Dynamically Adapting GUIs to Diverse Input Devices

Scott Carter[1] and Amy Hurst[2] and Jennifer Mankoff[2] and Jack Li[2]
[1]EECS Department
UC Berkeley
sacarter@cs.berkley.edu
[2]HCI Institute
Carnegie Mellon University
akhurst,jmankoff,jackli@cs.cmu.edu

ABSTRACT

Many of today's desktop applications are designed for use with a pointing device and keyboard. Someone with a disability, or in a unique environment, may not be able to use one or both of these devices. We have developed an approach for automatically modifying desktop applications to accommodate a variety of input alternatives as well as a demonstration implementation, the Input Adapter Tool (IAT). Our work is differentiated from past work by our focus on *input adaptation* (such as adapting a paint program to work without a pointing device) rather than *output adaptation* (such as adapting web pages to work on a cellphone). We present an analysis showing how different common interactive elements and navigation techniques can be adapted to specific input modalities. We also describe IAT, which supports a subset of these adaptations, and illustrate how it adapts different inputs to two applications, a paint program and a form entry program.

Categories and Subject Descriptors

K.4.2 [**Computers and Society**]: Social Issues: assistive technologies for persons with disabilities; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces: input devices and strategies

General Terms

Design, Human Factors

Keywords

Accessibility, toolkits, interaction techniques

1. INTRODUCTION

Graphical user interfaces (GUIs) are typically designed for a specific set of input and output devices: A keyboard, a pointing device, a monitor, and perhaps speakers. GUIs are usually built using toolkits that provide generic interactive elements (interactors), such as buttons and menus. These interactors have been optimized over the years for use with a keyboard and pointer. However, the resulting applications lack flexibility when a user's input needs change.

Input needs may change when someone has a disability, or is using an application in an off-the-desktop environment [19]. For example, a person with a motor impairment may be constrained to single switch input, in which she must manipulate the entire interface with the equivalent of a one-key keyboard. A person interacting with a projected display in a meeting may have a pointing device but no keyboard. How would these users interact with a text editing system? A sketching program? A form entry application?

The most commonly used approach is to display a special interface that can translate user input into mouse and keyboard events or can translate displayed output into audio. The user interacts with that interface, which then sends information to the application the user wishes to control. For example, a soft keyboard that turns pointer input into keyboard events would allow someone to control an interactive projected display with a pointing device. Single switch users typically use a "scanning interface" that functions similarly to a soft keyboard, but can generate both keyboard and mouse events and send them to any desktop application. Because they are very general, these solutions allow a user to control any GUI. But, because they are very general, they are not optimized for the specific interactive elements of different GUIs, and as a result may be difficult or slow to use. For example, a scanning interface may not have any way to select a menu directly without moving the cursor pixel by pixel across the screen until it is over that menu.

To make an interface accessible, two problems must be solved. First, the user must be able to select (*navigate to*) any of the interactors in the interface. Second, the user must be able to *control* the selected interactor. We present an analysis of options for navigation (global to an application) and control (at the level of interactors such as menus and buttons), available for different input configurations. The demonstration tool we developed, the Input Adapter Tool (IAT), automatically modifies a GUI to handle a variety of input devices it was not designed for. IAT enables the user to perform both navigation and control actions. Navigation in IAT is supported globally, either by traversing the interactor hierarchy or by directly selecting an interactor. Control is supported by dynamically customizing an application with interactors that are specifically designed to be usable by the available input device. For example, in Figure 1, a paint application (top) has been modified for use

ASSETS'06, October 22–25, 2006, Portland, Oregon, USA.

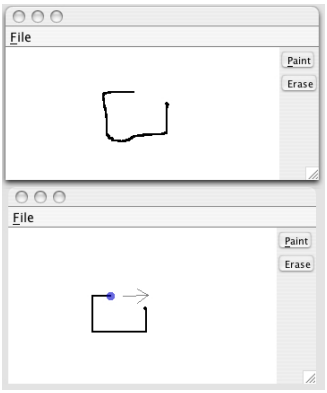


Figure 1: A paint program (top) adapted for switch input (bottom). The canvas has been augmented with a movable dot and a directional arrow.

