*In Proceedings of the 16th IEEE International Conference on Advanced Learning Technologies (*ICALT2016*)*

# Solution Authoring via Demonstration and Annotation: An Empirical Study

Karen Myers and Melinda Gervasio
Artificial Intelligence Center, SRI International
Menlo Park, CA
{karen.myers, melinda.gervasio}@sri.com

***Abstract*—A major impediment to the widespread deployment of intelligent training systems is the high cost of developing the content that drives their operation. Techniques grounded in end-user programming have shown great promise for reducing the burden of content creation. With these approaches, a domain expert demonstrates a solution to a task, which is then generalized to a broader model. This paper reports on a concept validation study that provides an empirical basis for the design of solution authoring frameworks based on end-user programming techniques. The study shows that non-expert users are comfortable with the approach and are capable of applying it to generate quality solution models. It also identifies constructs that, while important for accurate solution characterization, can lead to confusion and so warrant special care in tool design. Based on these results, we make recommendations for the design of solution-authoring tools in support of automated assessment for tutoring systems.**

***Keywords: automated assessment, end user programming, programming by demonstration, intelligent training systems***

## I. Introduction

Intelligent tutoring/training systems (ITS) have shown great potential for enabling self-directed learning. Organizations are intrigued by the potential of ITS to reduce training costs and to improve learning effectiveness. Individuals are further motivated by the promise of learning experiences adapted to their skill levels and learning styles.

Many training domains 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 [1]. Example-tracing tutors identify differences between a student's actions (recorded in an instrumented environment) and a predefined solution for completing a task. These differences can be used both to automatically assess student performance and to provide targeted feedback. Several training systems based on example tracing have been built, for domains that include collaborative visualization and planning [2], management of complex devices [3], and satellite operations [4].

At present, a significant impediment to the widespread use of ITS technologies is the substantial effort required to develop the content that drives their operation. Others have recognized the need for effective authoring frameworks to alleviate the burdensome cost of content development for ITS applications [5]. One of the appeals of example tracing is the potential for end-user programming techniques (such as programming by demonstration) to facilitate model acquisition. In particular, authoring is based on demonstrating solutions, augmented by mechanisms for generalizing from demonstrations to a comprehensive solution model. With this 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 [6] 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.

Our work seeks to provide an empirical basis for the design of content-authoring frameworks for example-tracing systems, with particular focus on support for automated assessment. This paper summarizes a concept validation study we conducted to determine whether domain experts (rather than ITS experts) can apply this approach and to investigate specific constructs for generalizing traces to broader solution models for training exercises. The results of the study are two-fold. First, it shows that domain experts are both comfortable with the approach and capable of applying it to generate quality solution models. Second, it identifies constructs that can lead to potential confusion and so warrant special care. Based on these results, we make recommendations for the design of solution-authoring tools for example-tracing assessment systems.

The paper is organized as follows. Section II presents a representation for solution models and a methodology for applying them, derived from our previous work on the Drill Evaluation for Training (DEFT) system [2]. Section III describes the design of our study and summarizes findings and implications. Section IV considers related work. Section V presents concluding remarks.

## II. Background: Automated Assessment in DEFT

### *A. DEFT overview*

DEFT is an automated assessment capability that, while domain independent, was developed originally to assist 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 [2]. As a student works on a training exercise in CPOF, DEFT logs a semantic trace of the student's actions. This trace is compared to a predefined exercise solution model to create assessment information with contextually relevant feedback: identifying conceptual errors or mistakes, providing guidance in the form of hints to help the student complete a task, and suggesting links to relevant training materials.

The exercise solution models are defined in terms of one or more *generalized action traces,* each consisting of (1) a sequence of *steps*, and (2) *annotations* that specify allowed variations. A step can be a parameterized action, a class of actions, or a set of options, each of which is itself a partially ordered set of steps. Annotations can be defined over individual steps (e.g., optionality of a step), sets of steps (e.g., step ordering constraints), and step parameters (e.g., a parameter must have a specific value or satisfy a property). Annotations support action ordering and grouping constraints; parameter type, value, and equality constraints; and state constraints, which capture requirements on the application state or on object properties that cannot be determined from actions themselves. These abstractions enable compact representations of large solution spaces.

