From Geek to Sleek: Integrating Task Learning Tools to Support End Users in Real-World Applications

Aaron Spaulding[1], Jim Blythe[2], Will Haines[1], Melinda Gervasio[1]

[1]Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
{spaulding, haines, gervasio}@ai.sri.com

[2]USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
blythe@isi.edu

ABSTRACT

Numerous techniques exist to help users automate repetitive tasks; however, none of these methods fully support end-user creation, use, and modification of the learned tasks. We present an integrated task learning system (ITL) that learns executable procedures based on user demonstration and instruction, constituting a first step toward a broader solution for procedure management. We discuss our deployment of ITL into a collaborative command-and-control system. In this complex domain, ITL's performance with end users doing real tasks indicates that providing multiple, integrated learning techniques both extends functionality and improves user experience. Our experience in integrating this system also provides key insights for future designs of domain-independent task learning systems, specifically in supporting users' ability to understand and edit lengthy procedures.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces, I.2.6 [Artificial Intelligence]: Learning – knowledge acquisition.

General terms: Design, Human Factors, Algorithms.

Keywords: Interaction design, task learning, programming by demonstration, end user programming, reasoning about actions.

INTRODUCTION

Despite the productivity gains, increases in efficiency, and workflow improvements enabled by modern computing hardware and software [7], today's information workers still must often perform repetitive tasks. This might be because a general procedure varies slightly with every execution or involves several separate applications, or simply because the user lacks the knowledge or time necessary to create a custom program to automate the task [13]. Various tools and techniques—including macro recorders, programming by demonstration, learning by instruction, and visual programming—have been developed to help users create and use automatic procedures. However, none of these adjuncts alone can fully support end user management of the entire life cycle of procedure creation, use, modification, organization, and sharing. Here we present an integrated task learning system (ITL) that learns executable procedures based on user demonstration and instruction, and that constitutes a first step toward a broader solution for procedure management. ITL provides a framework that can integrate additional task learning technologies and that can be extended to different application domains.

IUI'09, February 8–11, 2009, Sanibel Island, Florida, USA.

RELATED WORK

Macro recorders enable users to record a series of actions as a quick means of defining a rule that dictates how a certain input, maps to a desired output sequence. However, these macro recorders are notoriously brittle [11], requiring a precise system state and specific input to run successfully.

Programming by demonstration (PBD) [1,4,6,11] improves on macro learning by generalizing the recorded action sequence to extend its applicability. PBD systems observe a user executing the task to be learned one or more times and then induce a general procedure to accomplish the task. Early PBD approaches utilized strong background knowledge and some simple user annotations, such as indicating the start and end of an iteration, but relied primarily on the demonstrations for learning.

In task *learning by instruction*, a user specifies a procedural step or modification by entering a semi-formal text instruction (for example, "only buy the laptop if it costs less than $1000"). An early example of this method is Instructo-SOAR [10], which operationalized instructions as new SOAR productions. Compared to programming by demonstration, learning by instruction requires more user effort but can lead to faster procedure learning, because a condition and its threshold value need no delineating through examples.

A *visual programming* tool enables users to create programs via manipulation of graphical representations of program elements rather than through keying in code [3]. However, these applications are still programming tools

that leave the burden of designing and defining procedures completely with the user.

INTRODUCING ITL

The ITL system integrates independently developed learning components, harnessing them to enable task learning in a wide range of real-world client applications, such as the Firefox web browser and the Thunderbird email client, as well as applications specific to the military domain. Rather than working toward incremental improvements of isolated components, our primary focus is to extend utility and improve usability via a system that combines programming by demonstration and learning by instruction.

Our initial target community is users of CPOF [5], a collaborative command-and-control system presently used by the U.S. military to plan for and run military operations. ITL provides these users with the ability to demonstrate frequent or tedious procedures, edit them as needed, then call the procedures for execution as appropriate. As part of our design effort, we conducted interviews with representative users and observed as they used various versions of our prototypes to create procedures to automate key aspects of their work. During this iterative process, users created dozens of procedures using the ITL enhanced CPOF system.

