How to Serve Soup: Interleaving Demonstration and Assisted Editing to Support Nonprogrammers

Melinda Gervasio, Will Haines, David Morley, Thomas J. Lee, C. Adam Overholtzer, Shahin Saadati, Aaron Spaulding
SRI International
333 Ravenswood Avenue, Menlo Park, California 94025
{gervasio,haines,morley,tomlee,overholtzer,saadati,spaulding}@ai.sri.com

ABSTRACT
The Adept Task Learning system is an end-user programming environment that combines programming by demonstration and direct manipulation to support customization by nonprogrammers. Previously, Adept enforced a rigid procedure-authoring workflow consisting of demonstration followed by editing. However, a series of system evaluations with end users revealed a desire for more feedback during learning and more flexibility in authoring. We present a new approach that interleaves incremental learning from demonstration and assisted editing to provide users with a more flexible procedure-authoring experience. The approach relies on maintaining a "soup" of alternative hypotheses during learning, propagating user edits through the soup, and suggesting repairs as needed. We discuss the learning and reasoning techniques that support the new approach and identify the unique interaction design challenges they raise, concluding with an evaluation plan to resolve the design challenges and complete the improved system.

Author Keywords
Programming by demonstration, learning from demonstration, end-user programming, programming by direct manipulation

ACM Classification Keywords
H.1.2 User/Machine Systems (Human factors); I.2.6 Learning (Knowledge acquisition)

General Terms
Design, Human Factors

INTRODUCTION
Computers are becoming increasingly prevalent in the lives of an ever-widening spectrum of the population. Scaffidi, Shaw, and Myers estimate that by 2012, there will be over 55 million end users who will need to customize the behavior of spreadsheets, databases, and other applications they use on a daily basis [12]. While the human-computer interaction community has made great strides in improving general software usability, it has devoted less attention to end-user programming [8]. Thus, in spite of a growing need for end users to be able to create customized programs, this ability remains mainly in the realm of professionals.

In our earlier work, we attempted to address this issue with a procedure-learning system that integrates programming by demonstration with editing through direct manipulation [13]. The procedure-authoring workflow required the user to provide a complete demonstration from which the system learned a procedure that the user could then edit as desired. While this workflow enabled end users to successfully create procedures, feedback sessions with users of both desktop office and military applications revealed the need for a more flexible approach that would allow editing to be interleaved with demonstration and provide continual feedback on what was being learned during demonstration.

We present an enhanced approach based on incremental learning from demonstration interleaved with assisted editing. We begin with a review of previous work on procedure visualization and editing, followed by the motivation behind our work. We then present our approach to incremental learning and procedure validation, which provides the infrastructure to support an improved procedure-authoring workflow. The new framework greatly expands the authoring possibilities but raises a number of new interaction design challenges. We conclude with a description of the design space implied by these challenges and a discussion of our evaluation plans to resolve them.

RELATED WORK
A fundamental difficulty in programming by demonstration is communicating the learned program or procedure to the end user. The procedures are intended to generalize from the demonstration in order to apply to a variety of situations. However, nonprogrammers often have difficulty with abstract concepts like variables and loops, making direct presentation of learned programs problematic.

One approach is to not present the procedure at all, and instead to present its instantiation within the current computing environment (e.g., [3,6]). However, as learned programs or procedures become more complex, comprehensible depictions of the learned program become more important (e.g., [5,11,2]). An advantage of presenting learned procedures to users is that it offers them the

IUI 2011, February 13–16, 2011, Palo Alto, California, USA.

opportunity to edit the programs directly (e.g., Sheepdog [7], Koala [9], Adept [13]). Adept[1] differs in its focus on assisted editing: as users make changes, the resulting procedure is validated and fixes are suggested to ensure an executable procedure. However, neither Adept nor Sheepdog supported interleaved demonstration and editing, and while Koala does support interleaving, its generalization is limited to recognizing user-specific data.

Augmentation-based learning (ABL) [10] is designed specifically to address interleaved demonstration, generalization, and editing. To deal with potential inconsistencies between demonstrations and user edits, ABL favors user edits, discarding demonstrations that no longer align with the edited procedure. While it is ultimately always the user's responsibility to ensure that the correct program is learned, our work differs in its focus on end users with little or no programming experience who require more explicit assistance during editing.