The automated assessment capability in DEFT determines a mapping from the student's response to the predefined 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. The intuition is that the lowest-cost alignment corresponds to the specific solution the student is most likely attempting.

To use this graph matching approach in DEFT, the exercise solution model is represented as one or more graphs, each representing a family of possible solutions. Within the graphs actions and their parameters are nodes, 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 alignment allows DEFT to generate an assessment 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. In contrast to other example-tracing assessment capabilities, DEFT does not force a user 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.

## B. *Solution Authoring Methodology*

The training tasks targeted by DEFT can have numerous solutions that involve different actions and orderings between them, along with significant variability in the objects that are created or manipulated. This variability precludes an explicit enumeration of complete solutions—an approach that works for more constrained tasks.

The scope and complexity of the DEFT solution models motivated us to explore the use of programming by demonstration to facilitate their acquisition. This approach is in line with recommendations made by others that models should be created from data and interactions, to the extent possible, rather than handcrafted [5].

Our user study builds on the following process for solution authoring. Given the initial exercise setup, the content author demonstrates one or more solutions to the problem. The author then specifies annotations on the demonstration(s)—effectively adding, removing, or relaxing constraints—to define the space of acceptable solutions. While pedagogical information (e.g., learning hints) would be an important part of a comprehensive model, our focus here is on accurate characterization of the space of correct solutions.

Prior work on deploying end-user programming technology showed that most users are inclined to provide only one or two demonstrations for any particular task [7]. This finding motivates the use of an expressive set of annotations within a solution-authoring framework so that authors can specify generalizations directly that would otherwise require numerous demonstrations for them to be learned automatically.

## III. User Study

To inform the design of a solution-authoring tool grounded in demonstrations plus annotations, we conducted a user study to address the following questions:

1. *Can subjects understand a demonstration trace?*
2. *Can subjects understand annotations?*
3. *Can subjects apply these annotations appropriately?*
4. *How would subjects like to specify annotations?*

## A. *Process*

We conducted a two-on-one (facilitator + note-taker vs. subject) paper prototype study requiring approximately two hours for each subject. Because our target audience for the authoring tool is domain experts, it was important to select a domain in which subjects would have expert-level knowledge. We chose cooking as our domain, given that many people are experts in how they like to prepare their food, even if their preparations may be highly personalized. Cooking also shares several structural similarities with candidate application domains for DEFT:

- Solutions have similar complexity and length;
- Actions often need to be done in the order they are demonstrated, although with some variation;
- Alternative and optional actions exist;
- Some actions may be preferred to others.

Each subject was asked to demonstrate how to prepare two simple dishes and then to document how the demonstrations could be generalized while still remaining valid preparations for those dishes. The facilitator then led an open-ended discussion with the subject on what was easy or hard about

TABLE I. PREDEFINED ANNOTATIONS

| | Annotation | Example |
|---|---|---|
| Actions | **Group** | Prep steps; Cooking steps |
| | **In any order** | Chop onions and red pepper in any order |
| | **Optional** | Add cheese, if desired |
| | **Alternative** | Microwave rather than heat on stove |
| Parameters | **Any of** | Use any of butter, margarine, or oil |
| | **Range** | Use 4-6 eggs |
| | **Same** | Use the same pan |
| Either | **Preferred** | Prefer butter to margarine *(parameters)*<br>Prefer doing Prep steps before Preheat *(actions)* |

the tasks and on preferences over design features for a solution-authoring tool.

The four study subjects were drawn from our company's employee pool. They were not compensated for their participation. A one-page questionnaire on computer use showed that all subjects were comfortable or very comfortable with computers and used them for a variety of tasks, such as word processing, spreadsheets, web browsing, and games. Three had taken a programming class; two had minimal programming experience in the distant past.

## B. *Summary of Protocol*

Each subject was asked to describe how to prepare each dish in two phases. In the first phase, the subject specified the cooking process through a paper simulation. This involved selecting objects, such as a stove, a bowl, or ingredients, and then describing what to do with them. The facilitator documented these steps manually to create a trace of the subject's actions. The subject was free to correct mistakes by asking the facilitator to remove steps from the trace, reorder steps, or modify steps.