	Keyboard Only	Pointing device	Switch	Speech
Logical	Scan through interactors with key control cO(N) Directly select interactors with tooltips cO(1)	Snap cursor to interactors cO(log(N))	Scan through interactors cO(N)	Speak tooltip or interactor name cO(1) Interactor scanning controlled by voice cO(N)
Spatial	Pixel scanning with arrow keys, cO(XY) Grid navigation with scanning cO(log(WH))	Direct selection cO(XY)	Pixel scanning: cO(XY) Grid navigation with scanning cO(log(WH))	Voice controlled pixel scanning cO(XY) Voice controlled grid navigation cO(log(WH))

N = Number of interactors
X = Screen width (pixels) W = Grid width (pixels)
Y = Screen height (pixels) H = Grid height (pixels)

Replacement Key

Custom	No Change	IAT Library

Table 1: Example methods for navigation of a graphical user interface organized by input technique. With logical navigation, the user selects an interactor directly or via a logical hierarchy. With spatial navigation, the user selects a location, either hierarchically or by moving from pixel to pixel. Navigation either requires a custom extension to IAT ("Custom"), is the default technique ("No Change"), or is supported by IAT ("IAT Library").

with single switch input (bottom). The user is controlling a drawing canvas interactor and drawing a circle with it. In the switch case (bottom), the interactor has been augmented to support single switch input by displaying an arrow representing the current direction that a special drawing token is moving. The switch can be used to change the direction of the token, or toggle the drawing mode on and off.

Our contributions include an examination of possible augmentations for different input devices as well as IAT, which supports a subset of input configurations, including pointing (no keyboard, *e.g.*, a mouse only), keyboard without pointing, single switch, and speech input. IAT also supports two kinds of navigation: direct selection of interactors, and a hierarchical traversal of the interactor hierarchy. IAT uses a lookup table to determine if any interactor substitutions are necessary. When an interactor is replaced, IAT keeps a copy of it off screen and passes information to it. For example, when the user has a keyboard but no pointer, IAT replaces a combobox (a list of items from which the user selects one) with a text entry plus a display of valid inputs that is filtered as the user types. When the user types an item that was in the combobox, IAT causes that item to be selected in the combobox, triggering any callbacks to the application. Finally, IAT can be extended in multiple ways. Users can edit the lookup table to change interactor replacement mappings or the navigation approach per input device. Developers can also add to the library of custom interactors.

1.1 Overview

We begin with an analysis of the design space for augmenting interfaces to support different input devices. We then describe IAT, a demonstration implementation that handles an important subset of those approaches, and we conclude with related work in input adaptation.

2. NAVIGATION AND CONTROL

We can describe the efficiency of a navigation and control technique using standard O-notation [14]. This allows us to compare the behavior of different techniques in the worst case. The accuracy of an O-notation increases as the number of inputs increase – at low inputs, constants can overwhelm inputs. For example, searching for a particular number in an unordered list of n numbers is $c_1O(n)$ whereas finding a number in an ordered list using binary search is $c_2O(log(n))$. For large n, $c_1O(n)$ is always greater than $c_2O(log(n))$. However, for small n, if c_2 is larger than c_1, $c_2O(log(n))$ may be larger. For navigation and control techniques, different impairments or contexts can effect O-notation constants.

As stated above, adaptations that are not tailored to an application can be highly inefficient. For example, a scanning interface may require a user to move the cursor across the screen pixel by pixel. This is highly inefficient (it is approximately $cO(X * Y)$, where X and Y are the width and height of the screen, in pixels, and c is the very large constant associated with each scanning selection). Other forms of input may also be inefficient. As another example, keyboard access to a web page is linear in the number of links on a web page, because the user typically must tab through them one by one to move down the page [15].

For an accessibility tool to adapt to the context of use, two problems must be solved: *navigation* to interactors and *control* of interactors. We analyze how both can be approached for a representative range of input devices.

In our analysis, we focus on four kinds of input: keyboard (no pointer); pointer (no keyboard, *e.g.*, only mouse); switch; and speech. A switch is an input device with very few states, such as a button, toggle, or simple neural input (*e.g.*, [17]), that is likely to be used by people with sever motor impairments. Speech is any spoken input recognition. Typically, a command and dictation system is used to enter text and control the computer.

We chose these four inputs for two reasons: First, they are all in use today, by people with disabilities. Second, they are representative of the design space. They range from very low bandwidth (switch) to high bandwidth (speech, keyboard); from error prone (speech) to highly accurate (others); and from already partially supported by existing applications (keyboard, pointing device) to not at all supported (switch).

2.1 Navigation adaptation techniques

Jul explored the topic of navigation extensively [12]. While Jul's work points to the fact that carefully designed navigation can improve both the usability and speed of navigation, our focus here is on approaches to navigation basic enough to be easily automated.

Navigation cannot be handled with interactor-level substitutions, but must be dealt with globally. This is because replacing one interactor does not help the user to interact with another one. Additionally, there is no way to know ahead of time what interactor the user might wish to interact with next. The user must be able to get from any interactor to any other interactor at will.

