Interaction techniques for ambiguity resolution in recognition-based interfaces

Jennifer Mankoff[1], Scott E. Hudson[2], and Gregory D. Abowd[1]

[1] College of Computing & GVU Center
Georgia Institute of Technology
Atlanta, GA 30332-0280
{mankoff,abowd}@cc.gatech.edu

[2] HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213
hudson@cs.cmu.edu

ABSTRACT

Because of its promise of natural interaction, recognition is coming into its own as a mainstream technology for use with computers. Both commercial and research applications are beginning to use it extensively. However the errors made by recognizers can be quite costly, and this is increasingly becoming a focus for researchers. We present a survey of existing error correction techniques in the user interface. These *mediation* techniques most commonly fall into one of two strategies, repetition and choice. Based on the needs uncovered by this survey, we have developed OOPS, a toolkit that supports resolution of input ambiguity through mediation. This paper describes four new interaction techniques built using OOPS, and the toolkit mechanisms required to build them. These interaction techniques each address problems not directly handled by standard approaches to mediation, and can all be re-used in a variety of settings.

INTRODUCTION

Because of its promise of natural interaction, recognition is coming into its own as a mainstream technology for use with computers. Recognition is being used in personal assistants such as the PalmPilot™, as well as on the desktop (*e.g.* IBM's ViaVoice™). Research initiatives in areas such as multimodal computing are investigating how to build effective, usable applications involving recognizers.

However, the errors made by recognizers can be quite costly and annoying for users to correct, and this is increasingly becoming a focus of research [7,11,23,28,30] For example, when studying a speech dictation system, Halverson *et al*. found that input speeds decrease from the 120 words per minute (wpm) of conversational speech to 25 wpm due in large part to time spent correcting recognition errors [11]. These errors are corrected through explicit user interaction. For example, a user can delete misrecognized words and then repeat them.

This *repetition* of input is one of the two common classes of interaction techniques for correcting recognition errors. The other common class gives the user a *choice* of different possible interpretations of her input. We call repetition and choice *mediation techniques* because they are mediating between the user and the computer to specify the correct interpretation of the user's input. Tables 1 and 2 show some of the variations of both repetition and choice techniques that we found in the literature. Figure 1 shows an example of a hybrid mediator from IBM's ViaVoice™ system. It provides a choice technique (an *n*-best list) with an escape path to a repetition technique. Other commercial systems such as the Apple MessagePad™ and DragonDictate™ provide similar hybrids. Choice and repetition strategies have a fairly wide range of possible instantiations, making mediation techniques ripe for reusable toolkit-level support.

In general, the goal of perfect recognition is difficult to achieve because correct recognition is best defined as what the user intends. Since a system cannot know this *a priori*, we model possible interpretations of user input internally as *ambiguous* input. Mediation techniques then serve to resolve this ambiguity, helping to determine which of those potential interpretations is the correct one, the one the user intended. In order to do this properly, integrated architectural support for ambiguity at the input-handling level is required. This makes it possible to track which interactors use ambiguous information and will need to be

Figure 1: An n-best list from the ViaVoice™ speech system [3]. Note that it provides a text entry area for mediation by repetition. Illustration reprinted with permission from IBM Corporation.

notified when interpretations are accepted or rejected. With such integrated support, we can treat all events the same way whether they are generated by a recognizer, sensor, mouse, or keyboard. We have developed a toolkit, called the Organized Option Pruning System (OOPS) that provides these features, introduced in [19,20].

The important contribution of OOPS is the separation of recognition and mediation from application development. The separation of recognition leads to the ability to adapt mediation to situations which do not seem to be recognition-based, but where some problem in interaction causes the system to do something other than what the user intended (our ultimate definition of error). The separation of mediation allows us to develop complex mediation interactions independent of both the source of ambiguity and the application. It also allows us to defer mediation for arbitrary periods when appropriate. OOPS also provides hooks that facilitate application-specific mediation in situations that benefit from specific knowledge to do the right thing, such as dealing with errors of omission.

