Automatically Personalizing User Interfaces

Daniel S. Weld **Corin Anderson**[*] **Pedro Domingos** **Oren Etzioni**
Krzysztof Gajos **Tessa Lau**[†] **Steve Wolfman**
Department of Computer Science & Engineering
University of Washington, Box 352350
Seattle, WA 98195–2350 USA

Abstract

Todays computer interfaces are one-size-fits-all. Users with little programming experience have very limited opportunities to customize an interface to their task and work habits. Furthermore, the overhead induced by generic interfaces will be proportionately greater on small form-factor PDAs, embedded applications and wearable devices. Automatic personalization may greatly enhance user productivity, but it requires advances in *customization* (explicit, user-initiated change) and *adaptation* (interface-initiated change in response to routine user behavior). In order to improve customization, we must make it easier for users to direct these changes. In order to improve adaptation, we must better predict user behavior and navigate the inherent tension between the dynamism of automatic adaptation and the stability required in order for the user to predict the computers behavior and maintain control. This paper surveys a decade's work on customization and adaptation at the University of Washington, distilling the lessons we have learned.

1 Introduction

Today's software is mass produced with a plethora of features designed to satisfy every user. But since different people are working on different tasks with different styles, there is no way to organize features in a way that makes essential functionality convenient for everyone. Consider word processing, for example; most people will never use automatic line numbering, but legal secretaries couldn't survive without it.

The shift away from the desktop and towards pervasive computing greatly exacerbates the problem of "Steelcase Inspired" software for several reasons. First, the shift from quiet office to ubiquitous use introduces complex environmental factors that increase the differences between the usage patterns of distinct individuals; indeed, a single user will require different features depending on context. Second, the form factor of mobile devices is enormously more variable than that of desktop machines — for example, the ratio of large to small desktop display size is about four, but it varies by a factor of 625 between a cell-phone and a liveboard.

The wisdom of user-centered design have been well documented [Lewis and Reiman, 1993]. However, as user needs and device characteristics diverge, the traditional design methodology of intensive user studies becomes unscalable. While each user deserves a personalized interface, designed for the device at hand and providing simple access to the commands and features they need, there aren't enough ethnomethodologists and designers to manage, interpret, and respond to so many studies. Indeed, the only resources that scale with the number of users are the users themselves and the computing devices they are using.

We use the term *adaptation* to denote personalization which is automatically performed by the interface without explicit user directives. By *customization* we mean personalization which is directly requested by the user. Our argument, thus, is that adaptivity and customization are the *only* scalable approaches to personalization. Interfaces should automatically adapt to the capabilities of the device at hand, to network connectivity, and to the individual user's activities, location, and context; users should be able to guide and control adaptation through a variety of customization mechanisms.

1.1 Customization and Adaptation

Of course, many existing desktop applications allow limited customization by letting the user (1) select which menus are visible, (2) add buttons to toolbars, (3) define macros, and (4) even add custom functionality via scripting languages such as Visual Basic. While customization is important, most users fail to customize effectively. Few users are comfortable with macros, regular expressions, or scripting languages, and even programmers are often too busy to invest the necessary time *now* in order to speed future sessions. Thus, the challenge is to create improved methods for users to direct their interface, rearrange functionality as well as appearance, and recover from inappropriate adaptations.

Similarly, many systems support limited adaptation, but users don't always appreciate it. Defaults that remember the last option or directory selected are simple and can be helpful, but Microsoft's smart menus disorient, while the Office

[*]Current address: Google Inc, 2400 Bayshore Parkway, Mountain View, CA, 94043

[†]Current address: IBM TJ Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598

Assistant's guesses fall far from the mark. The danger of adaptivity is its potential to create confusing inconsistency and rob the user of control. Thus, the challenge is to develop datamining algorithms that can accurately predict a user's behavior and to navigate the tension between too rapid adaptation (disorienting) and too little (inefficient), and to seek interface metaphors that increase user control.

1.2 Deep Deployment

Successful customization and adaptation methods will maximize their benefit when used by a broad range of applications. Ideally, there will be *deep deployment*: a uniform layer at the operating systems level that records clickstream data across applications, and supports ubiquitous personalization. For this to work, the interfaces themselves must be described using declarative representations like those developed by the model-based interface community [Foley *et al.*, 1989; Puerta, 1996]

Deploying personalization methods at the OS layer will bring several advantages. The first is consistency of behavior amongst applications. Second, knowledge of a user's activity with one application may improve adaptation of another's interface. Third, cross-application personalization may offer the highest benefit to users, since many common tasks involve patterns connecting two or more programs.