Navigation can be handled using two approaches – logical and spatial. With *logical navigation* the user can directly or via a logical hierarchy select an interactor. With *spatial navigation* the user can select a location, either hierarchically or by moving from pixel to pixel. An interactor is implicitly selected by its presence at that location. Pointing devices are good at fast spatial navigation. A keyboard shortcut such as `Tab` for moving from one link to the next in a web page is an example of logical navigation. Table 1 summarizes our analysis of these methods. Note that the constant c will vary significantly for different users and input devices. This does not affect the relative comparison of different approaches for a given input device/user. Additionally, there may be a start up cost, k, that makes the differences between approaches negligible for small n.

Most interfaces have implicit support for navigation (no one interface element codifies how the interface as a whole is navigated). Additionally, they expect navigation to be done with a combination of pointer input and keyboard input (including tabbing and mnemonics). Although pointing devices easily support navigation without modification, most other input devices are not optimized for selecting an arbitrary interactor.

2.1.1 Logical Navigation

For logical navigation, we assume there are n interactors between which the user wishes to navigate. There are three possible approaches:

Direct $O(1)$ For input devices that can generate at least n signals, a signal can be mapped onto each interactor. For example, if each interactor is given a name, a user of a speech recognition system could access any interactor simply by saying its name. One problem with the direct mapping approach is that there may be fewer input states than interactors that could get focus (*e.g.*, there may be more interactors than there are characters on a keyboard). We use the term "focus" to refer to the interactor that has been designated by the user to receive input, act on input, and so on. This would only be the case in the most complex interfaces. A mixed approach may be appropriate when this case arises.

Linear $O(n)$ When only a small number of signals are available, or direct navigation is too complicated (too many different mappings must be remembered), interactors may be ordered linearly. In this case, the user simply indicates that selection should move forward to the next interactor until the desired interactor is selected. This is far slower than direct navigation.

Hierarchical $O(log(n))$ Typically, user interfaces are organized hierarchically, with simple interactors (such as buttons) grouped into more complex units (such as toolbars). If navigation reflects this hierarchy it can be much faster than other approaches while still being fairly intuitive. For example, Edwards *et al.* used this approach in Mercator [9].

2.1.2 Spatial Navigation

For spatial navigation, we make similar assumptions, but also label the screen pixel width and height as X/Y and the grid pixel width and height as W/H. Again, there are multiple approaches:

Navigate the pixels of the screen [12] $O(X * Y)$ The user can directly move the pointer from pixel to pixel. This approach can be potentially very slow, depending on the cost of pixel to pixel movement. In cases where the speed is not constant (such as typical mouse use, which conforms to Fitts' law [10]), this may be as fast as $O(log(X * Y))$.

Hierarchical $O(log(W * H))$ The user can use a grid navigation approach, such as those typical of speech recognition systems [7]. Typically, a $3x3$ grid is overlayed on the screen. Selecting a cell causes a new grid to be overlayed on just that cell, and so on, until the cell is small enough to entirely overlap the interactor of interest, at which point the user indicates that some action should take place.

2.1.3 Navigation adaptation for different devices

Table 1 makes some suggestions regarding navigation approaches for different input configurations. In this section we address each input configuration: keyboard (no pointer), pointer (no keyboard), switch, and speech recognition.

There are two techniques best suited to navigating an interface without a *pointer*: direct mapping and tab-stop. In an interface with direct mapping, each key on the keyboard is dynamically mapped to an interactor in the interface. For example, five pulldown menus may be mapped to the letters 'a' through 'e' and a toolbar could be mapped to the letter 'f'. In tab-stop navigation, the user is able to "scan" through the interactors by pressing the "tab" key.

We argue that direct mapping is best with keyboard input because it is so fast ($O(1)$). A typical implementation would respond to a special command sequence by displaying a key in a tooltip near each interactor. Upon pressing an interactor's associated key, that interactor would be selected. In cases where there are more interactors than keys, a combination method would be appropriate. However, this approach may become less effective as the number of interactors increases. For example, tooltips may interfere with other interactors, it may not be clear to which interactor a tooltip refers, or there may not be screen space available to display every tooltip. A hierarchical scheme in which tooltips are progressively revealed could address these issues, but would be $O(log(n))$. An additional solution would be use the techniques of explored in Fluid Documents [6] which can make small, automatic adjustments to the layout of an interface to make space for supplemental information and to make sure it is visually salient.

