CHAPTER 4 – Generalizing the Genres for ITS: Authoring Considerations for Representative Learning Tasks

Benjamin D. Nye[1], Benjamin Goldberg[2], Xiangen Hu[1,3]
[1]University of Memphis, [2]ARL-LITE Lab,[3]Central China Normal University

INTRODUCTION

Compared to many other learning technologies, intelligent tutoring systems (ITSs) have a distinct challenge: authoring an adaptive inner loop that provides pedagogical support on one or more learning tasks. This coupling of tutoring behavior to student interaction with a learning task means that authoring tools need to reflect both the learning task and the ITS pedagogy. To explore this issue, common learning activities in intelligent tutoring need to be categorized and analyzed for the information that is required to tutor each task. The types of learning activities considered cover a large range: step-by-step problem solving, bug repair, building generative functions (e.g., computer code), structured argumentation, self-reflection, short question answering, essay writing, classification, semantic matching, representation mapping (e.g., graph to equation), concept map revision, choice scenarios, simulated process scenarios, motor skills practice, collaborative discussion, collaborative design, and team coordination tasks. These different tasks imply a need for different authoring tools and processes used to create tutoring systems for each task. In this chapter, we consider three facets of authoring: 1) the minimum information required to create the task, 2) the minimum information needed to implement common pedagogical strategies, 3) the expertise required for each type of information. The goal of this analysis is to present a roadmap of effective practices in authoring tool interfaces for each tutoring task considered.

A long-term vision for ITSs is to have generalizable authoring tools, which could be used to rapidly create content for a variety of ITSs. However, it is as-yet unclear if this goal is even attainable. Authoring tools have a number of serious challenges, from the standpoint of generalizability. These challenges include the domain, the data format, and the author. First, different ITS domains require different sets of authoring tools, because they have different *learning tasks*. Tools that are convenient for embedding tutoring in a 3D virtual world are completely different than ones that make it convenient to add tutoring to a system for practicing essay-writing, for example. Second, the data produced by an authoring tool needs to be consumed by an ITS that will make pedagogical decisions. As such, at least some of the data is specific to the pedagogy of the ITS, rather than directly reflecting domain content. As a simple example, if an ITS uses text hints, those hints need to be authored, but some systems may just highlight errors rather than providing text hints. As such, the first system actually needs more content authored and represented as data. With that said, typical ITSs use a relatively small and uniform set of authored content to interact with learners, such as correctness feedback, corrections, and hints (VanLehn, 2006). Third, different authors may need different tools (Nye, Rahman, Yang, Hays, Cai, Graesser, & Hu, 2014). This means that even the same content may need distinct authoring tools that match the expertise of different authors.

In this chapter, we are focusing primarily on the first challenge: *differences in domains*. In particular, our stance is that the "content domain" is too coarse-grained to allow much reuse between authoring tools. This is because, to a significant extent, content domains are simply names for related content. However, the skills and pedagogy for the same domain can vary drastically across different topics and expertise levels. For example, Algebra and Geometry are both high-school level math domains. However, in geometry, graphical depictions (e.g., shapes, angles) are a central aspect of the pedagogy, while Algebra tends to use graphics very differently (e.g., coordinate plots). As such, some learning tasks tend to be

shared between those subdomains (e.g., equation-solving) and other tasks are not (e.g., classifying shapes).

This raises the central point of our paper: the learning tasks for a domain define how we author content for that domain. For example, while Algebra does not involve recognizing many shapes, understanding the elements of architecture involves recognizing a variety of basic and advanced shapes and forms. In total, this means that no single whole-cloth authoring tool will work well for any pair of Algebra, Geometry, and Architectural Forms. However, it also implies that a reasonable number of task-specific tools for each learning task might allow authoring for all three domains. To do this, we need to understand the common learning tasks for domains taught using ITS, and why those tasks are applied to those domains. In the following sections, we identify and categorize common learning tasks for different ITS domains. Then, we extract common principles for those learning tasks. Finally, we suggest a set of general learning activities that might be used to tutor a large number of domains.

WHAT IS A LEARNING TASK?

Before we begin, it is important to define what we mean by a learning task. Functionally-speaking, a learning task is an activity designed to help the participant(s) learn certain knowledge or skills. Any learning task has a three essential parts:

1) Task State (S_T) - the context and status of the task,
2) Task Interface (I_T) - the representation used to present the task and its available actions,
3) Task Goals (G_T) - importance or value given to states or state trajectories, which may be stated in the task, given prior to the task (e.g., by a teacher), or chosen by the learner.

A directed learning task, such as one run by a teacher or an ITS (as opposed to an undirected sandbox activity), also has complementary parts related to the instructor's control over the system:

1) Pedagogical State (S_P) - the context and status relevant to pedagogical decision-making,
2) Pedagogical Interface (I_P) - the pedagogical actions available during a task, and
3) Pedagogical Goals (G_P) - importance or value given to reaching certain pedagogical states.

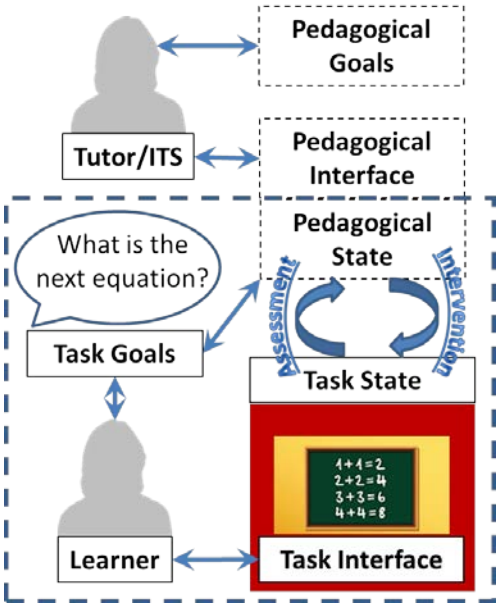

Figure 1: Tasks & Pedagogy

The relationships between these parts are noted in Figure 1. From an ITS authoring standpoint, both the task and the pedagogical model need to be authored. In this representation, the pedagogical state includes the task goals, task state, and the learner's state (e.g., a student model). In this respect, the pedagogical state is more complex than the task state. However, excluding the learner's internal state (which is only observable through the history of task interactions) and the task goals (which are typically not changed during a given task), the pedagogical state is *by definition* less complex than the task state. Considering a task as a Markov Decision Process, the pedagogical state trajectory S_P cannot consider any more information from the task beyond its trajectory of task states (S_T). In most cases, the representation of the pedagogical state is far simpler and based on features sets such as classifying good/bad answers,

identifying specific misconceptions or bugs, and other assessments that reduce even rich environments (e.g., 3D simulators) into streams of simpler features that form the pedagogical state used for triggering interventions such as hints (Kim et al., 2009; Nye, Graesser, & Hu, 2014; VanLehn, 2006).

This implies that ITS authoring should be greatly constrained by the learning task. At face value, it seems like there might be exceptions: a tutor for metacognitive skills might need to know almost nothing about learner's performance on their primary learning task, if it only suggests self-reflection and an unassessed summarization. However, it can be argued that such a task-agnostic tutor has that capability only because it generates its *own* learning tasks (e.g., journals for summarization, delays for self-reflection). This has two implications:

1. First, it implies that all ITS authoring is tied to a specific set of tasks.
2. Second, it implies that multiple learning tasks may be interleaved or even occurring simultaneously.