1.3 Outline

In the rest of this paper, we describe a sequence of our projects which are making progress on this vision. The underlying techniques range along a continuum of user-involvment, and our expository progression follows this vector from customization towards pure adaptation. Along the way we compare related endeavors and summarize lessons learned.

2 Customization by Command

The Internet Softbots project [Etzioni and Weld, 1994] marked the genesis of work on Intelligent User Interfaces at the University of Washington. By acting as a personal assistant, the softbot supported a qualitatively different kind of human-computer interface. Users were able to make high-level requests, and the softbot used search [Weld, 1996], inference [Etzioni *et al.*, 1997], and knowledge to determine how to satisfy the request. Furthermore, the softbot was able to tolerate and recover from ambiguity, omissions, and errors in these requests.

At its core, the softbot could handle goals specified in an expressive subset of first-order logic with modal operators for handling time and for distinguishing information gathering goals from those that requested state changes [Golden and Weld, 1996]. We labeled the softbot a *goal-oriented* interface, because human requests specified *what* the user wanted; the softbot was responsible for deciding *how* and *when* to satisfy the request. Since most users are uncomfortable with logical notation, we provided a forms-oriented interface front end. Unfortunately, we found the forms approach to be unscalable, and goal-specification was a challenge for many users. In response, we investigated two research directions: reliable natural-language interfaces [Popescu *et al.*, 2003; Yates *et al.*, 2003] and programming by demonstration [Lau *et al.*, 2000].

3 Programming by Demonstration

If it's too hard for users to specify goals, then a natural objective is the design of an interface that can watch a user's normal behavior and help as appropriate. In the case of repetitive tasks, this amounts to programming by demonstration (PBD). Of course, PBD has been studied extensively [Cypher, 1993], but most previous systems were heuristic and 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 a verbatim recorder's accompanying brittleness. While our PBD interface resembles a keystroke-based macro interface, it generalizes from the demonstrated actions to a robust program which is more likely to work in different situations.

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 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 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.

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.

3.1 Version-Space Algebra

Originally developed for concept learning, a *version space* is the subset of a hypothesis space which is consistent with a set of training instances [Mitchell, 1982]. If there is a partial order over candidate hypotheses, one may represent the version space implicitly (*i.e.*, with boundary sets) and manage updates efficiently. Version space algebra defines transformation operators (*e.g.*, union, join, *etc.*) for combining simple

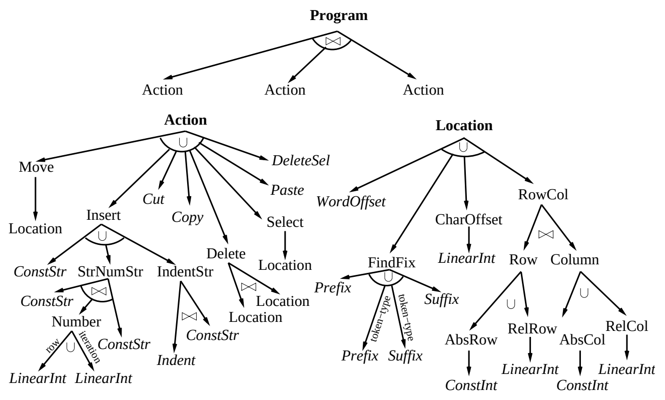


Figure 1: Algebraic specification of the version space for a SMARTEDIT program.

version spaces into more complex ones. We also developed a probabilistic 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 PBD implementation.

3.2 The SmartEdit Implementation

We verified the utility of version-space algebra for PBD with SMARTEDIT, an Emacs-like editor. When a user notices that she is about to perform a repetitive task, she clicks a button to start the PBD recorder. After completeing one instance of the task, she clicks another button to mark the first demonstration. SMARTEDIT initializes the version space using the recorded state sequence as the first training example. As the user continues to type, SMARTEDIT updates the version space with every action. In parallel, SMARTEDIT displays what action it thinks the user will next perform, and its confidence in the prediction. These probabilities are calculated by *voting* the version space. When the user is confident that SMARTEDIT has learned her procedure, she may let it execute in single-step or fully autonomous mode. We explored a number of ways of structuring the user's dialog with the learner, including decision-theoretic control [Wolfman *et al.*, 2001], but many aspects of facilitation require further study.

