Example-Tracing Tutors: A New Paradigm for Intelligent Tutoring Systems

Vincent Aleven, Bruce M. McLaren, Jonathan Sewall, Kenneth R. Koedinger.

Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA, USA.
aleven@cs.cmu.edu, bmclaren@cs.cmu.edu, sewall@cs.cmu.edu, koedinger@cmu.edu

Abstract. Key success criteria for an ITS authoring tool are that (1) the tool supports the creation of effective tutoring systems, (2) the tool can be used to build tutors across a wide range of application domains, (3) authoring with the tool is cost-effective, (4) the tool supports easy deployment and delivery of tutors in a variety of technical contexts, (5) tutors created with the tool are maintainable, and (6) if tutors are used in a research context, the tool must support research-related functionality. The Cognitive Tutor Authoring Tools (CTAT) address all of these requirements to a substantial degree, fully meeting most of them.

CTAT supports the creation of both Cognitive Tutors (Koedinger & Corbett, 2006) and a newer type of tutors called *example-tracing tutors*. This paper focuses on the latter. Example-tracing tutors evaluate student behavior by flexibly comparing it against examples of correct and incorrect problem-solving behaviors. Example-tracing tutors are capable of sophisticated tutoring behaviors: they provide step-by-step guidance on complex problems while recognizing multiple student strategies and maintaining multiple interpretations of student behavior. On that basis, they should be deemed intelligent tutoring systems. Example-tracing tutors can be built without programming, through drag-and-drop techniques and programming by demonstration. Example-tracing tutors have been built and used in real educational settings for a wide range of application areas.

Development time estimates from a large number of projects that have used CTAT suggest that CTAT improves the cost-effectiveness of ITS development by a factor of 4-8, compared to "historical" estimates of tutor development time. Although there is a lot of variability in these kinds of estimates, they nonetheless support our hope that lowering the skill requirements for tutor creation is a key step toward widespread use of ITS technology. The main contributions of the work are the example-tracing tutor technology and tools for building these types of tutors without programming.

Keywords. authoring tools, example-tracing tutors, cognitive tutors, ITS architectures, behavior of tutoring systems

INTRODUCTION

Intelligent tutoring systems (ITSs) have been shown to lead to impressive improvement in student learning in a range of domains, using a variety of different approaches (Beal, Walles, Arroyo, & Woolf, 2007; Graesser, Chipman, Haynes, & Olney, 2005; Koedinger, Anderson, Hadley, & Mark, 1997; Martin & Mitrovic, 2002; Mostow & Beck, 2007; Rickel & Johnson, 1999; VanLehn, Lynch, Schultz, Shapiro, Shelby, *et al.,* 2005; Mitrovic, McGuigan, Martin, Suraweera, Milik, & Holland, 2008). However, they have traditionally been difficult to build (e.g., Murray, 2003; Anderson, 1993). Many ideas have been offered to make ITSs easier to build: ITS design patterns (Harrer, Pinkwart, McLaren, & Scheuer, in press; Devedzic & Harrer, 2005), reusable components and learning objects (Koedinger, Suthers, & Forbus, 1999; Ritter & Koedinger, 1997; Ritter, Blessing, & Wheeler, 2003),

community authoring (Aleahmad, Aleven, & Kraut, 2008), use of off-the-shelf tools as an integrated part of the ITS development process (e.g., Aleven, Sewall, McLaren, & Koedinger, 2006), and authoring tools (e.g., Murray, Blessing & Ainsworth, 2003). In our view, all of these developments and approaches are likely to contribute to making ITS technology widespread in the years ahead. We expect that authoring tools will turn out to be the centerpiece of that endeavor.

Many ITS authoring systems have been developed, over 25 by most counts, since the earliest days of intelligent tutoring systems (Murray, 2003). Typically, each authoring tool focuses on a particular kind of ITSs, such as constraint-based tutors (Mitrovic *et al.*, 2006) and model-tracing tutors (Blessing, Gilbert, Ourada, & Ritter, 2007). In addition to trying to make tutors easier to build, there have been a variety of objectives behind ITS authoring systems, including helping authors transfer knowledge more accurately into the resulting tutoring system, supporting good (and consistent) design principles, and enabling rapid prototyping of ITSs. Different tools have focused on different aspects of tutoring, for instance, support of different tutoring strategies (Ainsworth *et al.*, 2003), tutoring within a simulation context (Munro, 2003), supporting multiple knowledge types (Halff *et al.*, 2003), and use of hypermedia (Brusilovsky, 2003).

Our "holy grail" is to create cost-effective tools that *non-programmers* can use to create and deliver, for real use, sophisticated tutors, a goal we have taken significant steps toward achieving. We report on our 6-plus years of experience with a suite of authoring tools called the *Cognitive Tutor Authoring Tools* (CTAT). Our experience is not just as tool developers, but also as users of our own tools, as consultants to learning science researchers who build tutors as a vehicle for learning science experiments, as consultants to developers of tutors for real-world use, as instructors of students learning about intelligent tutoring systems through practice with these tools, and finally, as academic advisors to graduate students using the tools for their research projects.

CTAT supports development of two types of ITSs, Cognitive Tutors (Anderson, Corbett, Koedinger, & Pelletier, 1995; Koedinger & Aleven, 2007; Koedinger & Corbett, 2006) and a much newer type of tutors called *example-tracing tutors* (Koedinger, Aleven, Heffernan, McLaren, & Hockenberry, 2004). Cognitive Tutors rely on cognitive theory and cognitive modeling; they require sophisticated AI programming skills to develop. They have a long history and are being used extensively in US schools (in more than 2,600 of them, at the time of this writing). Cognitive Tutors for high-school mathematics have proven in many studies to improve student learning (Koedinger & Aleven, 2007)[1]. Example-tracing tutors are often (though not always) behaviorally indistinguishable from Cognitive Tutors (Aleven, Sewall, McLaren, & Koedinger, 2006; Koedinger *et al.,* 2004). As discussed further below, example-tracing tutors implement many of the behaviors typical of ITSs (VanLehn, 2006). In fact, they go well beyond VanLehn's basic requirement that in order to be considered an ITS, a system must provide *step-by-step guidance* within problem-solving activities. Example-tracing tutors are much easier to build than Cognitive Tutors. They can be built entirely without programming, namely, through drag-and-drop and programming by demonstration techniques (Lieberman, 2001; Myers, McDaniel, & Kosbie, 1993). CTAT has been used to build a diverse set of example-tracing tutors that have been used in real educational settings (see for example Appendix A or the many figures in this paper). In this paper, we focus on example-tracing tutors. We consider the main contributions of the work presented in this paper to be both the example-tracing tutor technology and the tools for building these kinds of tutors without needing to do any programming.

[1] The Cognitive Tutors for high-school and middle-school mathematics that were built in our lab pre-date CTAT. They formed the inspiration for developing a set of authoring tools.

The main idea behind example-tracing tutors is straightforward. Just as model-tracing tutors work by tracing and interpreting student behavior with reference to a *cognitive model* (an executable simulation of student thinking that can solve problems in the way that students will learn how to), example-tracing tutors work by interpreting student behavior with reference to specific *examples* of problem-solving behavior. These examples comprise acceptable solution paths for a given tutor problem (i.e., the solution space for a given problem in the sense of Newell and Simon (1972)). Using CTAT, these behavioral examples can be created by teachers or subject-matter experts without programming – simply by demonstrating how problems should be solved. Example-tracing tutors were originally conceived as a tool for doing cognitive task analysis and for rapidly prototyping tutor behavior en route to developing a Cognitive Tutor. However, a number of extensions to the basic idea have turned it into a viable methodology for building tutors. In particular, we have added a number of techniques for *flexibly* matching student behavior against behavioral examples, so that a tutor can recognize a wide range of correct student behavior, rather than only the examples exactly as they were demonstrated.

CTAT belongs to a category of authoring tools (Murray, 2003) that focus on creating a model of domain expertise that the tutoring system uses as a comparison point for student behavior. The domain model is typically executable (i.e., one can run the model to solve a given problem), and student actions that diverge from the model are used to provide hints and generate feedback. Other examples of authoring systems that belong to this category are the Cognitive Model SDK (Blessing *et al.*, 2007), which supports authoring of an ACT-R-like cognitive model, and DIAG (Towne, 2003), which supports the development of a diagnostic model as the basis for tutoring. CTAT can be considered the "next generation" of DEMONSTR8, an early authoring system prototype that pioneered the idea of developing tutors through programming-by-demonstration (Blessing, 2003). DEMONSTR8 pre-dates example-tracing tutors; it focused on rule-based cognitive modeling and model-tracing tutors. To the best of our knowledge, however, it was not used to create a real-world tutoring system. Another early ITS authoring system that pioneered programming by demonstration techniques is DISCIPLE (Tecuci & Keeling, 1999). DISCIPLE was used to create a tutor for history problem solving that has been evaluated in an actual middle-school classroom, but we are not aware that DISCIPLE has seen much use since. Both systems use (interactive) machine learning techniques to learn from examples and explanations provided by an author. Example-tracing tutors, by contrast, directly use behavioral examples to provide tutoring, without the use of machine learning. Research is underway within the CTAT group on using machine learning to induce rule-based cognitive models from author-demonstrated examples (Matsuda, Cohen, Sewall, Lacerda, & Koedinger, 2007; Matsuda, Cohen, & Koedinger, 2005). In the current paper, however, we focus on example-tracing tutors.

CTAT is perhaps furthest down the path of providing a tool that non-programmers can use to develop real-world intelligent tutoring systems, although it is not the only such tool. The Assistments authoring tool (Razzaq *et al.*, 2005) and REDEEM (Ainsworth *et al.,* 2003) also support tutor authoring without programming, but of simpler tutors than CTAT. ASPIRE (Mitrovic *et al*., 2006) is another ITS authoring that makes it possible for non-programmers to create intelligent tutors, in particular constraint-based tutors. Using ASPIRE, complete constraint-based tutors can be built without programming, including a text-based user interface, a domain ontology, and a set of constraints used to evaluate student solutions. However, ASPIRE, unlike CTAT, is not integrated with a drag-and-drop interface development tool (i.e., off-the-shelf development tools such as NetBeans or Flash, as discussed below). With CTAT's predefined sets of tutor-enabled widgets, an author has

many more options for creating a user interface that "makes thinking visible" (Anderson, Corbett, Koedinger, & Pelletier, 1995) without needing to do any programming.

In this paper, we describe the behavior and inner workings of example-tracing tutors, and illustrate how they can be developed with CTAT. We also provide evidence of CTAT's effectiveness in the form of (a) "vote-with-your-feet" evidence (i.e., widespread use), and (b) estimates of tutor development times. We offer a number of general requirements for ITS authoring tools and discuss how CTAT (and specifically, its facilities for creating example-tracing tutors) stacks up with respect to these requirements. An authoring tool suite should:

1. support authoring of effective, computer-based tutors;
2. facilitate the development of tutors across a range of application domains;
3. support cost-effective tutor development (i.e., minimize time and money, compared to using different tools/methodologies);
4. support delivery of tutors in a wide variety of technical contexts characterized by different interface technologies, content management systems, e-learning frameworks, student management systems, and web technologies;
5. create tutors that are easy to maintain (e.g., easy to modify when requirements change);
6. support research use of tutors through such functionality as logging of student-tutor interactions, application of different treatments to different students, and on-line assessment.

These requirements represent some of the lessons learned in the CTAT project. We do not expect them to be controversial, although it is perhaps surprising that only one of the six requirements focuses on tutoring behavior (namely, requirement #1). During the CTAT project, we shifted from an early focus on "affordable authoring" of tutor behavior to the more encompassing problems of easy deployment, delivery, and maintenance of tutors. This shift was driven largely by the needs of CTAT users who were creating tutors for use in real educational settings.

THE BEHAVIOR OF EXAMPLE-TRACING TUTORS

Like many other ITSs, example-tracing tutors help students acquire a complex cognitive skill through guided practice (but see duBoulay, 2006 for a broader notion of ITSs). In this section, we establish that example-tracing tutors are capable of a large subset of the behaviors catalogued in Kurt VanLehn's (2006) paper "The Behavior of Tutoring Systems," the same subset, in fact, as that covered by Cognitive Tutors. (This is not to say that there are no behavioral differences between example-tracing tutors and Cognitive Tutors. Ultimately, Cognitive Tutors are more flexible. However, this flexibility is not always needed, and comes at the considerable expense of having to create a rule-based cognitive model. We briefly return to this topic in a later section.) In the next section, we show further that example-tracing tutors are capable of sophisticated behaviors not differentiated in VanLehn's categories. We illustrate our argument with example-tracing tutors that have been developed in our lab and that have been used in research studies in real educational settings, a tutor in the domain of chemistry (stoichiometry – see Figure 1), and one in the domain of 6th-grade fractions (see Figure 2).

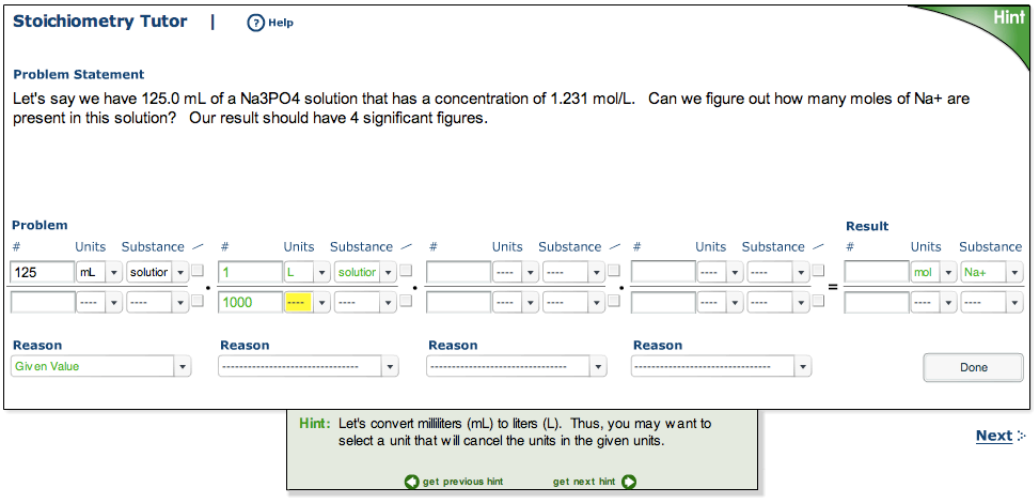

Figure 1: An example-tracing tutor for high-school stoichiometry. This tutor has been used in several studies in real high school classrooms. Approximately 200 students have used this tutor for an average of approximately 1 ¼ hours (McLaren, Lim, & Koedinger, 2008a).

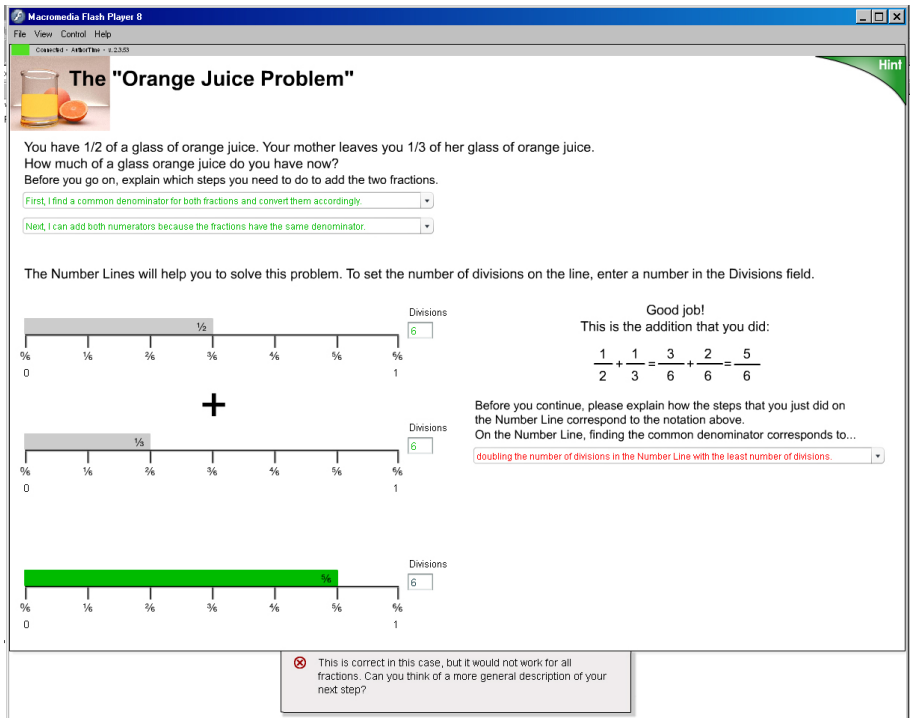

Figure 2: An example-tracing tutor for 6th-grade fractions learning. This tutor has been used in a study in a middle school with 130 participating students, who used the tutor for 2.5 hours.

As is typical of ITSs, CTAT-built tutors provide an interface that makes thinking visible (e.g., Anderson *et al.,*1995), in the sense that they lay out (or prompt students to enter) intermediate steps in the solution of a problem. For example, in the stoichiometry tutor of Figure 1, the student is prompted to provide terms of a ratio equation and elements of each term. In the fractions tutor of Figure 2, students are prompted to add fractions by first converting them to a common denominator using an interactive number line. In addition to an interface that makes thinking visible, example-tracing tutors provide various types of guidance that are typical of ITSs, divided by VanLehn (2006) into an "inner loop" and "outer loop" (see Table 1). The outer loop pertains to the selection of appropriate problems for a student to solve. The inner loop denotes the support that the tutor provides a student within a single problem, including step-by-step guidance while the student solves the problem and assistance with end-of-problem reflection. We review systematically which of the main inner and outer loop behaviors CTAT supports, because we believe that a good way of understanding an authoring tool is to look at the behaviors of the tutors that can be built with them.

Table 1: CTAT's Fall 2008 coverage of the categories of tutoring behavior identified by VanLehn (2006).

Inner loop (within-problem guidance)	
+	Minimal feedback on steps - classified as correct, incorrect, or suboptimal
+	Immediate feedback
+/–	Delayed feedback – not built-in, but certain forms can be authored
–	Demand feedback
+	Error-specific feedback
+	Hints on the next step
+	Assessment of knowledge
–	End-of-problem review of the solution
Outer loop (problem selection options)	
–	Student picks
+	Fixed sequence
–	Mastery learning
(+)	Macroadaptation

Legend	
+	CTAT supports it
(+)	CTAT will soon support it
+/–	CTAT supports a limited form of it
–	CTAT does not support it

With respect to the inner loop, example-tracing tutors provide correctness feedback on all problem-solving steps by the student, and may provide specific feedback messages for commonly occurring errors. This feedback is given after each step and is therefore immediate feedback. The CTAT example-tracing algorithm does not support delayed feedback, although some specific ways of delaying feedback can be authored within CTAT, as illustrated below with a demo tutor for factoring quadratics (see also Roll *et al.,* 2006; 2007). Second, example-tracing tutors provide next-step hints at the student's request. Third, they assess student knowledge. Following the ACT-R theory of cognition and learning (Anderson & Lebière, 1998), and following the way Cognitive Tutors operate (e.g., Koedinger & Aleven, 2007; Koedinger & Corbett, 2006), we consider a complex cognitive skill to be composed of many fine-grained "knowledge components" that can be acquired and strengthened

separately. In its inner loop, an example-tracing tutor maps student problem-solving behavior onto knowledge components, based on a mapping between problem steps and knowledge components provided by the author. Finally, CTAT does not have built-in facilities to support students in reviewing and reflecting on their solution at the end of a problem.