In simulation-based training for complex tasks, such as flight simulators or cross-cultural competencies, working on multiple tasks simultaneously might even be a major part of the pedagogy (Silverman, Pietrocola, Nye, Weyer, Osin, Johnson, & Weaver, 2012). So long as interactive feedback on each learning task is independent (i.e., feedback on one task does not directly impact the pedagogical state of other tasks), authoring for such tasks can typically be done independently as well.

So then, this is what we mean by a learning task from an authoring standpoint: 1) a task with a distinct pedagogical state, 2) whose dynamics during that task are mainly or wholly derived from the task state, and 3) which includes the actions performed by the learner (or learners). Moreover, as simplifying assumptions, we posit that the pedagogical goals and problem goals remain static for the typical ITS learning task. For example, even in complex training environments, switching the task goals typically implies ending a task and starting a new one. The counterexample to this case would be a learning task specifically targeting the learner's skills at goal-setting or at adapting to changing task goals. Such tasks are uncommon, and no major authoring tools target such tasks. Finally, we assume that changes to the learner during an ITS task (e.g., learning, affect changes) are primarily influenced by and observable based on interactions with the task interface. If they were not, this would be problematic: the ITS would have little ability to tell if its interventions are effective if some external factors are causing learning and/or task performance (e.g., a second user helping). With that said, such confounds are possible, such as when multiple users share an ITS intended for one user (Ogan, Walker, Baker, Rebolledo Mendez, Jimenez Castro, Laurentino, & de Carvalho, 2012). However, as no known authoring tools develop ITS content for such situations, these are also considered edge cases that we will exclude from this analysis authoring learning tasks for ITS.

A Review of Authoring-Relevant Characteristics of Learning Tasks

Significant literature focuses on taxonomies of learning tasks and the types of knowledge they are designed to convey to a learner. Notable examples include Bloom's Taxonomy and its revisions (Bloom, 1956; Anderson, Krathwohl, Airasian, Cruikshank, Mayer, Pintrich, Raths, & Wittrock, 2000), guidelines for learning activities and resources (R. Clark, 2002; R. Clark & Mayer, 2011), and theories of different types of knowledge components involved in learning (Koedinger, Corbett, & Perfetti, 2012). These three perspectives each look at a different facet of learning tasks: A) the task **activity** (Bloom, 1956), B) the **pedagogical goals** for the learning task (R. Clark, 2002), and 3) the **knowledge components** theorized to encode the knowledge (Koedinger et al., 2012). Figure 2 shows different possible combinations of learning tasks and pedagogical goals.

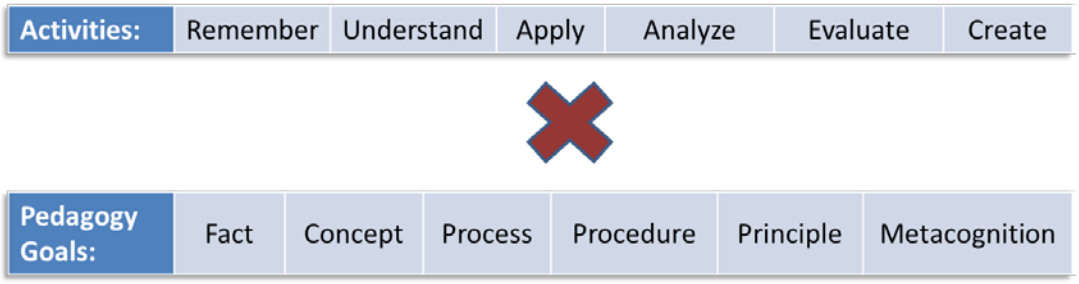

Figure 2: Combinatorial Combinations for Learning Tasks and Pedagogical Goals

Bloom's taxonomy is the most widely used taxonomy to label learning tasks, and has undergone a number of revisions (D. Clark, 2014). Bloom's revised taxonomy (2000) of cognitive knowledge considers six levels: remembering (e.g., list facts), understanding (e.g., summarize in own words), applying (e.g., solve a math problem), analyzing (e.g., identify a statistical trend), evaluating (e.g., select the best-value car for an average consumer), and creating (e.g., build a robot for some task out of parts). Ruth Clark (2002) presented a complementary taxonomy for different types of knowledge associated with pedagogical goals for learning tasks, which built on Merrill (1983). Her categories included facts (unique instances), concepts (classes of instances), processes (representations of how a system works), procedures (steps to reach a task state), and principles (causal relationships and general dynamics).

In addition to these pedagogical goals, metacognitive knowledge can also be a pedagogical goal: where the learner gains understanding or skills to monitor their own cognitive state or learning (Biswas, Jeong, Kinnebrew, Sulcer, & Roscoe, 2010; Azevedo, Johnson, Chauncey, & Burkett, 2010; Goldberg & Spain, 2014). While metacognitive knowledge may fall into other categories, it can involve learning to monitor an additional information channel other than the task state (i.e., their own mental state). As such, at least some types of metacognitive learning are probably qualitatively different than other types of knowledge (possibly closer to affective or psychomotor skills).

Koedinger et al. (2012) looked at the next step for learning activities, which was the cognitive components relevant to assessment and cognitive encoding of knowledge. They considered four facets for encoding knowledge: the task feature dynamics (static vs. variable), the required learner response (static vs. variable), the relationship between task features (explicit vs. implicit), and the availability of a rationale (e.g., a "why" justification for the relationship between features). These different categorizations determine when a learner would need to encode, such as a rule (e.g., y*x = x*y) or simply an association (e.g., x and y were observed together).

Considering these approaches to categorizing learning activities, key facets emerge for different learning tasks. These fall into three design concerns: pedagogical goals (what the student should learn), task design (the learning environment and its affordances), and task interface (how the task is represented and presented). Together, these concerns constrain the pedagogical interface for how an ITS interacts with learners and what needs to be authored.

Task Dynamics

A major constraint on ITS authoring is the dynamics of the task state itself. For example, some learning tasks are **static** and have *no* dynamic features (e.g., memorizing a shape or a fact). Koedinger et al. (2012) highlighted the distinction between tasks whose features are static (e.g., the same across all instances) versus those that are variable (e.g., some features vary across instances, requiring the learner to generalize across them). We further subdivide variable tasks into a few types, as shown in Figure 3. In our conceptualization, three types of variation can occur in the state of a task. The first type, which we call **variable instance**, is across presented tasks, such as presenting a series of pictures and requiring the

learner to identify which ones contain triangles. Other variable tasks change *during* the process of solving the task. The second type, **reactive**, is a task whose state changes due to the learners' actions (e.g., step-by-step equation solving). In the third type, **time-varying**, the task state changes over time regardless of user input (e.g., a video or simulation that unfolds over time). When the task is both time-varying and reactive to user input, we consider it **interactive** (e.g., a 3D game world).

