Personalized Adaptive Interfaces

Krzysztof Gajos
University of Washington
Seattle, WA 98195, USA
kgajos@cs.washington.edu

INTRODUCTION

Mass-produced user interfaces are inadequate for today's computing needs: interfaces shipped with today's complex applications are designed in a "one size fits all" manner and by aiming to address the needs of the *average* user they miss essential needs of most *individual* users. In particular, I note that:

- users own diverse display-equipped devices, which support different interaction methods and have displays varying in size over several orders of magnitude;
- different users use the same software to accomplish different tasks;
- people's work habits change over time;
- users differ in their perceptual, physical and cognitive abilities.

Therefore, I believe that custom-built interfaces need to be provided for each individual user, reflecting her usual tasks, preferred interaction devices, abilities and situation context.

My work makes the following contributions:

- Because user's work patterns and needs change over time, user interfaces need to be able to evolve to reflect those changes. While user-driven customization is one way to accomplish this goal, I show that automatic system-driven adaptation can be designed in a way that improves both user satisfaction and performance.
- Individual needs of different users depend on their dominant tasks, devices they use as well as their abilities. In order to provide each user with a custom user interface, those interfaces need to be generated automatically. The dominant paradigm is to use knowledge-intensive rule-based techniques. Unfortunately, these techniques require a new rule base for different devices and even different screen sizes. Instead, I propose decision-theoretic optimization as a much more flexible and general approach.
- While optimization-based systems can be re-parametrized to achieve a variety of different outcomes, choosing the right parameters for the objective function is a hard problem. I develop interactions and machine learning techniques that allow non-expert users to quickly produce the right parameters just by interacting with concrete user interfaces.

UIST '06, October 15-18, 2006, Montreux, Switzerland

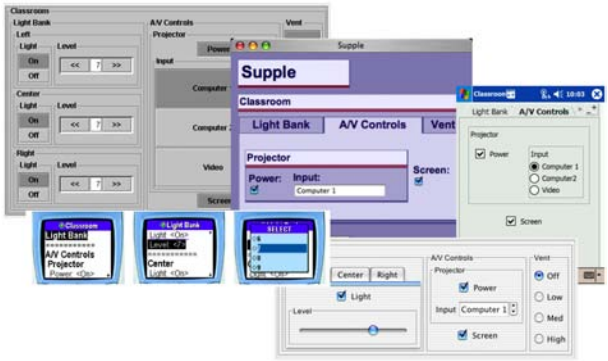


Figure 1: An interface for a simple application rendered automatically by SUPPLE for five different platforms: a touch panel, an HTML browser, a PDA, a desktop computer and a WAP phone.

While personalized adaptive interfaces can benefit regular users, there are two particular user populations – the mobile and the physically impaired users – that stand to gain even more from this approach. For mobile users it is technically possible to use a phone to locate nearby restaurants, check traffic conditions, find additional product information or access bus schedules, but in practice few people use such services simply because of the long and awkward interactions required to access them. Well designed adaptive interfaces can have large quantitative effects on these users' performance and thus bring about a qualitative change in how mobile users interact with their devices. For people with physical impairments, a lot of the difficulty in interacting with computers stems from the fact that the interfaces have been optimized for the "average" user able to easily manipulate keyboard, mouse and a small range of screen sizes. Custom interfaces that take into account less standard interaction techniques available to some users can make awkward assistive technologies unnecessary or at least easier to use.

I will first describe a system (currently under development) for automatically generating custom user interfaces for users with motor and vision impairments and use it to motivate and illustrate the contributions of this thesis.