With respect to the outer loop (i.e., the way in which problems are selected for students to solve), CTAT currently supports fixed problem sequences only, where the tutor is in charge of problem selection. CTAT will soon be extended to provide a form of individualized problem selection based on Corbett and Anderson's (1995) Bayesian "knowledge tracing" algorithm. Using this algorithm, CTAT-based tutors will track how well the student masters each knowledge component targeted in a particular unit of instruction, and will select problems that involve unmastered knowledge components. VanLehn (2006) refers to this kind of task selection based on fine-grained knowledge assessment as "macroadaptation," although to make matters confusing, in the Cognitive Tutor literature, it has always been referred to as "cognitive mastery learning" (Corbett & Anderson, 1995). CTAT does not provide facilities for what VanLehn calls "mastery learning," which under his definition is a less-sophisticated form of problem selection based on more aggregate or coarse-grained measures of student performance. At this point, there are no plans for adding this kind of problem selection algorithm to CTAT.

According to VanLehn (2006), the existence of an inner loop is what defines ITSs *vis-à-vis* other forms of computer-based instruction. However, we view this definition as overly broad, as it encompasses very simple forms of computer-based instruction (for example, simple tutors for two-step problems that do not allow for multiple different answers) that belie the historic roots of ITSs in artificial intelligence, cognitive science, and investigations into the epistemology of knowledge (e.g., Wenger, 1987). We propose that example-tracing tutors should be regarded as ITSs not just because they have an inner loop, but because of the flexibility with which they assess student behavior within the inner loop: They are capable of providing guidance with respect to *multiple strategies* for solving a given problem, regardless of which strategy the student decides to take. Further, example-tracing tutors are capable of entertaining *multiple interpretations* of student behavior, when a student action can be interpreted in multiple ways. These properties are illustrated below.

HOW EXAMPLE-TRACING TUTORS WORK

In this section, we focus on the workings of the inner loop of example-tracing tutors, the way in which they provide step-by-step guidance to students for a given problem. As mentioned, example-tracing tutors interpret a student's solution steps (and hint requests) with respect to a predefined solution graph for the given problem, which, following Newell and Simon (1972), we call a "behavior graph." As discussed in the next section, these graphs can be created without programming. We review the properties of behavior graphs, the process of interpreting student behavior against such a graph (called "example tracing," by analogy to model tracing), and the mechanisms in CTAT that enhance the flexibility of the example-tracing process, so that a wider range of student behavior can be recognized as correct than just literally the sequence(s) of steps captured in the behavior graph.

A behavior graph is a directed, acyclic graph that represents acceptable ways of solving a problem. The links in the graph represent problem-solving actions, and the nodes represent problem-solving states. A behavior graph may contain multiple paths, corresponding to different ways of solving a problem. It may also contain links that represent incorrect behavior, marked as such by the

author who created the graph. Let us consider, for example, the stoichiometry problem shown in Figure 3. The stoichiometry tutor is an example-tracing tutor developed using CTAT that has been used in a series of classroom studies (McLaren, Lim, & Koedinger, 2008a; 2008b, McLaren *et al.,* 2007, McLaren *et al*., 2006). Stoichiometry involves basic mathematics to solve elementary chemistry problems; solving problems in this domain requires applying concepts, such as unit conversions and molecular weight, in solving equations. The behavior graph of Figure 4 represents a section of the solution to the stoichiometry problem in Figure 3.

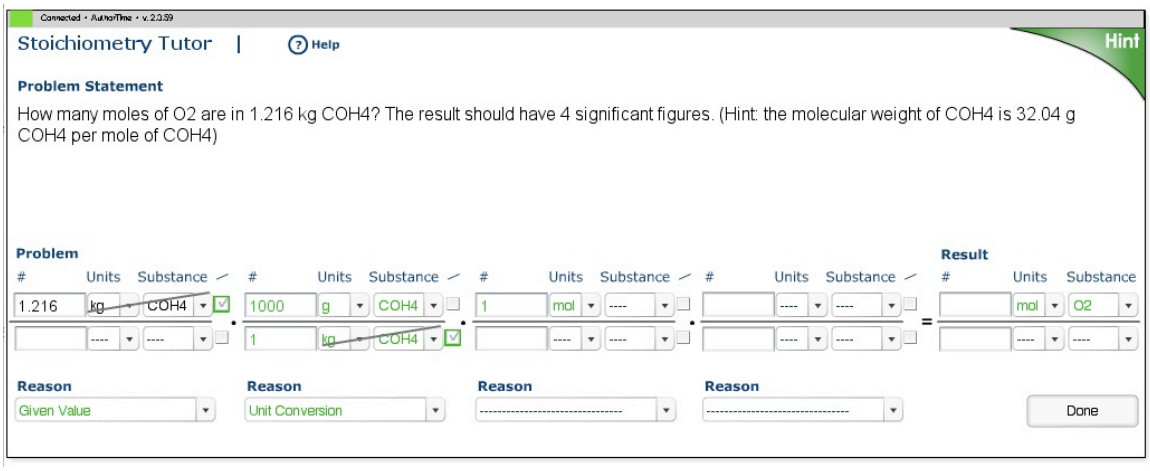

Figure 3: The stoichiometry tutor user interface[2]

The optimal solution to this problem, according to a chemistry instructor, is:

$$
\begin{aligned}
1.216 \text{ kg } COH_4 &\times (1000 \text{ g } COH_4 / 1 \text{ kg } COH_4) \\
&\times (1 \text{ mol } COH_4 / 32.04 \text{ g } COH_4) \\
&\times (1 \text{ mol } O_2 / 2 \text{ mol } COH_4) \\
&= 18.98 \text{ mol } O_2
\end{aligned}
$$

That is, the student is expected to take the given value (1.216 kg COH_4) and multiply it by three terms in order to calculate the final answer of 18.98 mol O_2: first by a term that represents a unit conversion (1000 g COH_4 / 1 kg COH_4, as has already been done in Figure 3), then by a term that represents the molecular weight of COH_4 (32.04 g COH_4 / mol COH_4), and finally by a term that represents a stoichiometric relationship (1 mol O_2 / 2 mol COH_4). The substances and units in the numerators and denominators of the terms cancel each other out, leading to a non-ratio as the final answer. There are other ways of solving the problem. Since the solution involves a series of

[2] There is no subscript notation in the stoichiometry tutor, as one of the reviewers remarked. Chemistry teachers who have worked with us advise that the notation used in this tutor is standard in other software tools and does not distract students. We are working to upgrade CTAT's facilities to handle a wider range of mathematical notation, which would be useful across a range of tutors.

multiplicative terms, the order of the terms does not strictly matter. Thus it is perfectly valid, for example, to swap the information entered as the third and fourth terms:

$$
\begin{aligned}
1.216 \text{ kg } COH_4 &\times (1000 \text{ g } COH_4 / 1 \text{ kg } COH_4) \\
&\times (1 \text{ mol } O_2 / 2 \text{ mol } COH_4) \\
&\times (1 \text{ mol } COH_4 / 32.04 \text{ g } COH_4) \\
= 18.98 \text{ mol } O_2
\end{aligned}
$$

Chemistry instructors, however, recommend the first strategy – to apply terms in an order that allows cancellation of parts (e.g., units and substance) of *prior* terms. For instance, the unit conversion (1000 g COH_4 / 1 kg COH_4, i.e., the second term in each of the equations shown above) is the best initial multiplier, since it allows the cancellation of parts of the first term (i.e., the given value), as shown by the "strike outs" in the interface of Figure 3.

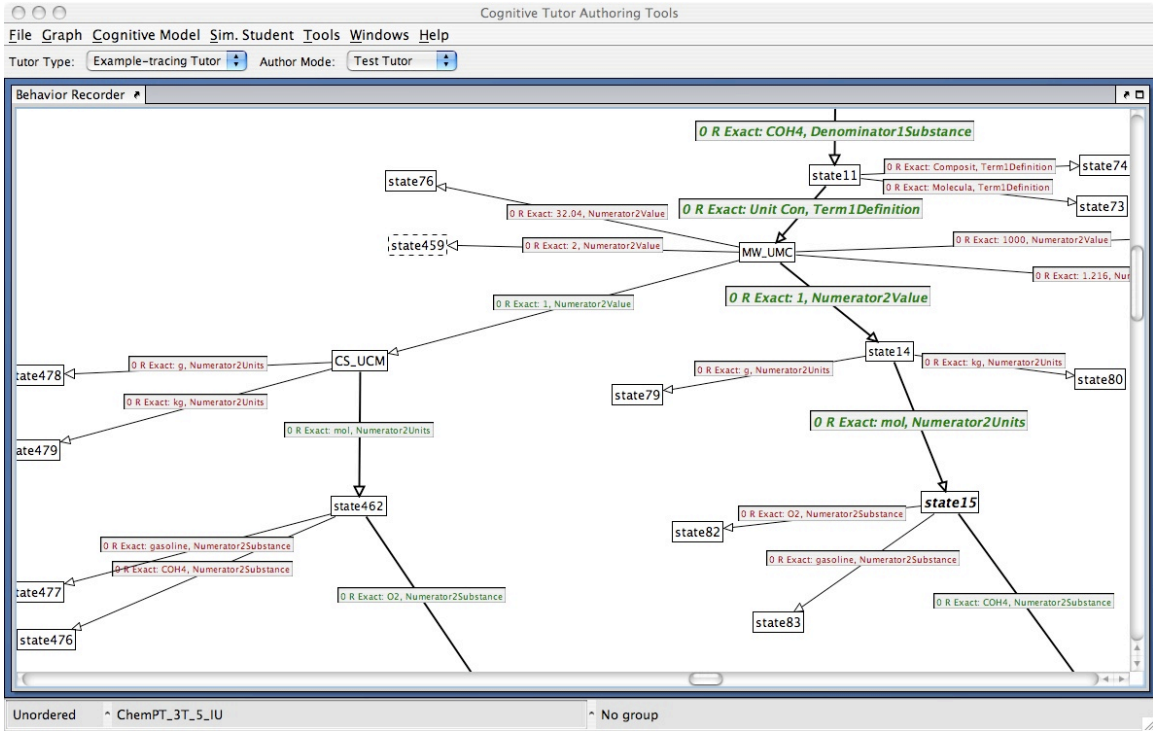

Figure 4: Part of an example behavior graph for the stoichiometry tutor

The instructor-recommended solution and the alternative solution are both represented in the behavior graph for this problem (see Figures 4 and 5 – the links representing correct actions are indicated with green text on the label, but this is not visible in the print version of this article). The instructor-recommended path is shown on the right. The boldface arrows indicate that the author has designated this path as representing the *preferred* way of solving this problem. The alternative solution path described above is encoded in the behavior graph as the path on the left, branching off of the preferred path in node MW_UMC. (CTAT authors can name the states in behavior graph. Even if only

a few well-chosen landmark states have meaningful names, it is considerably easier to navigate larger graphs.)

Let us consider in more detail how the example tracer uses the graph to monitor student problem solving and provide feedback. In the terminology of VanLehn (2006), the example tracer serves as *step analyzer*. It classifies each student action in the tutor's interface as correct, incorrect or suboptimal by flexibly comparing it against the actions stored in the graph. It also keeps track of the student's problem state by recording which links in the graph the student has "visited." Specifically, it keeps track of the *viable paths* through the graph, meaning start-to-finish paths that are consistent with the student's observed behavior thus far[3]. In order for a student action to be accepted as correct, it must correspond to an as-yet unvisited link on a viable path, and that link must represent correct behavior. (We use the term *viable link* to refer to such links.) Thus, once on a viable path, the student must stay on that path. If the student has visited a complete path through the graph, she has completed the problem.

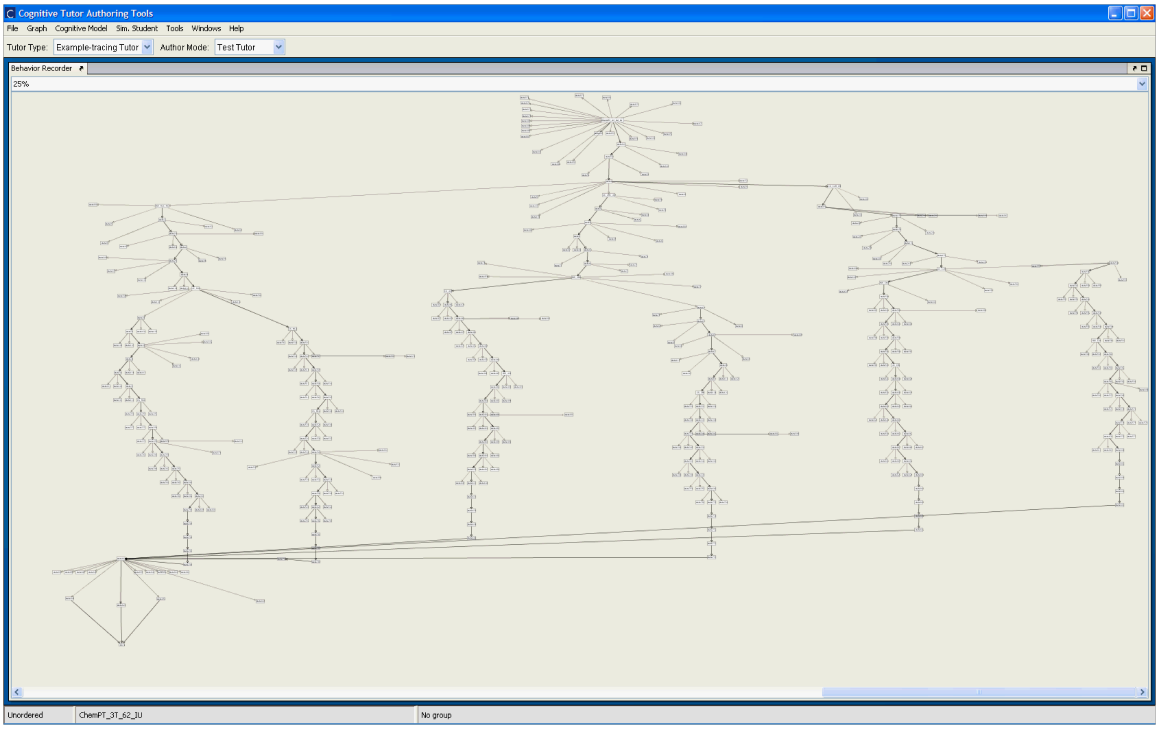

Figure 5: Complete stoichiometry behavior graph (zoomed-out view); this behavior graph is at the upper end of complexity of what has been achieved with CTAT

As one measure of the flexibility of the example-tracing process, the example tracer is capable (by means of the algorithm described above) of following a student through a problem even if the graph contains multiple paths, regardless of which of them the student actually takes. For example, suppose the student, working on the stoichiometry problem discussed above, has entered the second

[3] More specifically, a path is viable if all student actions that have been deemed correct so far correspond to links on this path. It is viable in the sense that it is a viable *interpretation* of the student behavior so far.

term, and is about to enter the third term. That is, the student is at the point in the problem where the expert and non-expert solutions paths diverge (represented by state MW_UMC in the graph, shown in Figure 4). At this point, both the start-to-finish path through the left branch, as well as that through the right branch, are viable paths, since all student actions that have been accepted as correct occur on both. Assume that the student enters the third term of the expert solution (1 mol COH_4 / 32.04 g COH_4). The six required actions all appear on the right branch (the expert branch), but not all are included on the left branch. Thus, after the tutor accepts these actions, the right branch is a viable path, but the left branch has been ruled out as a viable path. If the student were to follow up with actions that occur on the left branch but not on the right branch (such as, say, carelessly entering the same information again, but now as the fourth term), the tutor will reject these actions. Conversely, if the student (from the same state MW_UMC as mentioned above) goes down the non-preferred left path, the tutor will accept his/her actions as correct, but will not allow him/her to revert to the path on the right. In essence, once a path has become non-viable, it stays that way.

In addition to being able to follow students with respect to multiple paths, example-tracing tutors are capable of following the student in the face of ambiguity as to how a student action should be interpreted. This ambiguity occurs when a student action matches multiple viable links in the graph. When this happens, the example tracer will entertain multiple possible *interpretations* of the students' problem-solving behavior for as long as they are consistent with that student's problem-solving actions. This capability enables it to follow along smoothly with what the student is doing. For instance, this kind of ambiguity occurs at the point in problem solving illustrated in Figure 3, where the student has completed the second term, and has entered "1" and "mol" as part of the third term. The student could either be multiplying by (a) the molecular weight (with "1 mol COH_4" placed in the numerator), corresponding to the expert solution path (right branch), or (b) a stoichiometric relationship (with "1 mol O_2" placed in the numerator), corresponding to the non-preferred solution (left branch). The CTAT example tracer maintains *both* interpretations as possibilities at this point. It is therefore in a position to accept as correct student behavior, either "COH_4" or "O_2" as the substance in the numerator, regardless of which one the student decides to enter. By contrast, if the example tracer had committed to one interpretation or the other, at the moment that the ambiguous student input occurred, it would not have been able to accept, with equal smoothness, all valid subsequent student input. The ability to entertain multiple interpretations of student behavior in parallel sets it apart from Cognitive Tutors, which (although they can follow students along multiple strategies) do not at any point in time entertain multiple interpretations in parallel. Rather, they disambiguate on the spot (see e.g., the example in Anderson, 1993).

As a second mechanism that enhances the flexibility with which student behavior is matched against a behavior graph, an author can define ordering constraints (on the steps in the graph) that the tutor should enforce as a student solves a problem. In our experience, it is too restrictive to require that students enter the steps in the exact same order that they have been recorded in the behavior graph. Many problems have (clusters of) steps whose order does not matter, so that enforcing an order could confuse students by rejecting actions that are perfectly reasonable just because they do not occur in an somewhat arbitrary order selected by the author. Thus, CTAT allows an author to specify that all steps in the behavior graph can be done in any order. The behavior graph of Figure 4 is unordered; meaning, for instance, that the student can provide the value, unit, and substance of the numerator of the third term (the one currently being worked on in Figure 3) in any order. Alternatively, as described in more detail below, an author can specify more fine-grained ordering constraints, by defining (arbitrarily-nested) groups of actions that should be done in order, or that can be done in any order.

As a third key mechanism for increasing the flexibility of example tracing, links in the graph can be *generalized* so that they match a variety of student input values rather than a single value. An author may specify that a link represents a set of values, a numeric range, or a regular expression. Further, an author may write a formula (akin to formulas in Excel) that specifies a functional relation between problem-solving steps.

In addition to providing "generic" correctness feedback (i.e., "yes/no" feedback), example-tracing tutors can provide specific feedback on common student errors. A behavior graph may contain links that represent incorrect actions (marked with red font on the label, although the color may not be visible in Figure 4). For instance, the links that lead to nodes with no children at the top of Figure 4 (e.g., states 76 and 459) represent incorrect actions. When a student action matches no viable link in a graph, but does match an incorrect action link emanating from a viable path, an error message associated with that incorrect action link is shown to the student. If a student input value does not match any viable link in the graph, or any suboptimal or incorrect action link attached to a viable path, the tutor provides unspecific feedback indicating only that the student action is incorrect.

In addition to using their behavior graphs to generate feedback, example-tracing tutors use the behavior graph to generate hints as to what the student might do next. Essentially, CTAT's *step generator,* in the terminology of VanLehn (2006), works by finding an appropriate unvisited link in the behavior graph, and then displaying a hint message associated with that link. (Typically, all links in the graph have hint messages attached to them.) A key question is what hint to give (e.g., Van Lehn, 2006, pp. 242-243). When there are multiple interpretations of the student behavior, one is designated as the reportable interpretation – the one that most closely matches the preferred solution path through the problem. The first unvisited link in this viable path is chosen and the hints associated with this link are displayed to the student. Thus, in our stoichiometry example, if the student is in the (ambiguous) situation shown in Figure 3, the tutor's hints will focus on the expert solution (i.e., the right branch). If the student ignores the hint and follows the alternative strategy (i.e., the left branch), the right branch is no longer viable and the next hint will focus on the left branch.