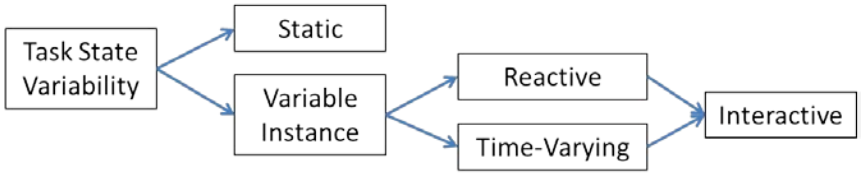

Figure 3: Different Types of Task Variability

These distinctions constrain authoring: static tasks are not typically taught by ITS at all, because many are rote learning that respond equally well to simpler drill-and-practice methods. However, some intelligent systems, such as Pavlik et al.'s (2007) FaCT system, improve recall-type tasks by optimizing spacing effects and the sequence of instance presentations. Static tasks that do not change based on user input are limited to interventions such as highlighting salient features, demonstrating how to find the right solution, responding to a single answer from the learner, or presenting different tasks (e.g., learning prerequisites). It is still possible to adapt to the learner's responses, such as with systems that provide hints and retry attempts in response to wrong answers on multiple choice questions (Conejo, Guzmán, de-la Cruz, & Millán, 2006). However, most ITS tend to focus on reactive and interactive tasks, because learner actions during the task allow a greater ability to target feedback and hints.

Task Assessment

A second major constraint is how well can the ITS measure progress toward pedagogical goals. Since tasks are used to assess learning, measuring progress toward pedagogical goals requires measuring progress toward task goals. In many ways, the ability to measure such progress distinguishes between well-defined and ill-defined domains (Fournier-Viger, Nkambou, & Nguifo, 2010; Nye, Bharathy, Silverman, & Eksin 2012). Any task has two possible levels of introspection: the value for the task state and the value of learner actions. When the goals are known, it is often possible to infer the value of actions from the state if the outcomes are predictable, but this is not always (e.g., due to competing goals to choose between). Table 1 categorizes different combinations of knowing the value of states and the value of learner actions. Knowing the value of states allows measuring good outcomes, while knowing the value of actions allows measuring good process.

	Action Values Known	Action Value Unknown
State Utilities Known	Can evaluate all actions and suggest good solution paths *Ex. Economic decision tree*	Know the value of states, but can't surely deduce what actions do *Ex. Stock market simulation*
State Transition Gradients Known	Can rank states relatively and can suggest next step or bug fix. *Ex. Solving an Algebra problem*	Know better/worse changes, but can't suggest concrete actions *Ex. Automated essay assessment*
State Values Categorical	Can detect good/bad actions, but can't rank improvement *Ex. Going out of bounds in a race*	Ill-defined task, due to emergent dynamics or subjective values *Ex. Building a better world*

Table 1: Measures for Task Goal Progress

If a **state utility** function is available, all states and transitions between states have a known value. For a completely measurable task, the relative value of actions is also known, such as a well-formed economics problem where some actions lead to more profit than others. In other cases, the ultimate impact of actions is uncertain (e.g., a chaotic system like the stock market), but good outcomes can still be measured. Generative simulations with emergent behavior often have this quality (Nye et al., 2012).

When an ITS can detect improvements between states but can't evaluate states exactly, then **state transition gradients** are known. So long as the relationship between learner actions and transitions is known (e.g., problem-solving in Algebra), formative assessments such as model tracing and example tracing can be used (Aleven et al., 2006). When specific learner actions cannot be evaluated easily (e.g., editing a learner's essay), ITS can still provide feedback on the task state. Design-based ITS often use this approach, such as essay-writing ITS that can assess an essay and suggest guidelines to improve the state of the essay (Roscoe & McNamara, 2013). Likewise, when relative value of overall task states is unknown, it is sometimes still possible to assess learner actions. This approach to measurement considers the process, rather than the outcomes. Constraint-based ITS are often applied to these kind of tasks (Mitrovic, 2003). If it is impossible to assess the quality of either the task state or the learner's actions, the task is ill-defined.

In general, when considering Figure 2, higher levels of Bloom's Taxonomy tend to involve tasks closer to the bottom and right of Table 1. The ability to measure progress on task goals is the first constraint on ITS authoring, since it directly constrains the types of feedback and interventions available to the ITS. If it is impossible to evaluate the quality of actions, it is impossible to provide a "correction" or suggest a concrete "next step" action. As such, there is no need to author one. As such, more complex tasks and generative models tend to offer fewer affordances for authoring traditional ITS content, and tend to rely on the natural dynamics of the simulation to provide reactive feedback (i.e., intelligent environments, rather than intelligent tutors).

Task Interface

The interface of the task consists of how it is presented to the learner and of how the learner presents input. More generally, task interfaces are part of the communication module of a classical 4-component ITS diagram (Woolf, 2010). They are the input and output with the learner for the learning task. Typical inputs to an ITS include: discrete selections (e.g., multiple choice), continuous selections (e.g., manipulating sliders for a simulation), formal representations (e.g., math equations, graphs), freeform input (e.g., natural language, freehand sketches), and controlling an avatar (e.g., 3D worlds). The modality of learner input is a further constraint on authoring: the pedagogical interface needs to turn input from the task into something actionable. Ironically, this means that feature-rich inputs, such as natural language, are typically simplified into much simpler representations such as discrete selections (e.g., good/bad answers). The representation of the user input is the final major constraint on authoring.

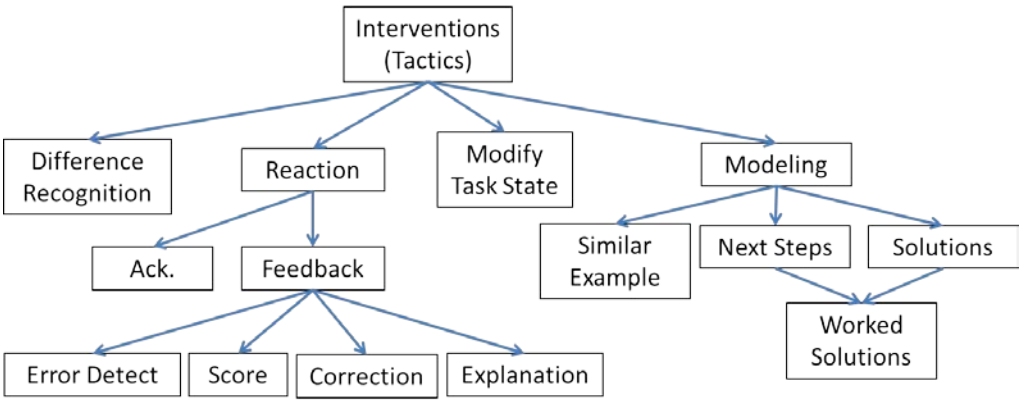

Figure 4: Common Interventions for an ITS

Based on the pedagogical features extracted from the task state and user input, the ITS needs to author various interventions. When extending an ITS to new learning tasks, these interventions are typically a major focus for authoring. When, why, and how the ITS applies its pedagogical strategies and tactics is the main repository of an author's domain pedagogy expertise. Figure 4 displays a variety of options for an ITS to intervene during a task. The most rudimentary of these is to **recognize a difference** between a detected state and some other state (e.g., "You seem to like chocolate ice cream more than the average learner."). Since this response assigns no specific value judgment, it can be used for entirely ill-defined domains by using techniques such as novelty detection (Markou & Singh, 2003). It is also possible to **modify the task state or features** even for tasks that lack clear assessments of state or action value, such as through random perturbations. However, typically an ITS changes the state of a task to make it easier or harder (elastic difficulty). This is done by reducing the degrees of freedom (less options), completing a task step, or increasing the salience of important task features (e.g., highlighting). Another approach is to **react** in response to user input. Even if no assessment can be made for that input, their input can always be **acknowledged** (Ack.). While this might seem like a weak tactic, it is often used to prompt learners to self-reflect, write in a journal, or mark the start or completion of other metacognitive activities.

