Dynamically Adapting GUIs to Diverse Input Devices

Scott Carter, Jennifer Mankoff, Jack Li and Nick Molchanoff
University of California, Berkeley
{sacarter,jmankoff,jack,nkm}@cs.berkeley.edu

ABSTRACT

Much past effort has investigated how to adapt a user interface to a different output device or modality. Far less attention has focused on the adaptations needed to handle a different input device or modality. We present a tool for automatically modifying a graphical user interface to enable or improve the use of a variety of input devices for which the interface was not designed. This can allow someone with a disability or in a unique environment to use existing applications that were previously inaccessible.

KEYWORDS: Accessibility, toolkits, interaction techniques

INTRODUCTION

Graphical user interfaces (GUIs) are typically designed for a specific set of input and output devices: A keyboard, a mouse, a monitor, and in some cases, speakers. GUIs are typically built using toolkits that provide generic interactive elements (interactors), such as buttons and menus, that can be easily manipulated and seen with those devices. These toolkits make it easy to create interfaces out of interactors that are easy to use with a keyboard and mouse, but 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 [6]. For example, a person with a motor impairment may be constrained to single switch input, in which she has no mouse, and can only control the equivalent of a one-key keyboard. Or someone interacting with a projected display in a meeting may have a mouse or pen but no keyboard. How would these users interact with a web browser? A text editing system? A sketching program? A form entry application?

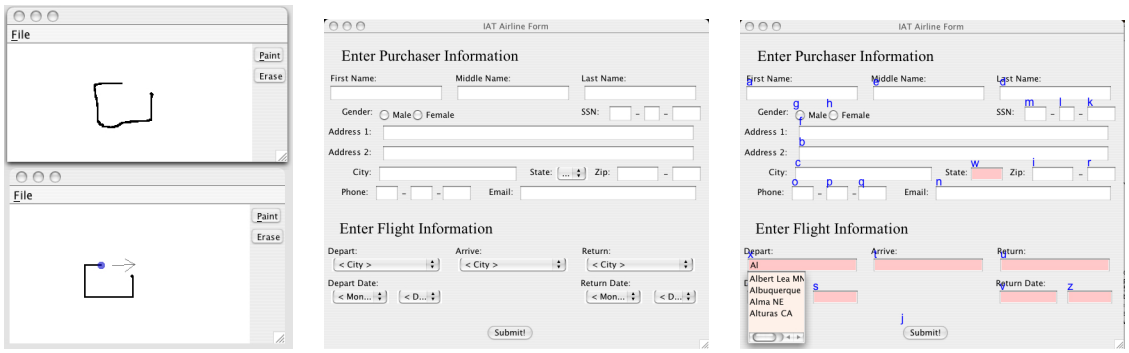
The most commonly used approach is to display a special interface that can translate user input into mouse and keyboard events, or displayed output into audio. The user interacts with that interface, which then sends information to the application the user wishes to control. For example, a soft keyboard that turns mouse input into keyboard events would allow someone to control an interactive projected display with a pen or mouse. 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 needs 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 mouse cursor pixel by pixel across the screen until it is over that menu.

We present IAT (Input Adapter Tool), which automatically modifies a GUI to handle a variety of input devices it was not designed for. 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. IAT allows the user to do both navigation and control. 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 Fig. 1(a), a paint application (top) has been modified for use 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 drawing mode on and off.

IAT currently supports three input configurations: mouse but no keyboard, keyboard but no mouse, and single switch input. However, our contribution lies not in the specific input devices we support, but in the approach we take. We support two kinds of navigation: Direct selection of interactors, and a hierarchical traversal of the interactor hierarchy. Additionally, we use a lookup-table to determine if any interactor substitutions are necessary. When an interactor is replaced, we keep a copy of it invisibly off screen and pass information to it. For example, when the user has a keyboard but no mouse, we replace 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, we cause that item to be selected in the Combobox. This triggers any callbacks to the application.

Overview

We begin by presenting related work in the domain of GUI adaptation. Next we describe the implementation of IAT, and illustrate it with an example. We end with a discussion of the pros and cons of our approach, and future directions that we would like to take.



(a) **(b)** **(c)**

Figure 1: Adapting applications. **(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 (only) input. Interactors that have been replaced in **(c)** have a darker (red) background and tool-tips showing key-bindings are shown in blue near each interactor. 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.

RELATED WORK

Adaptation of applications to handle unplanned input and output needs is an important problem. It can help to meet the needs of users with disabilities, and it is important in the ubiquitous computing world, for example in providing access to standard applications on small screen devices. Some common commercially available adaptations include soft keyboards (which let a user click on buttons that generate keyboard events), screen readers (which read the screen out loud, line by line), speech programs such as DragonDictate™(which allows a user to control applications with spoken commands), and scanning interfaces (which let a user with a switch simulate mouse and keyboard events). In addition, a variety of research is being done in making such adaptations more general (across applications or devices), or more usable (focusing on a specific problem such as audio access for the blind). Related work in this area can be broken up into two categories: Automatic adaptation of an interface to a different output device or modality, and support for a different input device or modality.