In sum, example-tracing tutors exhibit a desirable property of the inner loop not identified by VanLehn (2006), although perhaps assumed: they *flexibly* recognize student behavior within a given problem. Specifically, they are able to deal with multiple strategies for solving a given problem, and are able to maintain multiple interpretations of student behavior. In our opinion, this level of flexibility is highly desirable for an ITS authoring tool, as many ITS applications require it.

CREATING EXAMPLE-TRACING TUTORS WITH CTAT

In this section, we describe how an author uses CTAT's main tools for creating example-tracing tutors. (As mentioned, CTAT also provides tools for building Cognitive Tutors, including tools for building, testing, and debugging a cognitive model. Those tools however are outside the scope of the current paper.) The use of CTAT tools is part of a more-encompassing process aimed at tutor creation, outlined in Table 2. In the current paper, we focus on the steps in which CTAT is used, primarily steps 4-6. The other steps are equally important to building effective tutors (see e.g., Baker, Corbett, & Koedinger, 2007) but they are outside the scope of this paper.

Table 2: Developing an example-tracing tutor with CTAT

1. Definition of instructional objectives
2. Identify problem categories and problems for which to provide tutoring
3. Cognitive Task Analysis (e.g., think-alouds and difficulty factors analysis)
4. Tutor design and development
    a. Design and create interface
    b. For each problem (category)
        i. Demonstrate correct and incorrect behavior (i.e., create a behavior graph)
        ii. Generalize and annotate the behavior graph
        iii. (Optional) use template-based Mass Production to create multiple problems with isomorphic behavior graphs
    c. Organize curriculum and create curriculum files
5. Deploy prototype version
6. Pilot test
7. Iterate
8. Deliver final version

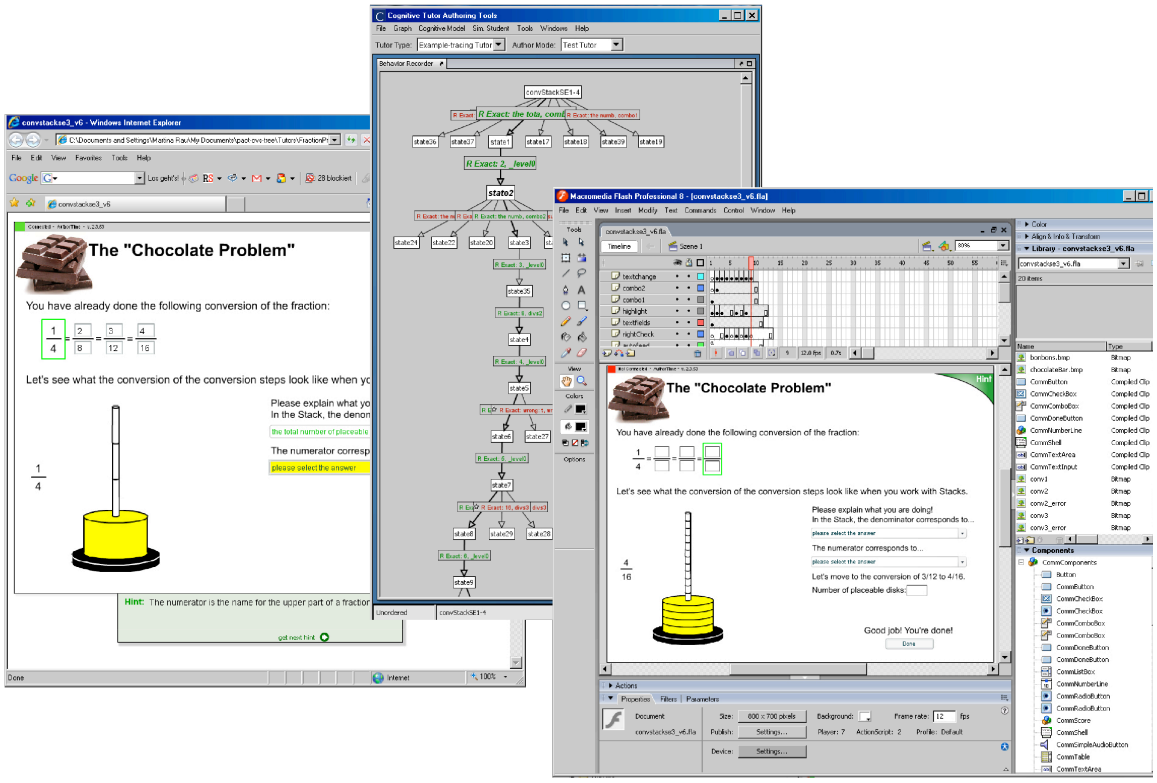

Figure 6. Creating a Flash-based example-tracing tutor with CTAT. An author creates an interface through drag-and-drop techniques in the Flash Interactive Development Environment (IDE) (shown on the right), runs the interface (shown on the left, in a browser), and uses CTAT's Behavior Recorder (middle) to create, generalize, and annotate a behavior graph. The tutor in this example is for 6th-grade fractions.

Creating the user interface for the tutor

Interface design is an important issue in tutor development, but here we focus on implementation. When using CTAT, an author uses an interface builder, in drag-and-drop fashion, to create a tutor interface that lays out some or all of the problem-solving steps that the student will go through (step 4.a in Table 2). The author has the choice between creating a Java-based interface and a Flash-based interface. Both can be created using an off-the-shelf Interactive Development Environment (IDE), either for Java development (e.g., Netbeans) or for Flash. For instance, the stoichiometry and fractions tutors, discussed and shown above, were developed using Flash (see right side of Figure 6).

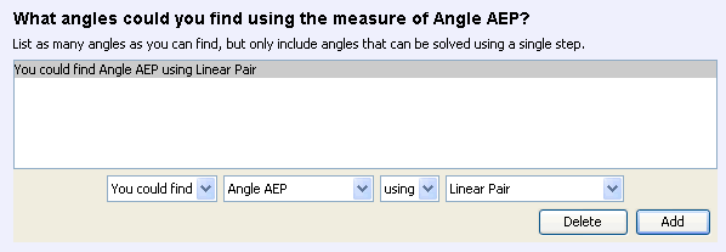

Figure 7: Composer widget (in Java) for building sentences from a sequence of menus

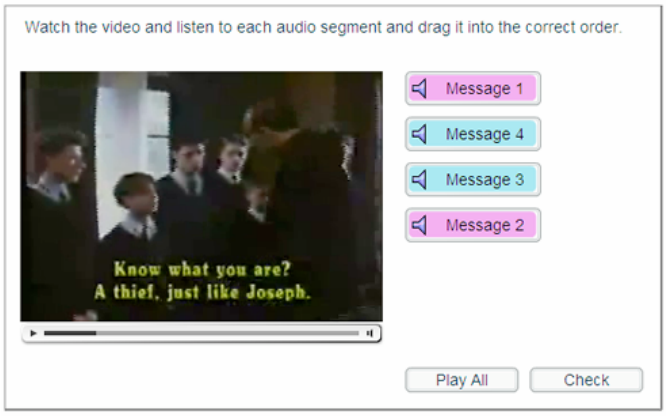

Figure 8: Multiple selection widget (in Flash) with a video prompt and audio responses

For each of these environments, Java and Flash, we have created a set of interface widgets that are "CTAT-enabled" in the sense that they communicate with the rest of the tools. The widget sets contain tutor-enabled versions of standard widgets such as buttons, combo boxes, text fields, text areas, labels, and lists. In addition, both sets contain composite widgets that we have created, such as a table widget, and (in the Java set only), a "Composer" used to build sentences from a sequence of menus (see Figure 7). Various widgets support often-used combinations of interface elements, such as a question followed by a menu to select the answer, or a question followed by multiple possible answers. The set of tutor-enabled Flash widgets (or "components" in the Flash terminology) contains a number of media-enabled components (see Figure 8). Finally, as an example of a domain-specific widget, an interactive number line was created for the fractions tutor illustrated in Figures 2 and 11.

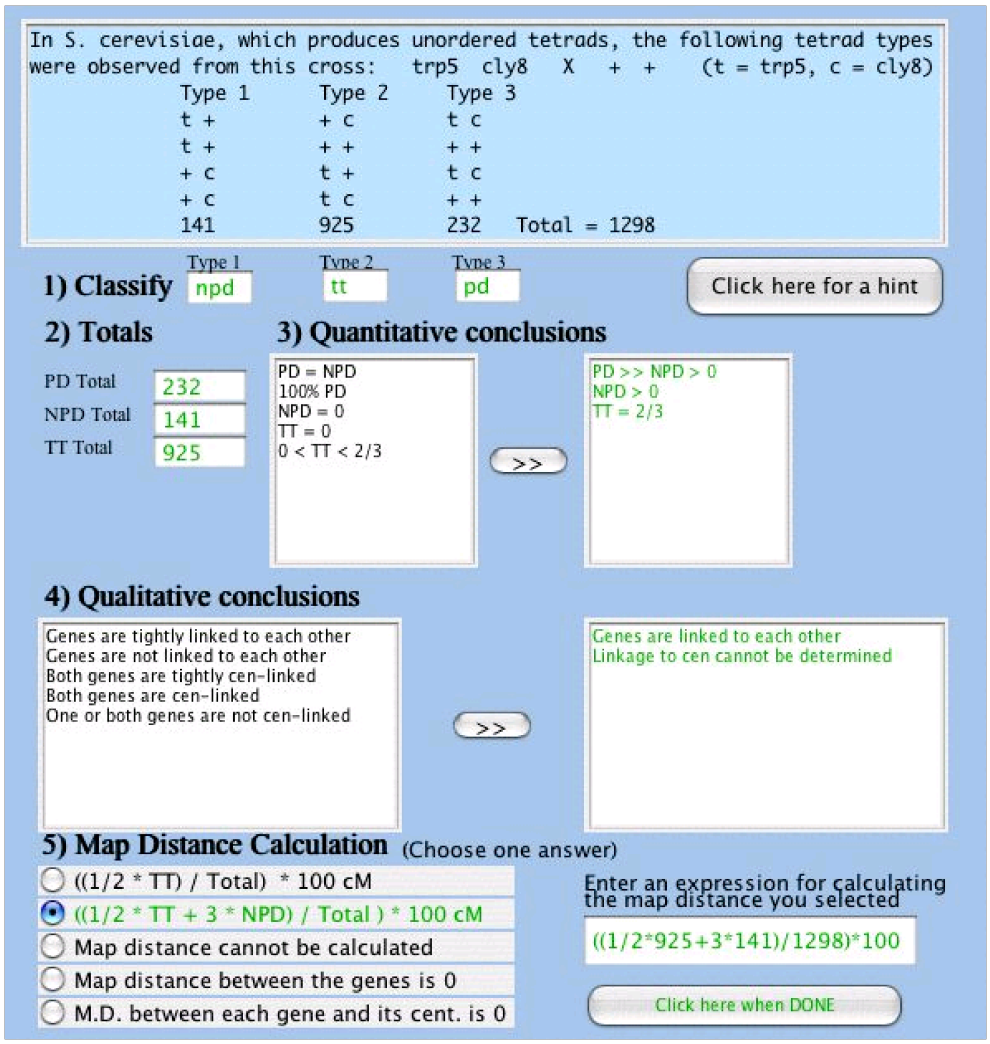

Figure 9: Tutor for genetics (with Java interface) built with CTAT.

Using these widgets, an author can construct interfaces that support complex reasoning by students, such as the tutor for introductory (college- and high-school level) genetics shown in Figure 9[4] and the stoichiometry tutor shown above. These tutor interfaces are made up entirely of standard widgets (with a small amount of custom programming in the case of the stoichiometry tutor, to show strike-out of terms in a stoichometry equation that cancel each other out). One of the media widgets was used in a tutor for intercultural competence (Ogan, Aleven, & Jones, 2008; see Figure 10). To the extent that the pre-defined widget sets are sufficient, tutor interfaces can be built entirely without

[4] The Genetics Problem Solving Tutor was implemented with CTAT by Albert Corbett and colleagues. It covers a wide range of genetics topics, and has been deployed and evaluated at 14 colleges and universities around the country. It is primarily a Cognitive Tutor, although for one of the units in the tutor curriculum, example-tracing tutors have been created.

programming, using the drag-and-drop GUI builders built into standard Java or Flash development environments. Programming is necessary only for tutor applications that require new widgets.

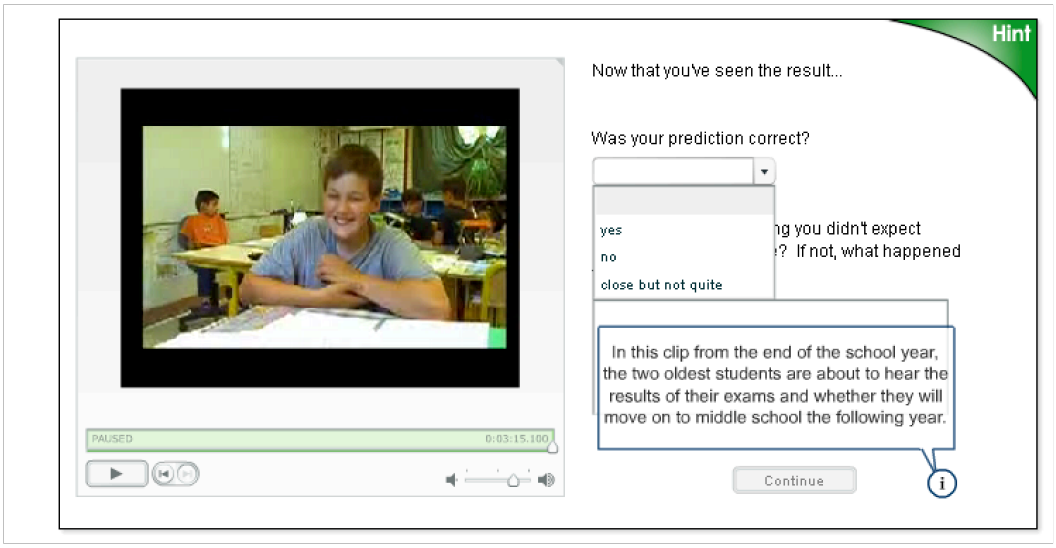

Figure 10: A tutor for French culture makes use of CTAT's video widget

Dynamic interfaces

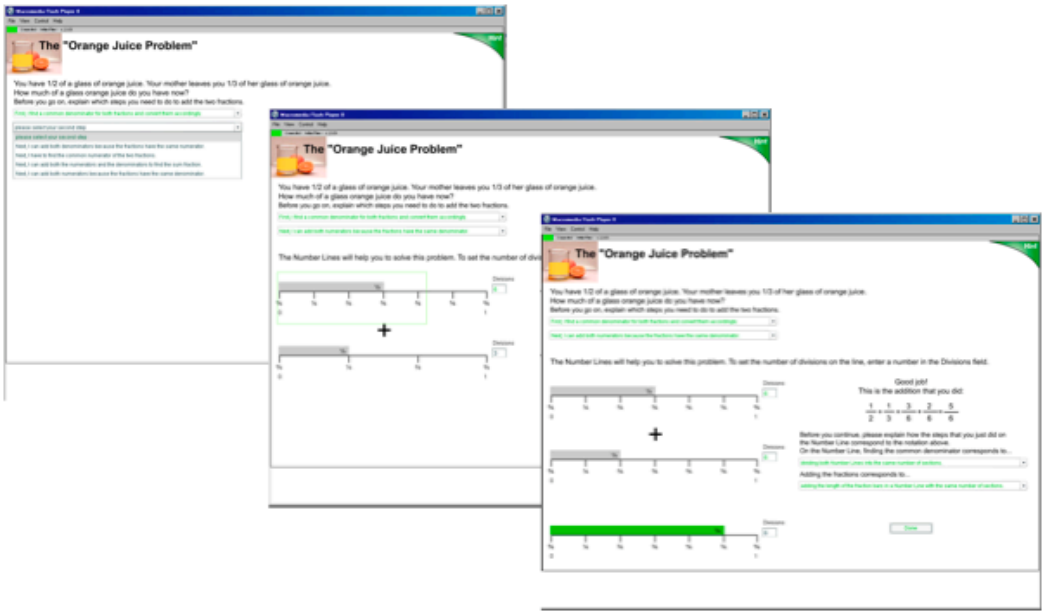

Figure 11: Sequence of screens in a tutor for fractions; this kind of dynamic interface can be authored without programming

CTAT also offers a capability to create *dynamic tutor interfaces,* meaning that the interface changes in response to student actions, for example by rearranging the screen layout, adding or subtracting widgets, or changing the content of existing widgets (see Figure 11). Using CTAT, dynamic interfaces can be created without programming. They serve many purposes, and CTAT users have requested this feature since day one. One purpose is to make effective use of limited screen real estate, which is important especially for tutors that run in a browser, and for tutors that (e.g., in less-affluent school districts) run on computers with low-resolution screens. The facility can also be used to provide dynamic scaffolding: if a student is struggling, the tutor may add steps in the interface to make thinking visible at a finer-grained level (e.g., Razzaq & Heffernan, 2006). As another example, the steps in the tutor's user interface could be revealed one-by-one, in sequence, as the student solves the problem, as a way of leading the student through these steps in order (as illustrated in Figure 11). Alternatively, the dynamic interface facility could be used to implement dynamically-linked representations (Moore, 1992; Reed, 2005): in tutors that display multiple representations of a problem. Also, the facility can be used for the tutor to communicate certain consequences of their choice to the student. Finally, as a relatively simple use of this facility, the system may take over some of the steps that are necessary to solve a problem but that do not address learning objectives (e.g., Brown, 1985). In order to create dynamic interfaces, an author can insert links in a behavior graph that represent "tutor-performed actions." Among the possible tutor-performed actions are showing and hiding of widgets, and changing the content of a widget.

Creating behavior graphs

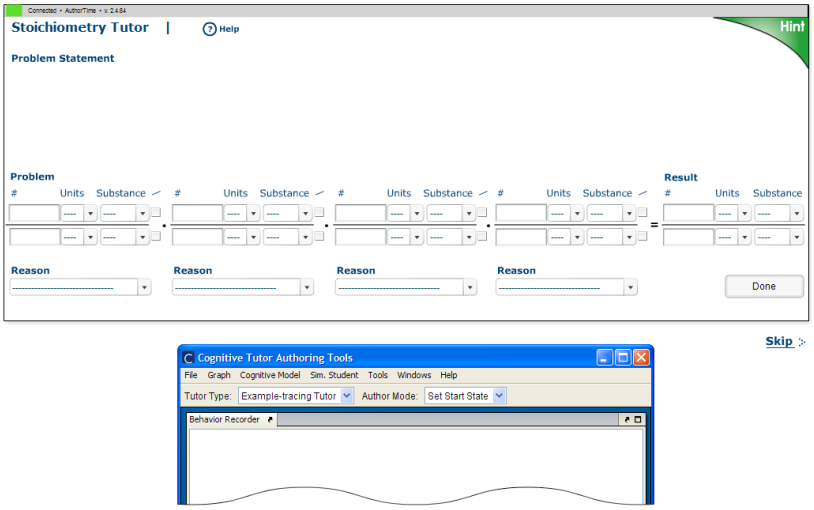

Figure 12: Empty interface and behavior graph, prior to entering a "start state."

An author must create behavior graphs for all problems-to-be-tutored, although a template-based "Mass Production" process (described below) substantially reduces the amount of problem-specific authoring needed to create multiple problems with the same behavior graph structure. CTAT's Behavior Recorder can be used for recording, editing, generalizing, and annotating behavior graphs.

