements and, sadly, ITS authoring tools still fall way behind those of commercially viable CAI authoring tools (Murray, 2003). Researchers should continue to engage in more studies on authoring and focus on accuracy of produced models, usability of tools, and increase efficacy studies that link authoring affordances to learning outcomes. In addition, an issue not addressed in this review but of high importance the need for greater emphasis on multi-party authoring. It is unlikely that one person will be qualified to do end-to-end authoring, and so explicit support for collaboration and workspaces is definitely needed. Authoring researchers should seek to provide explicit support for good pedagogy and the targeting the right level of complexity for desired learning outcomes. Together, and with general frameworks like GIFT to underlie them, it is likely that authoring tools for ITSs will continue to close the gap between research and real-world needs. As newer technologies, such as tablets and immersive games, work their way into educational technologies more deeply, authoring tools will evolve to meet those needs and research should be conducted to inform these inevitable trends.

References

Ainsworth, S. & Fleming, P. (2006). Evaluating authoring tools for teachers as instructional designers. *Computers in human behavior, 22*(1), 131-148.

Ainsworth, S., Major, N., Grimshaw, S., Hays, M., Underwood, J. & Williams, B. (2003). REDEEM: Simple Intelligent Tutoring Systems from Usable Tools. In T. Murray, S. Ainsworth & S. Blessing (Eds.), *Authoring Tools for Advanced Technology Learning Environments* (pp. 205-232).

Aleven, V., McLaren, B., Sewall, J. & Koedinger, K. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. In M. Ikeda, K. Ashley & T.-W. Chan (Eds.), *Intelligent Tutoring Systems* (Vol. 4053, pp. 61-70): Springer Berlin / Heidelberg.

Aleven, V., McLaren, B. M., Sewall, J. & Koedinger, K. R. (2009). A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. *International Journal of Artificial Intelligence in Education, 19*(2), 105-154.

Anderson, J. R., Corbett, A. T., Koedinger, K. R. & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *Journal of the Learning Sciences, 4*(2), 167-207.

Brusilovsky, P. (2012). Adaptive Hypermedia for Education and Training. *Adaptive Technologies for Training and Education, 46*.

Cypher, A. & Halbert, D. C. (1993). *Watch what I do: programming by demonstration*: MIT press.

D'Mello, S. K. & Graesser, A. C. (2012). AutoTutor and affective AutoTutor: Learning by talking with cognitively and emotionally intelligent computers that talk back. *ACM Transactions on Interactive Intelligent Systems, 2 (4)*(23), 1 - 38.

Devasani, S. (2011). *Intelligent tutoring system authoring tools for non-programmers.* Master's Thesis, Iowa State University.

Gilbert, S., Devasani, S., Kodavali, S. & Blessing, S. (2011). *Easy authoring of intelligent tutoring systems for synthetic environments.* Paper presented at the Proceedings of the Twentieth Conference on Behavior Representation in Modeling and Simulation.

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

Heffernan, N. T. & Heffernan, C. L. (2014). The ASSISTments Ecosystem: Building a Platform that Brings Scientists and Teachers Together for Minimally Invasive Research on Human Learning and Teaching. *International Journal of Artificial Intelligence in Education, 24*(4), 470-497.

Hoffman, R. R., Shadbolt, N. R., Burton, A. M. & Klein, G. (1995). Eliciting knowledge from experts: A methodological analysis. *Organizational behavior and human decision processes, 62*(2), 129-158.

MacLellan, C. J., Koedinger, K. R. & Matsuda, N. (2014). *Authoring Tutors with SimStudent: An Evaluation of Efficiency and Model Quality.* Paper presented at the Intelligent Tutoring Systems.

Mark, M. A. & Greer, J. E. (1995). The VCR Tutor: Effective Instruction for Device Operation. *The Journal of the Learning Sciences, 4*(2), 209-246.

Matsuda, N., Cohen, W. & Koedinger, K. (2014). Teaching the Teacher: Tutoring SimStudent Leads to More Effective Cognitive Tutor Authoring. *International Journal of Artificial Intelligence in Education*, 1-34. doi: 10.1007/s40593-014-0020-1

Matsuda, N., Yarzebinski, E., Keiser, V., Raizada, R., Cohen, W. W., Stylianides, G. J. & Koedinger, K. R. (2013). Cognitive anatomy of tutor learning: Lessons learned with SimStudent. *Journal of Educational Psychology, 105*(4), 1152.