Version-space algebra provides SMARTEDIT with several benefits: efficient methods for incremental update and voting, without explicit enumeration of the version space [Lau *et al.*, 2000], and flexibility. By varying the algebraic formulation for the hypothesis space, we experimented with several other version spaces for learning single-loop programs from traces with varying amounts of segmentation information, as well as learning programs with multiple nested loops.

3.3 Experiments and Observations

Empirical studies validated our approach on a range of text-editing scenarios (*e.g.*, bibliographic reformatting, HTML to LaTeX conversion, information extraction, *etc.*). We show that a program that generalizes correctly for each of these scenarios can be learned quickly in as few as one or two training examples. An informal user study confirmed SMARTEDIT's usability and usefulness, and showed that even novice SMARTEDIT users perform tasks more quickly and concisely with PBD than without.

However, we learned a negative lesson when we started widespread distribution — despite incorporation within Emacs, relatively few people used SMARTEDIT frequently. From this we conclude that the overhead of a macro recorder interface is relatively high. Since most repetitive tasks are short, it is often easier to do them manually and avoid the cognitive load of complex customization. Others have found similar results — users customize relatively little [Mackay, 1991] — perhaps because customization facilities are complex themselves [McGrenere *et al.*, 2002]. We conjecture that widespread PBD adoption requires automatic segmentation, and the adaptation mechanisms, described in the next section, may provide this capability.

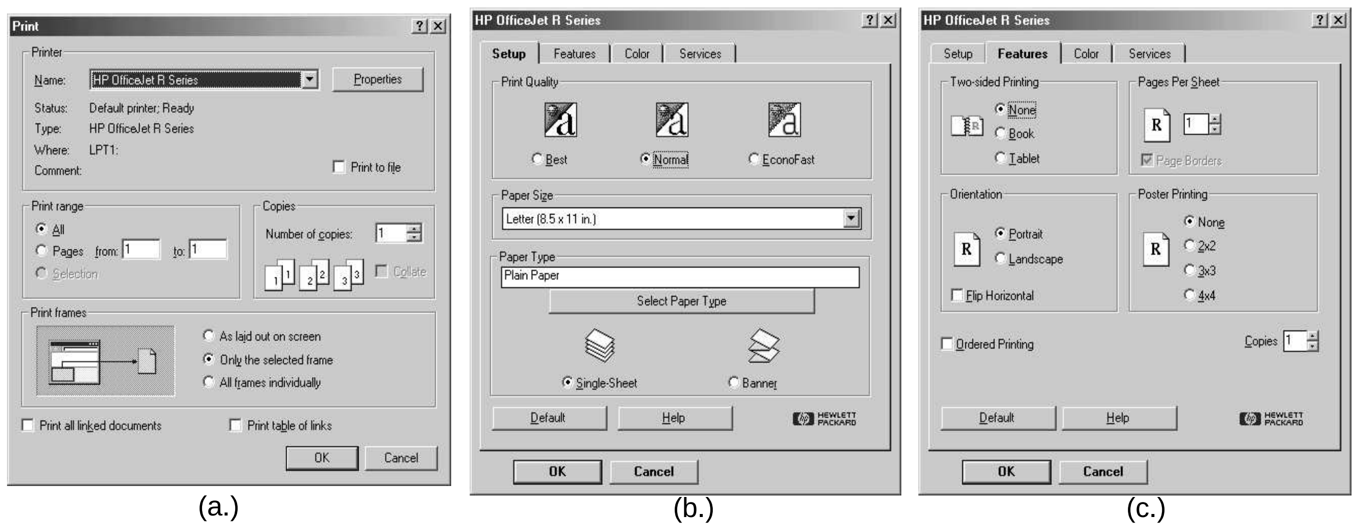
4 Adapting to User Behavior

The AI community has a long-standing interest in adaptive interfaces. The Calendar's Apprentice [Dent *et al.*, 1992] used machine learning to predict meeting location and durations. This research and similar work on email classification [Maes and Kozierok, 1993] led to an important principle about the incorporation of imperfect behavioral predictions in an interface: defaults are an effective way to minimize the cost to the user of (inevitable) poor predictions. Horvitz's decision theoretic framework [Horvitz, 1999] resulted in additional principles: graceful degradation of service prediction, expected utility of disambiguation dialogs as a function of user time and attention), and the use of timeouts to minimize cost of prediction errors.[1]