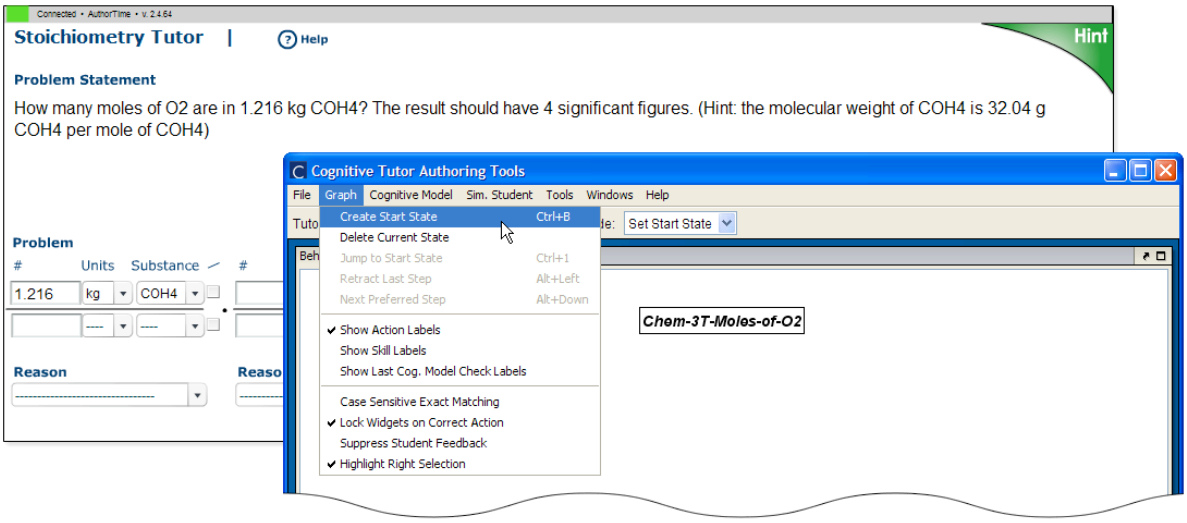

Figure 13: Interface after entering the start state information

The process of creating a behavior graph starts with defining a "start state" for the given problem. The author enters problem-specific information into the "empty" interface (see Figure 12), typically, a problem statement, values in some of the widgets, problem-specific graphical elements, etc. For instance, in Figure 13, the author of the stoichiometry problem discussed previously has typed in the problem statement (i.e., "How many moles of O_2 …") and the given initial value (1.216), and has selected the relevant units (kg) and substance (COH_4) from their respective pull-down menus. Next, the author selects "Create Start State" in the Behavior Recorder (Figure 13). CTAT prompts the author to enter a name for the problem, and the initial, solitary node of the behavior graph appears in the Behavior Recorder, i.e., the "start state."

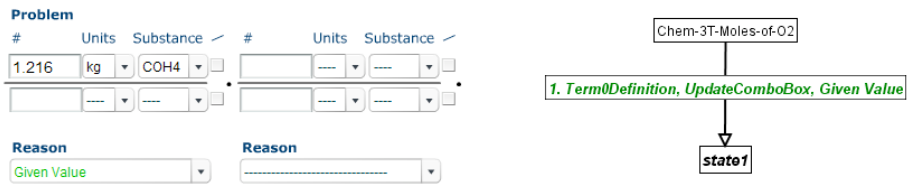

Figure 14: Interface and behavior graph after demonstrating the first step of the expert solution

Having defined the problem's start state, the author creates a behavior graph, simply by putting the Behavior Recorder in *Demonstrate mode* and demonstrating expected problem-solving behavior in the interface. The author's problem-solving steps are recorded in a behavior graph, one link per step, where a step is typically a single entry in an interface widget. So, for instance, in Figure 14, the author has just selected "Given Value" from the pull-down menu in the lower left of the stoichiometry user interface as the reason for the initial value, and this step has been captured by the Behavior Recorder as the first link of the behavior graph. A new node is added at the end of the link to represent the new

problem-solving state. The label on the link shows details about the specific action it represents. A green label (not visible in the print version) indicates that the link represents correct behavior.

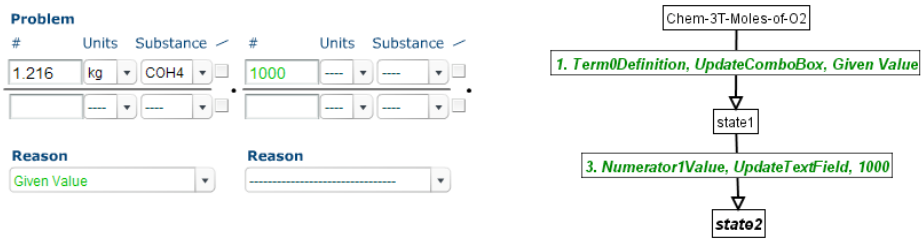

Figure 15: After demonstrating the second step of the expert solution

As the author demonstrates the second step of the solution (e.g., entering "1000" as the numerator value of the second term), that step is recorded as well (Figure 15), and so on. For instance, in Figure 16, the author has demonstrated part of the expert solution to the stoichiometry problem – up to and including the second term. Eventually, a complete solution path, from the start state to a final state, is captured in the Behavior Recorder.

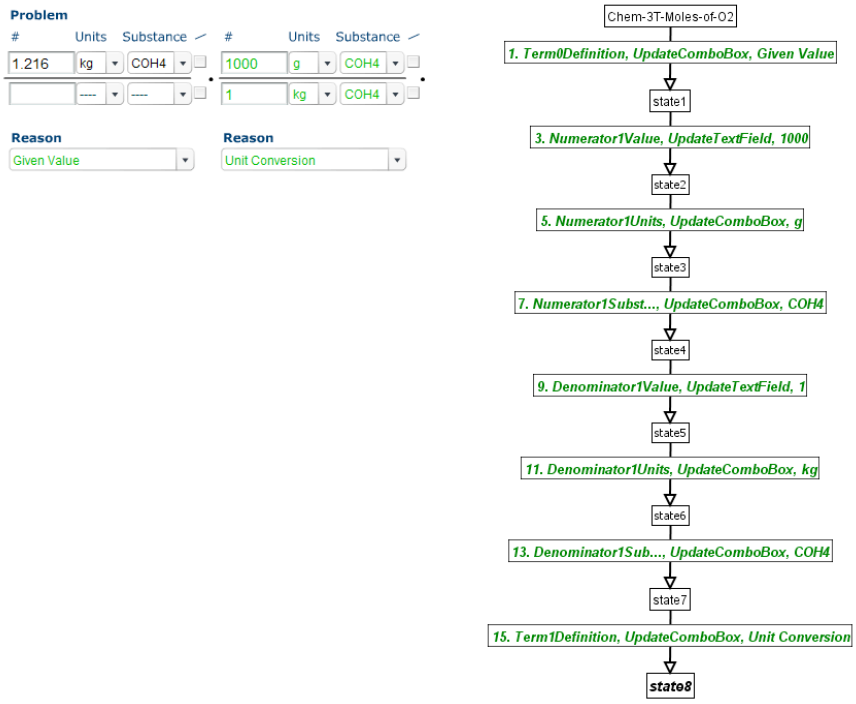

Figure 16: After demonstrating steps through the completion of the second term

When creating a behavior graph for a given problem, it is important to record all reasonable (or pedagogically desirable) solution paths that students may take. Otherwise, there is a risk that the tutor

will reject valid student input. As discussed further below, however, an author needs *only* record *true* alternative solution paths, not paths that differ only in minor ways. An author can create alternative paths in the graph by backing up to a problem-solving state that has been recorded previously – clicking on the state in the graph causes the interface to revert to that state – and then demonstrating a different way of solving the problem. The new solution steps will be recorded in a new branch in the behavior graph. For example, as discussed earlier, in our stoichiometry problem, a student can enter the terms of the stoichiometric equation in any order. After the initial step of entering "Given Value" as the explanation for the first term, therefore, valid next steps are (in addition to the already-recorded expert solution) to provide either the molecular weight of COH_4 (1 mol COH_4 / 32.04 g COH_4) or the relevant stoichiometric relationship (1 mol O_2 / 2 mol COH_4). To record these strategies as separate paths, the author may revert back to the state following the input of "Given Value" and then enter the alternative steps, which will be recorded in a new branch off the selected state (Figure 17).

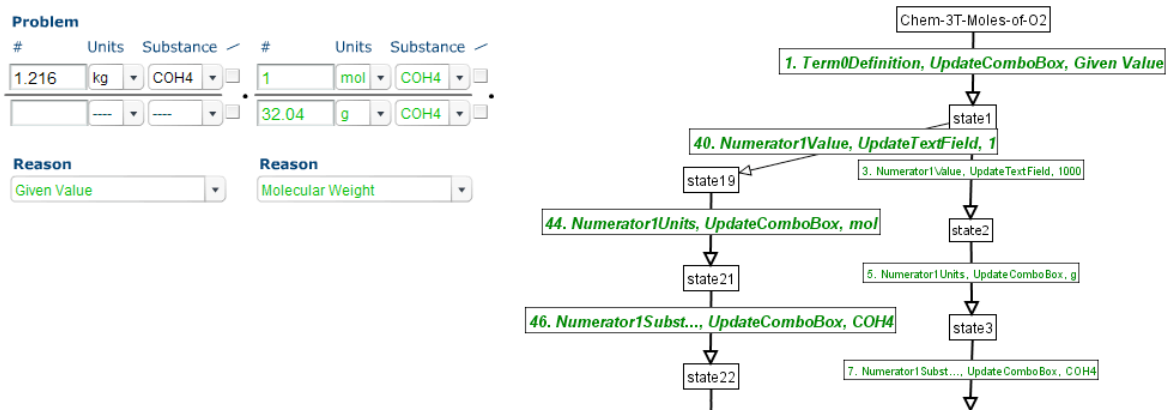

Figure 17: Demonstrating the first steps of an alternative, correct solution

In order for the tutor to provide specific feedback messages related to common errors that students are expected to make, an author can demonstrate incorrect steps. The author must mark the corresponding links in the behavior graph as representing incorrect behavior, and attach a feedback message that will be displayed when a student commits that particular error. For instance, in Figure 18, the author has just demonstrated an incorrect action by entering "32.04" in the numerator of the second term from the left. Such an error might easily be made by many students, since "32.04" is a relevant value to the problem (i.e., the molecular weight of COH_4), but it does not appear in the numerator of any term for any of the possible correct solutions. The author marks the link that results from demonstrating this erroneous step as incorrect behavior by selecting "Incorrect Action (Bug)" from the pull-down menu connected to that link (see Figure 18). She is then prompted to provide a specific error message (not shown).

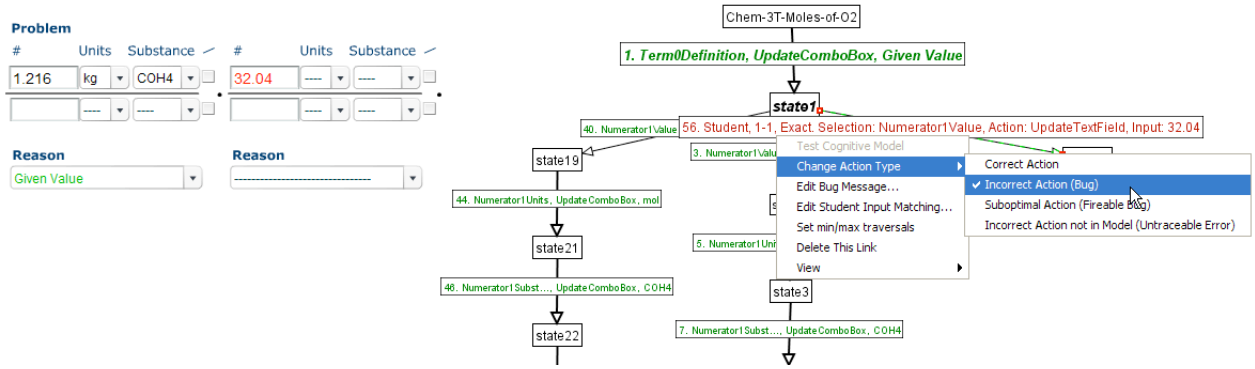

Figure 18: Demonstrating an incorrect step and marking it as an "incorrect action" in the behavior recorder.

Annotating a behavior graph

After a behavior graph has been created, the author's next step is to annotate the links in the graph with hints and knowledge component labels. It is in this step that the roots of example-tracing tutors in the Cognitive Tutor methodology are most clearly visible. In particular, example-tracing tutors share the assumption that a complex cognitive skill can be decomposed into small "knowledge components" that can be acquired and strengthened separately through practice (see Anderson & Lebière, 1998). Thus, the learning objectives targeted by these tutoring systems are (often) defined in terms of knowledge components. In Cognitive Tutors, the knowledge components are expressed as production rules. In example-tracing tutors, by contrast, the knowledge components are defined extensionally, by labeling steps in a behavior graph with the name(s) of the knowledge component(s) that they involve. An author can enter the knowledge component label in a dialog box that is accessible through the pull-down menu connected to the link (see Figure 19). The knowledge components are displayed in the behavior graph in separate link labels. In our stoichiometry task, there are (hypothesized to be) different knowledge components for applying a molecular weight term versus a unit conversion term versus a stoichiometric relation term. The knowledge components are subdivided further by the 6 different elements of each term (value, unit, and substance of the numerator and denominator), and by whether the step is an explanation step. Thus, as illustrated in Figure 19, the stoichiometry problem described above involves knowledge components for explaining unit conversion steps ("set-unit-conversion-as-reason") and for entering the numerator value of a molecular weight term ("set-numerator-value-of-molecular-weight").

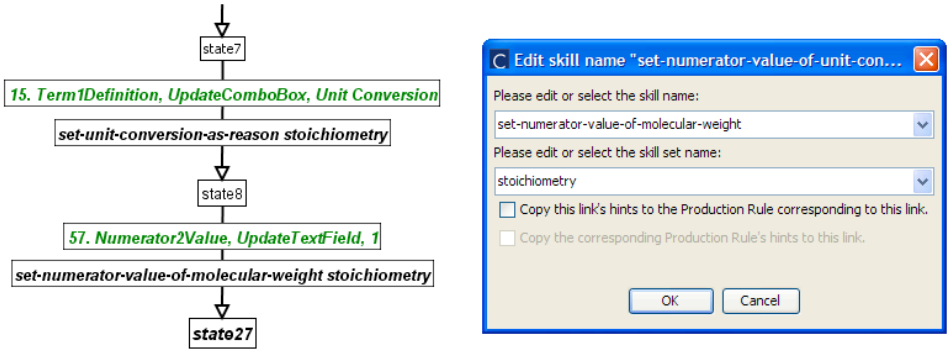

Figure 19: Labeling a step in the behavior graph with a knowledge component; the second (lower) label on each link indicates the knowledge component

Understanding problem solving in terms of underlying knowledge components is often an important goal of cognitive task analysis, especially as it is done in the service of tutor development. By labeling multiple steps in a behavior graph (or across behavior graphs) with the same knowledge component, an author makes a prediction that practice on the one step will lead to improved performance on the other (see Koedinger *et al.,* 2004 for a detailed example). In essence, she defines the applicability conditions of the knowledge component in an extensional manner. The knowledge component labeling helps in designing a curriculum and problem set for the tutor. It helps an author keep track of whether the problem set she has created so far provides sufficient practice opportunities with each of the targeted knowledge components. CTAT automatically generates a *skill matrix* to support this analysis; this report shows which problems involve which knowledge components. Also, the knowledge component labels enable the tutor to assess student knowledge in its inner loop. Currently, CTAT does not take advantage of this ability to support individualized problem selection, but we will soon add the Bayesian knowledge-tracing algorithm created by Corbett and Anderson (1995), so that CTAT can track individual students' knowledge growth, and select problems accordingly. However, CTAT *does* record the results of its inner-loop knowledge assessment in the logs of student-tutor activities. The primary purpose of these logs is to support research.

In addition to knowledge component labels, an author attaches hint sequences to each link in the behavior graph – these hints will be displayed when the student requests help from the tutor on the given step (see Figure 20). Typically, multiple levels of principle-based help are provided, going from broader hints (e.g., what goal or step to work on, what problem-solving principle to apply), to more specific advice (e.g., how does the problem-solving principle apply, what calculations need to be done), and finally, to a *bottom-out hint* that provides the resulting answer, or something close to it. The hints should be consistent with the knowledge component labels: steps that involve the same knowledge component should have similar hint sequences.

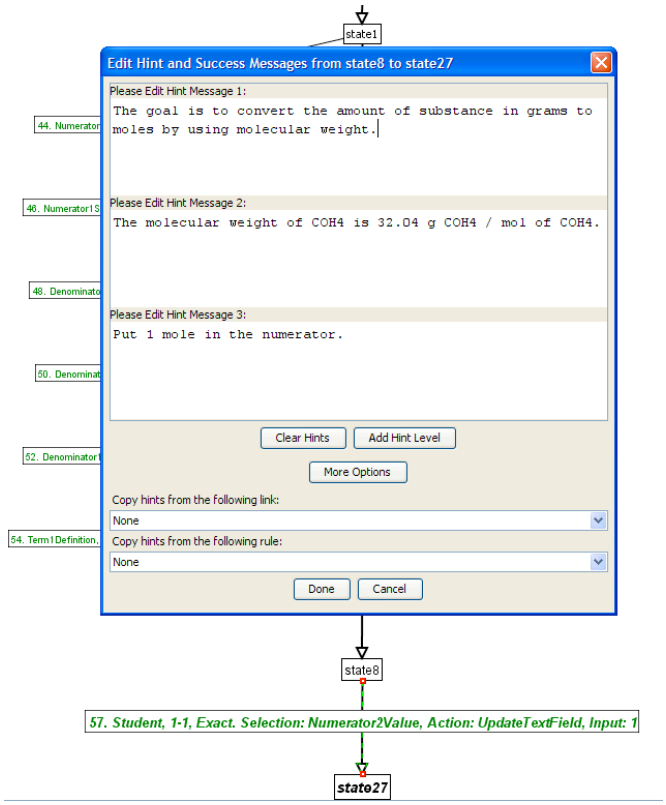

Figure 20: defining a hint sequence

Generalizing behavior graphs

As mentioned, an author can generalize a given behavior graph in a number of ways to extend the range of student behavior that the example tracer will recognize as matching the graph, beyond recognizing only the exact same steps in the exact same order. As mentioned, this flexibility has been key in making example-tracing tutors into a viable ITS paradigm. There are three mechanisms, illustrated below:

- Specifying constraints on the ordering of steps: the author can define arbitrarily nested groups of steps that should be "ordered" or "unordered."
- Defining a range of student input that matches a particular link; for example, an author can specify a numeric range, a set of values, or a regular expression.
- Specifying how one step depends on other steps, through CTAT's formula mechanism, by attaching Excel-like formulas to the links in a behavior graph

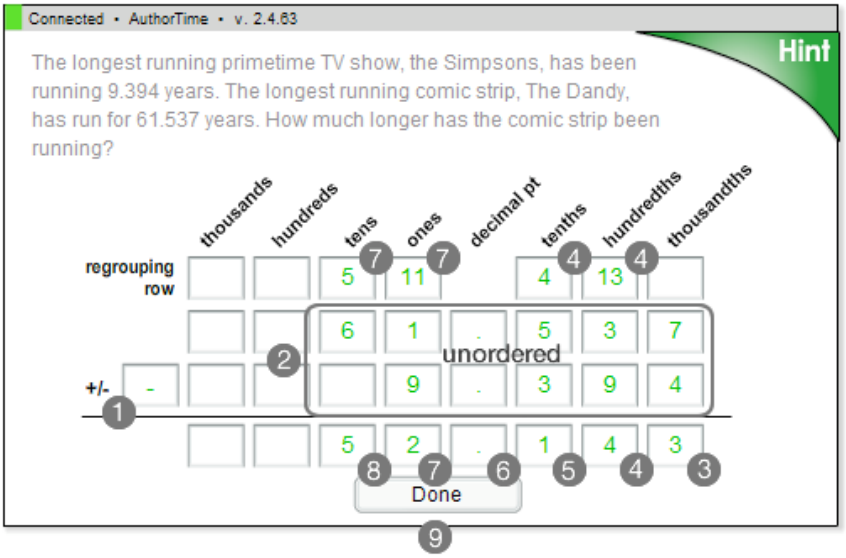

Figure 21: Desired step order in a tutor for decimal addition and subtraction; steps with the same number should be viewed as an "unordered group"

In many situations, the specific order of (some or all of) the steps needed to solve a problem does not matter. In such situations, a tutor should allow students the freedom to perform the steps in any order they may choose. On the other hand, where order *does* matter, the tutor should help students learn the right order, meaning that it should reject steps when they are out of order, even if they are otherwise correct. CTAT allows an author fine-grained control over the order in which problem steps will be accepted, using its Group Editor tool (see Figure 22, bottom left). An author can group links in a behavior graph and specify for each group whether it is "ordered" or "unordered." Groups can be nested within groups. Consider, for example, a tutor unit dealing with decimal addition and subtraction (see Figure 21; this tutor is part of a project, led by the authors of this paper, to create a comprehensive website with tutors for middle-school math). In this tutor, students, after reading the problem statement, must select the operation to be performed ("+" or "–"), copy the two numbers presented in the problem statement into the interface, making sure that the decimal points are aligned, and then add or subtract to find the final answer. There is probably no good reason to require that students enter the digits of the two numbers in any particular order, as long as they line up the decimal points correctly. On the other hand, it seems pedagogically desirable that they follow a systematic addition or subtraction strategy, processing columns in right-to-left manner. With respect to borrowing (in the case of subtraction), a small amount of flexibility is desirable: it does not matter whether the "borrower column" or the "lender column" is updated first (in the "regrouping row," see Figure 21), or whether the result in a column is written before or after the borrowing has been completed.

