Automatically Generating User Interfaces Adapted to Users' Motor And Vision Capabilities

Krzysztof Z. Gajos, Jacob O. Wobbrock†* and Daniel S. Weld**

*Dept. of Computer Science and Engineering
University of Washington
Seattle, WA 98195 USA
{kgajos,weld}@cs.washington.edu

†The Information School
University of Washington
Seattle, WA 98195 USA
wobbrock@u.washington.edu

ABSTRACT

Most of today's GUIs are designed for the typical, able-bodied user; atypical users are, for the most part, left to adapt as best they can, perhaps using specialized assistive technologies as an aid. In this paper, we present an alternative approach: SUPPLE++ automatically generates interfaces which are tailored to an individual's motor capabilities and can be easily adjusted to accommodate varying vision capabilities. SUPPLE++ models users' motor capabilities based on a one-time motor performance test and uses this model in an optimization process, generating a personalized interface. A preliminary study indicates that while there is still room for improvement, SUPPLE++ allowed one user to complete tasks that she could not perform using a standard interface, while for the remaining users it resulted in an average time savings of 20%, ranging from an slowdown of 3% to a speedup of 43%.

ACM Classification D.2.2 [**Design Tools and Techniques**]: User Interfaces, K.4.2 [**Computers and Society**]: Social Issues: assistive technologies for persons with disabilities

General Terms Algorithms, Human Factors

KEYWORDS: Motor impairments, vision impairments, multiple impairments, decision theory, optimization

INTRODUCTION

Designers today generally create graphical user interfaces (GUIs) for able-bodied users [19], who are presumed to have normal motor and vision skills, and a relatively narrow set of input and output devices. For example, users are presumed to have sufficient motor control for typing and operating a mouse, to perceive screen elements with adequate vision. We might say that desktop user interfaces are essentially "optimized" for people with fine motor control, adequate vision, and a stable work surface. Any deviation from these assumptions (e.g., hand tremor due to aging, use of a laser pointer for cursor control, or use of a laptop on a jostling bus) may drastically hamper users' effectiveness — not because of any *inherent* barrier to interaction, but because of a mismatch between users' actual parameters and the assumptions underlying the GUI designs.

UIST'07, October 7–10, 2007, Newport, Rhode Island, USA.

Generally speaking, assistive technologies address these challenges by giving users specialized devices that allow users to adapt to standard GUI designs [2, 3]. For example, screen readers allow 2D graphical designs to "become" 1D spoken designs better suited to blind users. Of course, the graphical design does not change, but the user is able to adapt to it. Unfortunately, assistive technologies can stigmatize their users; they are also impractical for people with temporary impairments caused by injuries; they do not adapt to users whose abilities change over time; and finally, they are often abandoned, even by users who need them, because of factors like cost, complexity, and the need for ongoing maintenance [6, 7, 20, 26].

Other approaches like universal design [23], inclusive design [19], and design for all [30] attempt to create technologies that have properties suitable to as many people as possible. We appreciate these approaches for their laudable goal, but find them impractical in many cases, particularly where complex software systems are involved [3]. A "one size fits all" approach often cannot accommodate the broad range of abilities and skills in vast and varied user populations.

In contrast to these approaches, we argue that interfaces should be personalized to better suit individual users. Many such interfaces will be needed because of the myriad distinct individuals, each with his or her own diverse abilities and needs [3]. Note that even people with the same medical diagnosis can have a wide range of motor capabilities [17, 22]. Therefore, traditional, manual UI design and engineering will not scale to such a broad range of potential users. Instead, we enable GUIs to *design themselves* for users based on users' own functional capabilities. This approach embraces a current trend in designing for what users can *do*, rather than for classes of users based on health conditions or presumed skill sets [5, 22].

This paper presents SUPPLE++, a system which automatically generates user interfaces tailored to an individual's functional motor capabilities. These interfaces are also easily adjusted to accommodate users' different vision capabilities. Importantly, these two types of adaptations can be used together, allowing SUPPLE++ to adapt to people with a combination of motor and vision impairments, a population that is poorly served by current assistive technologies. As an example, Figure 1 shows an interface rendered by our system for four different users. The majority of the paper explains how

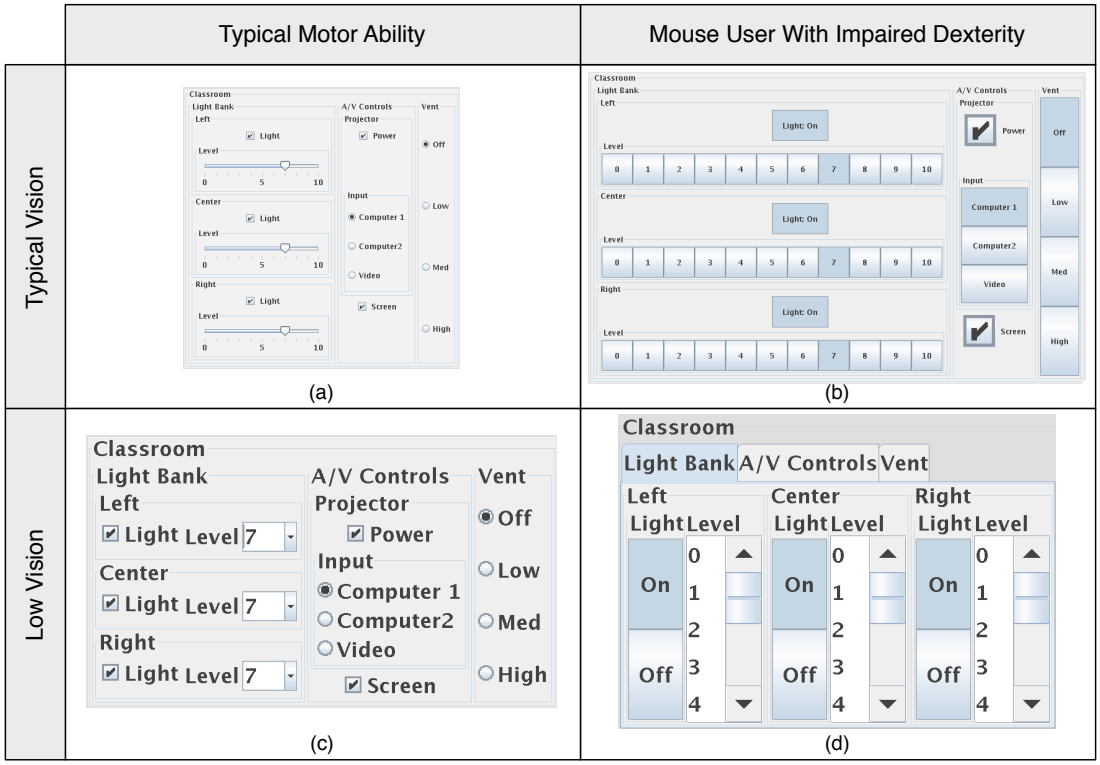


Figure 1: Four GUIs automatically generated under the same size constraints for four different users: (a) a typical mouse user, (b) a mouse user with impaired dexterity, (c) a low vision user and (d) a user with a combination of low vision and impaired dexterity. All but (a) were generated using SUPPLE++ described herein.