In contrast, we argue that tab-stop navigation is best for *switch* input because it only requires a small number of dis-

	Keyboard Only	Pointing Device	Switch	Speech
Binary: Button, Checkbox, Radio Button	Press key to make selection	Direct manipulation	Toggle state until switch is pressed	Spoken keyword makes selection
Lists: Pulldown Menu, Listbox, Combobox	Navigate options with tool tips & arrow keys	Direct manipulation	Scan through options until switch is pressed	Spoken tooltip
Scale: Slider, Scrollbar	Arrow keys adjust value	Direct manipulation	Scan through values until selection is made	Spoken keywords control values
Text Field	Direct manipulation	On-screen keyboard with word prediction	Comboboxes with word prediction	Natural language
Drawing Area	Grid Selection: specify cell with keypress	Point and click on start/end points	Movable Dot: scans through directions.	Grid selection: speak cell names
Context Menu	Press multiple keys	Right mouse button	Option in multimodal menu	Spoken keyword opens menu

Replacement Key

Standard Interactor	Custom	No Change	IAT Library

Table 2: Example methods for adapting some of the most commonly used interactors organized by input method. Interactors may either be substituted with other common interactors ("Standard Interactor"), substituted with a custom interactor ("Custom"), may not require any substitution ("No Change"), or substituted with an interactor in the IAT library ("IAT Library").

tinct signals. One of the most common strategies for dealing with switch interfaces is to deploy a scanning interface. Typically, a mostly static set of "keys" (which may include some navigation controls such as arrow keys and some application-specific options) is shown on screen, similarly to a soft keyboard. Each option is highlighted in turn. The highlight "scans" through all available options, and the user needs to activate the switch to make a selection when the appropriate option is highlighted. One of the main problems with a scanning interface is scale, since it may take a long time to arrive at the object of interest ($cO(n)$ where c is proportional to the dwell time on each object, and n is proportional to the number of "keys"). We argue that hierarchical navigation is best used with switch input when enough distinct signals are available, because it is $cO(log(n))$. Only three differentiable signals are needed to navigate a hierarchy: "move to a sibling", "move down", and "select." These can be mapped to "up", "down", and "pause" in some single switch interfaces, for example.

Pointers are ideally suited to navigation, and naturally provide a simple and direct way of selecting onscreen elements. Most modern graphical user interfaces are designed with a pointing device in mind to control navigation. Thus, we recommend not changing the navigation of an interface when there is only a pointer.

Speech can be used to navigate an interface in three ways: mimicking a pointer, direct selection of interactors, and through voice controlled scanning. Research has demonstrated that sound and speech can simulate a pointer, enabling a user to gain pixel-level control over an interface. Examples include modulating the pitch of the voice or the syllables spoken to control a cursor [16] and recursively dividing the screen into grid regions which are spoken by the user [7]. Direct manipulation can be accomplished using the same tooltip technique that we discussed in a keyboard only interface. Instead of pressing the keyboard letter corresponding to the interactor, the user can simply say the letter (or word). Voice controlled scanning is another reasonable way to navigate an interface with speech. This can be accomplished by having the interface automatically scan the interactors and wait for the user to say a key phrase such as "stop" to make a selection, or by allowing the user to simulate "tab-stop" navigation through the keywords such as "next" and "back." Further study is needed to understand the approach most effective for speech navigation.

2.2 Control adaptation techniques

Most of today's GUIs make use of generic interactors provided by the interface toolkit they are based on, such as menus, buttons, text entry areas and so on. Those interactors can typically be controlled using either a keyboard, a pointer, or both. Our analysis suggests how standard library elements can be replaced with similar interactors that are tailored to the needs of specific input configurations.

We derived interactors listed in Table 2 from the Macintosh manual [1], and Foley's description of interaction tasks [11]. The set of interactors we analyzed covered all of those in the Mac manual except outline triangles (used to expand hierarchical trees) and are representative of Foley's set of six general interaction tasks which include select (binary and list interactors), orient (movable dot), path (drawing area), and text (scale interactors and grid drawing). The navigation interactions that we described in the previous section also cover the navigation tasks that Foley describes including position (spatial navigation), quantify (grid navigation), and select (logical navigation).

There are two approaches to adapting these interactors. Interactor *substitution* involves replacing an interactor with one that is conceptually different to enhance access. An example is substituting a menu for a text entry area that has a fixed set of valid inputs. Alternatively, some interactors may only require *augmentation*, not replacement, to function with different input devices. In this case, extra controls can be visually overlayed on top of the interactor and events can be passed through these controls to the interactor (similar to [8] and [2]). An example is adding word prediction to a text entry area.

2.2.1 Control adaptation for different devices

Given these two approaches, the next question is what specific substitutions or augmentations should be made for different input devices. Here we provide suggestions, summarized in Table 2.

All binary interactors, such as buttons, radio buttons, checkboxes, can be activated or toggled by a single key, once they have the focus ("enter" is intuitive because it is often used already). Adjusting more complex interactors so they can be controlled with only a *keyboard* can be done using "direct mapping:" The same direct technique used for navigation with a keyboard can be used for selecting options in pulldown menus and listboxes. Arrow keys can be used for interactors such as scrollbars. A canvas can be supported

using Kamel and Landay's recursive grid drawing method [13]. Context menus can be accessed by a special key combination. If a user is not able to type multiple keys at the same time, "sticky" keys can be deployed so that keys may be pressed sequentially.