Mitrovic, A., Martin, B. & Suraweera, P. (2007). Intelligent Tutors for All: The Constraint-Based Approach. *IEEE Intelligent Systems, 22*(4), 38-45.

Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J. & Mcguigan, N. (2009). ASPIRE: An Authoring System and Deployment Environment for Constraint-Based Tutors. *Int. J. Artif. Intell. Ed., 19*(2), 155-188.

Mitrovic, A., McGuigan, N., Martin, B., Suraweera, P., Milik, N. & Holland, J. (2008). *Authoring Constraint-based Tutors in ASPIRE: a Case Study of a Capital Investment Tutor.* Paper presented at the World Conference on Educational Multimedia, Hypermedia and Telecommunications.

Munro, A. (2003). Authoring simulation-centered learning environments with RIDES and VIVIDS. In T. Murray, S. Blessing & S. Ainsworth (Eds.), *Authoring Tools for Advanced Technology Learning Environments* (pp. 61-91). Dordrecht, Netherlands: Kluwer Academic Publishers.

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 Journal of Artificial Intelligence in Education, 8*, 284-316.

Munro, A., Pizzini, Q. A., Johnson, M. C., Walker, J. & Surmon, D. (2006). Knowledge, models, and tools in support of advanced distance learning final report: The iRides performance simulation / instruction delivery and authoring systems: University of Southern California Behavioral Technology Laboratory.

Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education (IJAIED), 10*, 98-129.

Murray, T. (2003). An Overview of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art. In T. Murray, S. Blessing & S. Ainsworth (Eds.), *Authoring Tools for Advanced Technology Learning Environments* (pp. 491-544): Springer Netherlands.

Murray, T., Blessing, S. & Ainsworth, S. (2003). *Authoring Tools for Advanced Technology Learning Environments*. Dordrecht: Kluwer Academic Publishers.

Nye, B. D. (2013). *ITS and the digital divide: Trends, challenges, and opportunities.* Paper presented at the Artificial Intelligence in Education.

Nye, B. D. (2014). Barriers to ITS Adoption: A Systematic Mapping Study. In S. Trausan-Matu, K. Boyer, M. Crosby & K. Panourgia (Eds.), *Intelligent Tutoring Systems* (Vol. 8474, pp. 583-590): Springer International Publishing.

Shute, V. J. & Psotka, J. (1996). Intelligent tutoring systems: Past, present, and future. In D. H. Jonassen (Ed.), *Handbook for research for educational communications and technology* (pp. 570-599). New York, NY: Macmillan.

Shute, V. J. & Torreano, L. A. (2003). Formative evaluation of an automated knowledge elicitation and organization tool *Authoring Tools for Advanced Technology Learning Environments* (pp. 149-180): Springer.

Sottilare, R. A. (2012). *A modular framework to support the authoring and assessment of adaptive computer-based tutoring systems.* Paper presented at the Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), Orlando, FL.

Tecuci, G. & Keeling, H. (1998). Developing Intelligent Educational Agents with the Disciple Learning Agent Shell. In B. Goettl, H. Halff, C. Redfield & V. Shute (Eds.), *Intelligent Tutoring Systems* (Vol. 1452, pp. 454-463): Springer Berlin / Heidelberg.

VanLehn, K. (2006). The Behavior of Tutoring Systems. *International Journal of Artificial Intelligence in Education, 16*(3), 227-265.

VanLehn, K. (2011). The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. *Educational Psychologist, 46*(4), 197-221. doi: 10.1080/00461520.2011.611369

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., . . . Wintersgill, M. (2005). The Andes Physics Tutoring System: Lessons Learned. *Int. J. Artif. Intell. Ed., 15*(3), 147-204.

Woolf, B. P. (2009). *Building Intelligent Interactive Tutors: Student-centered Strategies for Revolutionizing E-learning*. Amsterdam, Netherlands: Morgan Kaufmann.