Our work on adaptive interfaces focussed initially on website design [Perkowitz and Etzioni, 1997] and led to algorithms for datamining web logs to discover aggregate patterns, which powered the automatic creation of index pages [Perkowitz and Etzioni, 2000]. Later we concentrated on mining individual behavior patterns, generating personalized sites for display on small, wireless devices [Anderson *et al.*, 2001b]. Our emphasis was on "information-goal seeking" behavior, common to wireless internet use, rather than general browsing or surfing. The PROTEUS system modeled adaptation as search through the space of possible websites. Site modification operators included highlighting text, adding shortcut links, and eliding parts of pages. We adopted the decision-theoretic approach, guiding search with expected utility calculations based on a model of the cost of manipulating the small PDA screen and fetching pages over the slow wireless connection.

A small user study revealed two important lessons. For the most part, the adaptations were good ones; PROTEUS suggested useful shortcuts and the elided content was almost always useless to the user. However, the cost of deleting important parts of a page was very high. Reweighting our utility function would have led to more conservative behavior, but

[1]It is ironic that the Offi ce Assistant violates most of Horvitz's principles, which were developed at Microsoft Research.



Print

Printer — Name: HP OfficeJet R Series; Status: Default printer; Ready; Type: HP OfficeJet R Series; Where: LPT1:; Comment:; Print to file

Print range: [✓ All] Pages from: 1 to: 1 Selection

Copies: Number of copies: 1 Collate

Print frames: As laid out on screen [✓ Only the selected frame] All frames individually

Print all linked documents; Print table of links

HP OfficeJet R Series — Setup

Print Quality: Best [✓ Normal] EconoFast

Paper Size: Letter (8.5 x 11 in.)

Paper Type: Plain Paper; Select Paper Type

[✓ Single-Sheet] Banner

HP OfficeJet R Series — Features

Two-sided Printing: [✓ None] Book Tablet

Pages Per Sheet: 1; [✓ Page Borders]

Orientation: [✓ Portrait] Landscape; Flip Horizontal

Poster Printing: [✓ None] 2x2 3x3 4x4

Ordered Printing; Copies: 1

Figure 2: (a) Main dialog box for print command. (b) After clicking the "Properties..." button, the default "Setup" tab is displayed. (c) After clicking the "Features" tab, the user can select double-sided printing; the user must now click "ok" twice.

the problem of accurate behavior prediction is central to interface adaptation; we discuss it in the next section.

A second problem stemmed from the fact that frequently users didn't find a shortcut link, even though PROTEUS had added it in the appropriate place. Shortcut-naming was usually to blame, and this raised the general issue of saliency, which we discuss in section 4.2.

4.1 Predicting User Behavior

High-quality adaptation requires the ability to learn an extremely accurate model of user behavior. Our comparison of existing techniques showed that a mixture of Markov models had the best predictive power [Anderson *et al.*, 2001a]. (Shortcuts generated by the mixture model saved users up to 40% of their navigational effort.) But all models performed poorly when data was sparse; hence none of these models would work well for situations where a user is exploring new parts of the interface — even if she is performing a task which is similar to one she has performed frequently in another context (*e.g.*, shopping cart checkout at a new store). To remedy this problem, we developed *relational Markov models* (RMMs), a generalization of Markov models that overlays relational structure on the states [Anderson *et al.*, 2002].