In addition to an architecture, OOPS includes a library of mediators that fill out the design space illustrated in Tables 1 and 2, described in the next section. In particular, we have built re-usable, generic choice and repetition mediators that can be easily modified along the dimensions shown.

The focus of the work presented here is on expanding the repertoire of mediation techniques into settings where standard techniques, such as the *n*-best list in Figure 1, face problems. The work described in this paper involves the design and implementation of re-usable mediation techniques that address some of the deficiencies or problems not handled well by the standard set of techniques found in the literature. We begin by describing the design space in more detail in the next section. After introducing the toolkit concepts necessary to understand our solutions to these problems in the following section, we will discuss each problem in detail. The first problem involves *adding alternatives* to choice mediators, as the correct answer (as defined by the user) may not appear in the list of choices. The second problem, *occlusion*, occurs because choice mediators may cover important information when they appear. The third problem we address is mediation for *target ambiguity*, which can arise when there are multiple possible targets of a user action (such as selecting part of a drawing). Finally, our fourth mediator illustrates one way to deal with *errors of omission*, where the user's input is completely missed by the recognizer.

DESIGN SPACE

Mediation is the process of selecting the correct interpretation of the user's input (as defined by the user). Because of this, mediation often involves the user in determining the correct interpretation of her input. Automatic mediation, which does not involve the user, is also fully supported by OOPS, although not a focus of this paper. Good *mediators* (components or interactors representing specific mediation strategies) minimize the effort required of the user to correct recognition errors or select interpretations. This section presents a survey of existing interfaces to recognition systems, including speech recognition, handwriting recognition, gesture recognition, word prediction, and others (expanded from [19]).

The survey led us to identify two basic categories of interactive mediation techniques. The first, and most common mediation strategy, is repetition. In this mediation strategy, the user repeats her input until the system correctly interprets it. In the second strategy, choice, the system displays several alternatives and the user selects the correct answer from among them.

Repetition

When the user specifies the correct interpretation of her input by explicitly repeating some or all of it, we refer to

I/O	System	Repair Modality	Undo	Repair Granularity
Handwriting / Words	MessagePad™[1], Microsoft Pen for Windows™	Soft keyboard, or individual letter writing	Automatic	Letters
Speech / Words, Phrases	ViaVoice™[3]	Speech, letter spelling, typing	Automatic	Letters or words
	Suhm speech dictation [30]	Voice, pen	User must select area to replace	Letters or words
Speech / Names (non GUI)	Chatter [22]	Speech, letter spelling, military spelling, with escape to choice	Automatic	Letters
Typing / Words	Word Prediction [2,10,25]	Letters (as user enters additional characters, new choices are generated)	Unnecessary (user has to explicitly accept a choice)	Letters
	POBox [21]			

Table 1: A representative set of systems (as defined by their input and output modalities) that vary along the dimensions of repetition mediators. All of these systems provide additionally unmediated repetition, in which the user deletes an entry and repeats it using the original system modality. In contrast, a system which does not provide mediated repetition is the PalmPilot™. I/O gives input/output of recognizer. Systems are representative examples. Military spelling uses "alpha" for 'a', "bravo" for 'b', *etc.*

this as repetition. For example, when a recognizer makes an error of omission (and thus generates no alternatives at all), this option is available to the user. Specific examples are given in Table 1. See [28] for a discussion of how such variations may affect input speeds. Below are the three dimensions of repetition:

Modality: The user often has the option of repeating her input in a different (perhaps less error-prone) modality. However, research in speech systems shows that users may choose the same modality for at least one repair before switching [11], despite the fact that the repair will have lower recognition accuracy [7].

Undo: In order to repeat her input, the user may first have to undo some or all of it. This is most often required of systems without explicit support for mediation (*e.g.* the PalmPilot™).

Repair granularity: Repair granularity may differ from input granularity. For example, the user may speak words or phrases, yet repair individual letters [30].