On well-defined tasks, ITS tactics often take the form of various types of **feedback** or **modeling** effective solution paths (VanLehn, 2006). Feedback is a response to a user action that either presents an answer or otherwise modifies the task state. Common methods of feedback include reacting to errors or good answers (binary assessments), scoring (continuous or ranked assessments), corrections (providing a fix to the answer), and explanations (stating why an answer is right or wrong). Modeling a good answer or solution path is also common. It can be used as feedback, or provided at some point during the task state (e.g., provide the worked solution if the user cannot solve the problem). A few types of modeling are possible. These include presenting the solution to a similar task example, providing the next step(s) to the current task, or providing a good final solution for the student to look at (e.g., a good essay on the same topic they are trying to write about). If the full set of steps and the solution is provided, then the intervention was a worked solution.

The authoring effort for these types of feedback varies significantly. Meaningful corrections and explanations require a much deeper connection to domain pedagogy than simpler feedback such as detecting the existence of errors. Likewise, adding explanations to modeling interventions greatly increases authoring effort, because it moves beyond simply assessing task performance and starts to model how a human teacher or instructor might correct student errors or explain the process of working on the task. However, this authoring effort probably supports some of the most effective ITS tutoring behaviors, since it is ideally based on the expertise accumulated from hundreds of hours of human teaching interactions.

DISCUSSION: COMMON LEARNING TASKS AND TOOLS

Based on these task features, it is possible to break down a variety of common ITS learning tasks and examine how their distinguishing characteristics are reflected in their authoring tools. Three types of tasks can be considered: well-defined tasks (mostly reactive to user input, values for user actions can be known, user inputs are formal and decidable), less-defined tasks (highly interactive, freeform input, lack well-defined goals etc.), and task sandboxes (e.g., complex simulators used to build pedagogical tasks).

Well-Defined Tasks

Table 2 lists common well-defined learning tasks. Two salient learning activities and pedagogical goals for each class of task are noted. With that said, specific tutors may use different types of scaffolding to use the same task to focus on different pedagogical goals and activities, so there can be significant variation on these. In terms of pedagogy, well-defined tasks probably allow the widest set of interventions: because the task state and user input allow clear assessments and have constrained solution paths, the ITS has a fairly clear view of the task and associated pedagogical state.

The most established ITS tasks center on multi-step problem-solving, such as step-by-step math or physics (Ritter, Anderson, Koedinger, & Corbett, 2007; Aleven, McLaren, Sewall, & Koedinger, 2006; VanLehn et al., 2005), diagnosing systems and repairing them (Lajoie & Lesgold, 1989), and building dynamical system models (Biswas et al., 2010; Iwaniec, Childers, VanLehn, & Wiek, 2014). Step-by-step problem-solving tasks can also be presented in 2D or 3D worlds (Rowe, Shores, Mott, & Lester, 2011). Step-by-step problem-solving ITSs tend to author hints and feedback that is conditional on the current task state and the current action (or actions). They also typically provide a full bottom-out worked solution when needed. In-game worked solutions are not common for ITS with avatar input (e.g., 3D worlds), though sometimes recorded cut-screen/video solutions are available. The specific ITS intervention content that is presented is typically tied to general rules that are shared across many task examples (e.g., hints related to the commutative property of addition). As such, authoring these tasks tends to require authoring: 1) A well-defined state representation (e.g., a chess board), 2) a set of domain rules that transform state (e.g., piece move rules), 3) a goal state for the task, 4) a set of expert rules that rely on features of the task state, 5) sometimes buggy rules that represent specific misconceptions to remedy, and 6) templates for feedback and hints that are associated with certain task states or production sequences. In some cases, authoring the task interface is also part of the ITS tool set. The Cognitive Tutor Authoring Tools (CTAT; Aleven et al., 2006) offers an example of fairly mature tools for problem-solving tasks.

Authoring tools for these tasks focus on defining ideal and buggy production rules that can be used to classify task states and learner behavior as they complete the task (Aleven et al., 2006). Ideal production rules can be used to identify points for positive feedback on a good action, or can be used to provide hints about good next steps for a problem. These ideal next steps are inferred by evaluating the chains of actions that are required to reach the task goal (i.e., solution). Similarly, buggy rules can be used to detect specific misconceptions for the learner when they perform certain sequences of actions. Instance-based authoring can be used to infer these rules instead of explicit authoring, through systems such as SimStudent (Matsuda, Cohen, & Koedinger, 2014). Feedback and hints tend to be provided through parameterized templates that can refer to task features. A simpler approach to authoring these tasks involves forcing the learner to stay on a linear or simple branching example-tracing approach, with hints that are specific to certain states or transitions in a problem template (Razzaq, Patvarczki, Almeida, Vartak, Feng, Heffernan, & Koedinger, 2009). At least for certain mathematics topics, tutoring a single solution strategy (or even a single path) is nearly as effective as a more complex structure (Waalkens, Aleven, & Taatgen, 2013; Weitz, Salden, Kim, & Heffernan, 2010).

As such, two alternatives exist to rule authoring: 1) instance-based inference and 2) template-specific tutoring. In the first case (e.g., SimStudent), rules are inferred from expert (and perhaps non-expert) solution paths. This requires an authoring tool that shows a complete interface to the problem, as well as an external judgment of the user's expertise level. This allows skipping explicit rule authoring. In the second case, task templates can be authored with tutoring associated with specific task paths. This type of authoring is also used for other tasks (e.g., constrained choice dialogs with branching), so it is a valuable general-purpose authoring interface in its own right. Much like making inferences across multiple instances, an authoring tool for integrating tutoring templates with task paths also needs to give the author a good view of the task state that is similar to the student's view.