AUTOMATICALLY GENERATING CUSTOM UIs FOR USERS WITH PHYSICAL OR VISUAL IMPAIRMENTS

A number of assistive technologies exist to address the needs of users with most common physical impairments: screen readers help blind users, software "magnifying lenses" help those with poor vision, while eye trackers, custom pointing devices and speech recognition systems help users with lim-

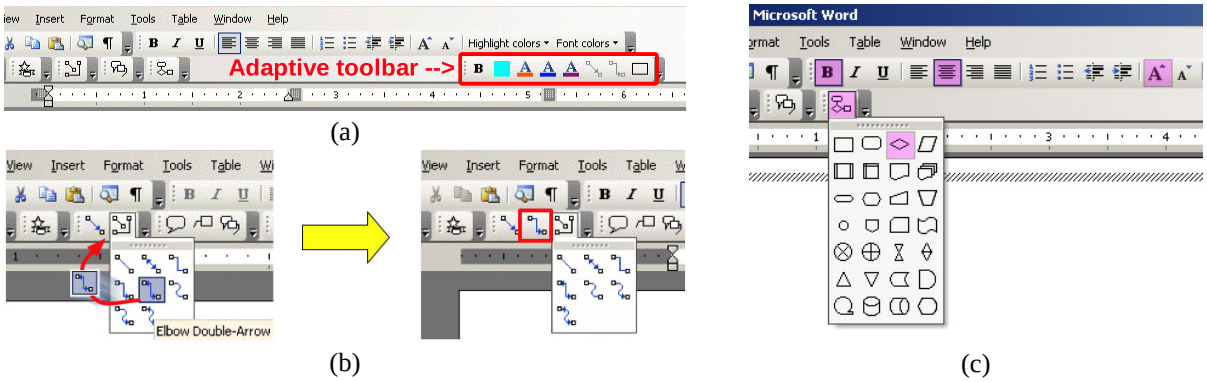


Figure 2: Three adaptive interfaces tested in our experiments (as implemented for Microsoft Word): (a) The Split Interface copies frequently used functionality onto a designated adaptive toolbar; (b) The Moving Interface *moves* frequently used functionality from inside a popup menu to a top level toolbar; (c) The Visual Popout Interface makes frequently used functionality more visually salient.

ited motor control. A common property of some of these solutions is that they address a single impairment at a time and often poorly combine with other assistive technologies. For example, software magnifying lenses assume a steady pointing device (like a mouse) but could be very hard to control with an eye tracker because of small involuntary eye movements. Furthermore, solutions like the magnifying lenses have a small number of discrete settings, ignoring the fact that there is a continuity of vision impairments. Finally, most of these solutions are poorly integrated with the interfaces and consequently provide an inefficient way to interact with the underlying applications. For example, screen readers are unaware of visual hierarchies of elements in dialog boxes. As another example, reducing screen resolution – with the intention of helping low-vision users – enlarges all parts of an interface and not just those that were too small to see clearly, thus wasting a lot of screen real estate.

In some cases no assistive technology might be required if an interface is rendered in a way that takes into account a user's capabilities. For example, to make an interface accessible to a user with a moderate vision impairment, it may be sufficient to make all the fonts and important visual cues larger, while appropriately rearranging the interface to make it fit in the available screen area. It is not necessary to elongate the slider tracks, for example, as long as the slider elements are made larger and the tracks are drawn with a thicker line. Similarly, users lacking very fine motor control will find it easier to interact with interfaces where only widgets with large targets and no need for dragging are used. In cases when the use of assistive technologies cannot be avoided, those technologies also work better with a dedicated GUI design: for example, users who explore a screen serially (with a magnifying lense or a screen reader) might find it easier to navigate a hierarchically structured interface, where each panel contains only a limited number of elements.

Thus the system I am building has to automatically generate user interfaces custom-tailored to any individual user's needs, taking into account not only individual impairments at their canonical stages but also combinations of impairments and their various intermediate stages, while enabling the most efficient interaction given the user's preferred modes of interaction. The system also has to adapt over time to make frequent tasks easier to accomplish. Finally, because physical impairments are, by definition, rare and often unique, it has to be easy for the consultants, care givers or even some users with impairments to quickly customize the system for the idiosyncratic needs of the imparied individual.

DO USERS WANT ADAPTIVE UIs?