LEARNING FROM DEMONSTRATION WITH LAPDOG

ITL provides a programming-by-demonstration (PBD) capability through its LAPDOG module, which learns straightline or iterative procedures from one or more examples [8]. In LAPDOG, a *demonstration* consists of a sequence of actions depicting some user procedure and capturing the flow of data from action outputs to inputs of succeeding actions. LAPDOG performs parameter generalization to replace constant arguments with variables that capture the support relationships between arguments. LAPDOG also performs structural generalization, inducing looping structures over sets and lists based on demonstrations of the complete loops.

Making Procedures Understandable

A correct understanding of what a procedure can do lets users assess the procedure's accuracy and facilitates the use and modification of the procedure. In early versions of the ITL interface, we displayed procedures that were still very similar to their native SPARK-L [12] procedural representation. This format was immediately derided as "a bunch of geek [stuff]" by our users, and we quickly realized that significant transformations would be necessary to make the procedures more presentable. Our solution, as illustrated in Figure 1, has two major components. First, we annotate the procedure action model with description templates for generating human-readable translations, leveraging an appropriately abstract action model, defined in terms of atomic user interactions and user-recognizable data types. For example, the template in Figure 1 indicates that the *dispenseFrame* action should display as "Dispense" plus the value of the second parameter, which in this case refers to an object in CPOF. Second, we include rules for the identifier data type, which queries the application to find a more specific representation—in this case, an icon and textual name. If the second parameter were a variable, we would instead display just the variable's name.

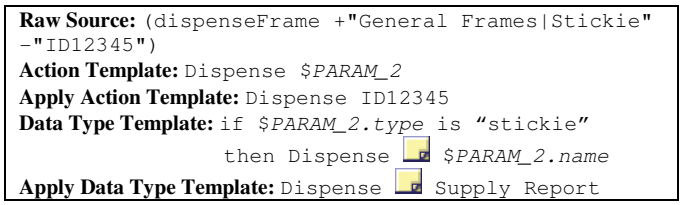

Figure 1. Action Metadata Application—Before and After

Note that this template does not present all arguments of an action to the user—in particular, the first argument of the *dispenseFrame* action is never shown. Interviews with the CPOF users revealed that some parameters simply complicate a user's understanding of the overall procedure flow. For example, though the procedure executor might need to know screen pixel positions, such information is irrelevant to most end users. So by default, we now suppress all such parameters.

Challenges in Learning Lengthy Procedures

Users were enthusiastic about the ITL's potential to automate many repetitive daily activities and were soon constructing surprisingly long procedures. Procedure durations of 30 minutes or longer were common. This highlights both the importance of capturing actions at a relatively high level of abstraction and the need to develop techniques for presenting learned procedures in an understandable manner.

The current action model captures actions at a level that makes sense to users, but the lengthy demonstrations often still result in learned procedures with hundreds of actions. To help users organize their demonstrations, ITL supports user-defined steps (subsequences of actions). We use these step annotations to impose a hierarchical structure on the procedure (Figure 4). By leveraging the user's view of the task's organization, we better represent the semantics of the procedure.

This hierarchical structure, together with the action and data type templates for translating actions into human-readable formats, greatly improved the presentation of procedures. But the sheer length of the procedures still rendered them unintelligible to most users. Through our interactions with the users, we determined that certain actions, though important to executing the procedure, are irrelevant to understanding its overall purpose. Our solution was to use templates to suppress such actions, similar to how we suppress some action parameters.

Another ramification of the long procedures was that learning from multiple demonstrations becomes unpractical, as users would likely be unwilling to demonstrate such long procedures more than once. Without the availability of additional demonstrations for refining the hypothesis space, a

learning system must employ a strong learning bias. LAPDOG currently relies on a combination of domain knowledge and control heuristics to limit the space of possible hypotheses and to select a single best generalization. This includes meta-information on action parameters indicating, for example, that a parameter is not generalizable (i.e., it should remain a constant) or that a structured argument should be treated as opaque (i.e., its individual elements should not be used to support subsequent inputs). LAPDOG also prefers more direct supports (e.g., it will prefer to support a list with another list rather than a list construction over individual elements) and shorter procedures (i.e., it will induce loops whenever possible).

In practice, this has led to successful generalization even from a single demonstration. However, the possibility that the wrong generalization is chosen—either because the heuristics do not apply or the demonstration was flawed—will always exist. Also, some procedural constructs, such as conditionals, are impossible to learn from one example. In these cases, ITL provides procedure editing capabilities, enabling users to modify procedures after they are learned.