MOTIVATION

The Adept Task Learning system was originally developed as part of the CALO project [1] to help busy office workers by automating repetitive tasks on their computers. The technology has since been transitioned into a number of other systems, including the U.S. Army's Command Post of the Future (CPOF), where Adept is used to automate time-consuming procedures in military command and control [4]. CPOF is a highly collaborative environment where procedures are shared, so users must be able to understand procedures created by others. CPOF users also typically have no programming experience, so assisted editing is important to ensure that they do not inadvertently break procedures [13]. Because CPOF procedures can be long and complex, Adept can learn from single demonstrations, relying on heuristics to select the best generalization and on user editing for further refinement.

Over the course of several sessions with end users to evaluate the procedure-learning technology, certain limitations of the demonstrate-then-edit paradigm of teaching procedures became apparent. Because users could see the learned procedure only after they completed their demonstration, they often did not catch demonstration errors or ambiguities as they occurred. When users did realize that they had demonstrated something incorrectly, they often wanted to demonstrate a small correction but had no recourse except to redemonstrate the entire procedure. Also, sometimes actions necessary for the proper execution of a program were not conducive to demonstration—for example, actions involving the user making a decision before execution can continue. Finally, loops were often problematic: for a loop to be learned, all iterations had to be demonstrated and each iteration had to consist of the same sequence of actions.

[1] Previously ITL (Integrated Task Learning).

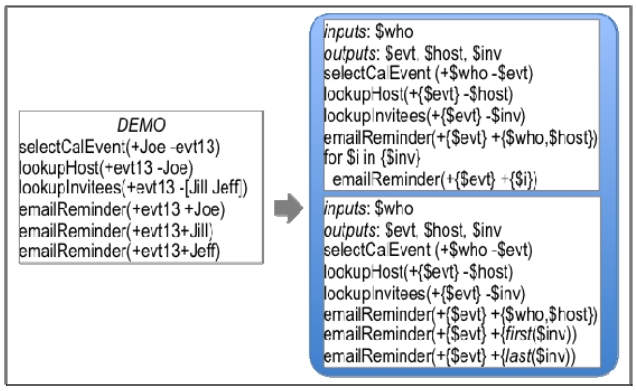

Figure 1. Learning generates a soup of alternative hypotheses. (+ *denotes input, - output, $ variable, and {} alternative supports*)

INTERLEAVED DEMONSTRATION AND EDITING

To address these issues, we began to develop a more robust, flexible approach to procedure authoring. The approach relies on incremental learning to provide continual updates about the procedure being learned. It accommodates user edits on partially learned procedures, employing structural reasoning techniques as before to ensure that user edits do not break procedures and to suggest fixes if they do. The approach also accommodates adding nondemonstrable actions into procedures during demonstration, allowing learning of a wider class of procedures. Finally, it supports proactive looping assistance to facilitate demonstration. Space constraints preclude more formal presentation of the algorithms involved, so we focus discussion on key ideas.

Incremental Learning from Demonstration

Under the new authoring paradigm, after each demonstrated action, Adept outputs not just a single procedure, but a *soup* of hypotheses comprising all possible structure and parameter generalizations consistent with the demonstration thus far. The possible structure generalizations of the demonstration are the different looping structures over collections that can be induced from the demonstration trace. For brevity, we refer to these alternative structures simply as *chunks*. Adept learns *dataflow* procedures, where action outputs support inputs of succeeding actions. Thus, the possible parameter generalizations of an input argument are output variables of previous actions (or expressions over those variables). Each chunk encodes the set of all possible parameter generalizations of all action inputs for a specific structure generalization. Because demonstrations consist of actions, the soup encodes action sequences directly, in contrast to the version spaces of [6], which encode procedural knowledge as functional state-to-state mappings. Figure 1 provides a simple example of the soup generated for a notional demonstration.

The sets of alternative parameter generalizations are independent: choosing a particular generalization for one input has no effect on the alternatives for any other input. Similarly, since each chunk represents a complete structural generalization of the demonstration, choosing one has no effect on the others. Using the same selection criteria

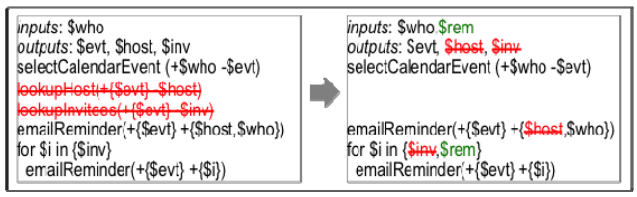

Figure 2. Deleting steps may invalidate subsequent supports, requiring new supports to be selected or created.

originally used to select a single best generalization [13], we can identify a preferred generalization and sort candidate generalizations for presentation to the user.