our system works; this explanation is then followed by a preliminary study for initial validation. We make the following contributions:

- We offer an algorithm to automatically select features and train a custom regression model to predict users' motor capabilities based on a one-time motor performance test. This accounts for the fact that some users with disabilities may not adhere to Fitts' law.
- We cast the generation of GUIs personalized to users' motor capabilities as an optimization problem in which users' expected movement times are minimized.
- We describe an efficient algorithm to solve this optimization problem.
- We present a modification to the SUPPLE system [8], which allows user interfaces to be manually adjusted to users' vision capabilities.
- We present the results of a pilot study, which indicate that using functional capabilities to personalize user interfaces can indeed improve users' performance. We describe these results, and the future directions to which they point.

We believe that, if it can be done successfully, software should automatically adapt to users and their abilities, rather than requiring users to adapt to software. This paper is a first step in demonstrating the viability of this approach, and the technology to make it possible.

RELATED WORK

Many others have recognized the benefit of creating GUIs specialized for people with unusual motor and visual capabilities. A number of specialized interfaces have been manually designed, with EyeDraw serving as an example of a drawing GUI optimized for eye trackers [16]. Others (e.g.,[11]) have developed new design methodologies for creating interfaces that can be dynamically adapted by the end-user for their individual needs. A few such interfaces have been created, particularly in the web domain, but most lack any such adaptation capabilities. A recent system [4] allows certain interactors to be dynamically swapped in unmodified third-party programs written in Java Swing to better support users of certain input technologies. The limitation of that system is that it can generally only replace GUI elements with similarly sized alternatives (e.g., swapping combo boxes for text fields) and cannot modify other attributes of the UI such as visual cue sizes or sizes of the interactors.

There exist a number of tools, such as the Personal Universal Controller (PUC) [25] or SUPPLE [8], which can automatically generate user interfaces from abstract specifications. In general, these tools have to be retargeted by their designers for each new platform or user type. A recent extension to SUPPLE is a notable exception – it allows the system to adapt itself to the end user's *preferences* [9], but not directly to their abilities or context.

The work on Layout Appropriateness [28] offers a useful technical insight: it shows how the layout of a dialog box can be automatically optimized for optimal performance. In our project we built on SUPPLE, which uses optimization to choose layout, appropriate widgets, and the structure of the

interface; we also built on the insight from Layout Appropriateness by using the expected movement time as the metric to be optimized.

DESIGN REQUIREMENTS

As noted, people with special needs differ widely in their motor and vision capabilities [3, 17, 22]. In this work, we set out to build a system that provides graphical user interfaces personalized to users' individual needs without any intervention on the part of designers or assistive technology specialists. Also, our system must be simple and fast to setup, use, configure, and maintain.

Currently, we focus on personalization for users who have difficulty controlling the mouse pointer (due to motor impairment, context, or input device) and users with low visual acuity. At this stage, we do not address text entry, blindness, or cognitive impairments.

A user's ability to control a GUI with a mouse depends on many factors. Among them are the distances among different on-screen elements, the complexity of navigation (e.g., using tabs or multiple widows), the types of operations required to use the elements (pointing, dragging, double-clicks), and the sizes of the elements. Thus, to support users with motor impairments, we need to provide them with user interfaces that strike the best balance among these competing factors. Complicating this are the complex tradeoffs involved. For example, widgets that are larger may be easier to manipulate but may result in a larger interface, which requires farther movements to use. To strike a balance among these tradeoffs, we use an optimization-based approach where SUPPLE++ "searches out" the interface that can be manipulated by an individual user in the least amount of time, as measured by that users' functional abilities. In order to do this, we need to elicit and accurately model these abilities for each user of the system. It is debatable, however, whether Fitts' law applies to individuals with motor impairments. Prior work suggests that in some cases it might [29], while in others it might not [13]. Due to these concerns, we need a new approach for modeling people's individual movement characteristics.

For people with vision impairments, size of the display, lighting, and distance from the screen have a large impact on usability. In such situations, the size of the visual cues can affect both task time and accuracy [27]. Most current solutions for low-vision users involve indiscriminately enlarging all the contents of the screen. Thus, useful content and empty space are enlarged equally, thereby wasting precious screen real estate. Solutions like screen magnifiers show only a fraction of the screen at a time and force serial rather than parallel exploration of the interface [21]. Our goal is to generate user interfaces that are legible and that can rearrange their content so that the entire interface fits on the user's screen for efficient exploration and interaction.

We note that a number of modern web browsers offer a "resize and reflow" feature where the user can enlarge the text on a page and the page is then instantaneously re-rendered. We intend to emulate this interaction for desktop GUIs in general, which requires that our system should be able to redraw interfaces for different sizes interactively, and that successive renderings resemble one another as much as possible so as to reduce potential for confusion.

Finally, our system needs to support people with *combinations* of motor and vision impairments – a population that is poorly served by the current technologies because screen magnifiers, the primary assistive technology for low vision users, often make assumptions about users' ability to control the mouse pointer.

MODELING POINTING PERFORMANCE OF PEOPLE WITH MOTOR IMPAIRMENTS

In this section, we present a study in which we collected motor performance data from 8 participants who exhibited a range of motor abilities and who used a variety of input devices. We also describe an automated method for developing and training a *personalized* predictive model for each participant.

Method

Eight people (3 female) aged 25 through 35 participated in our study. Six of our participants used their own personal input devices, and 3 used computers of their choosing. Table 1 lists the devices used and any health conditions that might affect participants' motor performance. The "code" column refers to a shorthand designation that we will use throughout the rest of the paper to refer to these individuals.