Task	Activities (Top-2)	Pedagogy Goal (Top-2)	Variability	State Values	Action Values	Task Inputs	Interventions to Author
Step-By-Step Math	Apply, Understand	Procedure, Principle	Reactive	Gradients/ Ranks	Known	Formal Expression	Feedback (Any), Next Steps, Similar Example, Worked Solution
Diagnosis & Repair	Analyze, Apply	Process, Procedure	Reactive or Interactive	Gradients/ Ranks	Known	Formal Model	Feedback (Any), Next Steps, Similar Example, Worked Solution
Dynamical Systems	Create, Analyze	Procedure, Process	Reactive or Interactive	Utility or Gradients	Known	Formal Model	Feedback (Any), Next Steps, Similar Example, Worked Solution
Classifying	Understand, Analyze	Concept, Process	Static	Category	Known	Discrete Selection	Feedback (Error, Correct, Expl.), Similar Example
Bug Detect	Analyze, Understand	Process, Concept	Static	Category	Known	Discrete/ Continuous Selection	Feedback (Error, Correct, Expl.), Similar Example
Represent-ation Map	Understand, Apply	Concept, Process	Reactive	Gradients/ Ranks	Known	Formal Models	Feedback (Any), Next Steps, Worked Solution
Concept Map Revise	Understand, Analyze	Concept	Reactive	Gradients/ Ranks	Known	Formal Model	Feedback (Any), Next Steps, Worked Solution
Constrained Choice	Analyze, Evaluate	Process, Principle	Reactive or Interactive	Utility or Gradients	Known	Discrete Choice	Feedback (Any)

Table 2: Common Well-Defined ITS Tasks

A second major class of problems includes pattern matching and classification of examples, such as biological taxonomies (Olney et al., 2012) or identifying errors in a complex task, such as bugs in a

computer program (Carter & Blank, 2013). These ITS tend to provide hints and feedback based on the difference in features between the chosen classification and the ideal one, with strong use of explanation but seldom presenting a step-by-step process. A third class of well-defined ITS tasks includes building formal semantic models from freeform representations (e.g., concept map revision) and converting between different well-defined representations, such as from a graph to an equation (Olney et al., 2012). These also tend to keep track of the difference in features between the current and ideal models, but can also suggest next-step changes because the model can be modified.

Authoring tools for these tasks tend to rely on defining ontologies, concept maps, or other structures that define the features of classes and examples. Each instance in a task can be authored by tagging its features or class memberships, after which hints, counter-examples, or other feedback need to be created. If the pedagogy goals also include following a certain step-by-step process to make classification distinctions, authoring may also require tutoring similar to branching example tracing. Across these types of tasks, a simplified ontology class and instance editor could be quite effective, if it provided clear intervention templates to target differences or similarities between patterns.

Finally, there are constrained choice problems, such as ITS-supported multiple choice or branching dialogs (Kim et al., 2009). These can actually be quite varied, but tend to provide interventions that are either dependent only on the current state (e.g., a hint for choosing the wrong answer) or that are exhaustively defined by a branching state path. This means that authoring such tasks tends to be easier up-front than a problem-solving ITS, but harder to reuse for related tasks. In general, authoring these tasks should be similar to linear example-tracing. However, because the tasks may involve fewer general principles that repeat across examples, the authoring is likely to have more explanations and need fewer templates and parameters.

Less Well-Defined Tasks

Ill-defined and less well-defined tasks are presented in Table 3. These tasks tend to be less-defined because either the goals are not fully-defined, the inputs require natural language processing or are otherwise not formally evaluatable, or the task requires the learner to produce a full artifact before it can be evaluated. Structured argument tasks, such as those used for law (Pinkwart, Ashley, Lynch, & Aleven, 2009) or policy (Easterday & Jo, 2014), work similarly to causal concept map tasks. However, they differ because the goals for argumentation are not always well-defined (i.e., the learner must first choose what to argue). As such, authoring typically requires generating an extensive formal model of free-text sources. This model may be hand-authored or extracted from the associated reference texts. Learners will then need to generate explanations that are logically or causally consistent with the underlying formal model, while supporting the argument goal that the learner has selected. These ITS tend to also require a reusable set of hint and error-correction templates (e.g., for different logical inconsistencies). For specific common misconceptions, rules or constraints may also be used to trigger explanations or modeling behavior (e.g., presenting an analogous case or example). Case-based reasoning is one mechanism for identifying similar examples (Kolodner, Cox, & González-Calero, 2005), and can also be used as a pedagogical strategy for these domains.

The next category of tasks requires the user to create significant artifacts, such as essays or computer programs (Roscoe & McNamara, 2013; Kumar et al., 2013). These ITS tend to calculate an overall quality score, based on a number of calculated features that it can highlight or give hints for improvement. However, unlike an ITS for Algebra, these tutors cannot explicitly correct most problems (e.g., a programming ITS typically cannot fix the learner's code). ITS authoring for these tasks requires defining a set of features that are used to determine quality of the task artifact. Typically, this training is done using supervised learning or hand-authoring. Tutoring often focuses on feedback and hints related to specific features that need improvement for the artifact, as well as an overall quality score.

Task	Activities (Top-2)	Pedagogy Goal (Top-2)	Variability	State Values	Action Values	Task Inputs	Interventions to Author
Structured Argument	Evaluate, Analyze	Principle, Concept	Reactive	Gradients /Ranks	Varies	Formal Model	Feedback (Error, Score, Explain), Similar Example
Functional Coding	Create, Understand	Procedure, Concept	Reactive	Gradients /Ranks	Not Known	Mixed Formal and Freeform	Feedback (Error, Score, Explain), Similar Example
Essay Writing	Create, Analyze	Procedure, Concept	Reactive	Gradients /Ranks	Not Known	Freeform (NLP)	Feedback (Score, Explain), Similar Example
Summaries	Understand	Concept, Process	Reactive	Gradients /Ranks	Not Known	Freeform (NLP)	Feedback (Score, Explain)
Expectation Coverage	Understand, Analyze	Concept, Principle	Interactive	Gradients /Ranks	Known	Freeform (NLP)	Feedback (Any), Next Steps, Worked Solution
Short Answer	Understand, Analyze	Concept, Fact	Reactive/ Interactive	Gradients /Ranks	Known	Freeform (NLP)	Feedback (Any), Similar Example
Open Self-Reflection	Understand	Concept, Process	Static	Not Known	Not Known	Freeform	Difference Recog., Acknowledge
Choice Search	Evaluate, Analyze	Process, Principle	Interactive	Utility or Gradients	Not Known	Freeform or Avatar	Feedback (Any), Similar Example
Setting Goals and Priorities	Evaluate, Analyze	Principle, Process	Interactive	Not Known	Not Known	Varies (Formal or Freeform)	Difference Recog., Similar Example

Table 3: Common Less-Defined ITS Tasks

The next set of less well-defined ITS focus on helping the learner understand, analyze, and evaluate information. They include self-reflection, expectation coverage tasks (Graesser, Chipman, Haynes, & Olney, 2005), summarization and paraphrasing tasks (McNamara, Levinstein, & Boonthum, 2004), and

short-answer tasks. All of these tasks focus on helping the learner understand semantic content and its relationships. Open self-reflection tasks focus on the metacognitive practice of reflecting on the content. As such, in many cases the quality of content is not assessed (e.g., a journaling task). Instead, ITS content focuses on encouraging the habit of self-reflection. In general, many metacognitive tutors focus on building habits, such as encouraging question-asking or hint button use (Azevedo et al., 2010; Roll, Aleven, McLaren, & Koedinger, 2011). Open self-reflection tasks and other content-agnostic tasks tend to require little content authoring, and often require only a set of simple prompt and acknowledgement templates. These reflection prompts can be triggered by task events, or even by general timers.