Choice

When the user selects the correct interpretation of her input from a set of choices presented by the system, we refer to this as choice mediation. The *n*-best list in Figure 1 is an example of this. Like repetition, choice mediators vary along a set of common dimensions. We illustrate the dimensions below by comparing two examples, the *n*-best list used in ViaVoice™ (Figure 1) and the Pegasus drawing beautification system (Figure 2). Pegasus recognizes user input as lines and supports rapid sketching of geometric designs [14]. Additional examples are given in Table 2.

Layout: The *n*-best list uses a menu layout. In contrast, Pegasus does layout "in place". Possible lines are simply displayed in the location they will eventually appear if selected (Figure 2(c&e)).

Instantiation time: The *n*-best list can be instantiated by a speech command, or can be always visible (even when no ambiguity is present). Pegasus shows the alternative lines as soon as they are generated.

Contextual information: Pegasus also shows contextual information about the lines by indicating the constraints that were used to generate them (Figure 2(c&e)). The *n*-best list, which is more generic, shows no additional information.

I/O	System	Layout	Instantiation	Context	Interaction	Feedback
Handwriting / Words	MessagePad™ [1]	Linear menu	Double click	Original ink	Click on choice	ASCII words
Speech / Words	ViaVoice™ [3]	Linear menu	Speech command / Continuous	None	Speech command	ASCII words
Speech/Commands (non GUI)	Brennan and Hulteen [4]	Spoken phrases	On completion	System state (audio icons)	Natural language	Pos.&neg. evid.-nat. lang.
Handwriting / Characters	Goldberg *et Al.* [9]	Below top choice	On completion	None	Click on choice	ASCII letters
Typing / Words (Word prediction)	Assistive Tech. [2, 10]	Bottom of screen (grid)	Continuously	None	Click on choice	ASCII words
	Netscape™ [25]	In place	Continuously	None	Returns to select, arrow for more	ASCII words
Gesture / Commands	Marking Menu [15]	Pie menu	On pause	None	Flick at choice	Commands, ASCII letters
Gesture / Lines	Beautification [14]	In place	On prediction / completion	Constraints	Click on choice	Lines
Context / Text	Remembrance Agent [27]	Bottom of screen, linear menu	Continuously	Certainty, result excerpts	Keystroke command	ASCII sentences
UI description / Interface spec.	UIDE [29]	Grid	On command	None	Click on choice	Thumbnails of results
Multimodal / Commands	Quickset [23]	Linear menu	On completion	Output from multiple recognizers	Click on choice	ASCII words
Email / Appointment	Lookout [12]	Pop up agent, speech, dialogue box	On completion	None	Click OK	ASCII words

Table 2: The layout, instantiation mode, context, selection, and representation used by commercial and research choice mediators. Note that feedback in the mediator may differ from the final output result of recognition. I/O gives input/output of recognizer. Systems are representative examples.

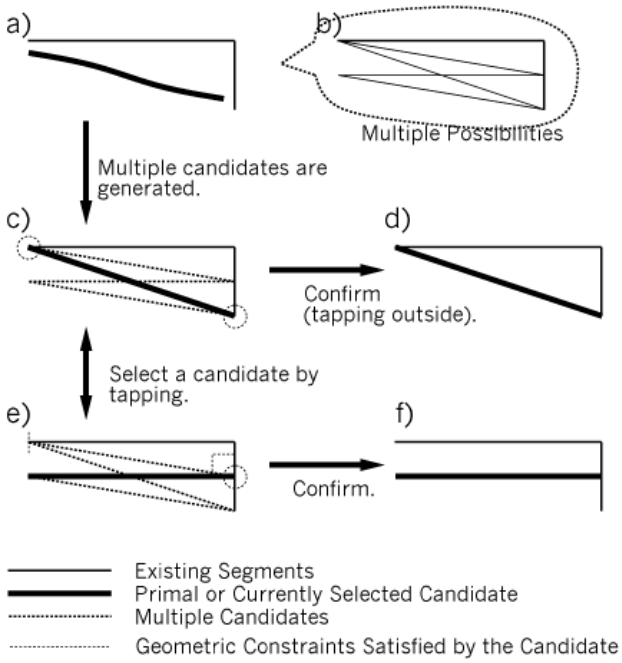


Figure 2: A choice mediator in the Pegasus drawing beautification system (Figure 7 in [14]). The user can click on a line to select it (e). © ACM (reprinted with permission).