In order to capture the desired tutoring behavior, an author would specify, after recording all the steps in a behavior graph, that overall, the graph should be viewed as ordered, and would then create three unordered groups of links: one for the steps involved in entering the two given numbers (group 2 in Figure 21), and one each related to the borrowing that occurs in the hundredths and ones columns (groups 4 and 7 in Figure 21), respectively. As illustrated in Figure 22, an author can use CTAT's

Group Editor to create and modify link groups. The author has created two groups and named them "Enter-Given-Numbers and "Subtract-Hundredths-Column," both listed in the Group Editor, and is working on creating the third. (The "Top-Level" group is created by CTAT and represents the whole graph. Note that it is an ordered group.) The author has selected three links in the behavior graph, has clicked on "Create Group," and is about to provide a name for the group and specify whether it is ordered or unordered.

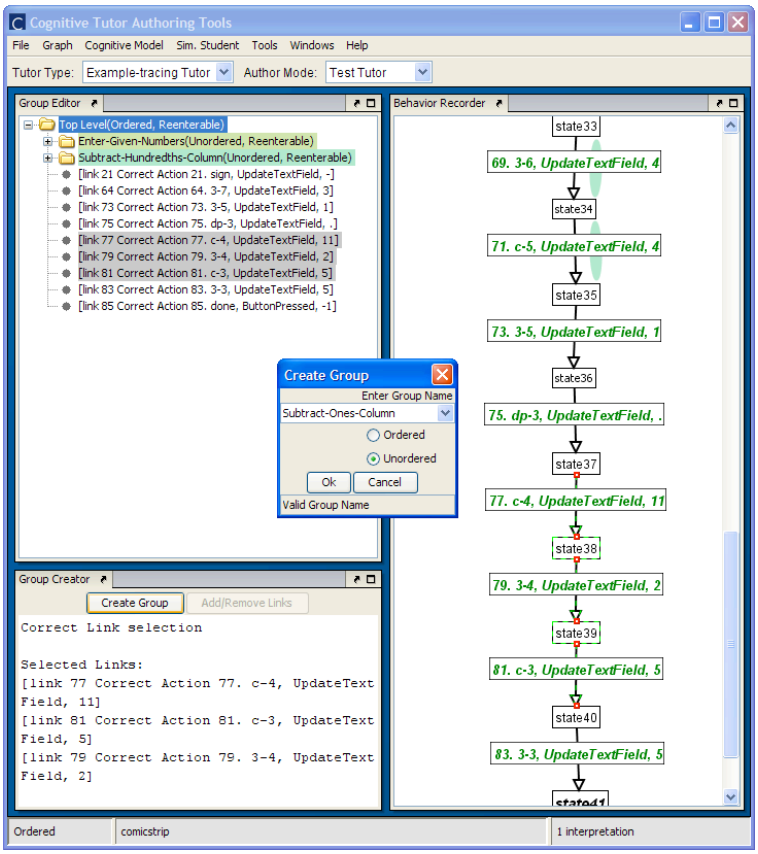

Figure 22: Defining an unordered group of links

Authors can also generalize a behavior graph by changing the matching criterion for a link, for example by attaching a formula, or specifying a range of valid values. We illustrate CTAT's formula facility with a demo tutor for factoring quadratics (see Figure 23). This tutor supports a guess-and-test strategy, reflecting the way that factoring is often done in real life – bypassing the quadratic formula[5]. In this tutor, the students themselves define the problems they want to solve with the tutor, by entering the coefficients a and b of a quadratic expression $x^2 + ax + b$. They must then factor the expression into the form $(x + N_1)(x + N_2)$. The tutor lets the student venture three guesses for the number pair N_1 and N_2. For each, the student must correctly enter the sum and product of N_1 and N_2. If the sum and

[5] The point is to illustrate what is possible with CTAT – we make no claim that the tutor for factoring quadratics is effective in real life.

product are equal to the coefficients a and b, the equation has been factored successfully. Otherwise, the tutor only accepts a click on the "Next Try" button, and adds input fields for the next attempt (through CTAT's dynamic interface facility). Thus, returning to a point discussed above, the tutor supports a specific form of delayed feedback: the correctness of the student's guesses for N_1 and N_2 is not revealed until later steps.

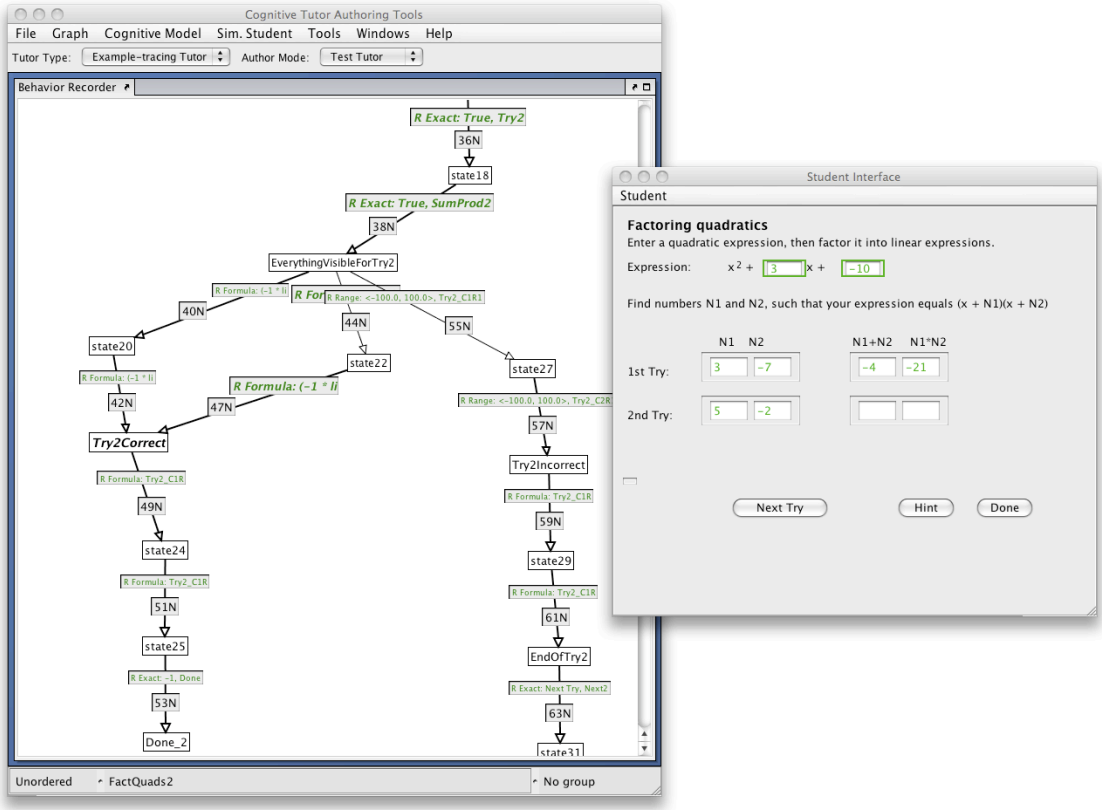

Figure 23: Demo tutor for factoring quadratics that supports a guess-and-test strategy, and part of its behavior graph. After the student's first attempt at factoring $x^2+3x-10$ as $(x+3)(x-7)$, which is incorrect, the tutor accepts a click on the "Next Try" button but will reject a click on the "Done" button. The student's second try, $(x+5)(x-2)$, is correct. After the student fills out the sum and product of 5 and -2, the tutor will accept a click on the "Done" button.

This tutor relies on CTAT's range mechanism in a number of places. Using this facility, an author can specify that a link in the graph can be matched by input within a predefined numeric range. Range matches are used on the steps where the student enters the coefficients a and b for the quadratic-expression-to-be-factored and the steps where students enter their guesses for N_1 and N_2.

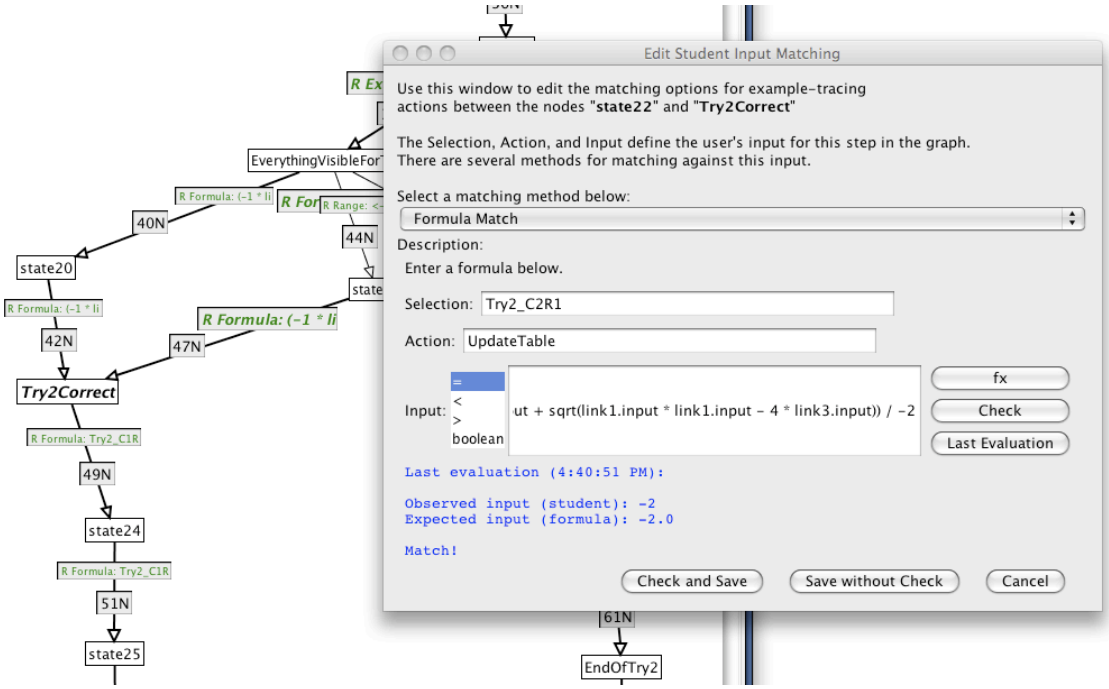

Figure 24: CTAT's formula facility

The tutor also relies on CTAT's formula mechanism, in two places. First, as one might expect, it is used on the steps where the student enters the sum and product of N_1 and N_2. Second, perhaps somewhat surprisingly, the formula mechanism is used on the steps where the student enters the guesses N_1 and N_2 to figure out (by means of the quadratic formula) whether these guesses are correct. In Figure 23, the left path branching off of state "EverythingVisibleForTry2" includes correct guesses for N_1 and N_2, followed by the sum and product of N_1 and N_2, and finally the Done button. (To accommodate the fact that N_1 and N_2 are interchangeable, this path bifurcates at the top, just below state "EverythingVisibleForTry2," and then converges again.) The path that branches off to the right of state "EverythingVisibleForTry2" includes *any* guesses for N_1 and N_2. It further includes the sum and product, followed by the TryNext button, but not the Done button. Thus, after the student has entered correct values for N_1 and N_2 (i.e., values that represent the correct factoring of the quadratic expression), both the left and right paths are viable interpretations. However, if the values for N_1 and N_2 are incorrect, only the right path is viable. Since only the left path ends in a Done button immediately after the sum and product, the student can finish only when she is currently on the left path.

Figure 24 shows the window in which an author enters formulas, displaying a formula expressing how the value in the N_2 field depends on other values. (This formula occurs on the left path in the graph; it is a straightforward application of the quadratic formula.) The formula refers to values accepted on other links (e.g., "link1.input") and also includes mathematical functions (e.g., "sqrt" for the square root). A formula wizard (not shown) lets an author browse the set of functions that can be included in formulas. They include the Java string and mathematics libraries and a (growing) CTAT-specific library. It is straightforward to write Java code to extend the function library.

Without the range and formula mechanisms, it would have been virtually impossible to build an example-tracing tutor that supports (1) students' entering their own expression-to-be-factored and (2) a guess-and-test strategy where the tutor can respond to a wide range of student input that cannot be anticipated in advance. Without the two mechanisms, the behavior graphs would need to contain an intractable number of paths. Thus, the example illustrates another way in which example-tracing tutors deserve to be called ITSs – they support *dynamic paths* where the steps within the path depend on each other, obviating the need to enumerate a massive number of paths.

Mass Production of behavior graphs

To facilitate the creation of many *isomorphic* tutored problems, CTAT provides a facility for template-based *Mass Production.* The idea behind Mass Production is to help an author generate many problems of the same structure without having to demonstrate solutions to each and every one of them. Rather, Mass Production lets an author take advantage of the (ample) reusable information contained in a behavior graph – all of the problem-solving states and links and possibly many of the hint messages and error feedback. The need for isomorphic problems arises in many task domains, given that ITSs tend to focus on recurring problem-solving tasks. For example, in fraction addition, a student typically completes multiple practice problems with the same underlying structure. As another example, consider the stoichiometry problem discussed earlier, i.e.

$$\begin{aligned} 1.216 \text{ kg } COH_4 &\times (1000 \text{ g } COH_4 / 1 \text{ kg } COH_4) \\ &\times (1 \text{ mol } COH_4 / 32.04 \text{ g } COH_4) \\ &\times (1 \text{ mol } O_2 / 2 \text{ mol } COH_4) \\ &= 18.98 \text{ mol } O_2 \qquad \textbf{(1)} \end{aligned}$$

Another problem provided in the stoichiometry tutor is the following:

$$\begin{aligned} 1 \text{ g } Fe_2O_3 &\times (1 \text{ mol } Fe_2O_3 / 159.692 \text{ g } Fe_2O_3) \\ &\times (2 \text{ mol Fe} / 1 \text{ mol } Fe_2O_3) \\ &\times (55.847 \text{ g Fe} / 1 \text{ mol Fe}) \\ &= 0.69943 \text{ g Fe} \qquad \textbf{(2)} \end{aligned}$$

While the specific terms used to solve these two problems are different – for instance, the first multiplier of equation (1) is a unit conversion term, as discussed earlier, while the first multiplier in equation (2) is a molecular weight term – the *structure* across the problems is identical. More specifically, the given value (i.e., the first term) is a single, non-ratio number multiplied by three ratios (i.e., 6-part terms of the kind discussed before) leading to a non-ratio solution. In CTAT we take advantage of this very common situation by providing a template-based approach to creating behavior graphs.

	A	B	C	D	
1	Problem Name	ChemPT_2T_04_PU	ChemPT_2T_04_IU	ChemPT_2T_	Cl
2	%(PRBM)%	Let's suppose we are	A water supply has be	The World He	Th
3	%(G_NV)%	6	6	0.58	
4	%(G_DV)%	100	100	100	
5	%(G_NU)%	g	g	mol	m
6	%(G_DU)%	g	g	kL	kL
7	%(G_NC)%	COH4	COH4	AsO2-	As
8	%(G_DC)%	H2O	H2O	solution	so
9	%(G_R_CV)%	Given Value	Given Value	Given Value	Gi
10	%(G_R_S)%	Well done!	NA	Well done. Y	NA
11	%(G_R_H1)%	Let's think of a reaso	Name the reason why	Let's think of	Th
12	%(G_R_H2)%	Is this term part of th	This term is part of th	Is this term p	Th
13	%(G_R_H3)%	You could select 'Give	Select 'Given Value' a	You could sel	Se
14	%(G_R_SK)%	Select-Given-Value-R	Select-Given-Value-R	Select-Given-	Se
15	%(SK_SET)%	Chemistry-Skills	Chemistry-Skills	Chemistry-Sk	Cl
16	%(UC_NV_CV)%	1000	1000	1	
17	%(UC_NV_S)%	Yes, 1000 is the corre	NA	NA	NA
18	%(UC_NV_H1)%	I suggest that you co	The goal here is to co	Our goal here	Th
19	%(UC_NV_H2)%	Let's convert g to kg.	The quantity provided	Let's convert	Th
20	%(UC_NV_H3)%	Since 1000 g is equiv	Since 1000 g is equiv	Isn't 1 kL eq	Re
21	%(UC_NV_SK)%	Set-Numerator-Value-	Set-Numerator-Value	Set-Numerat	Se
22	%(UC_NU_CV)%	g	g	kL	kL
23	%(UC_NU_S)%	NA	NA	Good. kL is t	NA
24	%(UC_NU_H1)%	Our goal here is to co	The goal here is to co	Perhaps we s	Th
25	%(UC_NU_H2)%	You may want to conv	Convert from g to kg	Let's convert	Co
26	%(UC_NU_H3)%	To convert from g to k	To convert from g to	To convert fro	To
27	%(UC_NU_SK)%	Set-Numerator-Unit-o	Set-Numerator-Unit-o	Set-Numerat	Se
28	%(UC_NC_CV)%	H2O	H2O	solution	so
29	%(UC_NC_S)%	NA	NA	Super job, ke	NA

Figure 25: Part of a Problems Table for the Stoichiometry Tutors; the rows represent variables, the columns represent problems; the cells contain problem-specific values for the variables.

The Mass Production process is straightforward: After the author has demonstrated and edited (and, ideally, debugged and pilot-tested) a behavior graph for a single problem, she turns this behavior graph into a template by introducing *variables* to given values and demonstrated input, effectively removing problem-specific information from the graph. The author may also insert the newly defined variables into hints and error messages, as appropriate, or may introduce variables that stand for entire hint or error messages. Once the template is fully generalized, the author edits a "problems table" in Microsoft Excel. For each problem for which a behavior graph is to be created, the author provides values for each of the variables in the template, as shown in Figure 25, which has all values filled in for a range of problems, including (1) and (2) above. The final step is for the author "to merge" the behavior graph template file with the problems table, in a fashion similar to the way mail merge works in Microsoft Word. The merge step yields multiple instances of the behavior graph, corresponding to individual problems in the problems file (shown as different columns in Figure 25).

Thus, once an initial behavior graph has been created and debugged, tutors of a similar structure can be created very efficiently. In the case of the stoichiometry tutors, we generated a total of 35 tutored problems, similar to those in equations (1) and (2), from three basic templates – a one-term template, a two-term template, and a three-term template. We have used Mass Production in a number of other projects as well. In one project for elementary-school whole-number division, over a thousand behavior graphs were created in this manner. In our experience, however, Mass Production is worthwhile even with very small numbers of isomorphic tutors (e.g., 4).

Not only does the Mass Production process make it easier to produce many example-tracing tutors that share similar structure, it also supports *content consistency* and *maintenance.* It is much easier to look across a row of a table, such as that shown in Figure 25, to check for consistent values and text, than it is to open separate behavior graphs and search for consistency. With respect to

maintenance, we have found that many changes involve a simple global replace. Without Mass Production, individual behavior graphs would have to be opened and edited; a time-consuming process highly prone to errors. (Of course, we have also experienced the usual problems with global replacement, especially when a Problems Table contains text, as does the one depicted in Figure 25. In particular, some text segments may share words but are subtlety different.) The Mass Production facility illustrates how an authoring tool can be combined synergistically with off-the-shelf software. Not only does Excel provide for convenient editing of tables, its formula mechanism can be used (in the Problems Table) to calculate problem-specific values, further enhancing the consistency and maintainability of the tutors[6].

CTAT'S ARCHITECTURE

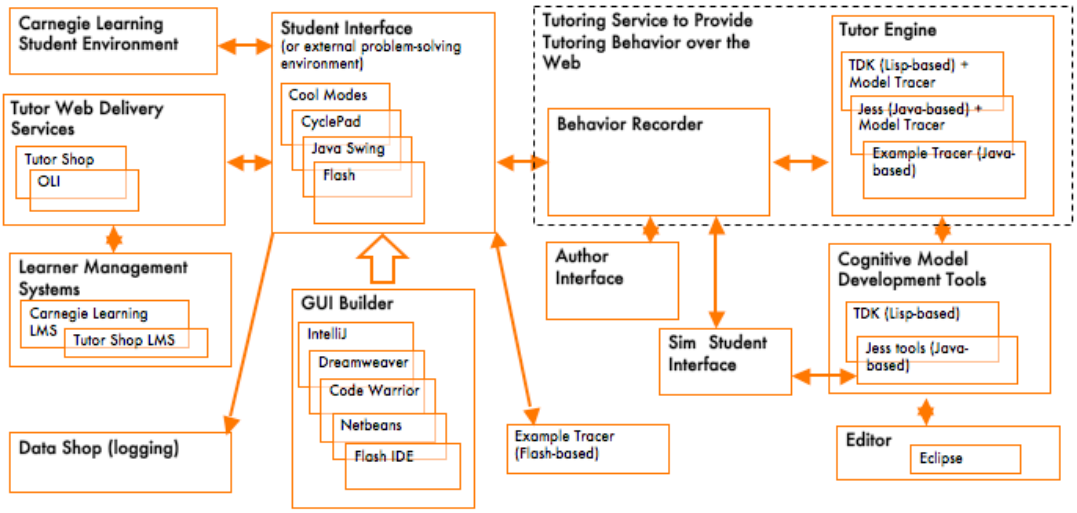

Figure 26: CTAT's modular architecture

Over the years of developing and using CTAT, we have come to realize that it is important to provide an architecture that is modular and supports plug-and-play of ITS components. An authoring tool such as CTAT will not satisfy all its users all the time, so it is important that the tool is (easily) customizable by programmers, even if so far we have emphasized users who are non-programmers. The wholesale plugging-in of new components may be the easiest way to customize. A modular architecture also adds to the versatility of an authoring tool, especially if multiple options are available for (some of) the modules. This versatility enables an author to select the tools appropriate to their particular situation and application. In this section, we briefly describe the main components in CTAT architecture, and illustrate ways in which it is modular and flexible.

---

[6] CTAT's new formula mechanism serves a similar purpose, and unlike the Excel formula mechanism, makes it possible to dynamically compute values, based on student input, as illustrated in the tutor for factoring quadratics above.

CTAT's open architecture (see Figure 26) supports the functions described above: building an interface, creating annotated and generalized behavior graphs, and using a behavior graph to provide tutoring. In addition, it supports the following functions:

1. Building cognitive models for use in Cognitive Tutors.
2. Delivering tutors and tutoring behaviors over the web and by other means.
3. Student management functions such as setting up student accounts, authentication, and teacher reports.
4. Logging of student-tutor interactions for research purposes. All CTAT- tutors have the built-in option to write and transmit detailed logs without requiring any extra effort from the author. Flash tutors in addition can log student-recorded audio clips to Flash's media server.

We already discussed the GUI Builder, Student Interface, Behavior Recorder, and Example Tracer, but have not yet fully discussed the following components:

- The ***Tutor Engine*** takes care of a tutor's inner loop (VanLehn, 2006), that is, it provides step-by-step guidance within a given problem. CTAT provides four different tutor engines: two versions of the example tracer described above, one in Java and one in Flash (although the Flash-based version has been deprecated), as well as two model-tracing engines for cognitive models in Jess (Friedman-Hill, 2003) and TDK (Anderson & Pelletier, 1991).
- The ***Tutoring Service*** provides access to inner-loop tutoring behavior over the web by running a tutor engine on a server machine. It reflects our desire to have a single example tracer compatible with different types of student interfaces running on the web (e.g., Flash, Java)[7].
- The ***Tutor Web Delivery Services*** serve up tutors or tutor interfaces embedded in web pages. They include authentication services and accommodate such functions as gaining students' consent for participating in an experiment. These services integrate with web servers to also provide explanatory text and images that an author might want students to see during a tutoring session.
- The ***Learner Management System*** provides services such as creation of class lists, student accounts, and teacher reports. It also provides "outer loop" functionality (VanLehn, 2006) in that it keeps track of each student's position in the curriculum and provides problem selection options (e.g., macroadaptation). We use an industrial-strength LMS system created in our lab and substantially perfected by Carnegie Learning, a company that markets Cognitive Tutor mathematics courses. This LMS module is gradually replacing the learner management functions of a home-grown module called the "TutorShop."
- The ***DataShop*** is a repository for student-tutor interaction data, primarily for research purposes. It also provides analysis tools for researchers, for example, tools to display and analyze "learning curves" (VanLehn, Koedinger, Skogsholm, Nwaigwe, Hausmann, Weinstein, & Billings, 2007), and an option to export log data to Excel or statistical packages for further analysis. The DataShop is not part of CTAT proper (e.g., it is used to

[7] It was easier to create a server-side tutor engine than to figure out how to integrate the tutor engine and student interfaces locally on the client, although recent developments in Java-Flash integration appear to make that option feasible.

collect data from sources other than CTAT tutors), but is a crucial service for CTAT tutors used in research experiments.

- The ***Cognitive Model Development Tools*** are used to create and debug rule-based cognitive models; they are outside the scope of the current paper, we refer the reader to Aleven, McLaren, Sewall, and Koedinger (2006).

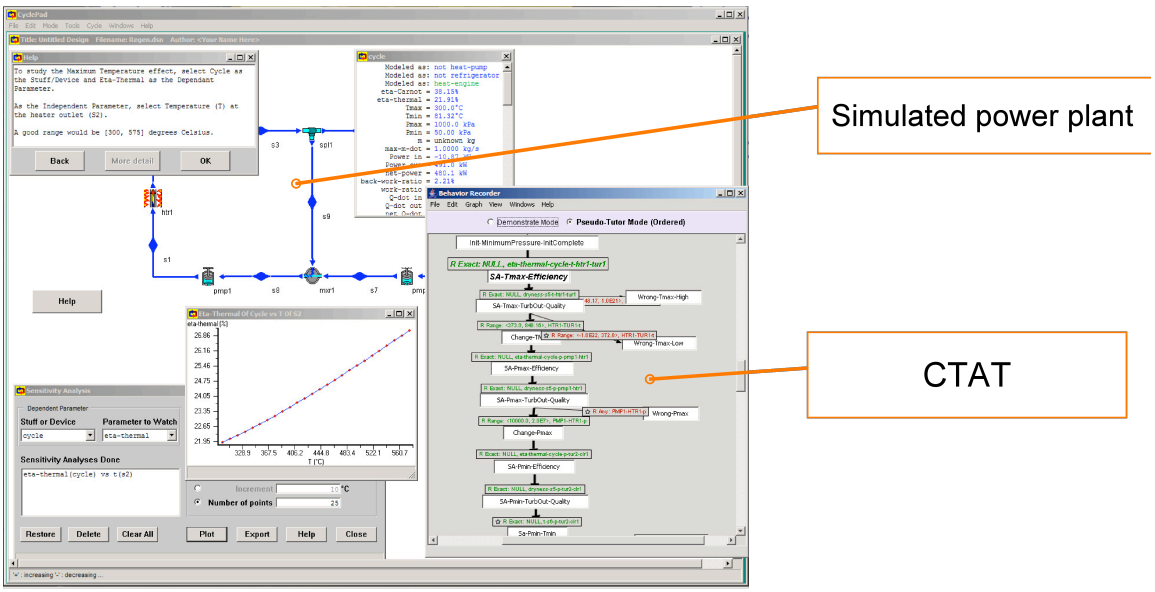

Figure 27: CTAT was used to add tutoring to the CyclePad thermodynamics simulator

As evidence of desirable modularity in the CTAT architecture, a number of components in the CTAT architecture have multiple instantiations, indicated, in the architecture diagram, with "stacked" rectangles, which represent configuration alternatives. We discuss each in turn.

Alternatives for the student interface: CTAT provides direct support for creating student interfaces in Flash and Java, in that its tutor-enabled interface widgets (or components) integrate into commercial GUI builders (integrated development environments, or IDEs) available for those languages. The choice between Java and Flash makes CTAT more versatile (e.g., Flash may be better for the web, Java is better for object-oriented software engineering; and there are many more factors affecting the choice). Furthermore, the modular separation of the student interface and tutor engine has permitted CTAT users to connect their own interactive simulators or user interfaces to our tutor engines, to create tutored versions of the original system. A simulator for thermodynamics (Rosé, Kumar, Aleven, Robinson, & Wu, 2006) (see Figure 27) and an interface for student collaboration (Harrer, McLaren, Walker, Bollen & Sewall, 2006) have been hooked up to CTAT in this way (cf. Blessing *et al.,* 2007; Ritter & Koedinger, 1997), and we are currently in the process of hooking up a chemistry simulator. As a somewhat different aspect of CTAT's modularity, the sets of interface components that can be used in CTAT tutors are easily extensible, because the API for these widgets is well-developed. At least one outside author, with our help, has created a custom component (Wylie, 2007), and we hope that others will contribute components as well, as we make the necessary sources and documentation available.

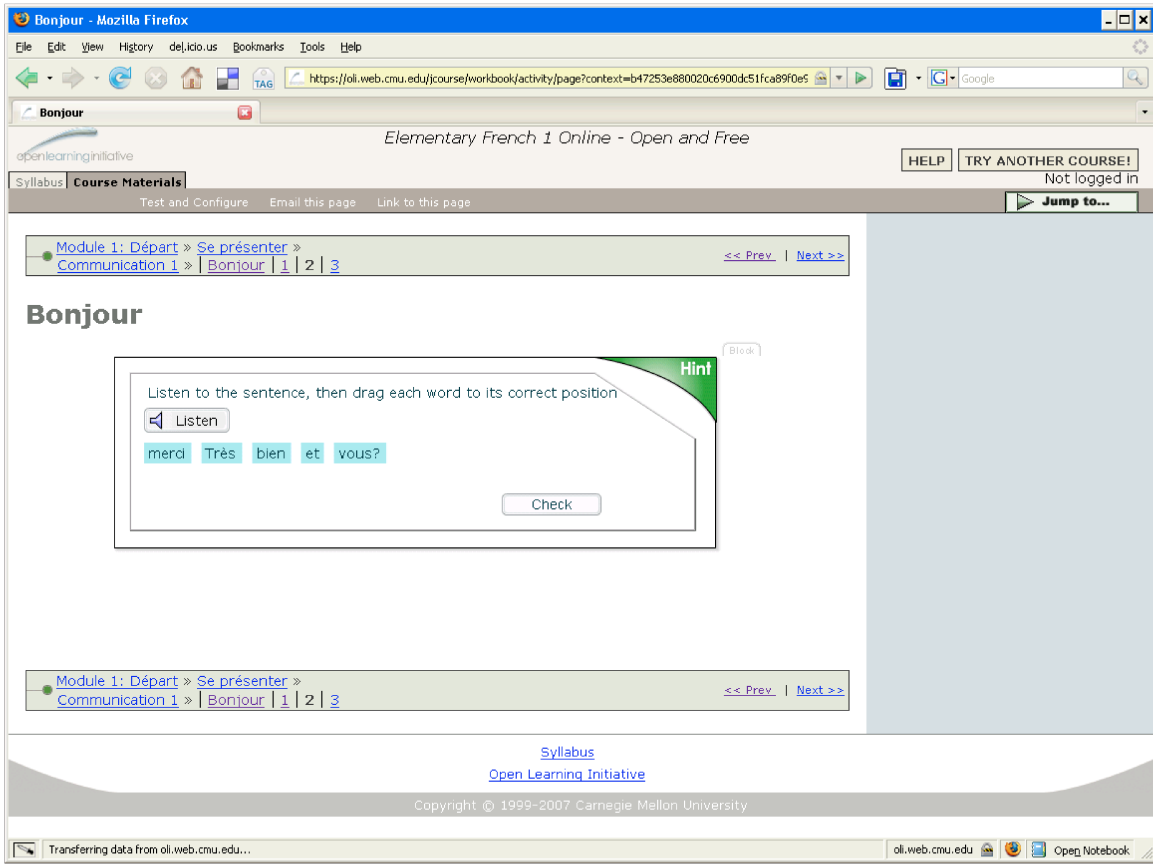

Figure 28: A simple Example-Tracing tutor embedded in an on-line introductory French course within the Open Learning Initiative; this tutor uses the Flash Jumble widget, which lets a student drag a scrambled set of items (here, words in a sentence) into the desired order

Alternatives for the tutor engine: The interface options available in CTAT can be combined in mix-and-match fashion with the four tutor engines mentioned above. So far, in real educational settings, we have seen example-tracing tutors with Flash interfaces and Java interfaces, an example-tracing tutor with a simulator, and TDK-based cognitive tutor (of the "traditional" kind) and with a CTAT-built interface. Jess-based cognitive tutors (with Flash and Java interfaces) have been created in courses, workshops, and are used heavily in the SimStudent project, which uses machine learning to infer a rule-based cognitive model from behavior graphs (Matsuda *et al.,* 2005; 2007).

The ability to mix-and-match interface options and tutor engines hinges on the message protocol between interface and tutor engine. This message protocol (http://ctat.pact.cs.cmu.edu/index.php?id=tool-tutor) is derived from Ritter and Koedinger's (1997) architecture for plug-in tutor agents and the concomitant *tool-tutor communication spec* – the basic idea is the careful separation of tool/interface functionality and tutor functionality.

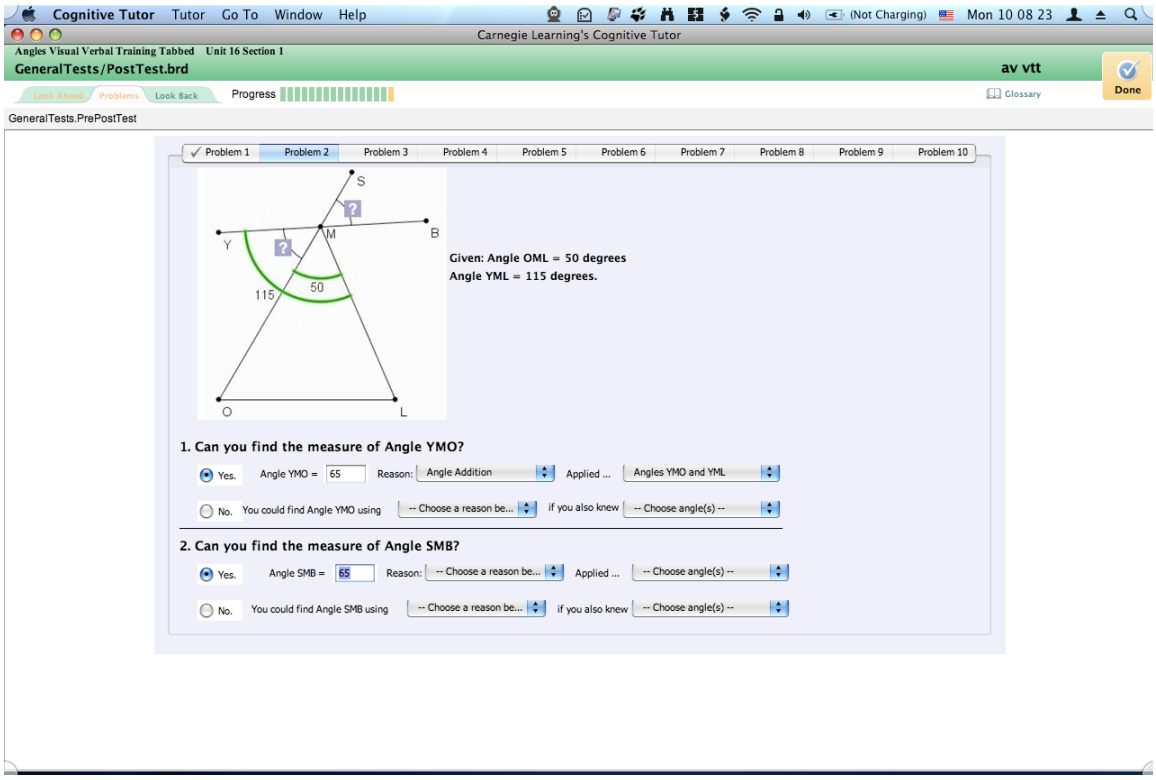

Figure 29: an on-line assessment authored with CTAT embedded in a Carnegie Learning interface (Butcher & Aleven, 2008)

Alternatives for the delivery environment: As another main driver of a flexible architecture, we have encountered the need to deliver tutors in a variety of technical contexts – reflecting the experience that CTAT tutors are often (but not always) embedded in a larger framework for e-Learning or other forms of technology-enhanced learning. For example, CTAT tutors have been used in the following contexts:

- Within an existing e-Learning framework, namely, on-line courses created as part of CMU's Open Learning Initiative (OLI) (see Figure 28).
- Web-based but separate from any existing e-Learning framework, with the tutor engine running either on the client or the server (as part of the Tutoring Service). The stoichiometry and fractions tutors described above belong in this category.
- Integrated with Carnegie Learning's Cognitive Tutors, so that CTAT and CL problems are interleaved (see Figure 29).
- Installed desktop applications: In settings where we have had access to the student machines, we have installed CTAT as a traditional desktop application. Although there are many advantages to running tutors on the web, there are situations (e.g., schools with poor network connectivity) where it is an advantage not to have to rely on the web.

EVIDENCE OF USE AND IMPROVED AUTHORING AFFORDABILITY

CTAT has been used extensively to create example-tracing tutors and on-line assessments. We know of approximately 370 people who have used CTAT to create working tutors and assessments, including people who have used CTAT in research and development projects, and people who have used CTAT in courses and summer schools at Carnegie Mellon, Worcester Polytechnic Institute, and the University of Edinburgh. In addition, there may be others who have used CTAT that we do not know about, since it is freely available on the web. In the past two years alone, CTAT was downloaded over 4,300 times and the CTAT website (http://ctat.pact.cs.cmu.edu) drew over 1.5 million hits from over 100,000 unique visitors. We regularly receive technical questions and requests for support via e-mail.

As shown in Appendix A, CTAT has been used to create tutors for mathematics (elementary, middle-school, and high-school, including whole-number division, fractions, algebra, and geometry), chemistry, genetics, thermodynamics, French language learning and culture, Chinese language learning, and English as a Second Language (ESL). This list only includes tutors that have seen *actual use in real educational settings*, either as part of classroom research, or as part of regular instruction. As the appendix indicates, CTAT-built tutors and assessments have been used in about 26 research studies in real educational settings. Further, CTAT will be used in a new project we have just started to create a comprehensive website for middle-school mathematics. To the best of our knowledge, this makes CTAT the most widely used ITS authoring tool in existence.

In order to get a sense for whether CTAT improves the cost-effectiveness of authoring ITS, we compare against "historical" estimates, reported in the literature, of overall development time and cost versus the project output (e.g., hours of instruction produced). This method provides some confidence that true gains are being achieved, but it is important not to lose sight of its limitations. The overall project cost-output estimates run the risk of comparing apples and oranges, for example, in that historical projects often have had time to produce high quality output, which may require more iteration and maintenance, whereas new projects may underestimate costs for such iteration and maintenance. On the other hand, historical projects often stretched out over longer periods of time, so that project development teams had time to become familiar with the tools and the technology, to hone their development skills, establish good working procedures, select the best tools for the job, re-use work done in earlier phases, etc. In other words, within these projects, there usually was sufficient time to achieve an economy of scale. Smaller projects typically do not reach that point. Further, the estimates of development time tend to be very rough, since they are often based on retrospective accounting of how time was spent, and they tend to be somewhat inconsistent across projects, since it is not easy to separate tutor development from other project activities. Finally, different projects used CTAT at different stages of its development – for instance, the stoichiometry tutors were built before functions were available, functionality that would have made their development much easier – which also introduces variability. The main point is that the development time estimates should be treated as rough estimates.

As our baseline, we use development time estimates reported in the literature, which for ITSs range from 200:1 (Woolf & Cunningham, 1987) to 300:1 (Murray, 2003). The "historical" development time estimates for the successful Algebra and Geometry Cognitive Tutors (see, e.g., Koedinger *et al.,* 1997) are in line with these estimates. A rough estimate puts the development time for these two Cognitive Tutors at 200 hours for each hour of instruction. It is important to note that these tutors *pre-date CTAT.* They were built with the Tutor Development Kit (TDK) (Anderson &

Pelletier, 1991), our initial Lisp-based cognitive modeling environment. In fact, these Cognitive Tutors (and their development times) were an important source of inspiration for CTAT.

The development times for CTAT-built tutors that have seen actual use in real educational settings are listed in Appendix A. CTAT staff built the tutors in one of these projects (stoichiometry). In all others, the tutors were built primarily by project staff or students not directly associated with CTAT, who typically had little prior experience developing tutors. In most of these projects, the CTAT staff had a consulting role, but did not do substantial tutor development. In order to provide a fair comparison against historical estimates, the development time estimates listed in the table include all tutor development activities, from the earliest conception of the tutor to the cognitive task analysis activities, tutor implementation, testing, pilot testing (sometimes), and debugging. However, they do not include general CTAT extensions implemented (by CTAT personnel) in the course of these projects, since we are interested in finding out how much the tool contributes to the affordability of authoring. Appendix A also lists, for those projects for which this information could be ascertained, the amount of instructional time covered by the CTAT tutors that were developed. This information was obtained from the lead researchers involved in each project. Since these projects were research studies in real educational settings, the amount of time covered by the tutors is known quite accurately, as arrangements needed to be made with collaborating teachers regarding the number of class periods that the study would cover.

The development time picture that emerges is very interesting. If we distinguish between small projects (at most one hour of instruction created) and larger projects (more than one hour), we see that small projects do not "beat" the 200:1 and 300:1 ratio reported in the literature. On the other hand, the larger projects consistently do better than that, with the Whole-Number Division project "weighing in" at a 107:1 ratio, an Algebra tutoring project at 87:1, the Stoichiometry project a 85:1 ratio, and the fractions project at 48:1. These estimates are higher than those reported in an early paper on CTAT (Koedinger *et al.,* 2004), which no doubt reflects the tougher standard used in the current paper, in particular, that the tutor must have been used in an actual educational setting. Even so, the larger projects that have used CTAT point to considerable savings in development time, with development time estimates 3-4 times better than the historical averages. The fact that the smaller projects have not consistently done better than the historical averages is likely due to the many factors discussed above: these projects have the start-up costs that all projects have but no opportunity to develop within-project economy of scale. The savings are even greater when we measure development effort in terms of cost rather than time. In many of the projects listed above, the CTAT tools were used by non-programmers, so if one estimates that PhD level AI programmers cost about 1.5-2 times as much as non-programmers, then our very rough estimate indicates that CTAT leads to 4-8 times more cost-effective development of ITSs than the "historical" results reported in the literature. We reiterate that there is a risk of comparing apples and oranges: the historical estimates were based on larger-scale projects that may have created higher-quality tutors through multiple iterations of use and revision, although we should emphasize that we included in our sample only tutors that were good enough for real educational settings, which is a high standard. Further, the historical projects may have developed a within-project economy of scale not possible for the shorter-duration projects being reviewed here.

DISCUSSION

In this section, we review how CTAT stacks up against the six requirements for authoring tools that we outlined in the introduction, summarizing and, in some cases, expanding upon points made earlier in the paper.

CTAT supports the authoring of effective tutors (criterion 1). Example-tracing tutors implement much of the behavior that is considered typical of ITSs, described by VanLehn (2006), focused on step-by-step guidance to students as they solve complex problems. However, example-tracing tutors go well beyond VanLehn's basic criteria. First, they follow students along multiple solution paths within a given problem, regardless of which one the student decides to pick. This capability sets CTAT apart from some other authoring tools, including Assistments (Razzaq *et al.*, 2005) though not from others, such as ASPIRE (Mitrovic *et al.*, 2006). This capability matters, because many recurring problem-solving tasks, such as the stoichiometry problems shown above, naturally give rise to multiple solution paths. Even in a relatively simple fraction addition problem (e.g., 1/4 + 1/6), a student might use different strategies to find common denominators (e.g., 12 or 24 are both correct). Example-tracing tutors' formula mechanism enables them to recognize a potentially very large number of different paths, too many to enumerate explicitly. As illustrated by the tutor for factoring quadratics, there is a large class of problem-solving tasks for which it is very difficult to predefine all paths, but that are relatively straightforward to address with formulas. As another example, if one wanted to build a tutor for fraction addition that will accept, as the common denominator, *any* common multiple of the denominators of the two fractions to be added (e.g., when adding 1/4 + 1/6, convert to common denominator 12, 24, 36, 48, 60, etc.), it would not be possible to enumerate all possible paths, but it would be straightforward to capture them all with a single formula.

A second way in which example-tracing tutors go beyond VanLehn's basic criterion is in their ability to maintain *multiple interpretations* of student behavior, for as long as these interpretations are consistent with the students' actions. This capability obviates the need for disambiguating student intent "on the spot," as for example Cognitive Tutors do. Delayed disambiguation enables the tutor to follow the student, "wherever they go" in the given problem, as long as they stay within the behavior graph, without interrupting them with disambiguation questions and without running the risk of steering them down a different solution path than they had in mind.

While example-tracing tutors support an interesting range of tutoring behaviors, there remains a class of applications for which Cognitive Tutors are the preferred choice, at least when personnel with the requisite rule-based programming skill is available. For example, Cognitive Tutors are often preferred for problems with variable or unpredictable subgoals, problems in which the relation between actions and subgoals is unclear, problems whose solutions constitute complex structures (e.g., the Lisp Tutor, Anderson, Conrad, & Corbett, 1989), or problems with subtle ordering constraints among steps. We anticipate that we will have more to say about the difference between example-tracing tutors and Cognitive Tutors as we gain experience using CTAT's new formula mechanism, which significantly narrows the gap.

Example-tracing tutors have proven their effectiveness on measures of learning. For instance, the stoichiometry tutors discussed earlier led to statistically significant learning gains from pre to posttest in *all* conditions of three separate studies (McLaren, Lim, & Koedinger, 2008a).

Example-tracing tutors have been built for a range of application domains (criterion 2). CTAT is a versatile tool, and has been applied not only in quantitative domains such as mathematics and science, in which ITSs have long had a strong track record, but also in language learning and even cultural learning. As the data presented in Appendix A indicate, the specific domains for which CTAT tutors have been built (and used in real educational settings) include mathematics (at the elementary school, middle-school, and high-school level), chemistry, genetics, thermodynamics, Chinese, French, and English as a Second Language. CTAT has been used to provide tutoring within a simulation for thermodynamics, and work is underway to hook up CTAT with a chemistry simulator.

In general, CTAT is useful for building tutoring systems that support complex problem solving and even systems that support not-so-complex problem solving. The example-tracing technology seems unnecessary for applications in which there is a single question and answer. For example, CTAT may not be particularly useful for teaching vocabulary items or historical facts, although it may well be used to create a system for historical analysis, although even for such applications, many components of the CTAT architecture may be useful (e.g., web delivery, logging, student management). CTAT proper is not currently geared toward supporting the authoring of tutorial interactions in natural language, or parsing of answers in natural language. Finally, CTAT is not geared towards building systems with large domain ontologies (e.g., Crowley & Medvedeva, 2006) or that draw on large stores of factual or conceptual knowledge (e.g. question generation systems).

Example-tracing tutors provide a cost-effective way of developing tutors (i.e., minimize time and money) (criterion 3). CTAT is one of very few ITS authoring tools that make it possible to author sophisticated tutors without programming. ASPIRE (Mitrovic *et al*., 2006) is another such tool; we briefly compared CTAT to ASPIRE in the introduction. Thus, CTAT can help ITS development projects avoid the cost of expensive programmers, those with, for instance, specialized skills in AI programming. In many of the CTAT projects reported in Appendix A, non-programmers used CTAT to create tutors or on-line assessments, including many authors who are not part of the CTAT development team, and who did not have extensive CTAT experience. Often, but not always, these authors worked in consultation with the CTAT team. Of course, authoring tools do not obviate the need for careful cognitive task analysis, interface design, or use of instructional design or learning sciences principles. Thus, one still needs personnel skilled in these areas. Further, although CTAT authors do not have to be programmers, so far, they typically have been tech-savvy people, not to be intimidated by relatively complex environments such as the Flash IDE, the Netbeans IDE, and CTAT proper. Finally, although the sets of interface-enabled widgets that CTAT provides are often sufficient to create interfaces that support complex reasoning tasks, for certain interactions new widgets are no doubt necessary. These new widgets will require programming.

Development time estimates from projects that have used CTAT over the past years indicate that CTAT considerably brings down the time needed to develop intelligent tutors. Our estimates of affordability savings come from comparing overall development costs of past large-scale development projects using prior authoring tools versus projects using CTAT. These estimates indicate an improvement of as much as 4-8 times greater cost-effectiveness, due to CTAT. It should be kept in mind that the reliability of such estimates is limited because of the many sources of variability (e.g., different personnel, different task domain, different quality of resulting tutors, different time scale of projects, and different opportunity to develop within-project economy of scale). It appears that the savings may be due in the first place to lower upfront implementation costs for each new problem type for which tutoring is to be provided, and to a lesser degree to lower implementation costs per tutored

problem. Creating a new problem type for a Cognitive Tutor requires programming extensions to the cognitive model, in the form of production rules, and often requires interface programming (although many re-usable interface components were developed). With example-tracing tutors, on the other hand, an interface and Mass Production template can be created without programming (at least if no new interface components are needed) and therefore typically take less time than the interface and cognitive model programming needed to add a new problem type to a Cognitive Tutor. Once the upfront work for a new problem type has been done, adding a new problem is rather easy both for Cognitive Tutors and example-tracing tutors, at least assuming that Mass Production is used. One way to take advantage of the relative ease with which new problem types and tutor interfaces can be created with CTAT might be to increase problem variability (e.g., Paas & van Merrienboer, 1994), which has been shown to lead to increased learning. In other words, students will encounter the same knowledge components in a problem set with greater variability, for example in terms of the steps or even the whole tutor interface. If we may be somewhat speculative, example-tracing tutors may lead to a more varied set of tutors; this greater variability may in turn lead to better student learning.

A number of developments are underway that may further improve the cost-effectiveness of authoring with CTAT, and the quality of the resulting tutors: the SimStudent module of CTAT (Matsuda *et al.,* 2005; 2007), which uses machine learning to create rule-based cognitive models, and a novel "bootstrapping" facility that lets an author create behavior graphs directly from log data of students. The bootstrapping facility is likely to lead to tutors with greater ability to recognize actual student strategies and errors, especially in domains with somewhat more open-ended activities such as the use of simulators (Harrer *et al.*, 2006; McLaren *et al.,* 2004).

CTAT supports delivery of tutors in a wide variety of technical contexts (criterion 4). Given the rise in Internet applications in present times, it is key that CTAT makes the process of building and delivering web-based tutors relatively easy. Being able to deliver in multiple contexts adds considerably to the complexity of CTAT, but CTAT's modular architecture helps achieve this goal.

CTAT supports ease-of-maintenance of example-tracing tutors in a number of ways (criterion 5). When tutors are built on a larger scale, and are used over longer periods of time, by successive cohorts of students, maintenance becomes an important concern. How easy or hard is it to modify tutors when requirements change? Although at this point in time, we have limited experience with regular maintenance of CTAT-built tutors over extended periods of time, we can point out that, serendipitously, many of the mechanisms in CTAT that enhance the ease of authoring, scaling up of tutored problems, and the flexibility of tutors also result in tutors that are easier to maintain. In particular, the Mass Production facility, designed to reduce repetitious authoring tasks and to support the production of many tutored problems, has the welcome side effect of making tutors easier to maintain. Similarly, many of the mechanisms that were designed to make example-tracing tutors more flexible have the effect of making them easier to maintain. Mechanisms such as unordered mode, range matches, and formulas effectively serve to "collapse" multiple behavior graph paths into a single one, which should significantly enhance maintainability. We will add that our views on maintainability of CTAT tutors are still developing. Few of the tutors built in CTAT have had a multi-year lifespan, and only a few of them have been updated regularly, a real test of maintainability. So far, however, our experience indicates that it is manageable to maintain a set of example-tracing tutors.

Part of the motivation for CTAT has been from the start that *it supports research use of tutors (criterion 6).* In particular, CTAT supports research into how people learn with ITSs, and what features make ITSs effective. In a typical research study, a researcher compares student learning with two tutor versions that differ only with respect to a single feature hypothesized to be a key influence on student learning. Many (but not all) of the tutors presented in this paper were built for such research purposes. For instance, the stoichiometry tutors were used to investigate the effect of polite problem statements, feedback, and hints (McLaren *et al.,* 2007). The fractions tutors were used to investigate the effect of having students work with multiple graphical representations of fractions and of prompts for students to self-explain the representations (Rau, 2008). CTAT supports research in two main ways: first, by lowering the skill threshold for tutor building, it makes it easier for researchers from non-technical disciplines (e.g., the learning sciences or educational psychology) to create and do experiments with CTAT-built tutors. Second, a number of key features are geared toward research use of tutors. Most importantly, CTAT tutors write logs of all student-tutor interactions, which allow researchers to study students' learning trajectories in great detail (VanLehn et al., 2007). CTAT also makes it possible to create on-line assessments, essentially tutors with the tutoring turned off. These "tutors" still evaluate students' problem-solving actions, but they do so silently, without providing feedback or hints. They write logs of student actions, and their evaluations of these actions, effectively offering a form of automated grading. Even the mass production capability facilitates the adapting of tutors for educational research: in the stoichiometry politeness study, for example, in which we studied the effect of polite language on student learning, polite and not-so-polite tutor versions were created by editing the problems table in Excel, much easier to do than editing individual behavior graphs. Finally, one of the CTAT components (i.e., the TutorShop) has the capability to automatically assign students to conditions.

CONCLUSION

In this paper, we report on our on-going project to develop the Cognitive Tutor Authoring Tools, highlighting two main contributions we believe this project has made in its first 6 years. First, CTAT supports the development of example-tracing tutors, a novel way of building intelligent tutoring systems. Second, it provides tools for building these types of tutors without programming.

Example-tracing tutors support sophisticated tutoring behavior. They provide step-by-step guidance in complex problem-solving activities, and thus meet VanLehn's (2006) minimal criterion for being regarded as ITSs. They go well beyond VanLehn's criterion, however, in that they are capable of recognizing multiple student strategies, of dynamically defining multiple solution paths within a problem, and of maintaining multiple parallel interpretations of student behavior. CTAT is one of the very few ITS authoring tools that supports the development of sophisticated tutoring behaviors *and* supports tutor building without programming. ASPIRE (Mitrovic *et al.,* 2006) is another such tool. Example-tracing tutors are built through drag-and-drop techniques, programming by demonstration, and template-based authoring. They have drastically reduced the skill level required to build tutors, which previously required PhD-level AI programming skill.

Evidence of CTAT effectiveness is its widespread and continued use ("vote-with-your-feet evidence"). At least 350 people have used CTAT, perhaps many more. Tutors have been built with CTAT for both real classroom and experimental use, across a wide range of domains, often by authors who were not directly associated with the CTAT project and had little prior experience in tutor

development. Development time estimates from a range of projects indicate that CTAT greatly reduces the cost-effectiveness of building ITS. Prior estimates of development time (Murray, 2003) have ranged from 200-300 hours of development time per one hour of instructional time. CTAT has managed to lower this ratio to 50-100 hours and thus reduced development time by a factor of 3-4. If one factors in the lower cost of personnel, then the savings may be as much as 4-8 times. We (or others who have used CTAT) do not yet have much experience with maintenance of an extended set of CTAT tutors over the course of a multi-year project. The CTAT tutors described in this paper have not yet seen multiple cycles of use and revision. We do believe however that maintenance of example-tracing tutors will turn out to be highly feasible and that the same tool features that limit the amount of problem-specific authoring (e.g., Mass Production) that needs to be done for real-world tutoring systems will also greatly facilitate maintenance.

Our experience with CTAT indicates that ITS authoring tools must satisfy many other criteria besides ease of authoring: (web) delivery and deployment of tutors, student management, and research use of tutors. Further, we have found that authoring tools can be fruitfully combined with other ways of facilitating authoring, or lowering its cost, such as use of application software (off-the shelf tools such as GUI builders and Excel), and an open architecture that supports plug-and-play of components (e.g., use of existing simulators, or an existing learner management system) and is extensible (e.g., we hope to see third parties contribute interface widgets/components, or provide Java code to extend the library of functions used in CTAT's formula mechanism). The CTAT authoring suite has now persisted (as well as continued to evolve) through over 6 years of use. Other ITS authoring tools also have reached or are reaching a mature state where they support development of real-world tutors. It is our strong belief that in the years to come, these authoring tools will contribute greatly to making ITSs widespread.

ACKNOWLEDGEMENTS

Brett Leber and Martina Rau contributed to this paper. The CTAT project was supported by grants from ONR Award No: N000140310220, the Grable Foundation, and the National Science Foundation under Award No. SBE0354420. Their contributions are gratefully acknowledged. The opinions expressed in the current paper do not represent any official position of the funding agencies.

REFERENCES

Ainsworth, S., Major, N., Grimshaw, S., Hayes, M., Underwood, J., Williams, B. & Wood, D. (2003). REDEEM: Simple intelligent tutoring systems from usable tools. In: T. Murray, S. Blessing, and S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (Chapter 8, pp. 205-232). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Aleahmad, T., Aleven, V., & Kraut, R. (2008). Open community authoring of targeted worked example problems. In B. Woolf, E. Aimeur, R. Nkambou, S. Lajoie (Eds), *Proceedings of the 9th International Conference on Intelligent Tutoring Systems, Lecture Notes in Computer Science*, 5091 (pp. 216-227). Berlin: Springer.

Aleven, V., Sewall, J., McLaren, B. M., & Koedinger, K. R. (2006). Rapid authoring of intelligent tutors for real-world and experimental use. In Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. G. Sampson, & W.

Didderen (Eds.), *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006),* (pp. 847-851). Los Alamitos, CA: IEEE Computer Society.

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. W. Chan (Eds.), *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (ITS 2006), (pp. 61-70). Berlin: Springer Verlag.

Anderson, J. R. (1993). *Rules of the Mind.* Mahwah, NJ: Lawrence Erlbaum.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP Tutor. *Cognitive Science, 13,* 467-506.

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

Anderson, J. R., & Lebière, C. (1998). *The Atomic Components of Thought.* Mahwah, NJ: Erlbaum.

Anderson, J.R., & Pelletier, R. (1991). A development system for model-tracing tutors. In *Proceedings of the International Conference of the Learning Sciences* (pp. 1-8).

Baker, R.S.J.d., Corbett, A.T., & Koedinger, K.R. (2007). The difficulty factors approach to the design of lessons in intelligent tutor curricula. *International Journal of Artificial Intelligence in Education, 17*(4), 341-369.

Beal, C. R., Walles, R., Arroyo, I., & Woolf, B. P. (2007). Online tutoring for math achievement: A controlled evaluation. *Journal of Interactive Online Learning, 6,* 43-55.

Blessing, S., Gilbert, S., Ourado, S., & Ritter, S. (2007). Lowering the bar for creating model-tracing intelligent tutoring systems. In the *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (AIED-07) (pp. 443-450).

Blessing, S.B. (2003). A programming by demonstration authoring tool for model-tracing tutors. In: T. Murray, S. Blessing, and S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments*. Chapter 4, 93-119. Dordrecht, the Netherlands: Kluwer Academic Publishers.

Booth, J.L., Koedinger, K.R., & Siegler, R.S. (2007). [Abstract]. The effect of prior conceptual knowledge on procedural performance and learning in algebra. In D.S. McNamara & J.G. Trafton (Eds.), *Proceedings of the 29th Annual Cognitive Science Society* (pp. 137-142). Austin, TX: Cognitive Science Society.

Brown, J. S. (1985). Idea amplifiers: New kinds of electronic learning environments. *Educational Horizons, 63,* 108–112.

Brusilovsky, P. (2003). Developing adaptive educational hypermedia systems: from design models to authoring tools. In: T. Murray, S. Blessing, and S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments*. Chapter 13, 377-409. Dordrecht, the Netherlands: Kluwer Academic Publishers.

Butcher, K., & Aleven, V. (2008). Diagram interaction during intelligent tutoring in geometry: Support for knowledge retention and deep transfer. In C. Schunn (Ed.) *Proceedings of the 30th Annual Meeting of the Cognitive Science Society, CogSci 2008.* New York, NY: Lawrence Erlbaum.

Crowley R. S., & Medvedeva, O. (2006). An intelligent tutoring system for visual classification problem solving. *Artificial Intelligence in Medicine, 36*(1), 85-117.

Corbett, A. T., & Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction, 4,* 253-278.

Devedzic, V. & Harrer, A. (2005). Software patterns in ITS architectures. *International Journal of Artificial Intelligence in Education*, *15*(2), 63-94.

du Boulay, B. (2006). Commentary on Kurt VanLehn's "The Behavior of Tutoring Systems." *International Journal of Artificial Intelligence in Education, 16*(3), 267-270.

Friedman-Hill, E. (2003). *Jess in Action.* Manning Publications Co.

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

Halff, H.M., Hsieh, P.Y., Wenzel, B.M., Chudanov, T.J., Dirnberger, M.T., Gibson, E.G., & Redfield, C.L. (2003). Requiem for a development system: reflections on knowledge-based, generative instruction. In: T. Murray, S. Blessing, and S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments*. Chapter 2, 33-59. Dordrecht, the Netherlands: Kluwer Academic Publishers.

Harrer, A., McLaren, B.M., Walker, E., Bollen, L., & Sewall, J. (2006). Creating Cognitive Tutors for collaborative learning: steps toward realization. *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* (UMUAI). 16. (p. 175-209).

Harrer, A., Pinkwart, N., McLaren, B.M., & Scheuer, O. (in press). The Scalable Adapter Design Pattern: Enabling Interoperability Between Educational Software Tools. *IEEE Transactions on Learning Technologies.*

Koedinger, K.R. & Aleven, V. (2007). Exploring the Assistance Dilemma in experiments with Cognitive Tutors. *Educational Psychology Review*. 19(3), 239—264.

Koedinger, K.R., Aleven, V., Heffernan, N., McLaren, B.M., & Hockenberry, M. (2004). Opening the door to non-programmers: authoring intelligent tutor behavior by demonstration. In the *Proceedings of the Seventh International Conference on Intelligent Tutoring Systems* (ITS-2004). (p. 162-174).

Koedinger, K.R., Anderson, J.R., Hadley, W.H., & Mark, M.A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8(1), 30-43.

Koedinger, K. R. & Corbett, A. T. (2006). Cognitive Tutors: Technology bringing learning science to the classroom. In K. Sawyer (Ed.) *The Cambridge Handbook of the Learning Sciences*. Cambridge University Press.

Koedinger, K. R., Suthers, D. D., & Forbus, K. D. (1999). Component-based construction of a science learning space. *International Journal of Artificial Intelligence in Education*, 10, 292-313.

Kumar, R., Rosé, C. P., Wang, Y. C., Joshi, M., & Robinson, A. (2007). Tutorial dialogue as adaptive collaborative learning support , *Proceedings of Artificial Intelligence in Education.*

Lieberman, H. (Ed.) (2001). Your wish is my command: Programming by example. Morgan Kaufmann, San Francisco, CA.

Martin, B. & Mitrovic, A. (2002). Automatic problem generation in constraint-based tutors. In S.A, Cerri, G. Gouarderes & F. Paraguacu (Eds.) *Intelligent Tutoring Systems: 6th International Conference* (ITS-2002). (p. 388-398). Berlin: Springer.

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2007). Evaluating a simulated student using real students data for training and testing. In C. Conati, K. McCoy & G. Paliouras (Eds.), *Proceedings of the International Conference on User Modeling* (LNAI 4511) (pp. 107-116). Berlin, Heidelberg: Springer.

Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2005). Building Cognitive Tutors with programming by demonstration. In S. Kramer & B. Pfahringer (Eds.), *Proceedings of the International Conference on Inductive Logic Programming* (Technical report, TUM-I0510) (pp. 41-46): Institut fur Informatik, Technische Universitat Munchen.

McLaren, B.M., Lim, S., & Koedinger, K.R. (2008a). When and How Often Should Worked Examples be Given to Students? New Results and a Summary of the Current State of Research. In B. C. Love, K. McRae, & V. M. Sloutsky (Eds.), *Proceedings of the 30th Annual Conference of the Cognitive Science Society* (pp. 2176-2181). Austin, TX: Cognitive Science Society.

McLaren, B.M., Lim, S., & Koedinger, K.R. (2008b). When is Assistance Helpful to Learning? Results in Combining Worked Examples and Intelligent Tutoring. In B. Woolf, E. Aimeur, R. Nkambou, S. Lajoie (Eds), *Proceedings of the 9th International Conference on Intelligent Tutoring Systems, Lecture Notes in Computer Science*, 5091 (pp. 677-680). Berlin: Springer.

McLaren, B.M., Lim, S., Yaron, D., & Koedinger, K.R. (2007). Can a polite intelligent tutoring system lead to improved learning outside of the lab? In the *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (AIED 2007), IOS Press. (p. 443-440).

McLaren, B.M., Lim, S., Gagnon, F., Yaron, D., & Koedinger, K.R. (2006). Studying the effects of personalized language and worked examples in the context of a web-based intelligent tutor; In the *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (ITS-2006). Jhongli, Taiwan, June 26-30. (p. 318-328).

McLaren, B.M., Koedinger, K.R., Schneider, M., Harrer, A., and Bollen, L. (2004). Toward Cognitive Tutoring in a Collaborative, Web-Based Environment. In Engineering Advanced Web Applications, From the *Proceedings in Connection with the 4th International Conference on Web Engineering* (ICWE 2004). Munich, Germany, 28-30 July, 2004. Rinton Press, (p. 167-179).

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. Accepted for ED-MEDIA 2008.

Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., & Holland, J. (2006). Authoring Constraint-Based Tutors in ASPIRE. In M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.), *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)* (pp. 41-50). Berlin: Springer Verlag.

Moore, J. L. (1992). Winch simulations: Multiple, linked representations of linear functions. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Proceedings of the Second International Conference on Intelligent Tutoring Systems, ITS '92* (pp. 111-115). Berlin: Springer-Verlag.

Mostow, J., & Beck, J. (2007). When the rubber meets the road: Lessons from the in-school adventures of an automated Reading Tutor that listens. In B. Schneider & S.-K. McDonald (Eds*.), Conceptualizing Scale-Up: Multidisciplinary Perspectives* (Vol. 2, pp. 183-200). Lanham, MD: Rowman & Littlefield.

Munro, A. (2003). Authoring simulation-centered learning environments with Rides and Vivids. In: T. Murray, S. Blessing, and S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (Chapter 3, pp. 61-91). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In: T. Murray, S. Blessing, and S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments*. Chapter 17, 491-544. Dordrecht, the Netherlands: Kluwer Academic Publishers.

Murray, T., Blessing, S., & Ainsworth, S. (Eds.) (2003). *Authoring Tools for Advanced Learning Environments: Toward Cost-Effective Adaptive, Interactive and Intelligent Educational Software*. Dordrecht, the Netherlands: Kluwer Academic Publishers.

Myers, B.A., McDaniel, R.G., & Kosbie, D.S. (1993). Marquise: Creating complete user interfaces by demonstration. In *Proceedings of INTERCHI'93: Human Factors in Computing Systems*.

Newell, A. & Simon, H. A. (1972). *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall.

Ogan, A., Aleven, V., & Jones, C. (2008). Pause, predict, and ponder: Use of narrative videos to improve cultural discussion and learning. In *Proceedings of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems* (pp. 155-162). New York, NY: ACM.

Paas, F., & Van Merrienboer, J. (1994). Variability of worked examples and transfer of geometry problem-solving skills: A cognitive-load approach. *Journal of Educational Psychology, 86,* 122-133.

Rau, M. A. (2008). Flexible knowledge of fractions with multiple graphical representations in intelligent tutoring systems. Albert-Ludwigs-Universität, Freiburg im Breisgau.

Razzaq, L. & Heffernan, N.T. (2006). Scaffolding vs. hints in the Assistment System. In Ikeda, Ashley & Chan (Eds.). *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*. Springer-Verlag: Berlin. pp. 635-644. 2006.

Razzaq, L., Feng, M., Nuzzo-Jones, G., Heffernan, N.T., Koedinger, K. R., Junker, B., Ritter, S., Knight, A., Aniszczyk, C., Choksey, S., Livak, T., Mercado, E., Turner, T.E., Upalekar. R, Walonoski, J.A., Macasek. M.A., Rasmussen, K.P. (2005). The Assistment Project: Blending Assessment and Assisting. In C.K. Looi, G. McCalla, B. Bredeweg, & J. Breuker (Eds.) *Proceedings of the 12th International Conference on Artificial Intelligence In Education*, 555-562. Amsterdam: ISO Press.

Reed, S. K. (2005). From research to practice and back: The Animation Tutor project. *Educational Psychology Review, 17*(1), 55-82.

Rickel, J. & Johnson, W.L. (1999). Animated agents for procedural training in virtual reality: Perception, Cognition, and Motor Control. *Applied Artificial Intelligence*, 13, 343-382.

Ritter, S., Blessing, S., & Wheeler, L. (2003). Tools for component-based learning environments. *Authoring Tools for Advanced Learning Environments*. Chapter 16, 467-489. Dordrecht, the Netherlands: Kluwer Academic Publishers.

Ritter, S. & Koedinger, K. R. (1997). An architecture for plug-in tutoring agents. *International Journal of Artificial Intelligence in Education*, 7 (3/4), 315-347.

Roll, I., Ryu, E., Sewall, J., Leber, B., McLaren, B.M., Aleven, V., & Koedinger, K.R. (2006) Towards Teaching Metacognition: Supporting Spontaneous Self-Assessment. *in Proceedings of 8th International Conference on Intelligent Tutoring Systems,* 738-40. Berlin: Springer Verlag.

Roll, I., Aleven, V., McLaren, B. M., & Koedinger, K. R. (2007). Can help seeking be tutored? Searching for the secret sauce of metacognitive tutoring. *International Conference on Artificial Intelligence in Education 2007,* Los-Angeles, CA.

Rosé, C. P., Kumar, R., Aleven, V., Robinson, A., & Wu, C. (2006). CycleTalk: Data driven design of support for simulation based learning. *International Journal of Artificial Intelligence and Education, 16,* 195-223.

Tecuci, G., & Keeling, H. (1999). Developing an intelligent educational agent with Disciple. *International Journal of Artificial Intelligence in Education, 10,* 221-237

Towne, D. (2003). Automated Knowledge Acquisition for Intelligent Support of Diagnostic Reasoning. *Authoring Tools for Advanced Learning Environments*. Chapter 5, 121-147. Dordrecht, the Netherlands: Kluwer Academic Publishers.

VanLehn, K., Koedinger, K.R., Skogsholm, A., Nwaigwe, A. Hausmann, R.G.M., Weinstein, A. & Billings, B. (2007). What's in a Step? Toward General, Abstract Representations of Tutoring System Log Data. *User Modeling* 2007: 455-459.

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

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

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge.* Los Altos, CA: Morgan Kaufmann.

Woolf, B. P., & Cunningham, P. (1987). Building a community memory for intelligent tutoring systems. *AAAI 1987* (pp. 82-89).

Wylie, R. (2007) Are we asking the right questions? Understanding which tasks lead to robust learning of the English article system. In R. Luckin, K. Koedinger, & J. Greer (Eds.), Proceedings of the 13th International Conference on Artificial Intelligence in Education (pp. 709-710). Amsterdam, the Netherlands: IOS Press.

Yaron, D., Evans, K. & Karabinos, M. (2003). Scenes and Labs Supporting Online Chemistry. Paper presented at the *83rd Annual AERA National Conference*, April 2003.

Appendix A: Tutors built with CTAT

Project Title	Domain	Tutor Description	Studies	Students	Instructional Time	Development Time	Time Ratio	Papers
The Self-Assessment Tutor	Geometry - Angles, Quadrilaterals	Tutor designed to tutor students on their self-assessment skills; to help students make appropriative learning decisions based on their self assessment; and to give students a tutoring environment, low on cognitive load, in which they can practice using their help-seeking skills.	1	67	45 mins	~9 weeks	540:1	Roll, Ryu, Sewall, Leber, McLaren, Aleven, & Koedinger, 2006; Roll, Aleven, McLaren, & Koedinger, 2007
Using Elaborated Explanations to Support Geometry Learning	Geometry	The assessment consisted of two types of problems. First, problem-solving items were used to determine if students could recognize solvable quantities and calculate the answers using known geometry principles. Second, reasoning items were used in which students reasoned about the relationships between diagram elements based on geometry principles.	1	90	30 mins	~2 months	720:1	Butcher & Aleven, 2008.
Fluency and Sense Making in Elementary Math Learning	4th-Grade Math - Whole-number division	The tutor provided practice with whole-number division, with two conditions, one aimed at fluency, one aimed at fluency and sense making, through self-explanation support, including an interactive multiplication table. On-line assessment was also created with CTAT.	1	~35	2.5 hrs each for 2 conditions plus 1 hr of assessment	~4 months	107:1	
Improving Skill at Solving Equations through Better Encoding of Algebraic Concepts	Middle and High School Math - Algebra	CTAT example-tracing tutors interleaved with Carnegie Learning tutor sections. Tutors supports self-explanation of worked examples with correct and incorrect equation-solving steps.	3	268	16 mins each for 2 conditions	~120 hrs	240:1	Booth, Koedinger, & Siegler, 2007.
Intelligent Writing Tutor	ESL - English article system (a, an, the, null)	Two computer-based systems to help students learn the English article system (a, an, the, null). The first system, a menu-based task, mimics cloze activities found in many ESL textbooks. The second, a controlled-editing task, gives students practice with both detecting errors and producing the correct response. The researchers created a custom CTAT widget for use in their systems.	3	~50				Wylie, 2007
Improving Cultural Learning By Predicting In French Film	French – Intercultural competence	This tutor uses the multimedia components of CTAT to display a French film and elicit predictions and responses from students about cultural behaviors.	2	45				Ogan, Aleven, & Jones, in press; 2006

Project Title	Domain	Tutor Description	Studies	Students	Instructional Time	Development Time	Time Ratio	Papers
Chinese PSLC-OLI Course	Chinese - Elementary Chinese	18-unit course includes various (simple) example-tracing tutors: Jumble, Media Multiple Selection, and the CTAT Video template.		Regular use				
French OLI Course	Introductory French	Various (simple) example-tracing tutors have been created that make use of the Jumble, Media Multiple Selection, and the CTAT Video template		Regular use				
Enhancing Learning Through Worked Examples with Interactive Graphics	Algebra - Equation Models of Problem Situations	Two types of example-tracing tutor units have been developed. In the Model Generation unit (16 problems), students are given a situation and write an equation. In the Model Analysis unit (4 problems), students are given a problem situation and an equation, and employ menus to describe what each component of the equation represents.	1	60-120	~3 hrs	~260 hrs	87:1	
Effect of Personalization and Worked Examples in the Solving of Stoichiometry	Chemistry Stoichiometry	Example-tracing tutors with Flash interface for stoichiometry problem solving. On-line assessment was also authored with CTAT.	4	223	12 hrs	1016 hrs	85:1	McLaren, Lim, Gagnon, Yaron, & Koedinger, 2006; McLaren, Lim, Yaron, & Koedinger, 2007.
Learning a tonal language-Chinese	Chinese	Example-tracing tutors for learning to recognize sounds in tonal language						
Genetics Cognitive Tutor:	Genetics	Java Cognitive Tutor (with one unit of example-tracing tutors) supporting genetics problem solving.		Regular use				
CycleTalk	Mechanical Engineering - Thermodynamics	Example-tracing tutors connected to a simulator that help students design and optimize thermodynamic cycles, such as the Rankine cycle used in power plants. A separate (non-CTAT) module supports natural language dialogue about thermodynamics design.	4	330				Rosé, Kumar, Aleven, Robinson, & Wu 2006; Kumar, Rosé, Wang, Joshi, & Robinson, 2007
Bridging the gap between comprehension and production in second language learning	French - Grammar	Training and tests for grammatical accuracy. In the experimental condition, comprehension and production items alternate, whereas in the control condition they are blocked. The CTAT Flash user interface was used to display feedback to students.	2	45				

Project Title	Domain	Tutor Description	Studies	Students	Instructional Time	Development Time	Time Ratio	Papers
Fostering fluency in second language learning: Testing two types of instruction	ESL – speaking fluency	The CTAT Flash framework was used for the user interface and problem assessment including sourcing of data (brd). Also student data was logged to the Data Shop using the CTAT Data Shop integration. The CTAT TutorShop study delivery server software was used to sequence the study.	1	30				
OLI Economics course	Economics – supply and demand	Example-tracing tutors use of a graph as a plotting input device.		Regular use				
Visual Feature Focus in Geometry: Instructional Support for Visual Coordination During Learning	Geometry - Angles	Study 1 Tutor: Visual examples and non-examples were shown either alone, or with a set of text statements from which relevant statements were selected. Feedback was provided on selected statements. Assessment: Conceptual training was assessed by quizzes following each unit. Problem solving also was tested by quizzes following each unit of study. Pre- and posttest performance was assessed using Butcher & Aleven's previous CTAT assessment.	2	60 + 90 = 150				
The Fractions Tutor	6th-Grade Math - Fraction Conversion, Fraction Addition	Students solve fractions problems with four tutor versions (all example-tracing tutors): single v. multiple representations of fractions, and self-explanation (with drop-down menu) v. no self-explanation.	1	132	2.5 hours each for 4 conditions	12 weeks	48:1	