Expectation coverage tasks unfold over many dialog turns, which assume a part-whole relationship for multiple expectations as part of a full explanation. As such, these ITS must detect multiple related subtopics and provide feedback and hints on each one. Short answer tasks are even more constrained, and their answers tend to be binned into good answers, specific misconceptions, or general bad answers. Expectation coverage tasks tend to contain short answer tasks inside of them, when specific knowledge needs to be assessed. A variety of authoring representations exist for evaluating semantic statements, which fall into three main categories: instance-based authoring, feature authoring, and grammar-based authoring. Instance-based authoring involves generating various classes of answers (i.e., good/bad), which are then used to match against using various algorithms. Feature-based authoring involves creating special features, such as regular expressions or keywords that capture key defining features between different types of answers. Finally, grammar-based authoring involves developing domain-specific parsers that extract domain-relevant relationships from the text.

In all cases, these techniques are used to bin learner answers into specific speech act categories, which can then be associated with feedback, hints, or modeling interventions. Summarization tasks work similarly, but require the learner to rephrase a passage. A successful summary requires the answer to have similar semantics, but dissimilar surface features (i.e., it cannot be identical). These tend to focus on understanding the content, but their quality tends to be rated on a continuous scale, because there are competing feature sets. In addition to assessments of learner input, rules are also required to allow the dialog to progress naturally. In general, a limited set of templates can be sufficient to handle typical ITS tasks. While there are some indications that different levels of knowledge might benefit from different dialog interactions (Nye et al., 2014), similar logical rules for managing dialog interactions can cover a variety of domains.

Task Sandbox Environments

As a final task category, open-ended searches and decision-points for choices are common, particularly in virtual worlds and scenario-based learning. These include looking for a satisfactory or optimal set of actions to some learning task. In many cases, the action sets vary by context and are not known apriori. If the quality of choices can be ranked or their component features ranked, the ITS can provide feedback, hints, and explanations about the quality of actions (Kim et al., 2009; Sottilare, Goldberg, Brawner, & Holden, 2012). However, for a simulation, this information may only be available after the completion of a scenario. The next level of complexity occurs when the task goals are not fully defined, but must be set or prioritized by the learner. For subjective or "wicked" tasks where actions change how goals are understood, goal selection tasks are almost unavoidable (Nye et al., 2012). This tends to occur almost exclusively in simulations or design tasks, where defining and monitoring goals are a major part of the tasks and learning content. These tasks tend to be very hard to tutor directly, and sometimes rely on detecting certain common or uncommon patterns, which are then brought to the attention of the learner.

For complex simulations, many current pedagogical methodologies focus on after-action review procedures. These tend to include a mixture of artifact evaluation (i.e., considering metrics collected from

a simulation run) and self-reflection. After-action reviews have historically been facilitated by a human-in-the-loop and are geared towards focused reflection and knowledge elicitation. The underlying task consists of a series of choices and decision points in a specific scenario, which are translated to overarching learning objectives. These choices are then considered similarly to other types of learning tasks, such as following procedural rules while receiving feedback about deviations from desired performance.

Rather than being more complicated to author, the pedagogy for complex choice tasks is often as simple (or simpler) than highly constrained domains such as mathematics. This is because well-defined domains give many opportunities for clear pedagogical interventions: the state of the task is fully known, completely based on the learner's inputs, and allows immediate feedback. By comparison, a game-based task requires game messages that are sent to assessment models that infer the pedagogical state. As a result, ITS authoring is limited to the data made available by a task interface that was not originally intended to offer pedagogically-useful assessments (e.g., a 3D game engine). As such, an additional authoring layer needs to convert the raw task state into much a pedagogical state. This requires an operational task analysis and authoring tools that transform various task events into pedagogically-useful assessments. This extra assessment layer makes complex environments more difficult to author, and which ultimately limits the interventions that can be authored (e.g., hints and feedback).

While serious games and simulation-based training environments can alleviate this problem in the development phase, many do not. In fact, simulation-based training solutions have increasingly moved toward commercial and open-source game engines to reduce production costs, such as Virtual Battle Space 3 (VBS3), Unity, and the Unreal Game Engine. These sandbox authoring environments enable developers to build complex task scenarios for both individuals and for collaborative/team-based interactions. However, the data generated by these systems follows generic protocols for distributed delivery, such as Distributed Interactive Simulation (DIS) and High-Level Architecture (HLA) that lack any concept of pedagogy or semantics (Hofer & Loper, 1995; Kuhl, Dahmann, & Weatherly, 2000).

The best solution so far to this problem has been to explicitly build a layer of metrics onto the task environment, which are then consumed by the ITS as its pedagogical state. Basically, a simpler task state is constructed from features in the task sandbox, which is then linked to assessments. For example, GIFT provides a generalized architecture that can consume game-message traffic and use this to infer pedagogical conditions linked to a concept hierarchy (Sottilare et. al., 2012). Much of the data captured associates with entity state (i.e., avatars, non-player characters, weapons, machines, vehicles, etc.) location, movement, and action. In short, much of the task state is too low-level or downright irrelevant to ITS behavior. A subset of this data is continuously communicated to GIFT and routed to the Domain Knowledge File (DKF) for managing assessment practices. The DKF is where an author structures: the concept hierarchy associated with a set of tasks, how data is integrated into a concept assessment, and how that data is managed at runtime. Assessments can be authored directly within a DKF or it can be supported by an external assessment engine, such as the Student Information Models for Intelligent Learning Environments (SIMILE; Goldberg, 2013), where the DKF acts by routing data to the appropriate concept assessments (see Figure 5).

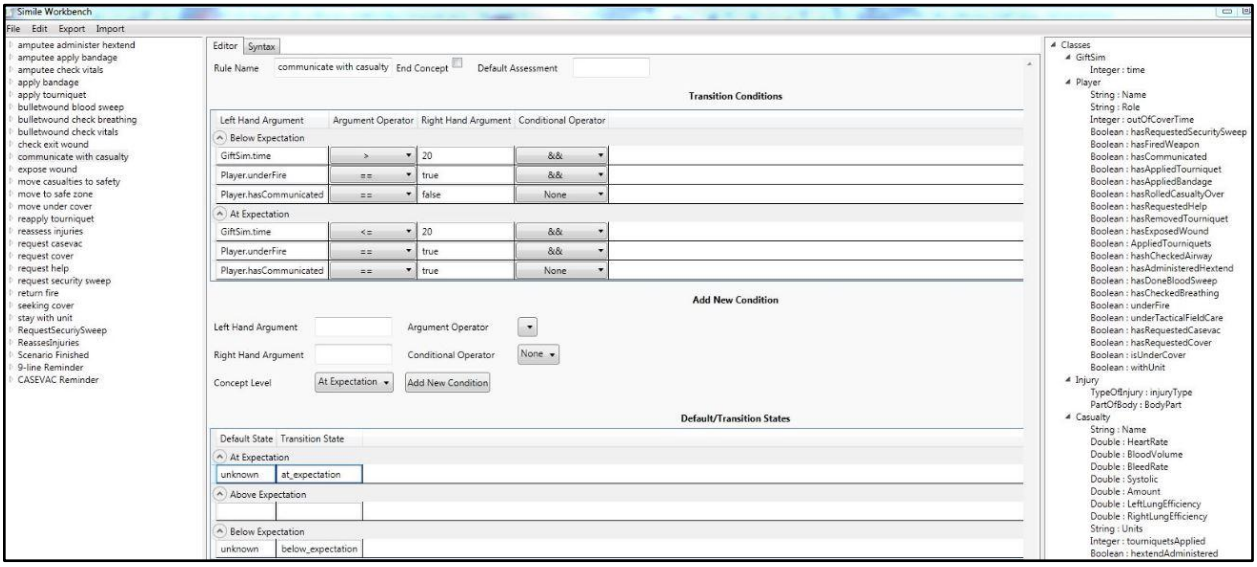

Figure 5. SIMILE Workbench with Authored Assessments for vMedic

However, the reverse direction (i.e., offering specific interventions) has the same complications. Conditions need to consider both the real-time performance and user intention, as well as the possible actions that are available to the user (which must also be relayed to the ITS, to enable suggestions). In GIFT's current use-cases, this information includes tagged locations for the user's position in the environment, the set of entities and objects that are around the user, what entities the user can currently observe, the actions available, and the timers related to task execution.