Interaction: In both examples, interaction is quite straightforward. In ViaVoice™, the user says "Pick [#]." In Pegasus, the user can select a choice by clicking on it (Figure 2(e)). Drawing a new line will automatically accept the currently selected choice in Pegasus (Figure 2(d&f)). The *n*-best list is only used to correct errors, the top choice is always sent to the current document as soon as recognition is complete.

Feedback: As stated earlier, feedback in Pegasus is in the form of lines on screen. Contrast this to the ASCII words used in the *n*-best list.

In summary

We have identified a rich design space of mediators which fall into two major classes of techniques. Each system we reference implemented their solutions from scratch, but as Tables 1 and 2 make clear, the same design decisions show up again and again. The space of mediation techniques is, therefore, amenable to toolkit-level support, and that is why we built OOPS. OOPS includes both an architecture and a library of mediators, including a generic repetition and a generic choice mediator that can be varied along the dimensions described above in a pluggable fashion [19,20]. We expand upon that work in this paper by identifying some significant gaps in the design space. We were able to use OOPS to create new mediators which address the problems responsible for those gaps.

TOOLKIT-LEVEL SUPPORT FOR AMBIGUITY

OOPS is an extension of the subArctic toolkit [6]. Here we will review the basic features of OOPS discussed in [20], and describe additional features that facilitated the development of the mediators described in this paper.

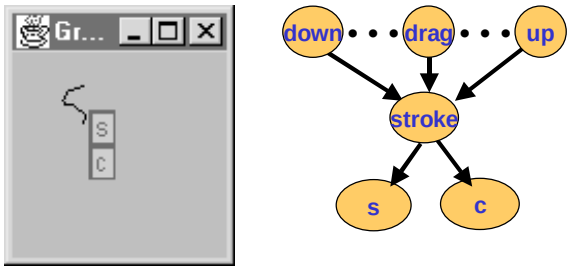
In the past, GUI toolkits have separated recognized input from mouse and keyboard input. Even when a recognizer generates the same data type as a device (such as text), the application writer has to take responsibility for informing interface widgets about information received from the recognizer. Both the Amulet [17] and the Artkit [13] toolkits go beyond this model for pen gesture recognition by allowing interactors to receive gesture results through the same API as mouse and keyboard events.

OOPS takes a step further by allowing recognizers to produce arbitrary input events that are dispatched through the same input handling system as any raw events produced by mouse or keyboard. Thus, they may be consumed by the same things that consume raw events including, possibly, other recognizers. Here we use the term "event" in the traditional GUI toolkit sense, to mean a single discrete piece of input (*e.g.* "mouse down" or "key press (a)").

A recognizer produces events that are *interpretations* of other events or raw data (such as audio received from a microphone). This is a very broad definition of recognition. Essentially, a recognizer is a function that takes events or raw data as input and produces interpretations (also events) as output. For example, a recognizer might produce text from mouse events (which, as described above, is dispatched and might then be consumed by a standard text entry widget such as the one in Figure 4). It could start with text and produce more text. Or it could start with audio and produce mouse events (which might cause a button to depress). It might also produce a new event type such as a "command" or "interaction" event.

As mentioned in the previous section, a recognition error is defined by the user's intent and neither the recognizer nor OOPS necessarily knows what the correct interpretation is. It is through automatic or interactive mediation that this is determined. Until mediation is completed, OOPS stores information about all known possible interpretations. We refer to the input as ambiguous at this point. Information about ambiguity is kept in a hierarchical ambiguous event graph in OOPS (which can be seen as an extension of the command objects described in [24]). Raw input such as mouse down, drag, and up events make up the root nodes of that graph. Whenever input is interpreted, a node representing the new interpretation is added to the graph. For example, the graph shown in Figure 3(b) represents a series of mouse events that have been interpreted as a stroke, and then recognized as either a '**C**' or an '**S**'. The '**C**' and '**S**' are ambiguous (only one of them is correct). A graph node is considered ambiguous when it is one of multiple interpretations.