Adapting a user interface to user's activities offers a potential to improve the quality of interaction, particularly if the frequently accessed functionality is hard to reach or if interacting with a device is slow and awkward (as is the case for some mobile or impaired users). User-driven customization is an effective approach but as user's needs change, recustomizing takes enough effort to practically deter many users from keeping their customizations up to date [8, 9].

Automatic system-driven adaptation is a complementary approach, but one that causes a fair amount of controversy: the proponents argue that automatic adaptation is a very effective technique to provide interactions optimized toward an individual, while the critics point out that a frequently changing interface is likely to disorient and confuse the user. Surprisingly, however, there is very little past research explicitly studying automatic adaptation in graphical user interfaces. The existing research includes both positive and negative examples of adaptation, sometimes reporting contradictory results without analyzing the reasons underlying the discrepancy (e.g., [11] and [1]).

We have conducted four laboratory studies (see [3, 7] for three of them) with two distinct applications (a software graphing calculator and MS Word) and three different adaptation techniques (illustrated in Figure 2 in the version for MS Word). We have synthesized our results with past research and began to outline how different design choices and interactions make some adaptive interfaces a pleasure to work with while others are frustrating impediments.

In the three studies that directly compared different adaptive techniques, Split Interfaces (where frequently used functionality is *copied* to a specially designated adaptive part of the

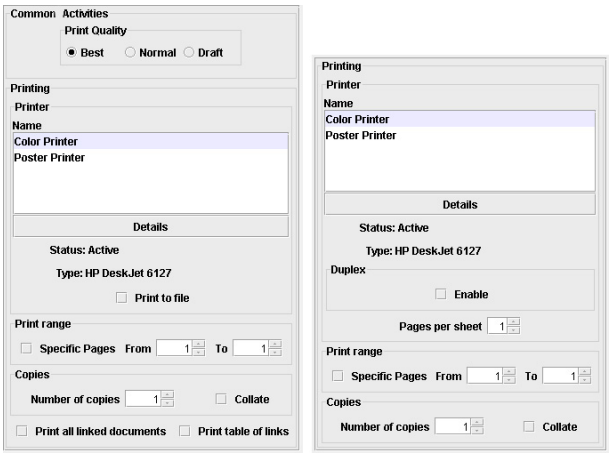


Figure 3: Two examples of personalization in SUPPLE: the left window features a dynamic section at the top whose automatically updated content reflects the most common activity; the right window was customized by the user: some functionality was removed while duplex printing and number of pages per sheet were added.

interface – see Figure 2a) were shown to result in significant improvement in both performance and satisfaction compared to the non-adaptive baseline. Our experiments and the analysis of past results also indicated that a number of specific design and context factors impact adoption of adaptive GUIs. Those factors included the accuracy and predictability of the adaptive algorithm, adaptation frequency, the frequency with which the user interacts with the interface, task complexity and the spatial stability of the interface (i.e., to what extent the original interface gets modified during the adaptation).

The results of our studies lead us to incorporate the split interface approach into our system. As illustrated in the left panel of Figure 3, all interfaces generated with our system can optionally include a "Common Activities" area, where frequently accessed but hard to reach functionality is automatically duplicated in response to the observed usage patterns. Unlike hard-coded GUI's, our automatically generated interfaces can promote arbitrary pieces of functionality (not just buttons or menu items) to the adaptive areas.

OPTIMIZATION FOR AUTOMATIC UI GENERATION

For the purpose of automatic generation of user interfaces, I use a *functional description* to define the *types* of data that need to be exchanged between the user and the application. The *device model* describes the widgets available on the device, as well as cost functions, which estimate the user effort required for manipulating supported widgets with the interaction methods supported by the device. Finally, I model a user's typical activities with a device- and rendering-independent *user trace*.

My goal was to build a system that can create a custom user interface for each user based on that user's current activity, her abilities and the available devices. SUPPLE, the system I have built, fulfills those requirements: it can quickly create interfaces on the fly, and the interfaces are immediately usable without any adjustments on the part of the user [4, 3].

