End-User Programming at the University of Washington

Daniel S. Weld **Pedro Domingos** **Raphael Hoffman** **Sumit Sanghai**
Department of Computer Science & Engineering
University of Washington, Box 352350
Seattle, WA 98195–2350 USA

Abstract

Over the past decade our research group at the University of Washington has investigated a number of techniques for improving end-user customization and programming. Much of this work has been reported in the AI literature, and we seek to participate in the Second Workshop on End-User Software Engineering in order to expand our understanding of existing work and alternative approaches.

1 Introduction

Starting with the Internet Softbots project [5], our research group at the University of Washington has been seeking new ways to facilitate end-user customization of their computational environment. Our work has included:

- Planning-based software agents, which synthesized and executed small programs from formal specifications [8; 7].
- The SMARTedit and SMARTpython programming-by-demonstration (PBD) systems, based on version-space algebra [10; 11].
- Relational Markov Models (RMMs), a learning method for predicting when a user may start executing a repetitious sequence of actions [1].
- Dynamic Markov Logic Networks, a statistical-relational learning engine, which improves on both version-space algebra and RMMs [17].
- The ASSIEME script recommendation engine.

End-user software engineering is especially important when programs are generated by demonstration with machine learning algorithms. Errors, debugging and visualization are important challenges for all programming environments, but are crucial when statistical or AI techniques are involved.

In the rest of this position paper we briefly describe some of our work and current directions.

2 Background

Mackay [13] studied the customization behavior of users of a Unix software environment and found that people do not take advantage of customization features, even if it made their work more efficient. The main barrier was the difficulty in making modifications, and people only did customize when something broke or they had to learn a new environment. Carroll and Rosson [3] suggest that users are biased towards making concrete, short-term progress. As a result, they are more likely to stick with known procedures than invest time learning about system features. In contrast, a survey on the use of a word processor by Page et.al. [16] showed that 92% of the participants *did* perform some form of customization. However, the authors remark that most participants were heavy users and many of the considered customizations were simple to do.

Although many people seem to be reluctant to customize their software environment, Mackay [12] and Gantt and Nardi [6] discovered that members of an organization tend to share customizations. Typically, some people experiment with the system and inform other users about useful customizations.

From this work, we draw two conclusions, which motivate our work:

- Users will customize more if it is easier to do so. We hope PBD will simplify customization.
- Users are often spurred to customize, when inspired by other users who suggest about useful customizations. Possibly the interface, itself, could make these suggestions?

3 Research at the University of Washington

Due to space constraints, we limit our discussion to two PBD systems (one powered by version-space algebra and the other by dynamic Markov logic networks) and a system for recommending relevant Web browser customizations.

3.1 Programming by Demonstration

In 1998, we started working on machine learning approaches to programming by demonstration (PBD). Of course, PBD has been studied extensively [4], but most previous systems were domain-specific. We sought a domain-independent approach suitable for deep deployment that offered the expressiveness of a scripting language and the ease of macro recording, without its accompanying brittleness.

It is useful to think of a PBD-interface as having three components: 1) *segmentation* determines when the user is executing an automatable task, 2) *trace induction* predicts what the

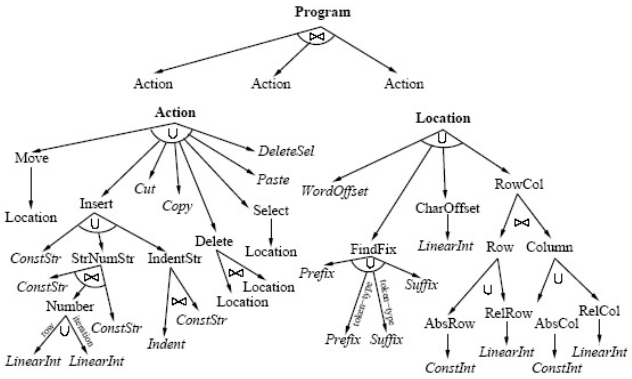

Figure 1: Composite version space for SmartEdit

user is doing from a prefix of her activity trace, and 3) *facilitation* manages user interaction to aid the user in completing her task. The next section treats segmentation in depth, but for our PBD work we assumed that the user would notify the interface when trace induction was desired, via "start" and "stop" buttons like those in a macro recorder. For the facilitation phase, we investigated decision-theoretic control [18], but many issues (*e.g.*, saving learned procedures for future use, means for convenient invokation, *etc.*) remain. The initial focus of our work was on the trace induction phase.