Intuitively, RMMs do for Markov models what Probabilistic Relational Models (PRMs) do for Baysian networks [Friedman *et al.*, 1999]. RMMs generalize Markov models by partitioning the states into classes; each class has an associated k-ary relation, R, and members of the class correspond to R instantiated with distinct ground values. The domain of each variable can be hierarchically structured, and a smoothing techniqued called *shrinkage* is carried out over the cross product of these hierarchies. RMMs make it possible to generalize beyond the observed states. For example, suppose a user surfing through an e-commerce Web site goes from a page about Nirvana's "Nevermind" CD to a page containing a biography of Nirvana. An ordinary Markov model can infer nothing from this about what the user will do when visiting a page about a Pearl Jam CD, but an RMM would be able to infer that the user might next go to a Pearl Jam biography. RMMs make effective learning possible in domains with very large and heterogeneous state spaces, given only sparse data. Experiments on academic and e-commerce websites show that RMMs trained on only ten instances perform as well as Markov models which were trained on ten thousand instances [Anderson *et al.*, 2002]. We are currently extending RMMs by incorporating structure in the same way that dynamic Bayes nets extend Markov models; see [Sanghai *et al.*, 2003].

4.2 Partitioned Dynamicity

As our experience with unnoticed shortcuts showed, saliency is essential — any adaptive mechanism that introduces new commands or options to an interface, must ensure that the user finds them. Furthermore, the mechanism must take care not to obscure important existing functionality.

One way to increase saliency, while minimizing the cognitive dissonance associated with adaptivity, is to *partition dynamism* — to segregate dynamic and static areas of an interface. Indeed, a number of well-designed adaptive interfaces exhibit this property. For example, the news stories and advertisements on Yahoo change continuously, but because the layout is fixed, the adaptation causes little distress to users. In contrast, the automatic menu shortening feature of Microsoft Office 2000 violates partitioned dynamicity, since the location of commands in the menus changes unpredictably.

Powerpoint XP's "Insert Symbol" command provides another example. Previous versions required a long sequence of clicks to select the correct font and navigate to the desired symbol, but the dialog box in the XP version has a "Recently used symbols" area.

We conjecture that partitioned dynamicity is a useful general principle for adaptation in interfaces. Because

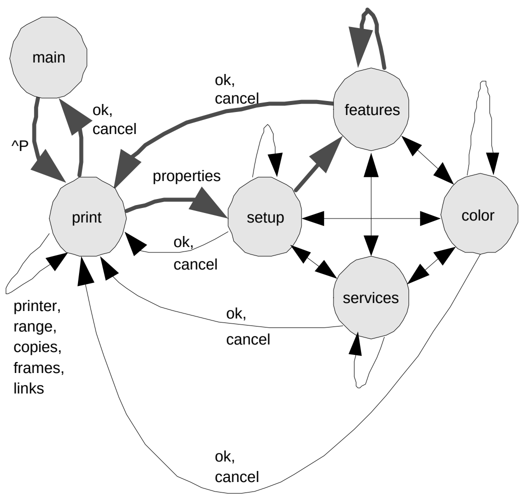
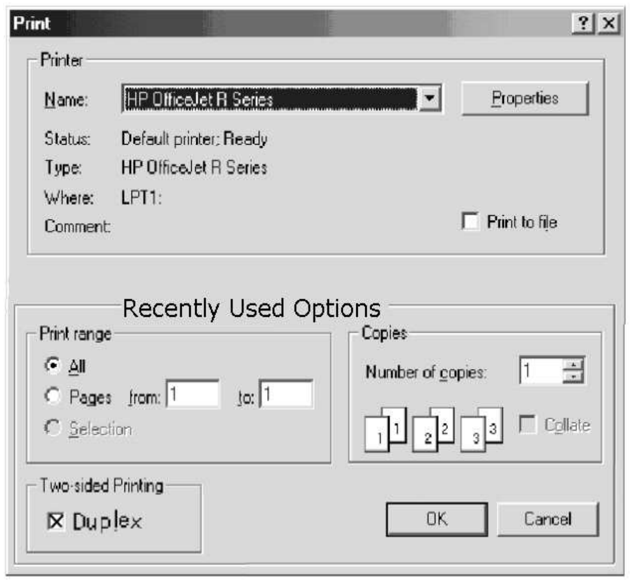


Figure 3: Abstract-state machine for the printing dialogs; the example user's path is darkened.