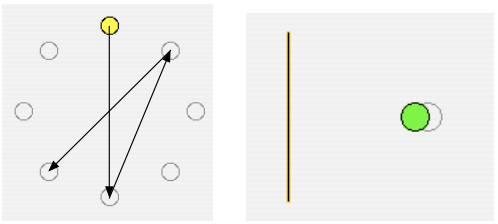
Code	Device used	Health condition
ET01	Eye tracker (ERICA) (click by dwelling)	
HM01	Head Mouse (click w th right fist using a switch)	spinal cord injury (incomplete tetraplegia)
M03	Mouse	muscular dystrophy (impaired dexterity)
M04	Mouse	
TB01	Trackball (Kensington Expert Mouse)	spinal cord injury (incomplete tetraplegia)
TP01	Trackpad (Apple MacBoook Pro)	
VJ01	Vocal Joystick [14]	
VJ02	Vocal Joystick [14]	

Table 1: List of participants

Throughout the rest of the paper, we will sometimes refer to *typical* users. These are users who utilize "typical devices" to interact with the computer and whose motor capabilities are unimpaired (i.e., M04 and TP01). In contrast, *atypical* users will refer to those whose motor abilities are impaired (i.e., HM01, M03, or TB01) and/or who use unusual input devices (i.e., ET01, HM01, VJ01 or VJ02).

We used a set of pointing tasks based on the ISO 9241-9 standard [18] where we varied target size (10-90 pixels), distance (25-675 pixels), and movement angle (16 distinct, uniformly spaced angles). A typical task sequence is illustrated in Figure 2a. In all, between 387 and 572 pointing actions were elicited per participant. (Some participants who tired faster had fewer trials than other participants.) The study took between 20 and 40 minutes per participant.

For each pointing action, we recorded the time taken to successfully acquire the target and the number of unsuccessful clicks. A click was "successful" if the mouse button was both pressed and released within the target, which is consistent with the way most widgets work in modern GUIs.



(a) (b)

Figure 2: The setup for the performance elicitation study: (a) for pointing tasks; (b) for dragging tasks; here the green dot was constrained to move in only one dimension, simulating the constrained one-dimensional behavior of such draggable widget elements like scroll bar elevators of sliders.

Results and the Inadequacy of Fitts' Law

We computed Fitts' law parameters for each of our participants. The average R^2 of the resulting models for the atypical participants was .51 ranging from .14 (ET01) to .81 (VJ02). As an example, Figure 3a (next page) shows how movement time varied with distance and target size for participant ET01. The high variance for target sizes of 10 and 15 pixels was due to the difficulty of acquiring small targets with this particular eye tracker. Therefore, ET01's performance markedly improved at the 25 pixel target size but did not change substantially beyond 40 pixels. The distance to the target only marginally affected this participant's performance. As illustrated in Figure 3b, Fitts' law poorly models the observed performance of ET01. This is not entirely surprising, because Fitts' law assumes rapid, aimed movements unencumbered by issues like eye-tracking jitter.

In fact, we observed a similar lack of fit to Fitts' law for HM01. His performance degraded sharply for distances larger than 650 pixels when he could no longer perform the movement with a single head turn and had to "skate". On the flip side, TB01 showed the opposite trend with respect to target size. His performance improved very slowly for small targets, but these improvements became much more pronounced for sizes larger than 25 pixels. Again, we see that Fitts' law is a poor fit to users like these.

Another matter that would complicate our use of Fitts' law occurs because our generator has the option of changing the sizes of widgets in a GUI and the distances between them. These distances (D) will change as a linear function of widget sizes (W) with $D = aW + b$, where the constant term b is due to unchanged components of the widgets (e.g., labels). In this case, Fitts' index of difficulty $ID = \log_2(\frac{aW+b}{W} + 1)$ makes it clear that as W shrinks, ID grows to infinity ($\lim_{W\to 0} \log_2(\frac{aW+b}{W} + 1) = \infty$). But as W grows to infinity, ID shrinks asymptotically to a constant ($\lim_{W\to\infty} \log_2(\frac{aW+b}{W} + 1) = a + 1$). Thus, if we were to model all users using Fitts' law, a single improvement strategy would be implied for all: grow target sizes to infinity unless size constraints force dramatic adverse changes in the choice of widgets or organization. But as the diversity in our observed results imply, our participants would likely benefit from more individualized adaptations.

For these empirical and theoretical reasons, we proceed to develop a different method for modeling each participant's unique motor performance abilities, as described in the next section.

Automatically Creating Models of Pointing Performance

Our process for automatically creating models of pointing performance for users with disabilities uses two steps. In the first step, our technique finds the best set of features to include in the model. In the second step, it trains a regression model that is linear with respect to the selected features.

We consider seven possible features in our model: a constant term, Fitts' index of difficulty (ID), its two individual components $\log_2(D)$ and $\log_2(W)$, as well as the raw measures W, 1/W and D. For each user's data set, SUPPLE++ evaluated the $2^7 - 1 = 127$ possible models with at least one feature using 10-fold cross validation[1] and ordered them by the mean squared error (MSE) they produced.

Cross validation is known to overestimate the algorithm's error on test data so a common practice is to look for the most parsimonious solution (i.e., one using the fewest features) in a small vicinity of the optimum predicted by the cross validation [15]. SUPPLE++ thus picks the most parsimonious model whose performance is within 1% of the best model. This one-time feature selection process takes less than two minutes on a standard computer. Table 2 summarizes the feature results for our participants.

	D	1/W	W	log(D)	log(W)	ID	1
ET01		x	x				
HM01		x	x			x	
M03		x				x	x
M04	x			x		x	x
TB01	x		x	x			
TP01		x	x			x	
VJ01	x			x		x	x
VJ02	x	x		x			

Table 2: Results of the feature selection process for different participants. Although $log(W)$ was not used in modeling the performance of any of the participants, it was used in modeling some devices tested by the authors.

Whereas the mean R^2 for Fitts' law models for atypical participants was .51 (ranging from .14 to .81), it improved to .71 (ranging from .49 to .88) with the personalized models. Unsurprisingly, for typical users the improvement was much smaller (from .79 to .85 on average).

Modeling Dragging and Multiple Clicks We also explicitly model participants' ability to perform dragging operations. These are constrained to one dimension, e.g., as for scroll bars or sliders (Figure 2b). We also model users' ability to execute multiple clicks over targets of varying sizes.

We model dragging time using the same user's pointing time MT_{point} (for the same distance and target size) as the only non-constant feature. This is because we observed that

[1]Because data were divided randomly into the 10 folds in each instance, we repeated the cross validation process 10 times and averaged the results.