The primary value of the incremental approach is the continual feedback it provides users regarding what is being learned. By seeing the demonstration trace along with the current generalization hypotheses, users can detect problems and take corrective action sooner (e.g., delete extraneous steps, reorder actions). The incremental approach can also detect loops over collections as early as the first action on an element, providing the opportunity for proactive looping assistance. Absent intervening user edits, learning can resume from the previous soup and the incremental approach incurs no significant additional cost.

Procedure Validation for Assisted Editing

Previously in Adept, users could edit a single procedure at a time. The soup of alternatives generated by learning raises the possibility of editing multiple hypotheses simultaneously. However, because users must understand enough of the procedure to be able to initiate edits, we expect users to edit only one chunk (representing the set of alternative parameter generalizations of a single procedure structure) at a time.

Some edits may simply involve the selection of alternative generalizations from the soup, such as choosing a loop over a linear sequence (i.e., a different chunk) or choosing a different support for an input (i.e., a different parameter generalization). Other edits can be performed locally without adversely impacting the rest of the chunk. For example, an action output may be designated as a procedure output and a variable may be given a new, globally unique name.

Many edits, however, require changes to be propagated through the rest of the chunk. A potential support for an input disappears when the action generating the support is deleted. If the deleted support was the preferred one, the next preferred support must be identified. If all alternative supports for an action's inputs are removed, the action can no longer be executed and the procedure is invalidated. To provide the required support, an action must be inserted, a new procedure input provided, or the input turned into a constant. Figure 2 illustrates some of these situations. The outputs of inserted actions and new procedure inputs also introduce new, potentially preferred, supports for subsequent actions in the procedure. As in ABL, user edits are preferred over the native selection criteria so user selections are preserved if possible when the demonstration is resumed. In contrast to ABL, validity is enforced and all alternative hypotheses are retained and reasoned over.

Although the procedure validation process on a chunk is naturally more complex than on a single procedure, it involves a subset of the reasoning already performed during learning to generate the soup in the first place. Thus, we can utilize the same techniques to propagate edits, store their effects, and select from alternative generalizations.

INTERACTION DESIGN CHALLENGES

While incremental learning interleaved with assisted editing supports a richer variety of procedure-authoring workflows, the increased flexibility poses a number of interaction design challenges. We discuss these challenges together with our design and evaluation plans to address them.

Visualization

The soup of incrementally generated hypotheses presents new challenges in visualizing the learned procedures:

- *Distinguishing between demonstration traces and procedures.* Incremental learning blurs the boundary between demonstration and learning, leaving users susceptible to confusing a procedure with the concrete trace of the actions that generated it. Traces and their generalizations are inherently similar, and our previous user studies indicate that end users are often unclear about their distinction. As such, the challenge is to convey that a trace is a record of a specific demonstration, while a procedure is an abstract program that can be executed in the future. We will test a series of low-fidelity prototypes to rapidly explore the large design space for distinguishing demonstration traces from procedures.

- *Navigating the space of alternative generalizations.* We expect the primary visualization to present the current best generalization, according to Adept's selection heuristics and any user edits to that point. However, users must be able to easily see alternative structure and parameter generalizations and to choose between them. The naive approach is to let users simply cycle through the alternative chunks and to use dropdowns to select alternative parameter generalizations within a chunk. However, we anticipate that this approach will not scale well, so we will explore visualizations that align related hypotheses to allow the user to think in terms of local rather than global decisions.

- *Understanding differences between candidate generalizations*. Alternative looping structures, especially during proactive loop completion, present a particular challenge. When the user's actions thus far correspond to multiple, possibly nested, loops, users may find it difficult to understand the differences. Previous work on proactive loop completion (e.g. [3,6,11]) provides valuable design direction, such as deferring looping assistance until the second iteration, but we expect to perform a longitudinal evaluation before reaching a definitive decision.