We formalized PBD trace induction as a learning problem as follows. A repetitive task may be solved by a program with a loop, where each iteration solves one instance of the task. The PBD system must infer the correct program from a demonstration of the first few iterations. Each action (*e.g.*, move, select, copy, paste, ...) the user performs during this demonstration causes a change in the state of the application (*e.g.*, defines a mapping between editor states). Therefore, we modeled this problem as one of inferring the function that maps one state to the next, based on observations of the state prior to and following each user action.

3.2 Version-Space Algebra

PBD presents a particularly challenging machine learning problem, because users are extremely reluctant to provide more than a few training instances. Thus the learner must be able to generalize from a very small number of iterations. Yet in order to be useful, a wide range of programs must be learnable. Thus the problem combines a weak bias with the demand for low sample complexity. Our solution, called *version-space algebra*, lets the application designer combine multiple strong biases to achieve a weaker one that is tailored to the application, thus reducing the statistical bias for the least increase in variance. In addition, the learning system must be able to interact gracefully with the *user*: presenting comprehensible hypotheses, and taking *user* feedback into account. Version-space algebra addresses this issue as well.

Originally developed for concept learning, a *version space* is the subset of a hypothesis space which is consistent with a set of training instances [15]. If there is a partial order over candidate hypotheses, one may represent the version space implicitly (*e.g.*, with boundary sets) and manage updates efficiently. Version-space algebra defines transformation operators (*e.g.*, union, join, *etc.*) for combining simple version spaces into more complex ones. We also developed a probabalistic framework for reasoning about the likelihood of each hypothesis in a composite version space. After constructing a library of reusable, domain-independent, component version spaces, we combined a set of primitive spaces to form a bias for learning text-editing programs (Figure 1), which was used in the SmartEdit implementation. Version-space algebra affords two benefits to a PBD system: 1) the ability to specify domain-specific details necessary to guide a learner with a simple algebraic expression (*i.e.*, a formula equivalent to the structure of Figure 1), and 2) a fast learning method which uses this expression to guide consideration of possible programs.

3.3 PBD with Dynamic Markov Logic Networks

More recently, we have employed *dynamic Markov logic networks* (DMLNs) [17] to do PBD. DMLNs are a probabilistic extension of first-order and temporal logic which consist of weighted first-order formulas describing the temporal relationships between the objects in a system. DMLNs can be used to model and learn stochastic processes, *i.e.*, the preconditions and effects of actions, the transitions between actions and the relationships between the hidden and observed properties of objects in the domain. In most real-world domains, the effects of an action are uncertain and a DMLN represents this using weighted first-order rules where the higher the weight, the more likely the effect.

The major advantage of using DMLNs for PBD is that one can learn first-order rules that capture the preconditions and effects of an action or transitions between them. For example, an expert can demonstrate the task of saving emails to a newly created folder and would like the PBD system to complete it for them. Using a DMLN, one can learn that the user was trying to save only those emails that belonged to his thesis based on the contents and the sender and recipients. Such tasks cannot be easily (if at all) learned using propositonal learners. Another advantage of DMLNs stems from its robustness to noisy training examples. It is capable of inducing a program even if the user makes a small error during demonstration (it can also identify these mistakes to verify that they *were* unintended.

DMLNs also allow us to combine the segmentation and trace induction phases of PBD. For example, we have modeled the desktop activity of a user simultaneously working on several tasks (*i.e.*, switching between them). We use a DMLN to look for common transition patterns (both at the propositional and first-order level) between the actions to segregate the tasks and then learn models for each task. Our DMLN learning method is implemented and works on examples of the form described above, but has not yet been implemented into a full PBD system.

3.4 Sharing Browser Customizations

While a PBD system might become easier than manual programming, program reuse is the focus of the ASSIEME project. Motivated by Mackay's observations [12], we seek ways for users to share browser customizations. In many ways our system is similar to alerting systems that advice novice users about system functionality that might be helpful, except that the likelihood of the user being unaware is even greater in our context.