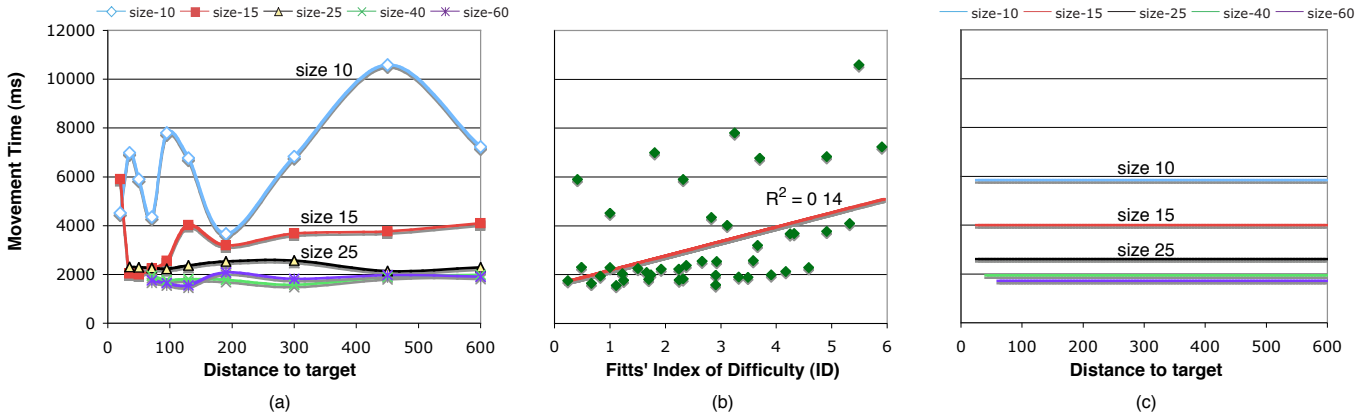


Figure 3: Movement time for the eye tracker user: (a) actual data, (b) Fitts' law model, (c) automatically selected custom model, which correctly captures the fact that this user was affected by the increasing target size but not by the movement distance. The same y-axis scale is used on all three graphs.

movement times for both types of operations follow similar patterns and the formulation we adopt for modeling MT_{drag} allows us to elicit only a small number of dragging operations (around 40) from the user to accurately estimate the parameters of the following equation:

$$MT_{drag}(\mathrm{D},\mathrm{W}) = a + b * MT_{point}(\mathrm{D},\mathrm{W}) \qquad (1)$$

The users have the option of skipping the dragging task if they cannot perform the dragging operation (the case with ET01), in which case our system sets the a parameter to a very large number to reflect the fact that dragging was not a practical option for the user.

Equation 2 below shows the model we found to accurately estimate the time per click when the user has to perform multiple clicks on a target of width W.

$$MT_{click}(\mathrm{W}) = a + b * \log(W) \qquad (2)$$

OPTIMIZING GUIs FOR USER'S MOTOR CAPABILITIES

We seek a method for generating a GUI which minimizes the user's expected movement time (EMT) given a description of a typical task performed on that interface. Since this problem is naturally one of optimization, we build our solution upon SUPPLE [8], a system that uses decision-theoretic optimization to automatically generate user interfaces. Note that past work [9] has shown that SUPPLE can generate user interfaces optimized for the user's qualitative *preferences*, but this is insufficient for our current needs. This section shows how to use the same general optimization framework to generate UIs which are optimized for a user's quantitative *performance*. As we show the resulting optimizations are significantly harder, involve continuous parameters (e.g., widget size), and require new heuristics to prune the space.

Review of SUPPLE

Similarly to the Personal Universal Controller [25] and similar systems, SUPPLE uses a hierarchical representation, called a *functional specification* ($\mathcal{S}_f$), to abstractly describe a user interface. Leaf nodes in the $\mathcal{S}_f$ correspond to individual pieces of data, described by their types (integer, string, enumeration, etc.), which should be presented to or manipulated by the user. Interior nodes in the $\mathcal{S}_f$ denote hierarchical groupings of child elements. SUPPLE treats interface generation as a discrete optimization problem, considering a finite set of concrete instantiations to use when rendering each node: type-compatible widgets for leaf nodes and layout "widgets" (e.g., horizontal, vertical, tabbed layouts, etc.) for interior nodes. SUPPLE searches the space of these configurations, ensuring that all constraints (e.g., size restrictions) are obeyed, and returns the one whose estimated cost, **$**, is minimal.

To do this optimization efficiently, previous SUPPLE implementations required that the cost function be factored in a specific way:

$$\$(R(\mathcal{S}_f), \mathcal{T}) = \sum_{n \in \mathcal{S}_f} M(R(n), \mathcal{T}) + N(R(n), \mathcal{T}) \qquad (3)$$

This equation states that the expected cost of executing a trace (sequence of interactions), $\mathcal{T}$, on a rendered interface, $R(\mathcal{S}_f)$ is equal to the sum of the match (M) and navigational (N) costs of each node $n \in \mathcal{S}_f$. The match cost measures how well the concrete widget, $R(n)$ enables changes to the value of the node, while the navigational cost penalizes extra interactions that might be required by tab panes or pop-ups; in both cases the component costs are weighted by the number of times they appear in the trace.

This factoring has two important properties: first, the cost of each widget can be computed independently of others and, second, the cost associated with navigating the interface is separated from the usability of individual interactors. These two properties were leveraged to develop an efficient branch-and-bound search algorithm for finding the optimal rendering. Specifically, SUPPLE explores the space of partial renderings; at each step in the search, SUPPLE uses the factored cost function to compute a *lower bound* on the cost of the best possible, reachable rendering. If that estimate is greater than the cost of the cheapest solution found so far, then this region of the search space may be pruned without sacrificing optimality. By combining this use of an *admissible* heuristic with efficient constraint propagation, SUPPLE avoids the

vast majority of possible renderings, finding optimal renderings in just one or two seconds [8].

Optimizing For A Person's Motor Abilities

A more complex cost function than the one described above is required, however, in order to optimize the interfaces for a user's expected movement time, EMT:

$$\begin{aligned} \$(R(\mathcal{S}_f, s), \mathcal{T}) &= EMT(R(\mathcal{S}_f, s), \mathcal{T}) \\ &= EMT_{nav}(R(\mathcal{S}_f, s), \mathcal{T}) \\ &\quad + \sum_{n \in \mathcal{S}_f} EMT_{manip}(R(n, s), \mathcal{T}) \end{aligned}$$

Here, EMT_{nav} is the expected time to navigate the interface, EMT_{manip} is the expected time to manipulate a widget (0 for layout widgets) and s is the minimum target size; that is, the minimum size for all control elements of a widget that can be manipulated with a pointer (see Figures 1 and 5 for how widgets are drawn depending on the different values of s). There are two important differences between this formulation of the cost function and that of Equation 3:

1. The lengths and the target sizes for the movements between the widgets, and thus the EMT_{nav}, cannot be computed until all interactors and layout widgets have been chosen. This is problematic because it makes it hard to estimate the minimal cost of renderings consistent with a partial assignment, blocking the use of branch-and-bound, and ruining search efficiency.