Most interactors are designed to be controlled using a *pointing device* through direct manipulation. However, interactors that have text fields need to be modified to be controlled by a pointing device. The most common techniques to input text with a pointer are gestures (such as graffiti) or written text, or clicking on a soft keyboard. Gesturing and handwriting may be error prone, and require learning either on the part of the user or the machine. A soft keyboard is highly accurate, but takes up significant screen space and may be slower or less intuitive for the user than handwriting or gestures.

Once the user has navigated to an interactor, controlling it with a *switch* is relatively straightforward. An optimally efficient scanning interface can be designed for each class of interactors. For example, binary interactors are easy to control with a switch because the interface can repeatedly toggle between the states until the user makes a selection. Listboxes or menus can be controlled by repeatedly scanning through the contents until the user makes a selection. Text can be entered using a combination of word prediction and an alphabet scanner. Context menus can be supported by adding an extra element to any scanning interface that triggers the menu. Similar to keyboards, canvas drawing can be supported using Kamel and Landay's recursive grid drawing method [13].

When controlling interactors through *speech*, keywords or sounds can be mapped to keyboard presses so the system emulates the behavior of a keyboard. Keywords such as "accept", "up" and "down" can control many interactors. Text entr]y is best supported by leveraging recognition of natural language. Pixel level manipulations such as drawing can be accomplished using the grid selection technique that we discussed for navigation. In this example, the user would draw a line by specifying where the line started, and then where it ended.

3. IAT

We built IAT to demonstrate that adaptations such as those described above are feasible. IAT is designed to work with any application written using the Java Swing toolkit. The application developer is not required to implement any special interfaces or use any special objects. IAT automatically grabs a handle to the application's top-level window, and then traverses the interactor hierarchy of that window to determine if any adaptations are needed given the currently available input devices. IAT makes the following assumptions about each input configuration:

Switch IAT assumes that a switch is a toggle with three states: left, right, and rest.

Keyboard IAT assumes a standard keyboard.

Pointer IAT assumes a standard mouse.

Speech IAT assumes speech recognition of characters and key phrases that would be performed by an off-the shelf speech recognition application such as IBM's ViaVoice.

3.1 Examples

To illustrate how IAT affects the end user experience, we present two examples illustrating how two typical GUI applications can be modified by IAT.

3.1.1 Switch control of a paint program

The application shown in Figure 2(a) is a paint program. The top image shows the original application, while the bottom image shows a version augmented to support switch input (for this example, assume that this input has three states: up, down, and rest). Note that a dot and arrow appear overlayed on the augmented canvas interactor. In this case, the dot provides feedback about the drawing state of the canvas: drawing, moving without drawing or not the current focus (indicating that some other interactor, such as the file menu, is being controlled). The arrow provides feedback regarding the drawing direction. When a user moves the switch to the down state the arrow moves counterclockwise and stops when the user releases the switch and a timeout occurs. Then, when the user moves the switch to the up state the dot and arrow move across the screen, sending appropriate mouse events to the component below if the interactor is currently in the drawing state. When the user releases the switch again another timeout occurs, the dot and arrow stop moving and a selection panel appears (see Figure 3(c)). The user may use the selection panel to switch tasks from control to navigation. For example, when the user is interacting with a drawing canvas, the selection panel allows the user to toggle the draw state of the drawing canvas to navigate to the next interactor in the ordered traversal or to return to her previous task.

3.1.2 Keyboard control of form entry

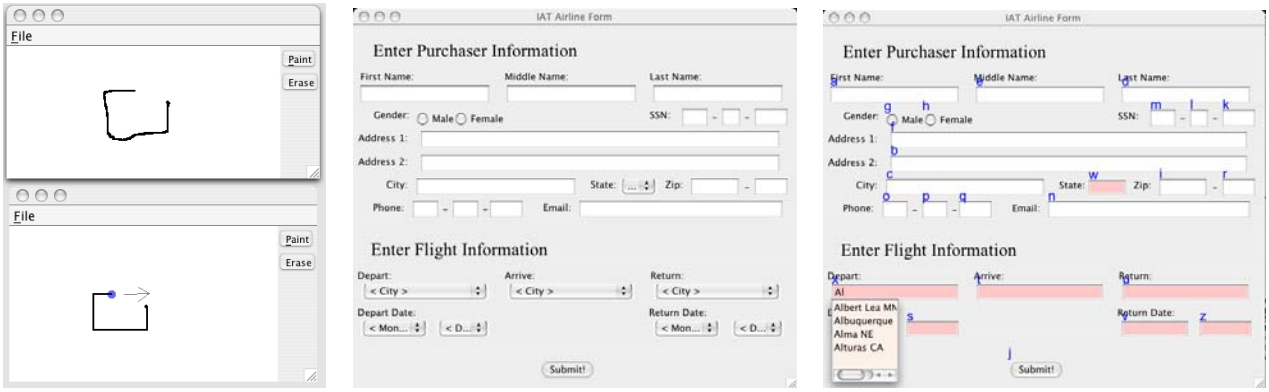
Figure 2(b) shows a form entry application as it was originally written. In Figure 2(c), the application has been modified to work with keyboard input. Note that the menu has been replaced with a textfield that automatically displays a list of possible selections given the current character input. Other interactors have not been modified. However, the user can directly navigate to any interactor by pressing its associated tool-tip key. When the user toggles tool-tips on, as they are in this example, a tool-tip key appears just above its associated interactor.