For example, consider the task of 'maintaining cover' while patrolling a compound, which requires time, location, and entity state data. Waypoints and areas of interest are defined around the compound wall so that the player can be tracked to monitor patrol progress. An author can then define assessments based on if the user has reached certain waypoints within a specific timeframe. In addition, if a scenario author determines that a user should adjust their entity's state within specific areas of interest, such as adjusting their stance from standing to kneeling due to a wall being low, then the author can associate assessments to inform student action in relation to performance criteria. By knowing this context, it is also possible to deliver real-time feedback based on the actions and current assessment information.

To further complicate the issue, these types of interactive environments are excellent for collaborative and team-based learning events. From the ITS perspective, this requires additional modeling dependencies that associate interaction and intention with team oriented skills and attributes. While there is extensive literature on what makes effective teams and effective team training approaches (Salas et al., 2008), how to establish these practices in an automated fashion is a challenge. Beyond modeling individualized tasks and how interaction in a virtual environment can infer competencies, additional assessments must analyze group-level data that is aggregated across a set of users. These assessments include team cohesion, trust, communication, and shared cognition. While this field is a wide-open research area, architecture's like GIFT must be designed to facilitate the type of modeling techniques that are based on trends across users rather than within users. In terms of authoring, the challenge is taking available data and translating it to team-based inferences that can designate performance across a set of concepts. In addition, how to react to these assessments needs to be explored, such as how interventions are handled and how they are communicated to a team.

RECOMMENDATIONS AND FUTURE RESEARCH

Across this book, examples and lessons learned for authoring each type of learning task are discussed. By identifying common learning tasks in ITSs, it should be possible to develop general authoring interfaces that make authoring for each type of task intuitive and effective. In some cases, highly effective authoring models already exist and might serve as exemplars for task-specific authoring tools in generalized ITSs such as GIFT. Ideally, these authoring tools should collect information in ways that are familiar to instructors and other domain pedagogy experts. Form-based authoring, example-based authoring, and supervised tagging are all reasonable approaches that are particularly attractive.

However, there are also learning tasks that do not yet have well-established techniques that allow non-technical domain experts to easily author content. For example, authoring real-time assessments for complex tasks remains more of an art than a science. At least some of this authoring involves mapping simulation or virtual world events to pedagogical features. For this type of authoring, even if game worlds had integrated pedagogical tools, a tool to easily map raw simulation data to metrics may be hard to use by the domain experts, even if it is well-designed. Similarly, authoring for ITS tasks with multiple learners is a poorly understood area. For example, team-based tutoring requires assessment and intervention at multiple levels (e.g., individual and group). Further research on such tasks and exploration of different types of authoring approaches may be needed before good examples of such authoring tools become clear.

REFERENCES

Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. R. (2006). The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley, & T. Chan (Eds.) *Intelligent Tutoring Systems (ITS) 2006* (pp. 61-70). Springer Berlin Heidelberg.

Azevedo, R., Johnson, A., Chauncey, A., & Burkett, C. (2010). Self-regulated learning with MetaTutor: Advancing the science of learning with MetaCognitive tools. In M. S. Khine & I. M. Saleh (Eds.) *New Science of Learning* (pp. 225-247). Springer New York.

Biswas, G., Jeong, H., Kinnebrew, J. S., Sulcer, B., & Roscoe, R. D. (2010). Measuring Self-Regulated Learning Skills through Social Interactions in a teachable Agent Environment. *Research and Practice in Technology Enhanced Learning*, *5*(2), 123-152.

Carter, E., & Blank, G. D. (2013). An Intelligent Tutoring System to Teach Debugging. In H. C. Lane, K. Yacef, J. Mostow, & P. Pavlik (Eds.) *Artificial Intelligence in Education (AIED) 2013* (pp. 872-875). Springer Berlin Heidelberg.

Clark, D. (2014). Bloom's taxonomy of learning domains. Retrieved August, 26, 2014.

Clark, R. C. (2002). Applying cognitive strategies to instructional design. *Performance Improvement, 41*(7), 8-14.

Clark, R. C., & Mayer, R. E. (2011). *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. John Wiley & Sons.

Conejo, R., Guzmán, E., de-la Cruz, J. L. P., & Millán, E. (2006). An empirical study about calibration of adaptive hints in web-based adaptive testing environments. In V. Wade, H. Ashman, & B. Smyth (Eds.) *Adaptive Hypermedia and Adaptive Web-Based Systems* (pp. 71-80). Springer Berlin Heidelberg.

Easterday, M. W., & Jo, I. Y. (2014). Replay Penalties in Cognitive Games. In S. Trausan-Matu, K. Boyer, M. Crosby, & K. Panourgia (Eds.) *Intelligent Tutoring Systems (ITS) 2014* (pp. 388-397). Springer Berlin Heidelberg.

Fournier-Viger, P., Nkambou, R., & Nguifo, E. M. (2010). Building intelligent tutoring systems for ill-defined domains. In R. Nkambou, R. Mizoguchi, & J. Bourdeau (Eds.) *Advances in Intelligent Tutoring Systems* (pp. 81-101). Springer Berlin Heidelberg.

Goldberg, B. & Spain, R. (2014). Creating the Intelligent Novice: Supporting Self-Regulated Learning and Metacognition in Educational Technology. In R. Sottilare, A. Graesser, X. Hu, and B. Goldberg (Eds.) *Design Recommendations for Intelligent Tutoring Systems, Vol. 2: Instructional Management* (pp. 109-134). U.S. Army Research Laboratory.

Goldberg, B. (2013). *Explicit Feedback Within Game-Based Training: Examining the Influence of Source Modality Effects on Interaction.* Ph.D., University of Central Florida.

Graesser, A. C., Chipman, P., Haynes, B. C., & Olney, A. (2005). AutoTutor: An intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions on Education, 48*(4), 612-618.

Hofer, R. C., & Loper, M. L. (1995). DIS today [Distributed interactive simulation]. *Proceedings of the IEEE, 83*(8), 1124-1137.

Iwaniec, D. M., Childers, D. L., VanLehn, K., & Wiek, A. (2014). Studying, teaching and applying sustainability visions using systems modeling. *Sustainability*, *6*(7), 4452-4469.