Print

Printer

Name: HP OfficeJet R Series — Properties

Status: Default printer; Ready

Type: HP OfficeJet R Series

Where: LPT1:

Comment:

☐ Print to file

Recently Used Options

Print range

(● All) ○ Pages from: 1 to: 1 ○ Selection

Copies

Number of copies: 1

☐ Collate

Two-sided Printing

☒ Duplex

OK Cancel

Figure 4: After adaptation, duplex printing is the default and visible on the main print dialog. By discarding the infrequently used "tablet" value, the variable can be cast as a Boolean and a simple, space-efficient check-box widget used.

users become accustomed to changes in the dynamic area, these changes don't disturb their conceptual model [Norman, 1998]. And because existing navigational patterns remain unchanged, users maintain control even if adaptation is unhelpful. The alternative, dual-interface approach of [McGrenere *et al.*, 2002] shares many benefits with partitioned dynamicity, but trades convenience for screen real estate.

4.3 Example: Adapting Desktop Dialogs

We believe that the PROTEUS architecture, developed for adaptive websites, would be even more useful in a desktop application setting. As a story-board example, suppose that a user wishes to print a Word document, doublesided. As Figures 2 and 3 show this requires six user actions.[2] First, the user presses `ctrl-P` to print the current document. Next, she must click on "Properties..." which yields "Setup" options. Clicking on the "Features" tab uncovers two sided printing options; another click selects book-style duplex. Then "Ok" must be clicked twice, once to close the Properties window and the other to confirm printing.[3] However, if an individual user executes this pattern hundreds of times, always choosing duplex printing, the system should recognize this and provide a shortcut.

Figure 4 shows one possible result, where the main screen has partitioned dynamicity, and the duplex options (with a new default value) have been added. In this example, we assume that the user had never chosen the "tablet" duplex value. For the shortcut, therefore, the system deleted the value, casting the duplex variable into a Boolean. As a result, the duplex option could be rendered with a simple, space-efficient checkbox. Of course, if the user later desired the tablet format, they could find it unchanged on the "Features" tab.[4]

Restricting a state variable's set of valid values and type casting are just two representative transformations; many more exist. Note that it is only possible to perform these transformations when the interface is represented in an abstract, declarative language. The next section discusses additional benefits from such a representation.

5 Adapting to Device Characteristics

The trend towards mobile and ubiquitous computing has resulted in an morass of device form factors and input mechanisms, and it is nearly impossible for a designer to target each of these manually. Furthermore, as shown in previous section, some parts of the interface may be generated dynamically in response to the particular usage patterns. Hence, we are building an automated solution, called SUPPLE. Others researchers have laid useful foundations. Starting with [Foley *et al.*, 1989], the model-based UI community has developed declarative, interface-representation languages; SUPPLE uses an extension of the representation developed for PEBBLES [Nichols *et al.*, 2002].

Researchers have explored many methods for "compiling" a declarative UI representation into concrete form for use on a specific target device. For example, PEBBLES uses a hand-constructed decision tree for this task; other approaches include hierarchical templates [Szekely *et al.*, 1993], stylesheets [Schreiber, 1994], and tools for facilitating

[2]Using Windows ME, Offi ce XP, and a HP R80 printer.

[3]Of course, one can quibble with the design of this particular interaction, but it is a result of today's software engineering process, and many similar designs exist —that's why adaptation is so promising

[4]Some might argue that this problem could be simply solved by using a MRU default for options like duplex, without restructuring the dialogs. But changing defaults is very dangerous unless they are visible to the user.