3.2 IAT adaptation support

While IAT does not support every cell shown in Table 2, we concentrated on supporting the full range of interactors for three common input devices (keyboard, pointer, and switch) and showing that other inputs (speech) are possible. We describe different types of input devices as well as the interactor replacement and augmentation and navigation augmentation we support. Note that our approach is flexible – users can both edit a lookup table to change interactor replacement mappings and add to the library of custom interactors.

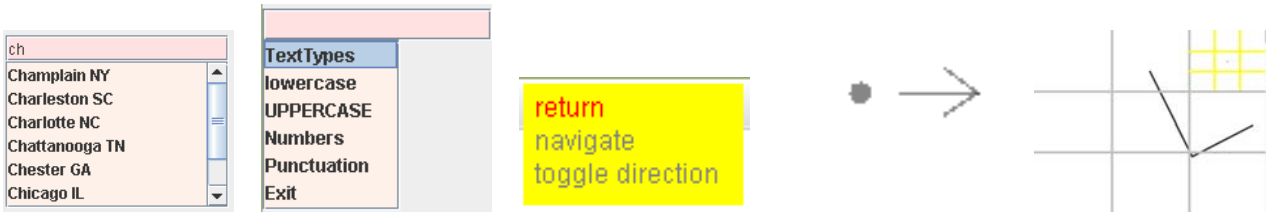
3.2.1 Navigation adaptation for different devices

For *keyboard* input, a special key command is defined that causes IAT to overlay a tooltip next to each interactor showing the character that can be used to access that interactor. IAT ensures that these are all unique from each other and from any existing key command bindings. IAT will cause an interactor to receive the "focus" when the user presses the associated key.



(a) (b) (c)

Figure 2: Adapting applications with IAT. (a) A paint program (top) adapted for switch input (bottom). The drawing canvas has been augmented with a movable dot and an arrow indicating direction. (b) A form entry program. (c) The same program adapted for keyboard input. Interactors that have been replaced in (c) have a darker (red) background. For example, the Departure combo box on the bottom left of (b) has been replaced with a predictive textfield in (c). A list of valid inputs, given what the user has typed, is shown below the textfield. Navigation is done using key-bindings, shown as blue tool-tips.



(a) (b) (c) (d) (e)

Figure 3: Example IAT adaptations. (a) For keyboard-only input IAT replaces a list with an interactor that prunes items based on entered text. (b) For switch input, IAT replaces a textfield with an interactor that allows the user to select specific characters one at a time. (c) For switch input, IAT uses a popup menu to allow the user to select between navigation or interaction. (d) For switch input, IAT augments canvases with a drawing dot and direction arrow that users can manipulate. (e) For speech input, IAT augments canvases with a grid-based selection tool (the black lines are on the canvas itself).

In the *pointer* case, no navigation adaptations are necessary. IAT does not augment navigation for *pointers*.

For *switch* input, each time the switch is in the *rest* state for longer than a set timeout, IAT temporarily overlays a selection list onto the interface (see Figure 3(c)). The user can use the list to indicate that he or she wishes to navigate or return to manipulating the interactor that currently has the focus. In navigation mode, IAT removes the selection list and the user can move the switch *left* to navigate the interactor hierarchy and *right* to select an interactor. When the user returns to the *rest* state the selection list returns and the user can indicate that he or she wants to manipulate the selected interactor or wants to navigate to a different interactor.

For *speech* input, IAT assumes a high-quality speech recognizer capable of disambiguating between characters and augments the interface using tooltips.

3.2.2 *Control adaptation for different devices*

For *pointer* input, IAT augments textfields with a standard soft keyboard.

For *switch* input, IAT replaces textfields with menu-based text entry systems (comboboxes, see Figure 3(b)). We chose to use a combobox instead of a soft keyboard because the combobox's selection maps *directly* onto the switch: up is up and down is down. We believe this direct mapping could facilitate adoption by users. Canvases are augmented with a movable dot that can send mouse events as well as an arrow indicating the current direction of movement (see Figure 2(a)). IAT also augments sliders, passing switch events onto the interactor to change the slider's value directly.