2. Both the expected time to manipulate a widget and the expected time to move between two widgets depend on the new continuous parameter, s, transforming a discrete optimization problem into a harder, hybrid (discrete/continuous) one.

The rest of this section explains how we confront these problems. We begin, however, with a brief description of the process for computing EMT_{manip} for widgets.

Computing EMT_{manip} Many widgets can be operated in different ways depending on the specific data being controlled and on the user's motor capabilities. For example, a list widget, if it is large enough to show every item, can be operated just by a single click. However, if some of the list elements are occluded, then the user may need to first use the scroll bar before selecting one of the invisible elements. Scrolling may be operated by dragging the elevator, clicking multiple times on the up/down buttons, depressing an up/down button for a short period of time or clicking multiple times in the scrolling region above or below the elevator. Which of these options is fastest, depends on how far the user needs to scroll and on how efficiently (if at all) she can perform a drag operation or multiple clicks.

To accommodate the uncertainty about what value the user will select while interacting with a widget, we assign a uniform probability to the possible values which might be selected and then compute the expected cost. To address the choice of ways the widget may be operated (e.g., dragging the elevator *vs.* multiple clicks on a button), we compute the EMT_{manip} for each possible method and choose the minimal value. We cannot decide *a priori* which interaction type is the fastest for a particular widget type because the outcome depends on the circumstances of a particular user (e.g., the particular eye tracking software used by ET01 did not provide support for dragging).

When computing movement times towards rectangular targets, we use the smaller of the two dimensions as the target size — as suggested by [24]. Although more accurate models for two-dimensional pointing have been developed for typical mouse users [1, 12], no such models are known for atypical users, and we found the approximate approach to be adequate for our purposes.

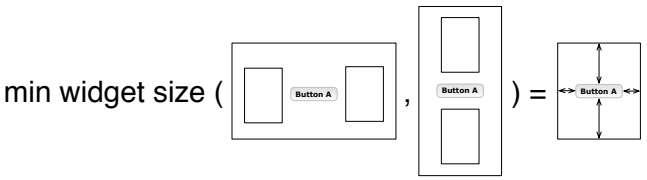
Finally, we note that in order to estimate the movement time *between* widgets, one must take into account the size of the target to be clicked at the end of the movement. That means that the first click on any widget counts toward the navigation time (EMT_{nav}) and not the time to manipulate the widget. Thus the EMT_{manip} for a checkbox, for example, is 0 and the size of the checkbox affects the estimated time to navigate the interface. This increases the urgency of bounding EMT_{nav} before all nodes in the $\mathcal{S}_f$ have been assigned a concrete widget; the next subsection explains how this may be done.

Computing a Lower Bound for EMT_{nav} The key to SUPPLE's efficient branch-and-bound search depends on being able to efficiently bound the cost, including EMT_{nav} for widgets that have not yet been chosen. Indeed, we confirmed it when our initial attempts to use blind methods to optimize layouts failed — search took many hours, for even simple specifications.

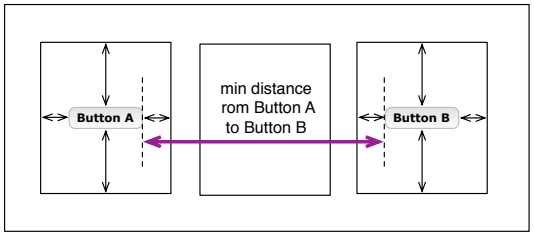
To compute a lower bound on EMT_{nav}, which is applicable even when some widgets and layouts have yet to be chosen, we proceed as follows.

First, for each unassigned leaf node, n, we compute a rectangular area which is guaranteed to be covered by all of the widgets which are compatible with n; that is, we compute the minimum width of all compatible widgets and separately find the minimum height, as illustrated below.

We may now propagate these bounds upwards to form bounds on interior nodes in the functional specification. For example, the width of an interior node with a horizontal layout is greater than or equal to the sum of the lower bounds of its children's widths. If an interior node has not yet been assigned a specific layout, then we again independently compute the minimum of the possible widths and the possible lengths.



Note, however, that in this case for each element contained within a layout element (like the Button A above), our estimate also provides the minimum distance from the edges of the layout element to the contained element. As a result, we can compute the most *compact* possible layout for an interface and thus the shortest possible distance between any pair of elements, as illustrated below:



To provide a lower bound on the time to move between elements n_s and n_t we use the shortest possible distance between the pair and the *largest* possible target size among the set of widgets which are compatible with the target, n_t. We update these estimates every time an assignment is made (or undone via backtracking) to any node in the functional specification during the branch-and-bound search process.

More complex layout elements such as tab panes, pop-up panes or pop-up windows make this process only slightly more complicated — most notably they require that multiple trajectories are considered if a node on a path between two widgets can be represented by a tab or a pop-up. However, the principle of our approach remains unchanged.

As documented in the Evaluation section, our lower bound on EMT_{nav} resulted in dramatic improvements to the algorithm performance.

Finding the Optimal Target Size. In contrast to the original SUPPLE implementation, SUPPLE++ must also optimize s, the minimum target size, which is a continuous parameter. Figure 4 shows how the cost (the EMT) of the best GUI varies as the minimum target size ranges between 0 and 100 pixels. Because these curves include numerous local minima, we can't apply any of the efficient, convex optimization, techniques — instead, it is necessary to search the space exhaustively. Fortunately, the specter of continuous optimization is only an illusion. In practice, only integer sizes are used. Furthermore, we may approximate matters by discretizing the space even more coarsely — e.g., at 5 pixel intervals — yielding 21 discrete settings (in the range between 0 and 100) for the size parameter. Because this approach lets us continue to exploit branch-and-bound, search is very fast in practice because a great many values can be pruned quickly. Figure 5 (next page) shows 2 GUIs generated by our algorithm.

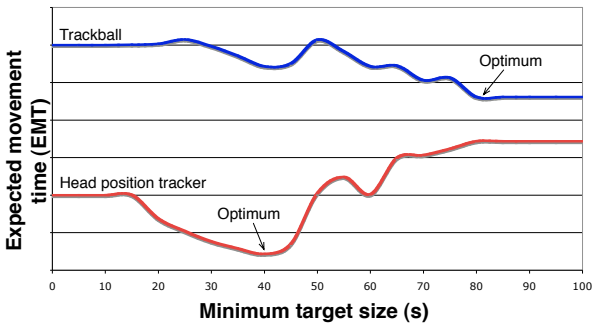
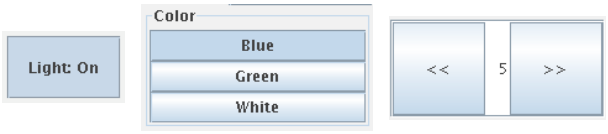


Figure 4: The estimated movement times (EMT) of the optimal GUIs generated with different values of the minimum target size parameter for two participants. Y-axis corresponds to the EMT but the curves were shifted to fit on one graph so no values are shown.