Output adaptation

Although work in output adaptation focuses on modifying the interface of a GUI for audio display or to fit on a smaller screen, this typically also entails providing support for alternate input devices. For example, the Mercator system could automatically transform any GUI written in XWindows to audio [2]. Mercator was solving the specific problems faced by someone with no monitor (or no ability to see a monitor), and (therefore) no mouse, but full keyboard access. To solve this problem, Mercator also had to provide keyboard-only access to the GUI. It supported hierarchically-based navigation of a GUI, 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 it's 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.

Several systems have explored transforming Web site interfaces designed for desktop machines to be more appropriate for PDA interfaces (see [4] for a summary of this work). For example the Power Browser system [1] supports Web page access from a cellphone or Personal Digital Assistant (PDA). The authors focused their efforts on output. They use widgets 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 hierarchical.

Input adaptation

Work in input adaptation has most often been inspired by the world of ubiquitous computing. For example, the Pebbles project [5] is an exploration of how a PDA (typically a PalmPilot™) can be used to control other devices including other PDAs, appliances, and desktop computers. In the case of desktop computing, Pebbles supports the use of a PalmPilot™to interact with specific applications (including *e.g.* PowerPoint™) using custom controls, or to control the mouse and keyboard (and through them any application). In the latter case, Graffiti™is used to generate keystrokes, and the stylus is used to control the mouse. Pebbles is an example of an application independent solution that can provide both general access to a desktop [5], and customized access to home appliances [7]. However, it is designed with assumptions about available input devices (stylus-based input via a PDA).

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 [8]; and speech [9]. 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. The clients render the application using interactors that are

customized for its input (and output) devices.

In earlier work, we developed web browsing support for people with limited input capabilities [4]. Our solution was general with respect to web browser and pages, but could not support other types of applications or input devices. We also developed an adaptation tool that could handle arbitrary mappings between input devices [10]. However it did not adapt application interfaces to better match the needs of the user's input device.

Summary

On both the input and output side, researchers are developing tools that can support automatic adaptation. Typically these tools focus on adapting an application to a specific input device (such as a PDA [5, 1], or switch [4]), output device (such as audio [2]), or application (such as a web browser [4, 1]). IAT builds on this past work, but is designed to generalize across both applications and input devices.

IMPLEMENTATION

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 currently supports three different input configurations: keyboard (no mouse), mouse (no keyboard) and switch.

IAT provides two sets of adaptations, described in the next two subsections: First, IAT finds any interactors that depend on input devices that are not available. It dynamically replaces those interactors with interactors that are accessible. Second, in the absence of mouse input, IAT provides the user with a way to navigate to any given interactive screen element. In combination, this allows the user to navigate to and control any interactors in the interface.

Interactor substitution

Most GUIs today 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 be controlled using either the keyboard, the mouse, or both. IAT makes applications constructed with standard library elements more accessible, by augmenting them or replacing them with similar interactors that are customized for the available input devices.

To identify interactors that require modification, IAT traverses the interactor hierarchy of a running application, making note of all of the interactors. It uses a lookup table to determine when modifications are needed. IAT supports the following types of modifications:

Replacement: When replacing an interactor, IAT removes the original interactor from the interface hierarchy, and replaces it with a modified, accessible interactor that is as similar as possible to the original. IAT gives the user feedback about substitutions by highlighting substituted interactors in light red. For example, in Figure 1(b), IAT substitutes a predictive Textfield, which only requires keyboard input, for a Combobox, which requires mouse input. When the user interacts with a substitute, IAT informs the original interactor (which is hidden, but still exists) by calling the methods that would be called were the user able to interact with it directly. This sets off any callbacks or other connections to the application.

Augmentation: When augmenting an interactor, IAT intercepts events at the top level of the application frame, visually overlays controls on top of the interactor, and saves state about those controls (*e.g.* in Figure 1(a), IAT overlays a dot and an arrow on top of the paint canvas). Those controls then pass events on to the underlying interactors.

By default, IAT makes the following types of substitutions and augmentations:

Switch Textfields are replaced with menu-based text entry systems (Comboboxes). We chose to use a Combobox instead of a soft keyboard because Combobox 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 an arrow indicating the current direction of movement (see Figure 1(a)).

Keyboard only Comboboxes are replaced with Predictive Textfields and augmented with information about the valid inputs. For example, in Figure 1(c), a Predictive Textfield has replaced the "Depart" Combobox on the lower left. When the user types 'A' 'l', valid entries beginning with "Al" are displayed below the interactor ("Albert Lea MN", ...). Canvases are augmented with a movable dot that can send mouse events .

Mouse only Textfields are augmented with a soft keyboard.