PROCEDURE EDITING WITH TAILOR

Users often want to change their procedures, either to correct mistakes or misinterpretations during demonstration, or to reflect changes in the desired behavior. ITL provides an intuitive interface for editing procedures by direct manipulation, using Tailor [2], which was designed to support procedure editing through text instructions. ITL currently enables users to delete a step; insert a step by dragging in an existing procedure; insert steps by dragging in sub-steps from within an existing procedure; and add conditions or simple list iterations around steps.

Users were generally able to perform these simple deleting and copying edits. However, they found adding conditions and iterations to procedures considerably more difficult. Tailor's original method of interaction used text instructions to describe the desired change. Although this is a powerful method, users find it difficult to formulate their command strings such that they are properly parsed. Our solution in ITL is to enable users to request a modification via direct manipulation in the UI, (e.g., selecting steps and clicking the "Add Loop"), then ITL suggests alternative modifications in a menu generated by Tailor.

After the user selects the steps and the modification, Tailor searches for possible command interpretations. For example, when "Add Condition" is chosen, it searches for short candidate conditions based on the variables bound at that point in the procedure, using its original search method for conditions [2], which can also add new information-producing steps. The set of candidates is filtered and ordered heuristically. ITL exploits the structure of the candidates to avoid showing a large flat list (Figure 2). This interface reduces the user's cognitive burden by not requiring descriptions of new steps or queries.

Providing candidate conditions illustrates a trade-off in proactive procedure editing, where the burden is shifted when possible from the user to the editing tool. We balance providing many computed hints to the user with the responsiveness needed for smooth user interaction. For example, ITL could compute all possible conditions on all steps in a procedure when it is considered for editing, then highlight steps for which conditions or loops could be added. The time required for long procedures, however, would limit its value. We are currently exploring ways to provide useful information more quickly or pre-compute it.

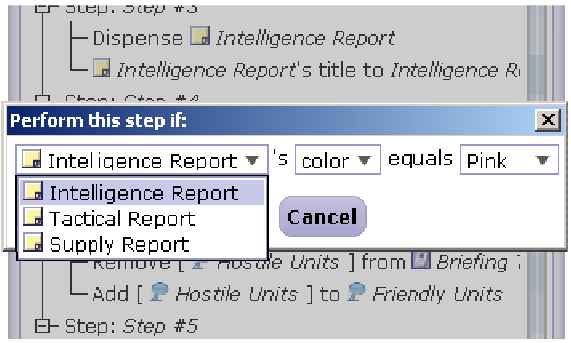

Figure 2. Tailor Provides Proactive Search Results

Managing and Presenting Errors

Tailor ensures that its suggested modifications are syntactically correct and obey type constraints but errors can still be introduced during editing. After each edit, Tailor checks the procedure for structural issues such as a variable being unbound when queried or a step whose results are no longer used. It provides a list of potential fixes, such as removing the problematic step, undoing the earlier modification that led to the problem, or using an alternative value for the missing parameter (Figure 3). If the user selects a suggested fix, Tailor computes a new procedure and re-checks for problems and suggested fixes. This approach resolves many structural errors in learned procedures.

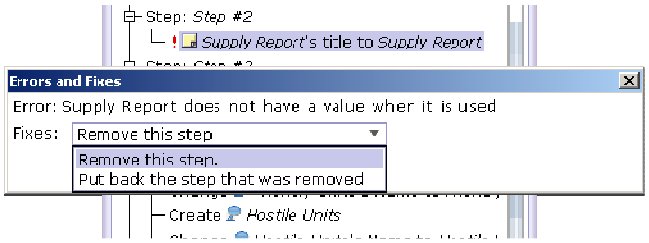

Figure 3. Edit Error. ITL's Tailor component warns of problems resulting from procedure edits and suggests remedies.

Copying steps between procedures

In interviews with users, a frequent request was the ability to copy steps from a previously learned procedure to a new one. By copying steps, users can reuse long demonstrations or complex constructs, such as conditions and loops. We were told that "plagiarism is good." The procedures learned in ITL use no global variables, so the variables in the cop-

ied steps must be replaced by terms in the target procedure, either (1) an existing variable, (2) a constant, or (3) a new variable through adding a new step. Tailor was already able to copy single steps using all three mappings, using the search methods developed to add new steps, conditions and loops. This method prefers to use an existing variable or constant for each copied variable, as this leads to a shorter solution. We extended this capability to enable copying sequences of steps, by composing the variable mappings of the component steps, and added domain-specific heuristics that replace variables with constants when the intended value is known (Figure 4).