For *keyboard* controls, IAT replaces comboboxes with predictive textfields augmented with information about the valid inputs (see Figure 3(b)). For example, in Figure 2(c), IAT has replaced the "Depart" combobox on the lower left with a predictive textfield. When the user types 'A' 'l', valid

entries beginning with "Al" are displayed below the interactor ("Albert Lea MN", ...). Lastly, IAT augments canvases with a movable dot that can be controlled with the keyboard arrow keys. The dot sends mouse events to the canvas as it moves.

Finally, for *speech* controls IAT overlays canvas interactors with a grid that can be used to move the mouse (see [7], Figure 3(e)).

4. RELATED WORK

Our analysis can be differentiated from past work analyzing the space of input devices (such as [5, 3]) in our consideration not only of physical manipulators such as the keyboard and mouse, but also virtual, error-prone "manipulators" such as speech recognition. Also, our focus is not on the properties of input devices *per se*, but rather on the ability of the user to leverage those properties, given the context of use, to achieve efficient access to an application.

As stated in the introduction, the most commonly available input adaptation tools are usually not highly tailored to the application in use. Probably the most flexible piece of technology currently available to most people with disabilities affecting keyboard input is the WiVik® virtual keyboard (`http://www.wivik.com`). While WiViK® is customizable, typically this would be done once, up front, by an assistive technology professional, and would not typically change as the user interacts with different applications. Also, Mankoff *et al.* developed web browsing support for people with single switch input capabilities [15]. This solution was general with respect to the web browser, but could not support other types of applications or input devices.

Some systems provide integrated support for commonly used applications such as Microsoft Office. Examples include the most sophisticated speech recognition packages (*i.e.* Dragon Naturally Speaking (`http://www.dragontalk.com`) and IBM's ViaVoice), which provide speech-specific support for navigation and have integrated support for commonly used applications. In the research literature, the Pebbles project provides similar top quality integrated support for using a PDA as a pointer and text entry device [18]. In addition to directly controlling applications, users of Pebbles can directly control the cursor using the PDA stylus, or generate key events using Graffiti®.

Researchers are also exploring approaches to dynamically adapting to the context of use. Pebbles has been extended to provide customized access to home appliances [20]. The XWeb project provides dynamic adaptation for a variety of applications, and does it across several different input devices including keyboard and mouse, pen, laser pointers [21], and speech [22]. XWeb is actually an interface specification language, and therefore only works with applications written using that language. Input device support is handled by implementing different clients for each set of input devices. Clients render an application using interactors that are customized for its input (and output) devices. The Archimedes project and the Total Access System have also provide a flexible system that is able to adapt to multiple input devices [23]. An approach taken by Wang and Mankoff is to provide general support for arbitrary mappings between input devices [24]. However, this approach does not adapt application interfaces to better match the needs of the user's input device and may therefore be more difficult to use.

Other work has investigated automatic adaptation of an interface to different *output* devices or modalities. Although work in output adaptation typically focuses on modifying the interface of a GUI for audio display or to fit on a smaller screen, this also entails providing support for alternate input devices. For example, the Mercator system could automatically transform any GUI written in XWindows to audio [9]. Mercator was solving the specific problems faced by someone with no monitor (or no ability to see a monitor), and therefore no pointer, but full keyboard access. To solve this problem, Mercator also had to provide keyboard access to the GUI. It supported hierarchically-based navigation of arbitrary GUIs, rendering relevant information to the user with a mixture of non-speech audio and spoken text. Once the user navigated to a given interactor, she could invoke its functionality with a single key press. Mercator's solution shares much with IAT: the user is interacting directly with an application, yet the tool that facilitates this is general in the sense that it can support access to any application.

Also, several systems have explored transforming Web site interfaces designed for desktop machines to be more appropriate for PDA interfaces (see [15] for a summary of this work). For example the Power Browser system [4] supports Web page access from a cellphone or Personal Digital Assistant (PDA). The authors focus their efforts on output. They use interactors customized for stylus-based input in the PDA, and do not report on a working input solution for the cellphone. In the cellphone case, they propose to support navigation by numbering lines and allowing the user to jump to a line by pressing the corresponding number. This is similar in spirit to the second form of navigation we support, direct access rather than logical.

Our work differs from past work in two ways. First, we present an analysis that looks at both navigation and control, and is general and applicable to a wide variety of interfaces and input devices, including those that are error prone. Second, our tool, IAT, is backwards compatible with any application written in Java. Given a specification for the user's input capabilities, and an application, it makes the necessary modifications.