Implementation

We were able to change the presentation of the widgets (to allow for resizing of interaction targets, icons and fonts) by re-implementing parts of the Java Swing's Metal Look and Feel. We have further implemented three additional simple widgets for our system to use as alternatives to a checkbox, a set of radio buttons and a spinner, respectively:



ADAPTING TO USERS WITH LOW VISION

To adapt to a person's vision capabilities, we vary only one parameter, namely the visual cue size. As mentioned previously, we allow users to directly control this parameter via a keyboard shortcut and a simple GUI — akin to the mechanism provided by most modern web browsers. This poses two challenges. First, the interaction should be nearly instantaneous. Second, as the font size varies, the changes between successive renderings must be kept to a minimum to avoid disorientation.

To address the first challenge, we added a caching system to SUPPLE++ and noted that only 8 discrete visual cue settings adequately cover the range from the font size 12 (the default font size used in Metal Look and Feel) and 50. Thus GUIs for the remaining 8 cue sizes can be pre-computed in the background while the user is inspecting the first one (a feasible option because our system requires less than two seconds per GUI when generating *preference*-based interfaces).

To address the challenge of maintaining consistency between successive renderings of an interface, we augment the cost function by adding a penalty to enlarged renderings that don't resemble the original (using similarity characteristics discussed in [10]).

Figure 6 (next page) shows an email client configuration GUI which has been automatically generated for a typical user (left) using original SUPPLE and for a low vision user (right) using SUPPLE++, with font sizes enlarged by a factor of 3 and other visual cues adjusted accordingly. The GUI with large cues uses the entire area of a 1440×900 screen. While not all elements could be shown side by side and some were thus put into tab panes, the logical structure of the entire interface, as well as the action buttons at the bottom of the interface, are all visible at a glance. If instead a full screen magnifying glass were used (or if the display resolution was lowered) to achieve the same font size, only a fraction of the original interface, outlined by the dashed line, would have

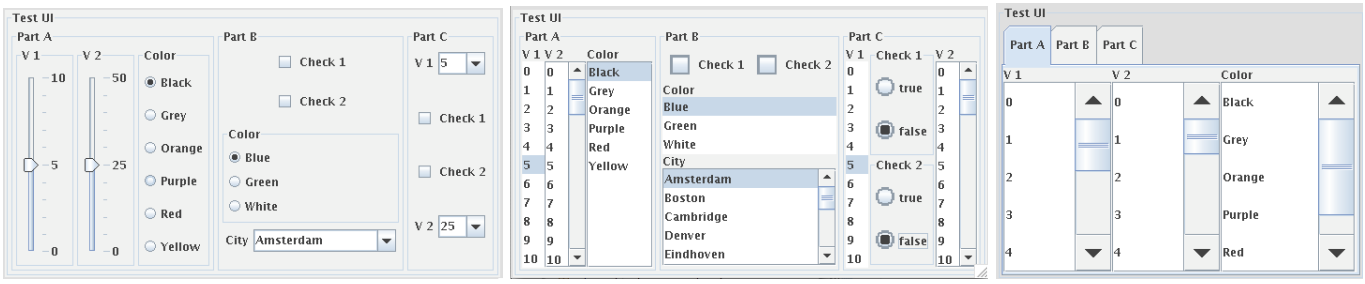


Figure 5: Three renderings of a synthetic interface used in the preliminary study and automatically generated under the same size constraint: (a) base line preference-optimized GUI; (b) personalized for the mouse user with muscular dystrophy (M03); (c) personalized for the eye tracker user (ET01). GUIs (b) and (c) were generated by SUPPLE++.

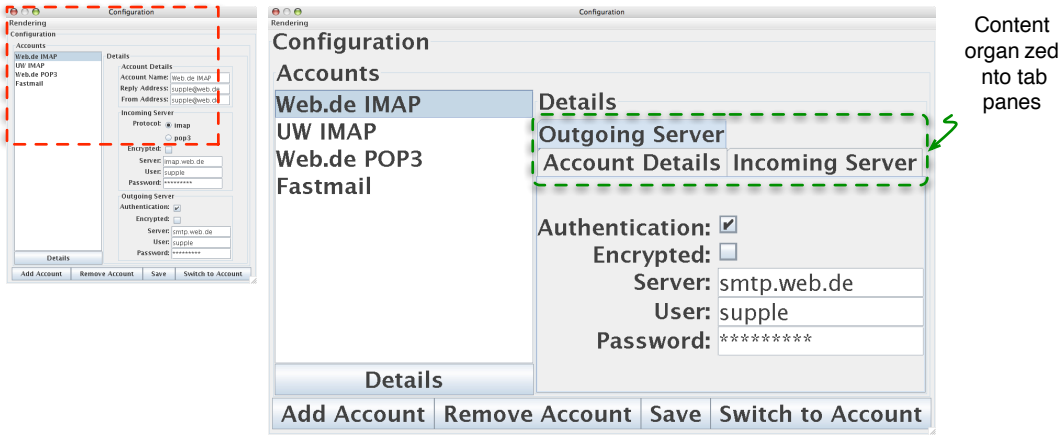


Figure 6: An email client configuration GUI automatically generated for a typical user (left) and for a low vision user (right) – both GUIs shown to scale. The latter interface allows the user to see the structure of the entire interface at a single glance. If a screen magnifier was used to enlarge the original interface to achieve the same font size, only a small fraction (marked with a dashed line) would have fit on the screen at a time.

been visible, forcing the user to explore the contents of the GUI serially rather than in parallel.

ADAPTING TO BOTH VISION AND MOTOR CAPABILITIES

We have now presented technical solutions for adapting GUIs to both a user's motor and vision abilities. We observe that our two approaches are orthogonal and combine trivially to provide custom GUIs for users who are atypical with respect to *both* their motor and vision abilities (refer to Figure 1 for an example). The only limitation being the fact that ability-optimized GUIs can take up to several minutes to generate so the instantaneous preview of interfaces generated for different cue sizes may not always be available so our system allows users to set a system-wide preference for visual cue size and that's the first value for which SUPPLE++ generates new interfaces.

EVALUATION

Algorithm Computation Time

For interfaces illustrated in Figures 1 and 5, SUPPLE++ needed between between 3.6 seconds and 20.6 minutes to compute the personalized GUIs for our participants. These results take advantage of our lower-bound estimation method for EMT_{nav}, which reduced the runtime for one of the less complex interfaces from over 5 hours to 3.6 seconds, and without which more complex interfaces would have required days to be rendered.