OOPS provides infrastructure for tracking ambiguity and for resolving ambiguity (mediation). By providing a consistent, recognizer-independent internal model of ambigu-



(a) (b)

Figure 3: (a) A sketched letter and associated mediator. **(b)** Our internal representation of the events making up the sketched, recognized stroke.

ity, OOPS is able to provide re-usable support for mediation. For example, when the stroke in Figure 3(a) is interpreted as a '**c**' or an '**s**', OOPS automatically sends the event hierarchy to the mediation subsystem because part of it is ambiguous. The default choice mediator (Figure 3(a)) simply displays the leaf nodes of the hierarchy. When the user selects his intended input, it is accepted, and the other interpretation is rejected, resolving the ambiguity.

Most interpretations in OOPS are generated and dispatched during an initial input cycle before the start of mediation. When a new interpretation is created after mediation has begun, it is dispatched as well, and the event hierarchy being mediated is updated, along with any current mediators. Any events that have already been accepted or rejected remain that way.

OOPS supports both automatic mediation and a variety of interactive choice and repetition techniques. Mediation in OOPS may occur immediately or at any later time determined by the current mediator. Thus, a mediator may choose to wait for further input, or simply defer mediation until an appropriate time in the interaction.

When separation is too separate

As stated, we provide a recognizer-independent internal model of ambiguous input in OOPS which allows the separation of recognition, mediation, and application development. However, there are times when two or more of these pieces may need to communicate. For example, recognizers may wish to know which interpretations are accepted or rejected by mediators in order to facilitate learning. OOPS stores information about who created each event in order to inform those recognizers about which of their interpretations are accepted or rejected by the user.

In addition to creating events and receiving accept/reject messages, recognizers in OOPS may support guided re-recognition. Guided re-recognition allows a recognizer to receive more detailed information than a simple reject. This information may be domain specific, and includes an event that should be re-recognized. The intent is to allow a recognizer to make a better guess as to how to interpret the user's input. Recognizers supporting guided re-recognition must implement the **rerecognize(event, Object)** method, where *event* is an event that the recognizer interpreted at some time in the past and *Object* may contain additional domain-specific information.

For example, a choice mediator could have a "none of the above" option. If the user selects it, that mediator could ask the recognizer(s) that generated the current set of choices to **rerecognize()** each of their source events. If there is more than one source event for a given interpretation, the mediator may call the **resegment(Set, Object)** method instead. This tells the recognizer that a mediator has determined that the events in *Set* should be treated as one segment and re-interpreted.

Thus far, we have described the minimal architectural support required by all of our example mediators. In addition to this architecture, we provide a library of standard mediators in OOPS. Since OOPS allows separation of mediation from recognition and from the application, it is possible to swap between different mediation strategies without redesigning any recognizers or the application.

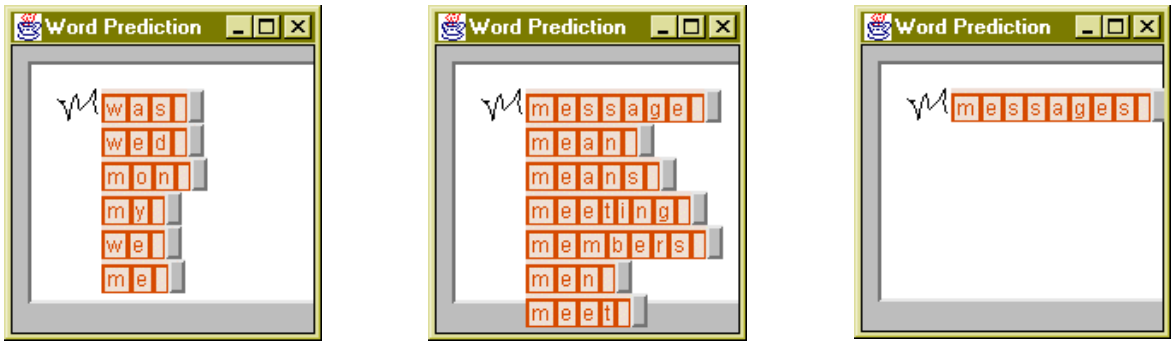
The next four sections consider four new mediation techniques. Each of these techniques is designed to illustrate a method for overcoming a problem with existing mediation techniques. In each section, after discussing the identified problem area (adding alternatives, occlusion, target ambiguity, and omission), and a new interaction technique that addresses it, specific toolkit mechanisms necessary to support these solutions will be considered.