5. CONCLUSION AND FUTURE WORK

We have presented an analysis of issues of navigation and control that arise as a user's input configuration changes. Our analysis includes suggested solutions and recommendations for different input configurations. We implemented a tool, IAT, on the basis of our analysis. While IAT does not completely cover the space of adaptations suggested in our analysis, it represents a validation of the fact that our analysis can be used to structure adaptation. IAT improves the accessibility of all of the standard interactors across different input types: keyboard, pointer, switch, and speech. These improvements can allow someone with a disability or in a unique environment to access existing applications with input devices that those applications were not originally designed to handle.

In future work, we plan to expand the library of substitutions and modifications available in IAT. In particular, we plan to extend support for speech input and add support for gesture recognition for pointer based text entry.

6. ACKNOWLEDGEMENTS

We thank Scott Hudson, Anind Dey, and Nick Molchanoff. This work was funded in part by IBM and NSF IIS-0511895.

7. REFERENCES

[1] Apple Computer Inc. *Macintosh Human Interface Guidelines*. Addison-Wesley Professional, 1982.

[2] L. Berry, L. Bartram, and K. S. Booth. Role-based control of shared application views. *Proceedings of UIST*, pages 23–32, 2005.

[3] W. A. S. Buxton. A three-state model of graphical input. *Proceedings of INTERACT*, pages 449–456, 1990.

[4] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd. Power browser: Efficient web browsing for PDAs. *Proceedings of CHI*, pages 430–437, 2000.

[5] S. K. Card, J. D. Mackinlay, and G. G. Robertson. A morphological analysis of the design space of input devices. *ACM Transactions on Information Systems*, 9(2), pages 99–122, 1991.

[6] B.-W. Chang, J. D. Mackinlay, P. T. Zellweger, and T. Igarashi. A negotiation architecture for fluid documents. *Proceedings of UIST*, pages 123–132, 1998.

[7] L. Dai, R. Goldman, A. Sears, and J. Lozier. Speech-based cursor control: a study of grid-based solutions. *Proceedings of ASSETS*, pages 94–101, 2004.

[8] J. Dan R. Olsen, S. E. Hudson, T. Verratti, J. M. Heiner, and M. Phelps. Implementing interface attachments based on surface representations. *Proceedings of CHI*, pages 191–198, 1999.

[9] W. K. Edwards and E. D. Mynatt. An architecture for transforming graphical interfaces. *Proceedings of UIST*, pages 39–47, 1994.

[10] P. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6), pages 381–391, 1954.

[11] J. D. Foley, V. L. Wallace, and P. Chan. The human factors of computer graphics interaction techniques. *IEEE Computer Graphics Applications*, 4(11), pages 13–48, 1984.

[12] S. Jul. *From Brains to Branch Points: Cognitive Constraints in Navigational Design*. PhD thesis, University of Michigan, 2004.

[13] H. M. Kamel and J. A. Landay. Sketching images eyes-free: A grid-based dynamic drawing tool for the blind. *Proceedings of ASSETS 2002*, pages 33–40, 2002.

[14] D. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley Professional, 1982.

[15] J. Mankoff, A. K. Dey, U. Batra, and M. Moore. Web accessibility for low bandwidth input. *Proceedings of ASSETS*, pages 17–24, 2002.

[16] Y. Mihara, E. Shibayama, and S. Takahashi. The migratory cursor: accurate speech-based cursor movement by moving multiple ghost cursors using non-verbal vocalizations. *Proceedings of ASSETS*, pages 76–83, 2005.

[17] M. Moore, P. Kennedy, E. Mynatt, and J. Mankoff. Nudge and shove: frequency thresholding for navigation in direct brain-computer interfaces. *Proceedings of CHI*, pages 361–362, 2001.

[18] B. A. Myers. Using hand-held devices and PCs together. *Communications of the ACM*, 44(11), pages 34–41, 2001.

[19] A. Newell and P. Gregor. Human computer interaction for people with disabilities. *Handbook of Human-Computer Interaction*, pages 813–824. Elsevier Science Publishers, 1988.

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

[21] D. R. Olsen and T. Nielsen. Laser pointer interaction. *Proceedings of CHI 2001*, pages 17–22, 2001.

[22] D. R. Olsen, Jr., S. Jefferies, T. Nielsen, W. Moyes, and P. Fredrickson. Cross-modal interaction using XWeb. *Proceedings of UIST 2000*, pages 191–200, 2000.

[23] N. G. Scott and I. Gingras. The Total Access system. *Proceedings of CHI*, pages 13–14, 2001.

[24] J. Wang and J. Mankoff. Theoretical and architectural support for input device adaptation. *Proceedings of CUU*, pages 85–92, 2003.