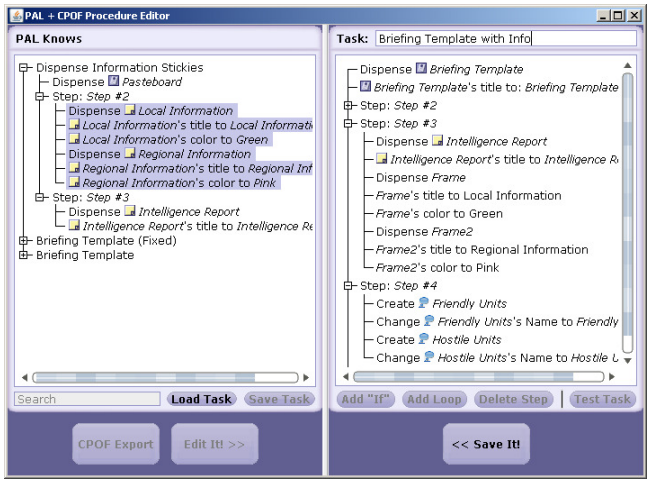

Figure 4. A New Procedure After Copying Steps. The ITL editor shows selected steps in the procedure browser (left window) and the result of the selection being pasted into Step #3 of the procedure being edited (right window). Note that some of the parameters have been generalized to correctly work in the edited procedure.

Synergy with other components

The integration of procedure learning by demonstration and editing in the same tool enables exploring interesting combinations of the capabilities. Most are future work; however, the editor already uses some information from demonstration learning. When a step is generalized, LAPDOG saves information about the initial values with which the step was demonstrated. If that step or procedure is later copied into another procedure, Tailor includes these demonstrated values as candidates when computing the mappings. This is useful when the particular values were salient or were good defaults. In addition, editing is useful in learning by demonstration to make small adjustments to procedures in which unwanted actions are unavoidable due to the application interface. Editing can also be used to add steps that are hard to demonstrate in the application UI.

INTEGRATED LEARNING WITH THE ITL FRAMEWORK

Fundamentally, ITL integrates disparate learning technologies to learn over any number of client domains. This requirement creates two major engineering challenges: 1) designing a client architecture that is general enough to support arbitrary clients yet simple enough to use for development, and 2) building a learner communications architecture that appears seamless to end users. Thus, the ITL framework serves two main functions: 1) providing a unified API to instrument and automate client applications and 2) providing unified GUI access to LAPDOG, Tailor, and the SPARK task execution engine [9].

Unified Client API

To facilitate communication with multiple clients, the ITL controller presents an API that enables clients to register a domain model, the set of domain-specific actions that their application can instrument and automate. After a client declares these actions and their parameter types, they are available for all learners to reason over. Then, the client can send instrumentation events to the controller's broadcast mechanism, making them available to the UI and learners. When learned tasks are executed, the controller notifies subscribed clients so that they can perform the automation for their registered actions.

One consequence of this client-server model is that ITL can learn procedures composed of actions from multiple domains. This capability necessitated a highly general ability to specify a domain model. Our approach enables clients to build arbitrarily complex actions and types from a limited set of universal primitives. By automatically translating these declarative structures into their primitive components, the ITL controller can transmit complex information to the learning components.

Further, ITL's intention to be a domain-independent framework necessitates an extensible display method. To this end, we tied user-interface metadata to the domain model, thereby placing all domain-specific logic in one module that can vary on a per-client basis. A major lesson learned during our deployment is that a good domain model and a good UI display both correspond to user-level atomic actions. As such, we recommend that future clients determine their actions by first observing and interviewing their users and then defining the domain model and UI annotations simultaneously.

Unified Access to Learners

ITL provides a unified user interface that lets users access multiple learners while learning a task. The ITL controller translates requests from this UI into a shared task representation that each learner can understand. Consequently, the user can demonstrate procedures and edit them within the same UI. To facilitate this integration, we represent procedures in a canonical format. Although this approach requires several translation layers, enforcing one canonical representation has enabled us to interoperate the learning systems more tightly than would otherwise be possible.