In the second phase, the subject was asked to annotate the trace to indicate how the demonstrated actions could be generalized to a comprehensive solution model. To assist with the annotation process, the subject was provided with cards showing predefined annotation categories (summarized in Table 1), with the facilitator explaining briefly the intended use of each annotation. The subjects were also free to create their own annotations as they felt necessary. For the second dish, subjects were allowed to mix the demonstration and annotation phases, as they liked. Throughout, subjects were asked to 'think aloud' so that the rationale for their actions could be documented.

For the first task, the subjects had to prepare scrambled eggs, a dish all the subjects felt significant expertise in preparing. For the second dish, subjects were free to select a dish of their own provided it did not require exotic ingredients, equipment, or techniques; it was of moderate complexity (i.e., it could be completed by someone with minimal cooking background in less than 20 minutes); and it supported some variability, such as ingredient swaps and

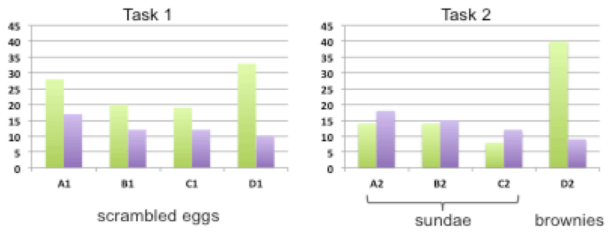


Figure 1. # of steps (green) and # of annotations (purple) by subject.

optional steps. Three subjects selected ice cream sundaes and one subject selected brownies for the second task. After each subject completed the two tasks, we conducted a semi-structured interview to gain further insight into how subjects preferred to specify annotations and to elicit feedback on preliminary design ideas.

## C. *Results*

Demonstrated solutions ranged in length from 8 to 40 steps. Figure 1 summarizes the number of steps and annotations provided by the four subjects (A through D) on a per-task basis. The number of steps included both simple actions such as "Turn the burner to medium" as well as control constructs for repetition, conditional branching, and termination (e.g., "Until the butter melts").

The ice cream sundae task had the fewest steps (8–14), although subjects provided more annotations for it than the more complex scrambled eggs or brownies tasks. Much of this was due to subject recognition that there was greater flexibility in solutions for creating an ice cream sundae: more optional steps (e.g., whether to add cherries or nuts) and more alternatives (e.g., vanilla vs. chocolate ice cream).

Figure 2 summarizes the average number of annotations per task, organized by category. The most frequently used annotations were *Optional* and *Alternative*. However, subjects consistently applied them to ingredients and cooking implements even though they were told that these applied to actions. Figure 2 splits the *Preferred* annotation into their two applications by the subjects: for ranking action orderings and for ranking the objects used in an action. It also includes a *Relation* category, which was introduced by one subject to indicate relationships between different objects (e.g., the amount of ice cream depends on the size of the bowl).

### 1) *Understanding Traces*

Our general design for a solution-authoring tool assumes that content developers will be able to understand computer-generated traces of their demonstrations and annotate those traces to show suitable generalizations. The study showed that subjects had no trouble understanding the traces of their actions that were created manually during the demonstration. This understanding was robust to semantic generalizations of verbalizations of user actions (for example, the user saying "Whip the eggs" while the trace contains "Mix the eggs with a whisk"). Subjects also found it natural to have the actions presented in the order they were demonstrated.

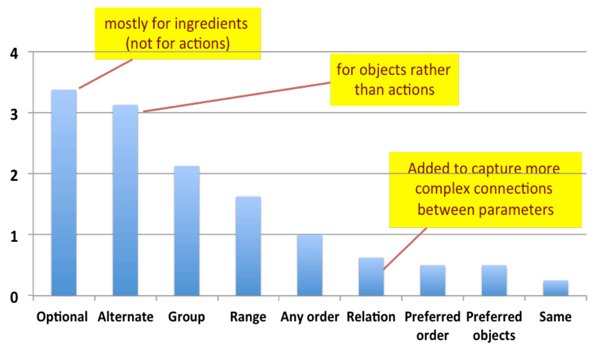
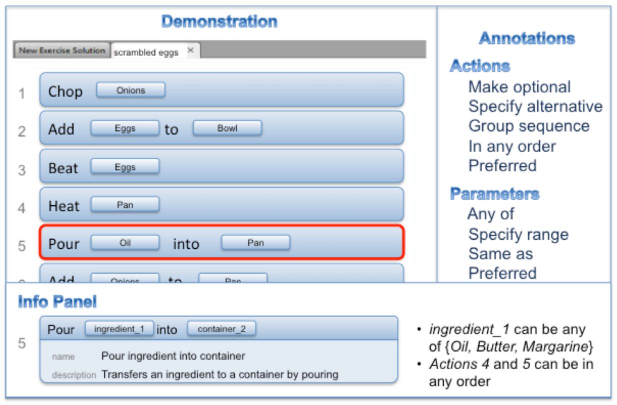


Figure 2. Average number of annotations per task by category.



- *ingredient_1* can be any of {*Oil, Butter, Margarine*}
- *Actions 4* and *5* can be in any order

Figure 3. UI mockup for a solution authoring tool.

### 2) *Specifying Annotations*

With minimal instruction, subjects were able to correctly specify annotations to generalize their demonstrations using our predefined set of annotations (Table 1). However, some of the intended distinctions between the different annotations were not well understood. In particular, *Alternative* and *Optional* were intended for generalizing actions but subjects used them almost exclusively, and also interchangeably, to generalize ingredients and implements instead.

Subjects were capable of correctly generalizing parameters. In specifying *Alternative* values, they sometimes gave explicit enumerations ("bananas, strawberries, cherries") but other times specified a category (e.g., "fruit", "nuts", "any type of cheese"). Subjects also provided good coverage of *Alternative* ingredients (vanilla vs. chocolate) but were less thorough with identifying alternative objects (spoon vs. whisk).

The *Same* annotation, intended to let subjects specify a constraint that the same object had to be used in multiple actions, was not well understood. Attempts to use it seemed motivated more by a sense of obligation rather than conviction that it was needed. Based on this experience, we believe that it would be better to have such constraints generated automatically via heuristics for a given domain. For example, in cooking, use of the same pan is usually intentional. Users could relax this constraint, as required.

Subjects often missed relaxing ordering constraints (*Any Order*) but responded appropriately when prompted by questions such as whether a later action could be performed before an earlier one. Thus, while subjects were capable of relaxing ordering constraints, this suggests the need for some type of assistance to help improve coverage.

Subjects showed a natural tendency to group actions by subtasks. Groups were sometimes used to collect and label related actions (e.g., "prep steps"). In other cases, they were used to aid in showing relaxed ordering constraints (e.g., one group of actions could be performed before or after another). Subjects readily grasped the concept of groups as an abstraction, which may facilitate the eventual use of 'macro' type capabilities in the solution specification process.

### 3) *Tool Design*

The semi-structured interview to elicit subjects' feedback on design ideas was facilitated through the presentation of a mockup of a candidate UI (Figure 3). Subjects showed significant variation in how they preferred to specify solutions. Some wanted explicit access to annotation tools while others preferred those to be available through menus, leaving more space within the editor to visualize solution elements. Some wanted a 'drill down' capability to see details of annotations for individual actions; others preferred to have a summary view that showed all annotations at once. Some annotations (e.g., alternative values) seemed to come more naturally during demonstration for some subjects; for others, after demonstration (e.g., specifying optionality or order relaxation).

There was strong support from all subjects for layering visual cues for annotations directly on the action trace. All subjects liked the idea of a prompting mechanism to help them identify missing annotations, provided that the prompting would not be 'annoying'. Furthermore, the prompting was expected to be smart enough not to ask questions with obvious answers. Interestingly, two subjects expected to learn something from the prompting. Prompting was viewed as much preferred to any kind of automated mechanism that would add annotations or make changes on its own; subjects felt strongly that they needed to validate any annotations that the system might suggest.

## D. *Design Implications*

The results of the study suggest several directions for the design of an authoring tool for exercise solution models.

*Provide available annotations*. The annotation cards served both as reminders to users about the available annotations and as cues to think through the different ways to generalize or relax their solutions. All participants repeatedly referred to the annotations; one methodically sought opportunities to apply each one. This suggests making the annotations prominent in an authoring tool.

*Contextualize annotations*. The study revealed that although the subjects knew what type of generalization they wished to make, they sometimes applied the "wrong" annotation to do so. Some of this confusion may be mitigated by the use of annotation labels more aligned with users' intuitions. A safer approach would involve contextualization: after an action or parameter is selected, the tool could highlight the relevant annotations and vice-versa.

*Visualize existing annotations*. The subjects understood the idea of a drill-down to provide annotation details, but all wanted visual cues of annotations directly on the action trace itself. A key challenge is to design visualizations that clearly communicate the range and number of annotations without overwhelming or confusing the user. Certain annotations have straightforward visualizations (e.g., grouping by indentation or blocking; optionality by an icon or coloring). Others will be more challenging to implement—e.g., intra- and inter-group ordering constraints. Layering of annotations would enable authors to customize views to align with their preferences and context-specific requirements.

*Prompt for annotations intelligently*. While subjects had no trouble specifying some annotations (e.g., *Group* and *Alternative* (*Any Of*)), they missed specifying others (e.g., *Any Order* and *Same*). Prompting could provide an effective mechanism for eliciting more problematic annotations. Because all participants expressed concerns about being overwhelmed with prompts, 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.

*Support a range of authoring styles*. Given the significant variation in how subjects preferred to specify solutions, an authoring tool would ideally be flexible enough to support individual preferences. Access to and interaction with the annotation tools should be configurable and users should be able to annotate both during and after demonstration, to more closely align with their natural tendencies.

## IV. Related Work

The use of end-user programming techniques for authoring ITS content originated with the work of Blessing [8], 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 [9] to develop content for example-tracing tutors requires explicit demonstration of all possible solution paths; in contrast, DEFT supports generalization through annotation. The work in [4], 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 [3]. 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 [10].

## V. Conclusions

We presented a concept validation study that provides an empirical basis for the design of authoring frameworks based on programming by demonstration. The study shows that while non-expert users are able to generate quality solution models with the approach, certain concepts present challenges for users, calling for special attention in tool design. We also provided recommendations for the design of future authoring frameworks based on our findings.

DEFT has recently been integrated into a broader intelligent training tool and content development framework for virtual environments called SAVE (Semantically enabled Assessment in Virtual Environments). Development of an authoring tool for SAVE is under way, informed by the results reported in this paper, that enables domain experts to generate, view, and modify solution models through programming-by-demonstration methods.

## Acknowledgment

This material is based upon work supported by the Advanced Distributed Learning Initiative (ADL) 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 ADL.

## References

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

[2] K. Myers, M. Gervasio, C. Jones, K. McIntyre, and K. Keifer, "Drill evaluation for training procedural skills," Proc. of the 16th Intl. Conf. on AI in Education, 2013.

[3] J. Rickel and W. L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. Applied Artificial Intelligence, vol. 13, 1999.

[4] J. Mohammed, B. Sorensen, J. Ong, and J. Li, "Rapid authoring of task knowledge for training and performance support," Proc. of I/ITSEC, 2005.

[5] K. Brawner, H. Holden, B. Goldberg, and R. Sottilare, "Recommendations for modern tools to author tutoring systems," Proc. of I/ITSEC, 2012.

[6] K. R. Koedinger, V. Aleven, N. Hefferman, B. McLaren, and M. Hockenberry, "Opening the door to non-programmers: authoring intelligent tutor behavior by demonstration," Proc. of 7th Annual Intelligent Tutoring Systems Conf., 2004.

[7] K. Myers, J. Kolojejchick, C. Angiolillo, et al., "Learning by demonstration for collaborative planning," AI Magazine, 33(2), pp.15–27, 2012.

[8] S. Blessing, "A programming by demonstration authoring tool for model-tracing tutors," Intl. Journal of AI in Education, vol. 8, pp. 233–261, 1997.

[9] K. R. Koedinger, V. Aleven, N. Heffernan, B. McLaren, and M. HockenBerry, "Opening the door to non-programmers: authoring intelligent behavior by demonstration," Proc. of the 7th Intl. Conf. on Intelligent Tutoring Systems, 2004.

[10] R. Angros Jr., W. Lewis Johnson, J. Rickel, and A. Scholer, "Learning domain knowledge for teaching procedural skills," Proc. of AAMAS, 2002.