Note that our approach is flexible, and could handle alternative substitutions (for example, we could augment a Textfield with a Graffiti recognizer instead of a soft keyboard without changing our underlying architecture).

Navigation

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 mouse input and keyboard input (including tabbing and mnemonics). Although the mouse easily supports navigation without modification, most other input devices are not optimized for selecting an arbitrary screen location to interact with an interactor.

Thus, navigation cannot be handled with interactor-level substitutions, but must be dealt with globally. There are two possible approaches to supporting navigation – a hierarchical interface traversal, such as that supported by Mercator, and a direct mapping, where each interactive element is accessible by a unique key, keyword, or other signal from the user. We support both types of navigation. However, we found that hierarchical navigation was best used with switch input because switches can generate so few signals, and direct mapping was best used with keyboard input because most interfaces have a limited number of interactors. Thus, for switch input we support a hierarchical interface traversal, and for keyboard input we support direct mapping.

Our support for hierarchical navigation takes two forms: Where the user can generate enough unique signals, we support direct tree navigation (one symbol for horizontal motion in the tree, another to move up or down the hierarchy). Where the user is limited to a single input signal (as with single switch input), we generate an ordered, depth-first traversal of the tree that the user can move through.

Our support for direct mapping makes use of interactor key-bindings. With the help of the Java accessibility infrastructure, IAT discovers any existing key-bindings for each interactor. It then creates key-bindings for selecting interactors that do not conflict with those existing bindings. The user can cause all key-bindings to be displayed as tool-tips near the interactor they select by pressing a pre-specified key.

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). This would only be the case in the most complex interfaces. We handle that case by homogeneously pruning the set of interactors to which to assign tool-tips. We assume users will navigate to non-tool-tip-assigned interactors from nearby, tool-tip-assigned interactors using standard navigation techniques (*e.g.* tabbing).

Summary

In summary, IAT supports four specific adaptations intended to make applications more accessible to specific input configurations: An interactor may be *replaced* by one that is more accessible or *augmented* with additional graphics or control characteristics. An interactor may be selected by *hierarchical navigation* or by a *direct mapping* using key-bindings.

EXAMPLES

This section describes how the adaptations supported by IAT play out in practice by focusing on two specific cases: Switch control of a paint program, and Keyboard control of a form entry application.

Switch control of a paint program

The application shown in Figure 1(a) is a paint program. Note that a dot and arrow appear overlayed onto the canvas interactor. In this case, the dot provides feedback about the drawing state of the interactor: drawing, moving without drawing or 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 begins moving 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 begin to 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 and another timeout occurs the dot and arrow stop moving and a selection panel appears. 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, navigate to the next component in the ordered traversal or returning to her previous task.

Keyboard control of form entry

Figure 1(b) shows the original, unmodified form entry application. In Figure 1(c), the application has been modified to work with keyboard (no mouse) 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.

CONCLUSION AND FUTURE WORK

IAT automatically modifies a graphical interface to enable or improve the use of a variety of input devices for which the interface was not designed. Specifically, IAT improves the accessibility of all of the standard interactors across three different input types: keyboard (no mouse), mouse (no keyboard) and a switch. 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 on the library of substitutions and modifications available in IAT. In particular, we plan to add support for speech-only input. We also plan to add support for gesture recognition for mouse-based text entry, and word prediction for Textfields. An alternate approach to non-mouse (*e.g.* keyboard or switch) drawing that we plan to support is Kamel's recursive grid drawing method [3].

REFERENCES

1. O. Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd. Power browswer: Efficient web browsing for pdas. In *Proc. of CHI 2000*, pages 430–437, 2000.
2. W. K. Edwards and E. D. Mynatt. An architecture for transforming graphical interfaces. In *Proc. of UIST 1994*, pages 39–47, 1994.
3. H. M. Kamel and J. A. Landay. Sketching images eyes-free: A grid-based dynamic drawing tool for the blind. In *Proc. of ASSETS 2002*, pages 33–40, 2002.
4. J. Mankoff, A. K. Dey, U. Batra, and M. Moore. Web accessibility for low bandwidth input. In *Proc. of ASSETS 2002*, pages 17–24, 2002.
5. B. A. Myers. Using hand-held devices and pcs together. *CACM*, 44(11):34–41, 2001.
6. A. Newell and P. Gregor. Human computer interaction for people with disabilities. In *Handbook of Human-Computer Interaction*, pages 813–824. Elsevier Science Publishers, 1988.
7. J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol. Generating remote control interfaces for complex appliances. In *Proc. of UIST'02*, pages 161–170, 2002.
8. D. R. Olsen and T. Nielsen. Laser pointer interaction. In *Proc. of CHI 2001*, pages 17–22, 2001.
9. D. R. Olsen, Jr., Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal interaction using xweb. In *Proc. of UIST 2000*, pages 191–200, 2000.
10. J. Wang and J. Mankoff. Theoretical and architectural support for input device adaptation. In *Proc. of CUU'03*, pages 85–92, 2003.