Editing

Incremental learning and the ability to make changes that affect multiple hypotheses also raise challenges for editing:

- *Editing demonstration traces vs. editing generalized procedures*. Users can sometimes achieve the same effect by editing either the demonstration trace or the learned procedure (e.g., when deleting an extraneous action). Some edits, however, are most natural in only one modality (e.g., appending an action is more naturally thought of as continuing the demonstration than as editing the procedure). While we have intuitions about the preferred modality for some interactions, in general, it is not clear which users will prefer. It is also unclear how users expect changes in one mode to be reflected in the other. We plan to explore these aspects as part of the rapid prototyping/evaluation exercise described above. By limiting the modalities available to users and measuring quantitative and qualitative task performance metrics, we can determine the best modality for each interaction.

- *Limiting available edits during demonstration*. While certain edits on a procedure are perfectly reasonable after demonstration is complete, they can introduce ambiguity or confusion if performed during demonstration. For example, if a user inserts a step into a partially learned procedure, there is no guarantee that subsequent steps would have been executable had the action been part of the demonstration. We could handle this insert by deleting all subsequent actions and appending the new action, but this treatment is inconsistent with the behavior for inserting steps into procedures outside of demonstration. Thus, we need to explore the trade-offs between having different behaviors for the same editing action versus prohibiting certain edits during demonstration. We expect to find some of these trade-offs in the rapid prototyping exercise and will supplement these findings with user studies on a high-fidelity prototype.

SUMMARY

We presented a novel procedure-learning approach that interleaves incremental learning with assisted editing to address usability issues with our existing Adept system. The approach relies on incremental learning to continually update a soup of alternative hypotheses and it utilizes procedure-reasoning techniques to propagate user edits and suggest repairs for problems. The approach supports rich authoring interactions but raises new interaction design challenges that we plan to address with a series of user studies. Through interleaved demonstration and editing, continual feedback about the learned procedure, and improved tools for visualizing the hypothesis space, the new Adept system will provide a more natural and powerful end-user programming workflow.

ACKNOWLEDGMENTS

This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract Nos. FA8750-07-D-0185/0004 and through Intelligent Software Solutions Inc. on Contract No. FA8750-06-D-0005/0008. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of DARPA or the Air Force Research Laboratory (AFRL).

REFERENCES

1. CALO: Cognitive Assistant that Learns and Organizes, 2008. Retrieved September 7, 2010, from SRI International: http://caloproject.sri.com/
2. Chen, J. H. and Weld, D. S. 2008. Recovering from errors during programming by demonstration. *Proc. IUI-08*, 159–168.
3. Cypher, A. 1991. EAGER: programming repetitive tasks by example. *Proc. CHI-91*, 33-39.
4. Garvey, T., Gervasio, M., Lee, T., Myers, K., Angiolillo, C., Gaston, M., Knittel, J., and Kolojejchick, J. 2009. Learning by demonstration to support military planning and decision making. *Proc. IAAI-09*.
5. Halbert, D. C. 1993. SmallStar: programming by demonstration in the desktop metaphor. Chapter 5 in *Watch What I Do: Programming by Demonstration*, A. Cypher and D. C. Halbert (Eds.), MIT Press, 1993.
6. Lau, T., Domingos, P., and Weld, D. S. 2000. Version space algebra and its application to programming by demonstration. *Proc. ICML-00*.
7. Lau, T., Bergman, L., Castelli, V., and Oblinger, D. 2004. Sheepdog: learning procedures for technical support. *Proc. IUI-04*.
8. Lieberman, H., Paterno, F., Klann, M. and Wulf, V. 2006. End-user development: an emerging paradigm. In *End User Development*, H. Lieberman, F. Paterno, and V. Wulf (Eds.), Springer, 1–8.
9. Little, G., Lau, T. A., Cypher, A., Lin, J., Haber, E. M., and Kandogan, E. 2007. Koala: capture, share, automate, personalize business processes on the web. *Proc. CHI-07*.
10. Oblinger, D., Castelli, V., and Bergman, L. 2006. Augmentation-based learning. *Proc. IUI-06*.
11. Paynter, G. W. and Witten, I. H. 2001. Domain-independent programming by demonstration in existing applications. Chapter 15 in *Your Wish is My Command: Programming by Example*, H. Lieberman (Ed.) Morgan Kaufmann, 2001.
12. Scaffidi, C., Shaw, M., and Myers, B. 2005. Estimating the numbers of end users and end user programmers. *Proc. VL/HCC'05*.
13. Spaulding, A., Blythe, J., Haines, W., and Gervasio, M. 2009. From geek to sleek: integrating task learning tools to support end users in real-world applications. *Proc. IUI-09*.