We note that execution times on the order of 10-20 minutes (in the worst case) will still allow practical deployment of the system if caching is used for users whose conditions do not change frequently.

Preliminary User Study

We conducted a preliminary study of our system with five of the participants who took part in the model eliciting study described above. We used a synthetic user interface designed so as to include many of the types of data commonly manipulated in standard GUIs. The functional specification included 11 leaf nodes: 4 numerical inputs (with different ranges and input accuracy requirements), 3 choice elements and 4 booleans. See Figure 5 for sample renderings.

Each participant was presented with two sets of automatically generated GUIs. Typically, one GUI in each set was generated using preference-based SUPPLE as a baseline (using preferences collected in earlier work [9]), one was optimized using SUPPLE++ for that user (the "personalized" GUI), and one or two others personalized for other participants for comparison. All GUIs in a set were rendered under the same size constraint (the "condition" column in Table 3)

participant	condition	SUPPLE++ (personalized)	SUPPLE (baseline)	personalized for others	
M03	small A	28.47	34.73	31.50 (ET01)	**27.45** (VJ02)
	full screen	**26.02**	30.04	26.88 (VJ02)	
ET01	small A	56.16	*not usable*	**48.03** (M03)	
	small B	59.04	*not usable*	**47.95** (M03)	57.99 (VJ02)
HM01	medium A	**29.96**	52.40		
	medium B	**29.94**	47.96		
TB01	small B	43.59	**42.52**	47.19 (M03)	43.44 (ET01)
	full screen	47.99	48.77	39.45 (VJ02)	**38.15** (ET01)
VJ02	small B	**55.55**	63.41	56.24 (M03)	61.52 (ET01)
	full screen	**52.76**	72.94	57.93 (M03)	70.45 (TB01)

Table 3: Study results (bold = fastest; underline = rated as easiest to use; VJ02 did not express a clear preference in one condition). SUPPLE++ allowed ET01 to complete tasks she was not able to accomplish at all with the baseline interface while for the remaining users it resulted in an average time savings of 20%.

and the sizes were chosen to accentuate differences among renderings within each set. The participants were not told until after the study how the interfaces were generated. With each GUI, participants performed 4 brief task sets, each requiring between 10 and 12 operations. The first task set was a practice set and was not included in the results. We were interested in expert performance, so the participants were led through the tasks by an animated visual guide, which limited the effect of visual search time and thinking time (for ET01, the experimenter read the instructions aloud for each operation). Participants performed the same tasks on all interfaces and the order of interfaces (personalized, baseline, others) was randomized. Participants were instructed not to use the keyboard during the study. Some of the concrete GUIs generated for the participants are shown in Figure 5 and Table 3 summarizes the results.

Results

On average, the personalized interfaces allowed participants to complete tasks in 20% less time than the baseline interface, with the time differences ranging from a slowdown of 3% to a speedup of 43%. In addition, ET01 was not even able to use the baseline interface because it required dragging, which her particular software configuration did not support. In 5 out of 10 conditions, participants were fastest using a personalized GUI, and in 6 out of 10 cases, they rated the personalized GUI as easiest to use.

In 4 cases, however, interfaces optimized for a *different* participant resulted in fastest performance pointing to limitations of our current performance model. We find that the current version of our model significantly underestimates the time necessary to manipulate those list widgets where only a small fraction of items is visible at a time because it does not take into account the visual verification time. We believe that this limitation is primarily responsible for the observed shortcomings and that it can be remedied in future versions.

Observations of individual participants HM01's (Head Mouse, incomplete tetraplegia) performance improved by up to 43% with respect to the baseline when using the personalized GUIs. His main difficulty in operating the baseline interface was caused by combo boxes which caused long lists with scroll bars to drop down – any accidental clicks while operating those lists would cause the combo box to collapse the list and force the participant to start over with that widget.

The particular software setup used by ET01 made it impossible for her to perform drag operations. This made the baseline GUIs unusable for this participant because they included sliders which, in the particular Swing implementation used, could not be set to a desired value with just a single click.

For both sets of interfaces, this participant preferred the personalized interfaces over the alternatives. However, she performed the slowest with those interfaces. The personalized interfaces included list widgets that showed fewer values at a time than the alternative GUIs. The time to manipulate those lists was much longer in practice than what our model predicted. For larger target sizes, both the heights and the widths of list cells were enlarged thus moving the values away from the slider toward the left of the list. That made it much harder for this participant to use her peripheral vision to monitor the values as she was using her gaze to click on the slider. She suggested that in some cases larger fonts for list would be beneficial because peripheral vision is naturally less acute.

TB01 (trackball, incomplete tetraplegia) was often able to work quickly with GUIs containing small targets, but he tired more quickly when using them and had to take longer breaks compared to when he was using GUIs with larger targets.

M03 (mouse, impaired dexterity) perceived the slider to be difficult to use and in the small condition ranked the VJ02 interface, which used it, as least easy to use even though she was most efficient using it. She was most satisfied with the ET01 interface, which used the largest targets (but which also required more scrolling to operate the lists and subsequently was second slowest to use). The baseline interface was the only one not to use tabs but despite being the easiest to navigate, it was slowest to use for this participant.

VJ02 (VocalJoystic, able-bodied) perceived the preference-optimized GUI as most aesthetically pleasing and most familiar looking and said that this probably made him perceive it as more usable than it perhaps was in practice.

Implications For Future Work

While personalized interfaces generated by SUPPLE++ were generally faster and easier to use than the baseline, the study suggested a few improvements to the system.

In particular, we will extend our model to better predict list selection times. The results of an initial follow-up study with TB01 and 4 additional able-bodied participants suggest that list selection times can be modeled accurately at the expense of adding additional tasks to the one-time motor performance elicitation test for each user. The strategies employed by the different participants varied widely so, just as with modeling pointing times, we have to automatically develop personalized models for each participant.

HM01's difficulty with combo box lists, which would disappear after an unintended or misplaced click, suggests that we should also explicitly model the cost of recovering from errors. It will affect the expected costs of individual widgets (those for which errors are costly to recover from will

become more expensive), as well as layout and spacing of interactors.

We are currently planning a larger study of this system. We will broaden the diversity of motor differences represented (in particular by recruiting participants with tremors and age-related impairments) and we will also evaluate our system's ability to adapt to low vision requirements and combinations of vision and motor impairments.