With SUPPLE, I cast UI rendering as a constrained optimization problem, where the metric to be optimized is the estimated ease of use of the rendered interface, and the constraints reflect the available widgets and the screen size. This approach is a radical departure from the dominant paradigm of using knowledge-based techniques for user interface generation. Unlike the previous approaches, SUPPLE trivially adapts to devices with vastly different screen sizes and using SUPPLE on a novel device only requires specifying a new device model listing what widgets are available on that device. Finally, by modifying the cost function, SUPPLE can be made to produce very different styles of user interfaces or to accommodate other objectives, such as similarity to previously generated versions of the interface (even if they were generated for a different device) [6].

Although there was some previous work that used optimization methods for laying out widgets within a dialog window (*e.g.*, [10, 2]), my rendering algorithm does much more: it chooses the widgets, it chooses the navigation structure of the UI (i.e., puts things into tab panes or pop-up windows if not everything can fit on one screen) and chooses the layout.

Despite computational complexity, my algorithm is very efficient and renders even complex interfaces in less than two seconds on a standard desktop computer.

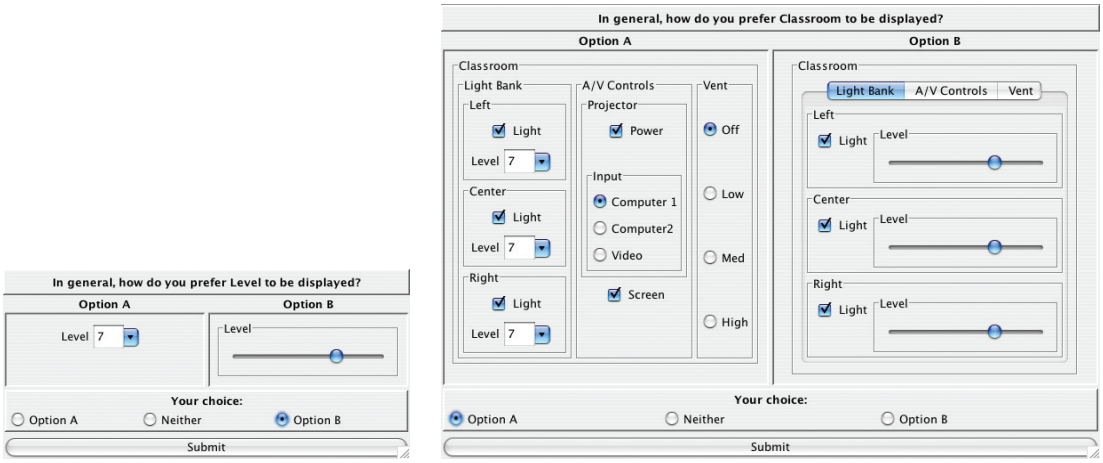
PERSONALIZING THE PROCESS

Decision-theoretic optimization is becoming a popular tool in the user interface community, but creating accurate cost (or utility) functions has become a bottleneck — in most cases the numerous parameters of these functions are chosen manually, which is a tedious and error-prone process. SUPPLE's cost functions, for example, typically rely on more than 40 parameters reflecting complex and interacting decision trade-offs. These parameters have to be chosen anew for each new target device and interaction style.

I have thus built ARNAULD, a system that allows users to quickly come up with the right parameters just by providing feedback about concrete outcomes [5]. ARNAULD uses two types of interactions: system-driven elicitation and user-driven example critiquing.

Users can freely switch between the two types of interactions. During the system-driven elicitation, ARNAULD presents the user with a pair of outcomes, always starting with a pair where only one easily-identifiable difference exists between the two alternatives (Figure 4a). The user is asked to express preference for one outcome or the other. If the difference causes rippling effects in the larger context, a followup query is issued illustrating those effects (Figure 4b). These *isolated* and *situated* queries allow ARNAULD to identify not only absolute preferences but also trade-offs: for example, in the two queries shown in Figure 4, the user indicated that he preferred sliders to combo boxes but not if they caused the interface to grow so large that it had to be split into separate tab panes.