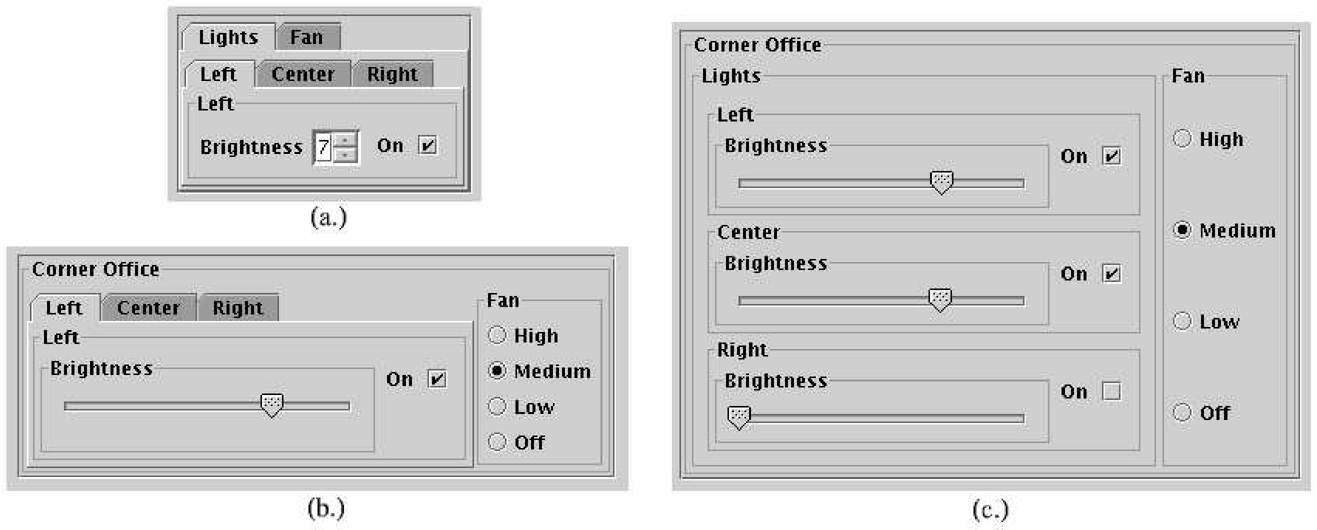


Figure 5: Given different PDA screen resolutions, SUPPLE generates different interfaces for a household controller.

manual design [Eisenstein *et al.*, 2001]. Yet another method is based on a higher-level encoding of the design in terms of *design patterns* [Lin and Landay, 2002].

In contrast, we adopt the decison-theoretic framework and search for the design with the highest expected utility among those that satisfy the device constraints. Our SUPPLE prototype focusses on screen-size constraints and uses a utility function that models the expected ease of interface operations for each widget. If multiple interface actions are necessary to complete a task (*e.g.*, if tabs must be selected, panes navigated, or confirmation buttons pressed) the costs are combined linearly. Figure 5 shows three different interfaces (from amongst the 3,888,000 possible) which were generated by SUPPLE and deemed optimal for differing amounts of available screen space.

While the current SUPPLE implementation accepts as input an arbitrary probability distribution over command usage, our framework supports user-based adaptation as described in the previous section. For example, at present SUPPLE makes an independence assumption regarding command use, but in practice people use commands in sequence and human designers recognize this by colocating the corresponding widgets. By computing the expected utility of an interface relative to a recorded trace of individual user commands, SUPPLE should be able to automatically duplicate this form of design optimization, but in a personalized manner.

6 Conclusions

Effective personalization requires improved methods for both adaptation (change based on implicit user behavior) and customization (change guided by explicit user requests). Interfaces should automatically adapt to the capabilities of the device at hand, to network connectivity, and to the user's activities, location, and context. If they wish to provide customization guidance, users should be able to control the adaptation process at any level.

This paper has briefly surveyed some of the projects at the University of Washington, which are investigating these issues, and codified several emerging principles:

- Because users find it hard to specify their prefences and goals, it is often more effective to induce them from user behavior. Version-space algebra allows an application designer to 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.
- Many benefits of model-based UI design have been long recognized [Foley *et al.*, 1989], but it is increasingly apparent that these declarative models are essential to a set of transformations (*e.g.*, Currying and casting) that facilitate adaptation.
- Since adaptive mechanisms are imperfect, the cost to users of errors must be considered alongside the advantages. Decison theory provides a powerful framework for such analysis [Horvitz, 1999], and the requisit utilities can be learned from user behavior or derived from device models.
- Certain interface mechanisms, such as defaults and automatic timeouts, can minimize the cost of errors and so increase the value of adaptation. Partitioned dynamicity offers a way to manage adaptivity while minimizing a user's cognitive load.
- Another way to enhance the user experience by improving prediction of user behavior. Because RMMs smooth, based on an abstraction lattice defined by the relational structure, they are exceptionally accurate when predicting sequential behavior (*e.g.*, a user's next action) from sparse data.

Acknowledgments

We thank James Landay for helpful discussions. This research was funded in part by Office of Naval Research Grants

N00014-98-1-0147 & N00014-02-1-0932, National Science Foundation Grants IRI-9303461, IIS-9872128, DL-9874759, and a CAREER Award.

References

[Anderson *et al.*, 2001a] C. R. Anderson, P. Domingos, and D. S. Weld. Adaptive web navigation for wireless devices. *IJCAI-01*, Aug 2001.