CONCLUSION

We have presented SUPPLE++, a system which automatically generates user interfaces adapted to the user's motor and vision capabilities. Noting the great diversity of abilities among users, our system does not rely on profiles of health conditions, which would necessary stereotype users into discrete classes, but instead it automatically adapts to the user's *individual* motor performance and allows manual adjustment to accommodate varying vision capabilities. Importantly, SUPPLE++ supports *combinations* of vision and motor impairments.

Our technical contributions include a method for automatically selecting features of a custom regression model for each user, noting that Fitts' law often does not adequately describe the performance of people with unusual motor abilities or devices. We also developed a novel optimization-based algorithm for automatically generating GUIs adapted to user's motor performance.

Results of a preliminary study indicate that while there is still room for improvement, SUPPLE++ shows promise as it allowed one user to complete tasks, which she could not perform using a standard interface while for the remaining users it resulted in an average time savings of 20%.

Acknowledgments We acknowledge the following individuals (in alphabetical order) for their advice, help in recruiting the participants and comments on this manuscript: Eytan Adar, Anna Cavender, Dan Comden, Susumu Harada, Mark Harniss, Curt Johnson, Kurt L. Johnson, Kayur Patel, Ali Rahimi, Joe Stuckey and Michael Toomim. Jing Jing Long helped develop early prototypes of the system. This research was funded in part by NSF grant IIS-0307906, ONR grant N00014-06-1-0147, DARPA project CALO through SRI grant number 03-000225, the WRF / TJ Cable Professorship and a Microsoft Graduate Research Fellowship.

REFERENCES

1. Accot, J. and S. Zhai. Refining Fitts' law models for bivariate pointing. *Proc. CHI'03*, 193–200, New York, NY, USA, 2003. ACM Press.
2. Anson, D. *Alternative Computer Access: A Guide to Selection*. Davis FA, 1996.
3. Bergman, E. and E. Johnson. Towards Accessible Human-Computer Interaction. *Advances in Human-Computer Interaction*, 5(1), 1995.
4. Carter, S., A. Hurst, J. Mankoff, and J. Li. Dynamically adapting GUIs to diverse input devices. *Proc. Assets'06*, 63–70, New York, NY, USA, 2006. ACM Press.
5. Chicowski, E. It's all about access. *Alaska Airlines Magazine*, 28(12):26–31, 80–82, 2004.
6. Dawe, M. Desperately seeking simplicity: how young adults with cognitive disabilities and their families adopt assistive technologies. *Proc. CHI'06*, 1143–1152, 2006.
7. Fichten, C., M. Barile, J. Asuncion, and M. Fossey. What government, agencies, and organizations can do to improve access to computers for postsecondary students with disabilities: recommendations based on Canadian empirical data. *Int J Rehabil Res*, 23(3):191–9, 2000.
8. Gajos, K. and D. S. Weld. Supple: automatically generating user interfaces. *Proc. IUI'04*, 93–100, Funchal, Madeira, Portugal, 2004. ACM Press.
9. Gajos, K. and D. S. Weld. Preference elicitation for interface optimization. *Proc. UIST'05*, Seattle, WA, USA, 2005.
10. Gajos, K., A. Wu, and D. S. Weld. Cross-device consistency in automatically generated user interfaces. In *Proceedings of Workshop on Multi-User and Ubiquitous User Interfaces (MU3I'05)*, 2005.
11. Gregor, P., A. Newell, and M. Zajicek. Designing for dynamic diversity: interfaces for older people. *Proc. Assets'02*, 151–156, 2002.
12. Grossman, T. and R. Balakrishnan. A probabilistic approach to modeling two-dimensional pointing. *ACM Trans. Comp.-Human Interaction (TOCHI)*, 12(3):435–459, 2005.
13. Gump, A., M. LeGare, and D. L. Hunt. Application of fitts' law to individuals with cerebral palsy. *Percept Mot Skills*, 94(3 Pt 1):883–895, June 2002.
14. Harada, S., J. A. Landay, J. Malkin, X. Li, and J. A. Bilmes. The vocal joystick:: evaluation of voice-based cursor control techniques. *Proc. Assets'06*, 197–204, New York, NY, USA, 2006. ACM Press.
15. Hastie, T., R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.
16. Hornof, A., A. Cavender, and R. Hoselton. Eyedraw: a system for drawing pictures with eye movements. *Proc. Assets'04*, 86–93, 2004.
17. Hwang, F., S. Keates, P. Langdon, and J. Clarkson. Mouse movements of motion-impaired users: a submovement analysis. *Proc. Assets'04*, 102–109, New York, NY, USA, 2004. ACM Press.
18. International Organization for Standardization. 9241-9 Ergonomic requirements for office work with visual display terminals (VDTs)-Part 9: Requirements for non-keyboard input devices, 2000.
19. Keates, S., P. Clarkson, L. Harrison, and P. Robinson. *Towards a practical inclusive design approach*. ACM Press New York, NY, USA, 2000.
20. Koester, H. Abandonment of speech recognition by new users. *Proc. RESNA'03*, 2003.
21. Kurniawan, S., A. King, D. Evans, and P. Blenkhorn. Design and user evaluation of a joystick-operated full-screen magnifier. *Proc. CHI'03*, 25–32, 2003.
22. Law, C., A. Sears, and K. Price. Issues in the categorization of disabilities for user testing. *Proc. HCI Intl.*, 2005.
23. Mace, R., G. Hardie, P. Jaine, and N. C. S. U. C. for Universal Design. *Accessible Environments: Toward Universal Design*. Center for Accessible Housing, North Carolina State University, 1990.
24. Mackenzie, S. I., and W. Buxton. Extending fitts' law to two-dimensional tasks. *Proc. CHI '92*, 219–226, New York, NY, USA, 1992. ACM Press.
25. Nichols, J., B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol. Generating remote control interfaces for complex appliances. *Proc. UIST'02*, Paris, France, 2002.
26. Phillips, B. and H. Zhao. Predictors of assistive technology abandonment. *Assist Technol*, 5(1):36–45, 1993.
27. Scott, I. U., W. J. Feuer, and J. A. Jacko. Impact of graphical user interface screen features on computer task accuracy and speed in a cohort of patients with age-related macular degeneration. *Am. J. of Ophthalmology*, 134(6):857–862, December 2002.
28. Sears, A. Layout appropriateness: A metric for evaluating user interface widget layout. *Software Engineering*, 19(7):707–719, 1993.
29. Smits-Engelsman, B., Rameckers, E., Duysens, and J. Children with congenital spastic hemiplegia obey fitts law in a visually guided tapping task. *Experimental Brain Research*, 177(4):431–439, March 2007.
30. Stephanidis, C. Towards the Next Generation of UIST: Developing for all Users. *HCI Intl.*, 473–476, 1997.