The example critiquing interaction allows users to change the widget choice or layout of any part of the interface through direct manipulation (using the customization framework, in



(a) (b)

Figure 4: Two consecutive steps in the active elicitation process. (a) ARNAULD poses a *ceteris paribus* query, showing two renderings of light intensity control in isolation; this user prefers to use a slider. (b) Realizing that the choice may impact other parts of the classroom controller interface, ARNAULD asks the user to consider a concrete interface that uses combo boxes for light intensities but is able to show all elements at once, and an interface where sliders are used but different parts of the interface have to be put in separate tab panes in order to meet the overall size constraints.

the case of SUPPLE) – these interactions also provide input to the learning algorithm.

The learning algorithm uses the max-margin approach to find a set of parameters that optimally matches the preferences expressed by the user through the two types of interactions.

CONCLUSIONS

Numerous trends make the design of user interfaces increasingly more challenging: increasingly complex applications, people's continually changing work habits, fast growing number of diverse computational devices, awkward interaction styles afforded by some of the devices, and the growing need to offer the same access to computation to all members of the society regardless of their physical impairments.

The complexity of the software, people's shifting work patterns and awkward interactions imposed by some computing platforms motivate the need for interfaces that can adapt to individual usage patterns. The diversity of devices, individual interaction needs and preferences create the need for designing multiple versions of interfaces to address needs of different users, some of which cannot even be anticipated in advance.

I believe that automatically generated personalized adaptive user interfaces are an effective approach to address those challenges. In my thesis I intend to demonstrate that such interfaces are in fact feasible both from the usability and technology points of view. Indeed, my user studies have demonstrated that it is generally possible to design adaptive user interfaces that improve user performance and satisfaction; these studies have also shed light on what design choices influence the success or failure of adaptive UIs. I have also developed SUPPLE, a novel optimization-based approach for automatically generating interfaces. The approach is flexible enough so that it can be adapted to different devices, interaction methods and users. This adaptation, while hard in principle, can be performed easily with ARNAULD – a system that allows end-users to quickly re-parametrize SUPPLE to match their individual interaction requirements.

While designers are unlikely to give up pixel-level control over the design of user interfaces for commercial applications running on standard platforms, I intend to demonstrate that the approaches presented in this thesis can, even in near term, significantly change how physically impaired or mobile users interact with computation.

REFERENCES

1. L. Findlater and J. McGrenere. A comparison of static, adaptive, and adaptable menus. In *CHI'04*, 2004.
2. J. Fogarty and S. E. Hudson. GADGET: A toolkit for optimization-based approaches to interface and display generation. In *UIST'03*, Vancouver, Canada, 2003.
3. K. Gajos, D. Christianson, R. Hoffmann, T. Shaked, K. Henning, J. J. Long, and D. S. Weld. Fast and robust interface generation for ubiquitous applications. In *Ubicomp'05*, 2005.
4. K. Gajos and D. S. Weld. Supple: automatically generating user interfaces. In *IUI'04*, 2004.
5. K. Gajos and D. S. Weld. Preference elicitation for interface optimization. In *UIST'05*, 2005.
6. K. Gajos, A. Wu, and D. S. Weld. Cross-device consistency in automatically generated user interfaces. In *Workshop on Multi-User and Ubiquitous User Interfaces (MU3I'05)*, 2005.
7. K. Z. Gajos, M. Czerwinski, D. S. Tan, and D. S. Weld. Exploring the design space for adaptive graphical user interfaces. In *AVI '06*, 2006.
8. W. E. Mackay. Triggers and barriers to customizing software. In *CHI'91*, 1991.
9. J. McGrenere, R. M. Baecker, and K. S. Booth. An evaluation of a multiple interface design solution for bloated software. In *CHI'02*, 2002.
10. A. Sears. Layout appropriateness: A metric for evaluating user interface widget layout. *Software Engineering*, 19(7):707–719, 1993.
11. A. Sears and B. Shneiderman. Split menus: effectively using selection frequency to organize menus. *ACM Trans. Comput.-Hum. Interact.*, 1(1):27–51, 1994.