[Anderson *et al.*, 2001b] C. R. Anderson, P. Domingos, and D. S. Weld. Personalizing web sites for mobile users. *WWW-01*, May 2001.

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

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

[Dent *et al.*, 1992] L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski. A personal learning apprentice. *AAAI-92*, pages 96–103, July 1992.

[Eisenstein *et al.*, 2001] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of UIs for mobile computers. *IUI 2001*, pages 14–17, 2001.

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

[Etzioni *et al.*, 1997] O. Etzioni, K. Golden, and D. Weld. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1–2):113–148, January 1997.

[Foley *et al.*, 1989] J. Foley, W. C. Kim, S. Kovacevic, and K. Murray. Defining interfaces at a high level of abstraction. *IEEE Software*, Jan 1989.

[Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. *IJCAI-99*, Aug 1999.

[Golden and Weld, 1996] K. Golden and D. Weld. Representing sensing actions: The middle ground revisited. *KR-96*, pages 174–185, 1996.

[Horvitz, 1999] Eric Horvitz. Principles of mixed-initiative user interfaces. *CHI-99*, pages 159–166. ACM Press, May 1999.

[Lau *et al.*, 2000] Tessa Lau, Pedro Domingos, and Daniel S. Weld. Version space algebra and its application to programming by demonstration. *ICML-00*, pages 527–534, June 2000.

[Lewis and Reiman, 1993] C. Lewis and J. Reiman. *Task-Centered User Interface Design: A Practical Introduction*. University of Colorado, Boulder, CO, 1993.

[Lin and Landay, 2002] J. Lin and J. A. Landay. Damask: A tool for early-stage design and prototyping of multi-device user interfaces. *8th International Conference on Distributed Multimedia Systems*, pages 573–580, 2002.

[Mackay, 1991] W. Mackay. Triggers and barriers to customizing software. *CHI-91*, pages 153–160. ACM Press, 1991.

[Maes and Kozierok, 1993] Pattie Maes and Robyn Kozierok. Learning interface agents. *AAAI-93*, pages 459–465, 1993.

[McGrenere *et al.*, 2002] J. McGrenere, R. M. Baecker, and K. S. Booth. An evaluation of a multiple interface design solution for bloated software. *CHI-02*, 2002.

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

[Nichols *et al.*, 2002] J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol. Generating remote control interfaces for complex appliances. *UIST-02*, pages 161–170, 2002.

[Norman, 1998] D. Norman. *The Invisible Computer*. MIT Press, 1998.

[Perkowitz and Etzioni, 1997] M. Perkowitz and O. Etzioni. Adaptive web sites: an AI challenge. *IJCAI-97*, Aug 1997.

[Perkowitz and Etzioni, 2000] M. Perkowitz and O. Etzioni. Towards adaptive web sites: Conceptual framework and case study. *Artificial Intelligence*, 118:245–276, 2000.

[Popescu *et al.*, 2003] A. M. Popescu, O. Etzioni, and H. Kautz. High precision natural language interfaces to databases: a graph theoretic approach. *IUI-03*, Jan 2003.

[Puerta, 1996] A. Puerta. The MECANO project: Comprehensive and integrated support for model-based interface development. *CADUI'96*, pages 19–36, June 1996.

[Sanghai *et al.*, 2003] S. Sanghai, P. Domingos, and D. S. Weld. Dynamic probabilistic relational models. *IJCAI-03*, Aug 2003.

[Schreiber, 1994] S. Schreiber. Specification and generation of user interfaces with the boss-system. *EWHCI-94*, pages 107–120, 1994.

[Szekely *et al.*, 1993] P. Szekely, P. Luo, and R. Neches. Beyond interface builders: Model-based interface tools. *INTERCHI-93*, pages 383–390. ACM Press, 1993.

[Weld, 1996] D. Weld. Planning-based control of software agents. *AIPS-96*, May 1996.

[Wolfman *et al.*, 2001] Steven A. Wolfman, Tessa Lau, Pedro Domingos, and Daniel S. Weld. Mixed initiative interfaces for learning tasks: Smartedit talks back. *IUI-01*, pages 167–174, Santa Fe, USA, January 2001.

[Yates *et al.*, 2003] A. Yates, O. Etzioni, and D. Weld. Reliable natural language interfaces to household appliances. *IUI-03*, 2003.