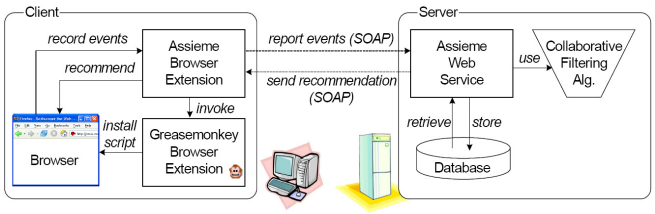

Figure 2: Architecture of ASSIEME.

Specifically, ASSIEME is a recommender system [2] for client-side Webpage customizations. ASSIEME— designed as an extension to the Firefox browser — records event traces of user browsing behavior. This recorded information is transmitted to a central server, and the server computes customization recommendations based on the similarity of multiple user models, which consist of event traces, installed customizations, and user responses to previous recommendations. Recommendations are transmitted back to the user who may accept or reject the installation of a new customization. We currently support client-side Webpage customizations written in JavaScript for the Greasemonkey Firefox extension.

Since the development of the client-side customization scripts requires programming skills, our system does at this point not yet offer the same flexibility as a PBD system. However, we believe that there are many customizations which have been developed and made publicly available. Our system facilitates sharing of these customizations, which often exhibit very complex behavior, because they are written by sophisticated programmers. Our main challenges lie in the design of an accurate recommendation algorithm and a secure communication protocol that respects every user's privacy.

Our work is not the first to addess sharing of customizations. Kahler [9] developed a system that allows users to explicitly share word processor customizations with colleagues. Unlike our approach, Kahler's system does not automatically track customization usage nor provides personalized recommendations. Client-side customization for webpages has also been previously proposed. Miller and Myers [14] integrated a command shell into a web browser to enable simple forms of automation. Today, the Greasemonkey extension to the Firefox webbrowser enables simple installation of more than 3000 publicly available customization scripts.

4 Conclusions

We aspire to the CHI workshop on EUSE, because we stand to learn much from the community. In particular, our work has not yet paid sufficient attention to problems, such as informing the user the nature of the program induced by the PBD algorithm — this is a critical weakness and we believe that visual programming languages may be a key component of the solution. Furthermore, we hope that our background in AI and machine learning could contribute to the workshop discussions.

References

[1] C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov models. *KDD-02*, August 2002.

[2] John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *UAI*, p43–52, 1998.

[3] John M. Carroll and Mary Beth Rosson. Paradox of the active user. Interfacing thought: cognitive aspects of human-computer interaction. pages 80–111, MIT Press, 1987.

[4] Allen Cypher, editor. *Watch what I do: Programming by demonstration*. MIT Press, 1993.

[5] O. Etzioni and D. Weld. A softbot-based interface to the Internet. *C. ACM*, 37(7):72–6, 1994.

[6] Michelle Gantt and Bonnie A. Nardi. Gardeners and gurus: patterns of cooperation among cad users. CHI-92, p107–117, 1992.

[7] K. Golden and D. Weld. Representing sensing actions: The middle ground revisited. KR-96, p174–185, 1996.

[8] Keith Golden, Oren Etzioni, and Dan Weld. Omnipotence without omniscience: Sensor management in planning. AAAI-94, p1048–1054. 1994.

[9] Helge Kahler. More than words - collaborative tailoring of a word processor. *J. UCS*, 7(8):826–847, 2001.

[10] Tessa Lau, Pedro Domingos, and Daniel S. Weld. Version space algebra and its application to programming by demonstration. ICML-00, p527–534, June 2000.

[11] Tessa Lau, Pedro Domingos, and Daniel S. Weld. Learning programs from traces using version space algebra. K-CAP-03, p36–43, 2003.

[12] W. E. Mackay. Patterns of sharing customizable software. CSCW-90, p209–221, 1990.

[13] W. E. Mackay. Triggers and barriers to customizing software. CHI-91, p153 – 160, 1991.

[14] Robert C. Miller and Brad A. Myers. Integrating a command shell into a web browser. *USENIX Annual Technical Conference*, p171–182, 2000.

[15] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[16] Stanley R. Page, Todd J. Johnsgard, Uhl Albert, and C. Dennis Allen. User customization of a word processor. CHI-96, p340–346, 1996.

[17] S. Sanghai, P. Domingos, and D. Weld. Learning models of relational stochastic processes. ECML-05, October 2005.

[18] Steven A. Wolfman, Tessa Lau, Pedro Domingos, and Daniel S. Weld. Mixed initiative interfaces for learning tasks: Smartedit talks back. IUI-01, p167–174, January 2001.