When the user is satisfied with her procedure, she can register it for execution. This registration creates a new entry in the controller's procedure repository, providing a handle for any registered client to request that the procedure be exe-

cuted. By querying the repository, clients can receive types for each procedure parameter, enabling any client to construct an appropriate UI widget for input. Likewise, by using the output types, any client can decompose procedure outputs into a format appropriate for display in their UI.

The end result of this architecture is that users can access a consistent interface to the learners when demonstrating or editing a task. By leveraging consistency between learners and within applications, we hope to make task learning easy for non-technical end users.

CONCLUSIONS AND FUTURE WORK

We have presented ITL, an extensible framework for procedure learning, and discussed its user interface and integration with LAPDOG and Tailor. The contributions of this work are (1) to demonstrate an integrated task learning system that is more powerful than the combination of its components due to their interaction; (2) to demonstrate a task learning system applied to several application tools, using a separable domain model, execution environment, and user interface; and (3) to develop a model for the user interface and interaction that enables end users to successfully understand and edit complex procedures. Initial observation of end users suggests that the unified interface and integration of these tools improves the usability of each component both by permitting repair of demonstration errors and through improved interpretation during editing based on information gathered during demonstration.

Future Work

We plan to continue collaborating with end users to further simplify procedures. Additionally, we are attempting to identify and group functionally significant sequences of steps. Currently we rely on user-entered step names to provide structure to the procedure display. By automatically grouping sequences, we hope to provide a clearer procedure overview.

In editing, we intend to cover more modifications supported by Tailor, such as step reordering and adding new steps. We will use ITL as a platform to explore deeper integration between learning by demonstration and editing (such as modifying a procedure while it is being learned or switching to demonstration to describe new steps while editing).

An alternative to editing imperfect generalizations is to improve the initial generalization by involving the user more in learning from demonstration, such as by enabling the system to ask key disambiguation questions. We are also developing new algorithms to broaden the types of procedures learnable from demonstration.

Finally, we are actively working to extend the ITL system to work with other applications, as well as integrating other learning technologies, such as PrimTL [10], which provides structured access to semi-structured data from the web, effectively representing data sources as procedures with input and output types drawn from a standardized ontology. By extending the ITL system, we hope to find more learners and domains and to ultimately bring intuitive end-user programming into the real world.

ACKNOWLEDGMENTS

This material is based on work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or the Air Force Research Laboratory (AFRL).

REFERENCES

1. Allen, J. et al. PLOW: A Collaborative Task Learning Agent. Proc. AAAI 2007.
2. Blythe, J. Task Learning by Instruction in Tailor. *Proc IUI 2005*.
3. Burnett, M. Software Engineering for Visual Programming Languages. *Handbook of Software Engineering and Knowledge Engineering*, Vol. 2, World Scientific Publishing Company, June 2001.
4. Burstein, M., Laddaga, R., McDonald, D., Benyo, B., et al. POIROT—Integrated Learning of Web Service Procedures. *Proc. AAAI-08*.
5. Command Post of the Future (CPOF) http://www.darpa.gov/sto/strategic/cpof.html
6. Cypher, A. (Ed.). *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, 1993.
7. Dedrick, J., Gurbaxani, V., and Kraemer, K. L. 2003. Information technology and economic performance: A critical review of the empirical evidence. *ACM Comput. Surv.* 35, 1 (Mar. 2003), 1-28.
8. Gervasio, M., Lee, T. J., and Eker, S. Learning Email Procedures for the Desktop. *Proc. AAAI 2008 Workshop on Enhanced Messaging*.
9. Huffman, S. and Laird, J. Flexibly Instructable Agents, *Journal of AI Research*, 3, 1995
10. Lerman, K., Plangrasopchok, A. and Knoblock, C. Semantic Labelling of Online Information Sources, *IJSWIS 2007.*
11. Lieberman, H. (Ed.). *Your Wish is My Command: Programming by Example*. Morgan Kaufmann, San Francisco, CA, 2001.
12. Morley, D. and Myers, K. The SPARK Agent Framework. *Proc AAMAS 2004*.
13. Nardi, B. A Small Matter of Programming: Perspectives on End User Computing. MIT Press, Cambridge, MA, 1993.