Kim, J. M., Hill, Jr, R. W., Durlach, P. J., Lane, H. C., Forbell, E., Core, M., ... & Hart, J. (2009). BiLAT: A game-based environment for practicing negotiation in a cultural context. *International Journal of Artificial Intelligence in Education*, *19*(3), 289-308.

Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2012). The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, *36*(5), 757-798.

Kolodner, J. L., Cox, M. T., & González-Calero, P. A. (2005). Case-based reasoning-inspired approaches to education. *The Knowledge Engineering Review*, *20*(03), 299-303.

Kuhl, F., Dahmann, J., & Weatherly, R. (2000). *Creating computer simulation systems: an introduction to the high level architecture*. Prentice Hall PTR Upper Saddle River.

Kumar, A. N. (2013). Using Problets for problem-solving exercises in introductory C++/Java/C# courses. In *IEEE* 2013 *Frontiers in Education Conference* (pp. 9-10). IEEE Press.

Lajoie, S. P., & Lesgold, A. (1989). Apprenticeship Training in the Workplace: Computer-Coached Practice Environment as a New Form of Apprenticeship. *Machine-Mediated Learning*, *3*(1), 7-28.

Markou, M., & Singh, S. (2003). Novelty detection: a review- part 1: Statistical approaches. *Signal Processing*, *83*(12), 2481-2497.

Matsuda, N., Cohen, W. W., & Koedinger, K. R. (Online First). Teaching the Teacher: Tutoring SimStudent Leads to More Effective Cognitive Tutor Authoring. *International Journal of Artificial*

Intelligence in Education, 1-34.

McNamara, D. S., Levinstein, I. B., & Boonthum, C. (2004). iSTART: Interactive strategy training for active reading and thinking. *Behavior Research Methods, Instruments, & Computers*, *36*(2), 222-233.

Merrill, M. D. (1983). Component Display Theory. In C. M. Reigeluth (Eds.), *Instructional Design Theories and Models: An Overview of their Current States* (279-333). Hillsdale, NJ: Lawrence Erlbaum.

Mitrovic, A. (2003). An intelligent SQL tutor on the web. *International Journal of Artificial Intelligence in Education*, *13*(2), 173-197.

Nye, B. D., Bharathy, G. K., Silverman, B. G., & Eksin, C. (2012). Simulation-Based training of ill-defined social domains: the complex environment assessment and tutoring system (CEATS). In S. A. Cerri, W. J. Clancey, G. Papadourakis, & K. Panourgia (Eds.) *Intelligent Tutoring Systems (ITS) 2012* (pp. 642-644). Springer Berlin Heidelberg.

Nye, B. D., Graesser, A. C., & Hu, X. (2014). AutoTutor and Family: A review of 17 years of natural language tutoring. *International Journal of Artificial Intelligence in Education*, *24*(4), 427-469.

Nye, B. D., Rahman, M. F., Yang, M., Hays, P., Cai, Z., Graesser, A., & Hu, X. (2014). A tutoring page markup suite for integrating Shareable Knowledge Objects (SKO) with HTML. In *Intelligent Tutoring Systems (ITS) 2014 Workshop on Authoring Tools*, (pp. 1-8). CEUR.

Ogan, A., Walker, E., Baker, R. S., Rebolledo Mendez, G., Jimenez Castro, M., Laurentino, T., & de Carvalho, A. (2012). Collaboration in Cognitive Tutor use in Latin America: Field study and design recommendations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1381-1390). ACM.

Olney, A. M., D'Mello, S., Person, N., Cade, W., Hays, P., Williams, C., ... & Graesser, A. (2012). Guru: A computer tutor that models expert human tutors. In S. A. Cerri, W. J. Clancey, G. Papadourakis, & K. Panourgia (Eds.) *Intelligent Tutoring Systems (ITS) 2012* (pp. 256-261). Springer Berlin Heidelberg.

Pavlik Jr., P. I., Presson, N., Dozzi, G., Wu, S., MacWhinney, B., Koedinger, K. R. (2007). The FaCT (Fact and Concept Training) System: A New Tool Linking Cognitive Science with Educators. In McNamara, D., Trafton, G. (eds.) *Proceedings of the Twenty-Ninth Annual Conference of the Cognitive Science Society*, pp. 397–402. Lawrence Erlbaum: Mahwah.

Pinkwart, N., Ashley, K., Lynch, C., & Aleven, V. (2009). Evaluating an intelligent tutoring system for making legal arguments with hypotheticals. *International Journal of Artificial Intelligence in Education*, *19*(4), 401-424.

Razzaq, L., Patvarczki, J., Almeida, S. F., Vartak, M., Feng, M., Heffernan, N. T., & Koedinger, K. R. (2009). The Assistment Builder: Supporting the life cycle of tutoring system content creation. *IEEE Transactions on Learning Technologies*, *2*(2), 157-166.

Ritter, S., Anderson, J. R., Koedinger, K. R., & Corbett, A. (2007). Cognitive Tutor: Applied research in mathematics education. *Psychonomic Bulletin & Review*, *14*(2), 249-255.

Roscoe, R. D., & McNamara, D. S. (2013). Writing pal: Feasibility of an intelligent writing strategy tutor in the high school classroom. *Journal of Educational Psychology*, *105*(4), 1010.

Roll, I., Aleven, V., McLaren, B. M., & Koedinger, K. R. (2011). Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and Instruction*, *21*(2), 267-280.

Rowe, J. P., Shores, L. R., Mott, B. W., & Lester, J. C. (2011). Integrating learning, problem solving, and engagement in narrative-centered learning environments. *International Journal of Artificial Intelligence in Education*, *21*(1), 115-133.

Salas, E., DiazGranados, D., Klein, C., Burke, C. S., Stagl, K. C., Goodwin, G. F., & Halpin, S. M. (2008). Does team training improve team performance? A meta-analysis. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 50(6), 903-933.

Silverman, B. G., Pietrocola, D., Nye, B., Weyer, N., Osin, O., Johnson, D., & Weaver, R. (2012). Rich socio-cognitive agents for immersive training environments: case of NonKin Village. *Autonomous Agents and Multi-Agent Systems*, *24*(2), 312-343.

Sottilare, R. A., Goldberg, B. S., Brawner, K. W., & Holden, H. K. (2012). A modular framework to support the authoring and assessment of adaptive computer-based tutoring systems (CBTS). In *Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2012*.

Vanlehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, *16*(3), 227-265.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., ... & Wintersgill, M. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, *15*(3), 147-204.

Waalkens, M., Aleven, V., & Taatgen, N. (2013). Does supporting multiple student strategies lead to greater learning and motivation? Investigating a source of complexity in the architecture of intelligent tutoring systems. *Computers & Education*, *60*(1), 159-171.

Weitz, R., Salden, R. J., Kim, R. S., & Heffernan, N. T. (2010). Comparing worked examples and tutored problem solving: Pure vs. mixed approaches. In S. Ohlsson & R. Catrambone (Eds.) *Proceedings of the Thirty-Second Annual Meeting of the Cognitive Science Society* (pp. 2876-2881).

Woolf, B. P. (2010). *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann.