Advances in Cognitive Systems 5 (2017) Submitted 3/2017; published 5/2017

Demonstration-based Solution Authoring for Skill Assessment

Melinda Gervasio MELINDA.GERVASIO@SRI.COM
Karen Myers KAREN.MYERS@SRI.COM
Michael Wessel MICHAEL.WESSEL@SRI.COM
Artificial Intelligence Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025 USA

Abstract

The high cost of developing content has been a major impediment to the widespread deployment of intelligent training systems. To enable automated skill assessment, traditional approaches have required significant time investment by highly trained individuals to encode first-principles domain models for the training task. In contrast, approaches grounded in example-based methods have been shown to significantly reduce authoring time. This paper reports on an approach to creating solution models for automated skill assessment using an example-based methodology, specifically targeting domains for which solution models must support robustness to learner mistakes. With this approach, a content author creates a baseline solution model by demonstrating a solution instance and then specifies a set of annotations to generalize from that instance to a comprehensive solution model. Results from a user study show that domain experts are comfortable with the approach and capable of applying it to generate quality solution models.

1. Introduction

Intelligent tutoring/training systems (ITS) have shown great potential for enabling self-directed learning adapted to individual skill levels and learning styles. However, a significant impediment to the widespread use of ITS technologies is the substantial effort required to develop the content that drives their operation. As noted by others, effective authoring frameworks are essential to alleviate the high cost of content development for ITS applications (Brawner et al., 2012).

This paper reports on work that addresses a key portion of the content authoring problem, namely the creation of models for supporting assessment of learner performance. We focus on training domains that have a *procedural* flavor in that students seek to master skills that involve performing sequences of context-dependent actions to achieve desired effects. Proficiency in such domains improves with practice, which can be enabled through automated assessment capabilities grounded in ITS technologies.

Example-tracing tutors have emerged as a promising technology for training procedural tasks (Aleven et al., 2009). Example-tracing tutors identify differences between a student's actions (recorded in an instrumented environment) and a predefined solution trace for completing a task. These differences can be used both to assess student performance and to provide targeted feedback. Several training systems based on example tracing have been built, including our prior work on a framework called Drill Evaluation for Training (DEFT).

DEFT uses approximate graph-matching technology to align learner actions with predefined solution models. This alignment provides the basis for generating assessment information with contextually relevant feedback: identifying errors, providing hints to help the student complete a task, and suggesting links to relevant training materials. Importantly, our flexible graph matching capability provides tolerance to learner mistakes, avoiding the need for strict adherence to specific solutions. Using this assessment technology, we created prototype training tools for two significant application domains. One of these assists with training for the Command Post of the Future (CPOF)—a collaborative geospatial visualization environment system used extensively by the U.S. Army to develop situational awareness and to plan military operations (Myers et al., 2013). The second application is to rifle maintenance, drawing on requirements from a U.S. Army soldier training publication (Greuel et al., 2016).

One of the appeals of example tracing is the potential for using solution demonstrations as the basis for solution model acquisition. With a demonstration-based approach, content creation no longer requires deep understanding of the knowledge representation and inference mechanisms within the ITS. As such, assessment-related content can be created directly by domain experts after a modest amount of training. Studies reported in (Koedinger et al., 2004) show an order-of-magnitude reduction in development time for demonstration-based creation of solution models for example-tracing tutors compared to direct authoring of first-principles models.

In prior work, we conducted a paper study to inform the design of authoring frameworks for solution models based on end-user programming techniques (Myers & Gervasio, 2016). The study showed that domain experts (i.e., not ITS experts) are comfortable with the approach and are capable of applying it to generate quality solution models for moderately complex tasks. It also identified constructs that, while important for accurate solution characterization, can lead to confusion and so warrant special care in tool design.

The work in this paper builds on that initial study. In particular, we describe the design of an implemented authoring tool for solution models that was informed by the study. The tool supports an authoring process in which a user first demonstrates a baseline solution and then generalizes the demonstration to a comprehensive model by providing appropriate annotations that designate allowed variations from the specifics of the demonstration. We then present results from a follow-up study in a cooking domain that provides evidence for the practicality, effectiveness, and viability of our solution authoring approach.

The paper is organized as follows. Section 2 presents background information on our approach to assessment. Section 3 describes our demonstration-based approach to solution authoring along with the specific tool that we developed to support this process. Section 4 presents the design of our user study and summarizes findings and implications. Section 5 summarizes related work and Section 6 presents concluding remarks.

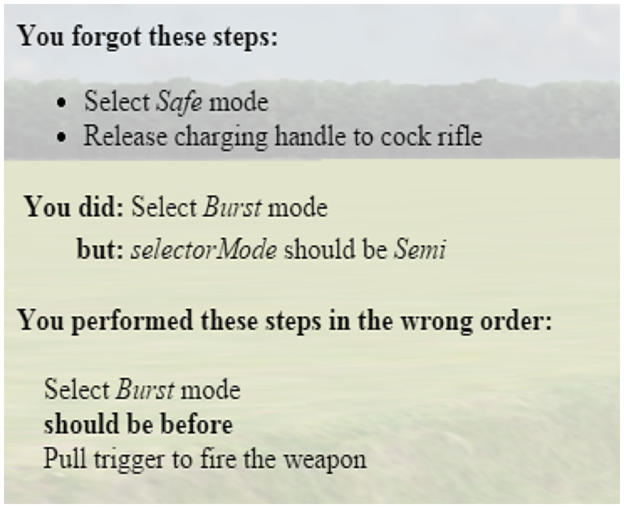
2. Automated Assessment via Flexible Graph Matching

We represent solution models for a training exercise (i.e., a task to be achieved) in terms of one or more *generalized action traces,* each consisting of a sequence of *steps* and *annotations* that specify allowed variations. A step is a *parameterized action*, a *family of actions* (e.g., all the different ways that a delete action can be performed), or a *set of options*, each composed of a partially ordered set of steps. Annotations defined over steps include action ordering and grouping constraints to support specification of, for example, a partial order between steps, some of which may be specific actions and others of which may be any of a family or actions. Annotations over parameters include type, value, membership, and equality constraints; these support the specification of constraints such as a parameter taking on any of a specified set of alternative values of a particular type or that the output of one action be the same as the input of another. The action traces and accompanying annotations define a set of constraints on the space of all possible solutions and thus a solution model implicitly defines a set of specific valid solution instances for a particular exercise.

Our automated assessment capability determines a mapping from the student's response to the exercise solution model. This alignment problem is formulated as *approximate graph matching*, using graph edit distance to rate the quality of the mappings. Graph edit distance measures the cumulative cost of graph editing operations needed to transform the student response into an instance consistent with the exercise solution model.

To use this graph matching approach, the exercise solution model is encoded as one or more graphs, each representing a family of possible solutions. Actions and their parameters are nodes within these graphs, parameter roles are links, and required conditions within the solution (e.g., action orderings, parameter values) are constraints. The student response is represented similarly as a response graph. Alignment involves finding the lowest-cost mapping between the response and a solution graph, with costs incurred for missing mappings and violated constraints. The intuition is that the lowest-cost alignment corresponds to the specific solution the student is most likely attempting. From this alignment, an assessment is generated that identifies differences between the response and the exercise solution model, which translate to specific errors the student has made (e.g., out-of-order, missing, or extra actions; incorrect parameter values) and to the corrections needed. The alignment algorithm is described in more detail in (Gervasio et al., 2017).

The display of assessment feedback in our original DEFT system presented the full action trace to the learner with overlaid markings that identified learner mistakes. However, users showed a preference for a summary view of their mistakes: they did not want to see everything they had done, rather only what they had done incorrectly (Myers et al., 2013). For this reason, we transitioned from the original trace-based presentation of feedback to a summary-based approach. Figure 1 shows sample assessment feedback of this type for a training exercise from our rifle maintenance domain.


You forgot these steps:

- Select *Safe* mode
- Release charging handle to cock rifle

You did: Select *Burst* mode
but: *selectorMode* should be *Semi*

You performed these steps in the wrong order:

Select *Burst* mode
should be before
Pull trigger to fire the weapon

Figure 1. Sample assessment feedback

Our approach to automated assessment is similar to that of other example-tracing tutors, e.g., (Aleven et al., 2009), which compare learner actions to a graph representing alternative solution paths. In contrast to other example-tracing assessment capabilities, our graph-matching approach does not force users down a limited set of solution paths. Rather, the flexible matching provides tolerance to user mistakes, enabling assessment for domains in which more exploratory styles of task completion are an important part of the learning experience.

3. Solution Authoring

In an online training application, a content author develops exercises for the learner, where the exercises include solution models for automated assessment (Figure 2). As part of the framework we developed for our prototype training applications (Greuel et al., 2016), we developed a tool for authoring solution models. Our approach to authoring involves two steps: (1) demonstrating one or more specific solutions to a training exercise, and (2) specifying annotations that generalize the demonstrations to the full range of allowed solutions.

The author performs Step 1 within the same application students will use for training (e.g., a virtual environment or a complex piece of software with instrumentation to capture interactions). The output of Step 1 is a *demonstration trace*, which provides the input to Step 2. Step 2 is performed within our authoring tool, the *exercise solution editor* (ESE). The ESE provides annotation mechanisms that let the solution author generalize the specific sequence of steps in the demonstration trace to a comprehensive solution model specification.

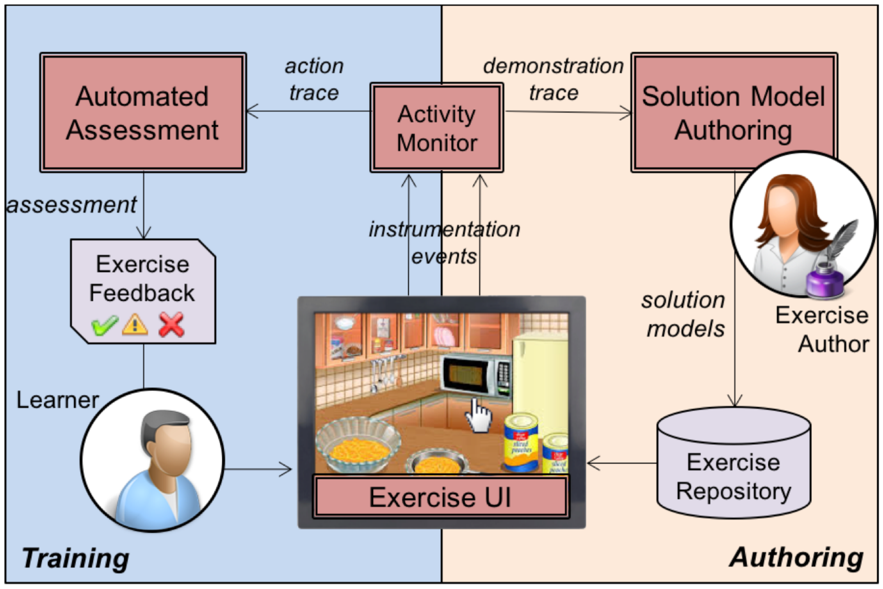
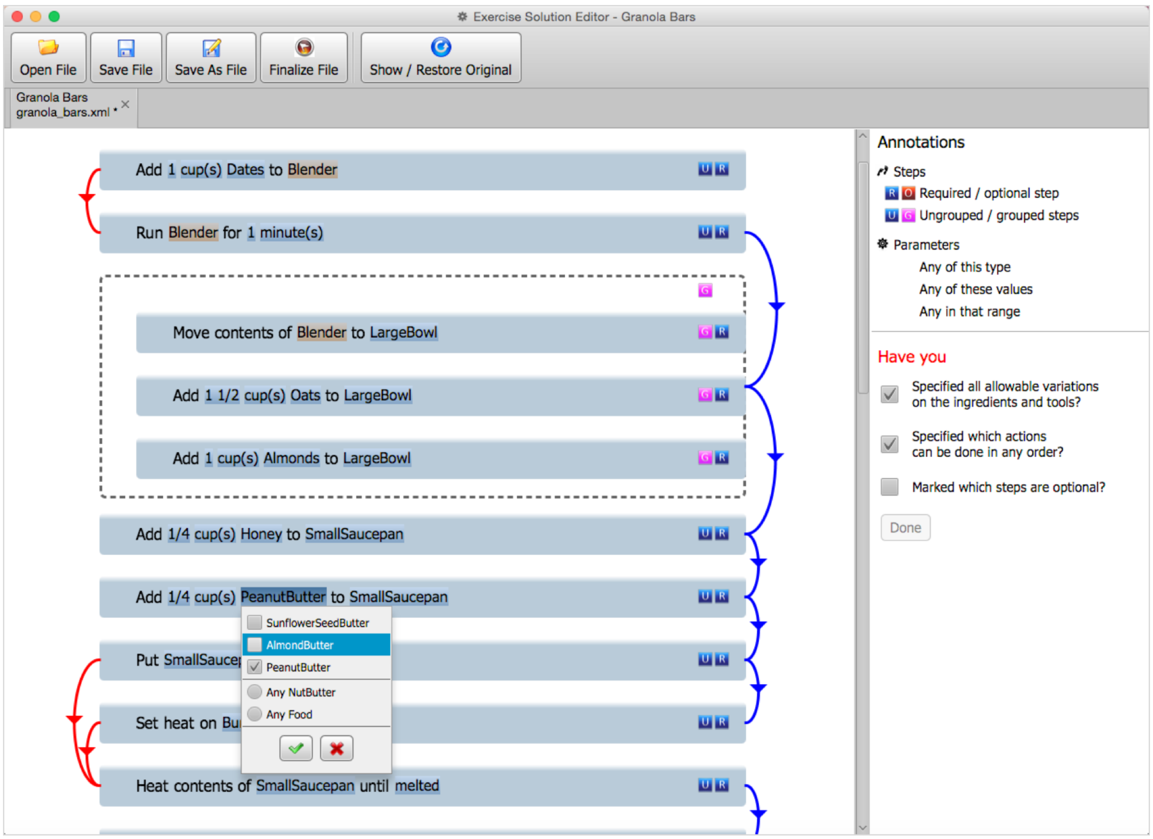


Figure 2. Solution model authoring for online training.

3.1 The Exercise Solution Editor (ESE)

The ESE is a direct-manipulation interface that provides both a visualization of the solution model (i.e., a demonstration trace + annotations) and a variety of mechanisms for editing it. The screenshot of the ESE in Figure 3 shows a solution model being developed for a cooking task. On the left is the main pane, containing the annotated demonstration trace. It shows the sequence of steps, consisting of single actions and groups of actions, as well as the parameters of each step. The steps are decorated with various annotations, including arrows representing different types of ordering constraints and buttons indicating step optionality and grouping. These annotations can be made through drag-and-drop gestures on the demonstration trace or through the *Annotations* pane on the right, which also serves as a legend for the semantics of the buttons on the steps (e.g., 'R' and 'O' mean 'Required' and 'Optional' respectively).

As discussed previously, annotations on a solution model include annotations on individual steps, sets of steps, and step parameters to support the specification of optionality constraints, ordering and grouping constraints, and various parameter constraints. We now discuss each of these in more detail.



Add 1 cup(s) Dates to Blender U R

Run Blender for 1 minute(s) U R

G

Move contents of Blender to LargeBowl G R

Add 1 1/2 cup(s) Oats to LargeBowl G R

Add 1 cup(s) Almonds to LargeBowl G R

Add 1/4 cup(s) Honey to SmallSaucepan U R

Add 1/4 cup(s) PeanutButter to SmallSaucepan U R

Put SmallSauce U R

Set heat on Bu U R

Heat contents of SmallSaucepan until melted U R

- ☐ SunflowerSeedButter
- ☐ AlmondButter
- ☑ PeanutButter
- ○ Any NutButter
- ○ Any Food

Annotations

Steps

- R O Required / optional step
- U G Ungrouped / grouped steps

Parameters

- Any of this type
- Any of these values
- Any in that range

Have you

- ☑ Specified all allowable variations on the ingredients and tools?
- ☑ Specified which actions can be done in any order?
- ☐ Marked which steps are optional?

Done

Figure 3. Screenshot of the exercise solution editor showing a partially annotated demonstration trace.

3.1.1 Annotations on Single Steps

By default, the steps in a demonstration trace are ‘Required’, as signified by the ‘R’ on the upper right of the step. Clicking on the ‘R’ toggles it into an ‘O’, indicating that the step is now ‘Optional’. Marking multiple steps as optional may be achieved more efficiently through the controls provided by the Annotations pane. Clicking on the ‘R/O’ option here switches to a specialized editing mode limited to toggling optionality of steps. During assessment, students are only penalized for missing steps that are marked as ‘Required’. The ESE currently only supported ‘Required/Optional’ annotations on steps. Future annotations, based on user feedback, could include ‘Preferred’ to indicate optional but desirable steps, and ‘Benign’ to explicitly indicate extraneous but allowable actions.

3.1.2 Annotations over Multiple Steps

The ESE supports two types of multi-step annotations to specify sequencing constraints: *ordering* and *grouping*. These annotations allow the solution author to relax the order of actions in the demonstration trace, generalizing the solution model to a larger set of acceptable solutions. The ESE uses arrows to visualize two types of ordering constraints: *red arrows* that depict *fixed ordering constraints* (i.e., non-editable constraints imposed by the physical world and derived from preconditions in the background action model for a domain) and *blue arrows* that depict *implied ordering constraints* (initially, the ordering in which the actions were demonstrated). For example, Figure 3 shows that putting dates into the toaster *must* be done before they can be toasted. It also specifies that sugar should be added to the pan before the almond butter; however, there is no physical reason that the order cannot be inverted and the author may choose to do so.

Steps can be reordered by dragging a step to the beginning or end of the sequence, or by moving it to any position in between steps. However, a step cannot be dropped at a location where a fixed (red) ordering constraints would be *violated*—ESE will revert this drag-and-drop operation after also informing the user of the violation with a pop-up message.

After a successful move operation, the implied (blue) ordering constraints are automatically recomputed. To avoid visual clutter, a blue arrow is established between a step and its direct successor only if they are not already connected via a red arrow. Blue arrows represent *direct succession* in the visual/temporal sequence of steps, unlike red arrows, which do not necessarily encode direct succession.

While arrows visualize ordering constraints, a *group of steps* (depicted as a dashed box in Figure 2) represents the *lack of ordering constraints*. That is, a group is a set of actions that can be performed in any order. For example, in Figure 2, the group involving adding ingredients to a bowl precedes the step of adding sugar to the pan, but the order in which the ingredients are added to the bowl does not matter.

The group as well as individual steps in a group can be involved in ordering constraints with steps outside the group. By dragging a step into or out of a group, ordering constraints can be relaxed or imposed between the step and the group. However, because grouping implicitly specifies any ordering of the steps in the group, there cannot be fixed ordering constraints between members of the same group. Consequently, the ESE prevents the creation of groups that contain red arrows—for example, moving the first two steps into the subsequent group would not be allowed as the fixed ordering between them would be inconsistent with the non-ordering implied by the grouping.

An earlier version of the ESE supported grouping but did not explicitly show implied ordering constraints (blue arrows). We decided to add the visualization of implied ordering for two reasons. First, the initial user study showed that users assume that actions will be performed in the same order as in the trace, unless expressly indicated otherwise. While grouping effectively communicated that the steps in the group could be performed in any order, there was no corresponding visual cue for the implicit ordering between steps not contained in groups. Pilot tests found that this approach could lead to confusion. Second, we were concerned that the lack of an explicit visualization for the ordering constraints would lead to content authors forgetting to relax ordering constraints (through grouping) that are not required for correct solutions. The earlier version also did not support fixed ordering constraints (red arrows). This addition led to the potential contradictions with groups described earlier. However, we decided to retain the concept of explicit groups of unordered steps because we believe it improves understandability of ordering requirements compared to approaches without groupings that would require many more ordering annotations to be added. This meant generalizing the concept to support the retention of required ordering constraints from steps within the group to steps outside of the group.

3.1.3 *Annotations on Parameters*

The ESE supports generalization of action parameters relative to an underlying type hierarchy, which is defined as part of the underlying *action model* for the training domain. Authors can generalize specific parameter values by selecting (super)types or by specifying (sets of) enumerated instances. In Figure 3, the author is in the process of replacing the ‘PeanutButter’ parameter value in the first action with ‘AlmondButter’, a different instance of the ‘NutButter’ type. Alternatively, the user could generalize to ‘Any NutButter’ (type) or ‘Any Food’ (supertype). The menu options are dynamically determined based on the current parameter value and its range of values. The ESE also supports generalizing numeric quantities through specialized parameter editors that allow, for example, changing ‘1 cup’ to ‘between 2 and 3 cups’.

Sometimes, it is desirable to change a particular value not only on one step, but in all the steps that use that value. For example, instead of using the ‘LargeBowl’ throughout the exercise, the user might decide that the ‘MediumBowl’ is sufficient. To support this function, when a changed parameter value is entered and confirmed by the user and there are multiple occurrences of the value, the ESE asks the user whether all occurrences should be changed. This is convenient for editing longer solution models and also helps reduce errors. Such a global replace is converted by the ESE into an equality constraint between the parameters involved. A further feature in the ESE is highlighting all modified parameters (for example, ‘Blender’ in Figure 3) by comparing the current model with the original model to identify differences.

3.1.4 Prompting

The preliminary user study revealed the need and desire for some sort of intelligent prompting mechanism to help the solution author identify overlooked annotations. To this end, we implemented a simple prompting mechanism in the form of a checklist presented to the author after a distinguished 'Finalize' step. The checklist covered whether all possible parameter variations has been specified, which steps could be done in any order, and which steps were optional. Authors had to check off all three to complete the finalization step (see the checklist in the bottom right of Figure 3). This idea was inspired by findings from the assessment community that simple cueing mechanisms can generate small but measurable performance gains for tasks that involve creative thinking (Green et al., 2012).

4. User Study: Solution Authoring

To evaluate our approach to solution authoring, we conducted a user study involving subjects using the ESE to create solution models for a number of tasks. Our approach to solution authoring consists of demonstration and annotation; this study is focused on the annotation process, under the assumption that experts should have no trouble demonstrating a solution to a training exercise. Because our target audience consists of domain experts, we needed a study domain where subjects would have expert-level knowledge.[1] Building on our earlier paper study, we chose cooking, given that many people are experts in how they like to prepare their food, even if their preparations may be highly personalized.

To support the user study, we developed an action model comprising 16 actions and a simple object ontology with 40 classes defined over 5 levels, including 113 instances representing different ingredients, kitchen equipment and appliances, and concepts. The ontology is not intended to be comprehensive and the actions are not as specific as might be ideal (e.g., we use the single action SPREAD to capture cooking actions more commonly distinguished as SPRINKLE, SPREAD, and GREASE). Overall, though, the action model provides sufficient coverage and variability while supporting generalizations required for recipes used in the study.

4.1 Experimental Protocol

We conducted a two-on-one (facilitator + note-taker vs. subject) study involving five subjects, with each subject session taking one to two hours. To start each session, subjects were given a short tutorial on the ESE and then asked to practice using the editor on a simple test trace to let them achieve basic competence with the tool. The subjects were also asked to save their annotated trace but we omitted the prompted review step so that we could better evaluate the effects of prompting during the study tasks themselves. Subjects were introduced to the practice of "thinking aloud" and were instructed to think out loud as they annotated the traces.

The subjects were then asked to create generalized solution models suitable for learner assessment on three cooking tasks of moderate complexity. First, subjects were presented with a recipe and a demonstration trace in the ESE representing one specific instance of the recipe (a completely ordered sequence of actions with full specification of the arguments for each action). Because the subjects did not create the traces themselves, they were given time to familiarize themselves with the recipe and the trace, and instructed to treat the recipe as a starting point but to

[1] We did not use the two real-world domains for which we had previously built significant prototypes because of the unavailability of subjects with expert-level knowledge in those areas.

feel free to make annotations reflecting what they felt were the acceptable ways of performing the cooking task represented by the recipe.

Next, the subject was asked to use the ESE to generalize the trace to a comprehensive model of solutions for the cooking task. This involved adding annotations to generalize trace actions and modifying the default sequencing information (the order the actions were performed). Table 1 summarizes the specific annotations supported in the ESE for the study.

When subjects indicated that they were done, they were asked to click the 'Finalize' button. Doing so triggered a review phase where subjects were presented with the checklist designed to prompt them into considering whether they had considered all relevant annotations. Subjects were then left to complete the process without any further explanation or intervention. After the subject completed the three tasks, the facilitator led an open-ended discussion to elicit feedback on what was easy or hard about the task, and suggestions for capabilities that would make the task easier. Finally, subjects were asked to complete two questionnaires: one on computer use (Schroeders & Wilhem, 2011) and another on perceived system usability (Brooks, 1996).

Table 1. Annotations for generalization of the demonstration trace from the user study.

	Annotation	**Semantics**
Steps	Reposition	Change the position of a step within the overall trace
	In any order	Perform a group of steps in any order
	Optional	Step is not required as part of the solution
Parameters	Any of values	Use any of an enumerated set of values (e.g., red pepper or onion)
	Any of type	Use any values of a particular type (e.g., Cheese or Pasta)
	Range	Use a range of values (e.g., 1-5)

4.2 Results

The primary result of the study is that the ESE supports effective solution authoring: subjects were able to understand the traces and to effectively create their intended solution models through the annotation mechanisms. The five subjects (P1, P2, P3, P4, P5) were all SRI employees who cooked regularly and were self-reported to be comfortable or very comfortable with computers. All regularly use a variety of computer devices and software and had had exposure to some form of programming (e.g., writing an Excel macro) in the past, but none were computer programmers or scientists. We believe they are representative of the target audience for solution authoring—people who are knowledgeable about the task to be encoded but not necessarily familiar with formal encodings such as computer programs for interpretation by a computer.

4.2.1 Subjects understood the annotation task and found it to be straightforward

The thoughts spoken out loud by the subjects revealed that they understood the task of generalization and were thinking about the various ways in which to correctly generalize the recipe (e.g., "Is order important…", "Is it still grilled chicken even without the seasoning and sauce?", "Can you have granola bars without peanut butter?"). During the sessions, we realized that a particularly important concept to convey to the subjects was that they were developing a solution model to be used by a computer for automated assessment rather than creating an instructional guide (a 'recipe') for a student to follow. With this clarification, subjects were quickly able to change their mindset to think about a computer evaluating someone's performance rather than someone following a prescribed process.

Subjects quickly grasped the idea of parameter generalization but due to our limited ontology, they were sometimes uncertain about whether generalizing to a type was the same as generalizing to the all the individual instances listed as alternative values. Thus, subjects would sometimes elect to check off all instances (*Any of values*) rather than simply selecting the type (*Any of type*).

For the most part, subjects understood the ordering constraints implied by the two types of arrows. However, the red arrows (fixed ordering constraints) often required explaining more than once and some subjects remained unsure about whether they needed to check them and whether they could edit them ('no' to both). As discussed earlier, the blue arrows (implied ordering constraints) are redundant with the natural ordering people infer when presented with a list of actions but are useful for other reasons. Here, we observed that they appeared to help users quickly more quickly grasp the concept of grouping—i.e., that the actions in a group can be performed in any order but that they all have to occur before the next step in the sequence. However, the subjects also expressed a desire to be able to group actions for other reasons—for example, to represent subtasks or a set of related optional tasks.[2]

4.2.2 The annotations in ESE provide good coverage of desired generalizations

Subjects were able to use the existing annotations to make most of the generalizations they wanted although they also expressed the desire to be able to specify additional, more sophisticated generalizations. The annotations made by the subjects over the three solution traces each are summarized in Table 2. The five subjects made an average of 16.53 parameter annotations and 5.87 action annotations per recipe, for a total of 336 annotations overall. Subjects varied in the number of annotations they made, with P2 making the fewest annotations at 36 total, and P3 making the most at 101 total. The predominant parameter annotation was allowing values beyond the one demonstrated (*Any of values*); there were 172 such annotations overall. The predominant action annotation was marking a step as optional; there were 36 such annotations overall. For the grouping (*In any order*) annotation, the individual groups contained 2–4 steps each, with an average of 2.67 steps.

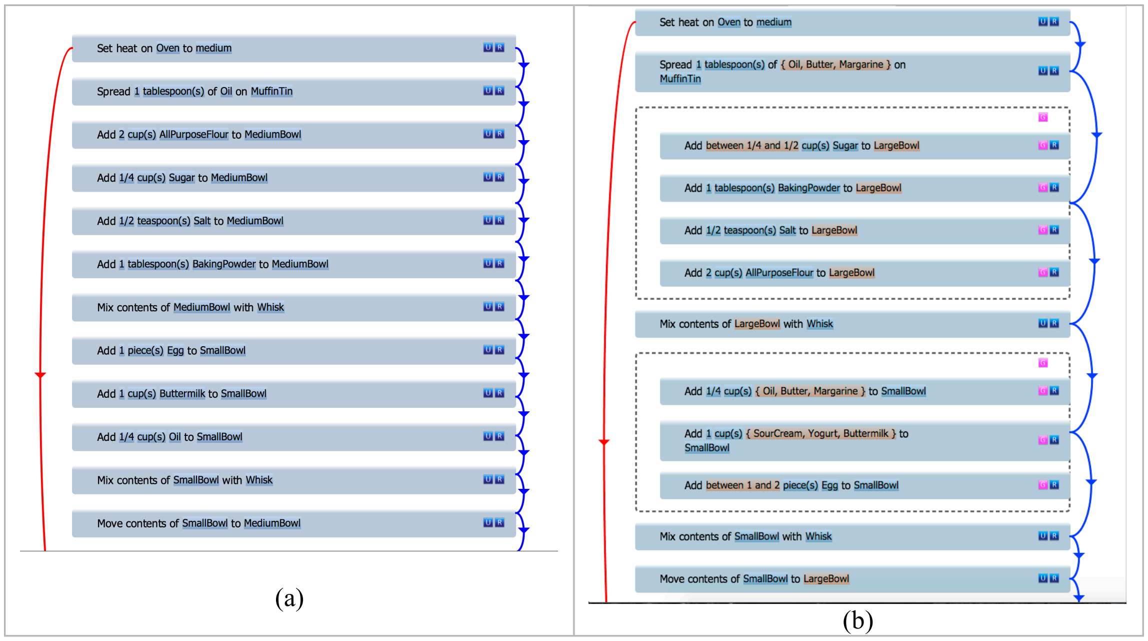
[2] An initial version of the ESE supported a notion of grouping of steps independent of ordering constraints. We removed that functionality for the user study out of concern that having two types of groupings might confuse users. In retrospect, retaining that functionality would have been desirable.

Table 2. Summary of annotations made by subjects over three solution traces each.

Subject	#Param annot.	*Any of Values*	*Any of Type*	*Diff value*	#Action annot.	*Opt'l*	*Diff order*	*Any order*
P1	**51**	21	25	5	**16**	9	2	5
P2	**23**	10	3	10	**13**	5	3	5
P3	**80**	70	10	0	**21**	7	7	7
P4	**36**	25	4	7	**26**	7	14	5
P5	**58**	46	12	0	**12**	8	2	2
Totals	**248**	**172**	**54**	**22**	**88**	**36**	**28**	**24**

An example of an original trace and an annotated version for a muffin recipe are shown Figure 4. Here, P2 has relaxed the order in which the dry ingredients (sugar, baking powder, salt, and flour) are added to the large bowl; and similarly for the wet ingredients. P2 has also generalized the type of fat and the type of dairy to use as well as the amount of sugar and eggs.

The recipes naturally provided some expected generalizations (e.g., "1/4 teaspoon pepper, optional,", "peanut butter or almond butter," "choice of cheese"). However, based on the results from a pilot study, we decided that the subjects' think-alouds would provide a more reliable rubric for scoring their annotations against their intentions. All the subjects made all the



(a)

Set heat on Oven to medium
Spread 1 tablespoon(s) of Oil on MuffinTin
Add 2 cup(s) AllPurposeFlour to MediumBowl
Add 1/4 cup(s) Sugar to MediumBowl
Add 1/2 teaspoon(s) Salt to MediumBowl
Add 1 tablespoon(s) BakingPowder to MediumBowl
Mix contents of MediumBowl with Whisk
Add 1 piece(s) Egg to SmallBowl
Add 1 cup(s) Buttermilk to SmallBowl
Add 1/4 cup(s) Oil to SmallBowl
Mix contents of SmallBowl with Whisk
Move contents of SmallBowl to MediumBowl

(b)

Set heat on Oven to medium
Spread 1 tablespoon(s) of { Oil, Butter, Margarine } on MuffinTin
Add between 1/4 and 1/2 cup(s) Sugar to LargeBowl
Add 1 tablespoon(s) BakingPowder to LargeBowl
Add 1/2 teaspoon(s) Salt to LargeBowl
Add 2 cup(s) AllPurposeFlour to LargeBowl
Mix contents of LargeBowl with Whisk
Add 1/4 cup(s) { Oil, Butter, Margarine } to SmallBowl
Add 1 cup(s) { SourCream, Yogurt, Buttermilk } to SmallBowl
Add between 1 and 2 piece(s) Egg to SmallBowl
Mix contents of SmallBowl with Whisk
Move contents of SmallBowl to LargeBowl

Figure 4. Partial trace for muffin recipe (a) and participant P2's annotated version (b).

parameter annotations and optional step annotations that they wanted to (within the limitations of the available annotations) and they did so correctly. Except for P5, who missed two action groupings she expressed in the think-aloud, the subjects were also able to make the action sequencing annotations they intended (both changing the order and grouping actions that could be performed in any order).

4.2.3 Prompting is useful

Although prompting elicited few additional annotations, all the subjects found it useful and desirable. Of the 336 total annotations made, only four were made after prompting—two by P1 (for one recipe) and two by P5 (one each for two recipes). However, we observed that prompting made all the subjects pause to think about whether they had made all the generalizations in the checklist (P1: "I examined everything…. I grouped everything…. I did [check all the optional steps]."). All but P4 then reviewed the solution models for at least one of their cooking tasks.

When asked during the open discussion whether they found the prompting useful, all expressed the strong opinion that it was definitely useful. This was somewhat surprising given that most of the subjects seemed to just check off the items on the checklist, particularly with the second and third recipes. Based on the subjects' responses, there seemed to be a consensus that checklists are useful in general, particularly when one is distracted or the resulting product is important. We also observed that subjects seemed to learn from seeing the first checklist, appearing to become more conscious about the scope of the generalizations they needed to make for subsequent tasks.

4.2.4 Subjects found the ESE to be highly usable

The usability survey revealed high perceived usability of the ESE (86.50 on the *System Usability Scale* (Brooke, 1996)).[3] In particular, subjects found the ESE easy to learn and use, and felt confident in using the system. Four of the five subjects agreed or strongly agreed that they would like to use the ESE for authoring solution models and no one felt they would need technical support or a lot of training to use the ESE.

The subjects did express a desire for some generalizations that would require more significant extensions and design work—for example, specifying groups within groups, conditional optionality (e.g., when adding an optional ingredient, the succeeding Mix step should only optional if adding the ingredient is omitted), dependencies between allowable values (e.g., an appropriate bowl size depends on the amount used), and state-driven conditions for action execution (e.g., cook "until the cheese is bubbling"). In spite of these limitations, the subjects were all able to create solution models that described a reasonable space of acceptable solutions for each of the cooking tasks and that they were satisfied with.

When asked about procedural tasks they performed as part of their work or were otherwise aware of for which they could imagine creating solution models for training through the annotation process embodied in the ESE, the subjects had no trouble coming up with numerous examples, including running simulation experiments, safety/security procedures, payroll processing, and proposal preparation.

[3] SUS scores range from 0 to 100, with average scores around 65–69 and scores below 60 considered low.

5. Related Work

The use of end-user programming techniques for authoring ITS content originated with the work of Blessing (Blessing, 1997), who applied it to learn production rules for a cognitive tutoring system, in contrast to our focus on exercise solutions. The use of programming by demonstration in (Koedinger et al., 2004) to develop content for example-tracing tutors requires explicit demonstration of all possible solution paths; in contrast, our approach supports generalization through annotation. The work in (Mohammed et al., 2005), which is more closely aligned with the approach considered in our study, describes an authoring framework based on demonstration plus annotation that was built to support an ITS for satellite management.

The Steve system teaches hierarchical procedures for operating electro-mechanical devices (Rickel & Johnson, 1999). Its domain models combine precondition/effects models to characterize behaviors of individual actions and hierarchical task networks (HTNs) to describe solution processes built on those actions. Demonstrations are used for learning HTNs, supplemented by queries to the demonstrator to determine required effects (Angros et al., 2002).

6. Conclusions

The results of our user study provide initial evidence that our demonstration-based solution authoring process is a practical, effective, and viable approach for enabling domain experts to create solution models for automated assessment. We believe that there are several additional directions to take this work to provide a comprehensive solution authoring capability.

One is to extend the simple prompting mechanism to a more sophisticated capability aimed at eliciting more problematic annotations. Participants in our preliminary study expressed concerns about being overwhelmed with prompts, hence prompting should be used carefully. A combination of logical and heuristic reasoning could reduce the need for exhaustive prompting. For example, a prompting mechanism could build on causal reasoning techniques from the automated planning community to identify those relationships that are not crucial to maintaining the causal coherence of the action sequence and hence are good candidates for relaxation.

Another interesting direction is to integrate more tightly the authoring with assessment and demonstration, to enable content authors to interactively ‘try out’ and refine solution models. In creating a model, an author could run assessments over sample learner traces to see what kinds of assessment feedback gets generated and refine models accordingly, including providing additional demonstrations. Ideally, the editor would also support a broader complement of functions, such as being able to add/delete steps directly within a trace or to reuse models from other exercises rather than having to re-demonstrate from scratch to perform such modifications.

Acknowledgments

This material is based upon work supported by the United States Government under Contract No. W911QY-14-C-0023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Government. The authors thank Chris Greuel and Chris Jones for their activities in support of this work.

References

Aleven, V., McLaren, B., Sewall, J. & Koedinger, K. (2009). A new paradigm for intelligent tutoring systems: example-tracing tutors. *Intl. J. of AI in Education*, 19(2).

Angros Jr., R., Johnson, W.L., Rickel, J. & Scholer, A. (2002) Learning domain knowledge for teaching procedural skills. *Proc. of the 1st Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Brawner, K., Holden, H., Goldberg, B., & Sottilare, R. (2012). Recommendations for modern tools to author tutoring systems. *Proc. of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC).*

Blessing, S. (1997). A programming by demonstration authoring tool for model-tracing tutors. *Intl. Journal of AI in Education*, 8, 233–261.

Brooke, J. (1996). SUS — A quick and dirty usability scale. *Usability Evaluation in Industry*, *189*(194), 4–7.

Gervasio, M., Jones, C., & Myers, K. (2017). Approximate graph matching for mistake-tolerant skill assessment. *Proc. of the 5th Annual Conf. on Advances in Cognitive Systems*.

Green, A., Cohen, M., Kim, J. & Gray, J. (2012). An explicit cue improves creative analogical reasoning. *Intelligence*, 40, 598-603.

Greuel, C., Myers, K., Denker, G., & Gervasio, M. (2016). Assessment and content authoring in semantic virtual environments. *Proc. of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC).*

Koedinger, K.R., Aleven, V., Hefferman, N., McLaren, B., & Hockenberry, M. (2004). Opening the door to non-programmers: authoring intelligent tutor behavior by demonstration. *Proc. of 7th Annual Intelligent Tutoring Systems Conference*.

Mohammed, J., Sorensen, B., Ong, J. & Li, J. (2005). Rapid authoring of task knowledge for training and performance support. *Proc. of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC).*

Myers, K. & Gervasio, M. (2016). Solution authoring via demonstration and annotation: An empirical study. *Proc. of the 16th IEEE Intl. Conf. on Advanced Learning Technologies*.

Myers, K. Gervasio, M., Jones, C., & Keifer, K. (2013). Drill evaluation for training procedural skills. *Proc. of the 16th Intl. Conf. on AI in Education*.

Rickel, J. & Johnson, W. L. (1999). Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, *13*.

Schroeders, U. & Wilhem, O. (2011). Computer usage questionnaire: structure, correlates, and gender differences. *Computers in Human Behavior, 27*, 899–904.