ADDING ALTERNATIVES TO CHOICE MEDIATORS

One problem with choice mediators is that they only let the user select from a fixed set of choices. If none of those choices is right, the choice mediator is effectively useless. For example, as Figure 1 illustrates, if the user intended to say '**for**', the choice mediator cannot help her—she must escape to a repetition technique (spelling the word).

Our goal is to support a smooth transition from selection of an existing choice to specification of a new one. Our solution is to extend an *n*-best list to include some support for repetition. We allow the user to specify new interpretations as well as to select from existing ones using the same mediator.

For example, in the application shown in Figure 4, the user can sketch Graffiti™ letters, which are recognized as characters (by [18]). A word predictor then recognizes the characters as words, generating many more choices than can be displayed by the mediator. When the graffiti letters are ambiguous, the word-predictor returns words starting with each possible letter. Once mediation is completed, the text edit window updates to show the correct choice. Our goals in this mediator are:



(a) (b) (c)

Figure 4: A choice mediator which supports specification. The user is writing 'messages'. **(a)** The user sketches a letter which is interpreted as either a '**m**' or a '**w**' by a character recognizer. A word predictor then generates options for both letters. The user clicks on the '**e**' in '**me**.' **(b)** This causes the mediator to filter out all words that do not begin with '**me**.' The word '**message**' is now the top choice, but it needs to be pluralized. The user clicks on the space after '**message**' indicating that the mediator should generate a word beginning with '**message**' but one character longer. **(c)** The resulting word.

Provide choice at the word level: The user can select from among choices as he did in a standard choice mediator, by clicking on the gray button to the right of a word.

Allow users to control filtering: By clicking on a character, the user specifies a prefix. The mediator reflects this by displaying only words starting with the same prefix. Although word-predictors support dynamic filtering, in most cases, a prefix can only be specified by entering each letter in the prefix in turn. If the user filters repeatedly on the same prefix, the mediator will display a new set of words each time.

Allow users to specify length: by clicking on the space at the end of a word, the user causes the mediator to add a character to that word.

Allow users to specify individual characters: The user can right-click on a character to cycle through other possible characters. This can be used to generate words not returned by the word-predictor.

Allow users an escape: if the user sketches a new letter, only the current prefix will be accepted.

Suppose the user enters an (for which the graffiti recognizer returns '**m**' and '**w**'). The word predictor returns words starting with '**m**' and '**w**' (derived from a frequency analysis of a corpus of email messages), of which the mediator displays the top choices: **was**, **wed**, **mon**, … (Figure 4(a)). The user, who intended '**messages**', filters for words starting with '**me**' by clicking on the '**e**' in '**me**.' The resulting list (Figure 4(b)) includes '**message**', but not '**messages**.' The user indicates that a word one character longer than '**messages**' is needed by clicking on the space at the end of the word, and '**messages**' appears as the top choice (Figure 4(c)). The user selects this choice by clicking on the gray button to its right.

Was this mediator really useful? Without word prediction, the user would have had to sketch 8 characters. Given an 85% accuracy rate (typical for many recognizers), she would have to correct at least one letter (with a total of at least 9 strokes). Here, the user has sketched one letter and clicked 3 times. While this is only one data point, it should be noted that the mediator is built with appropriate defaults so that if the user simply ignores it and sketches the 8 characters, the result will be identical to a situation without word-prediction.

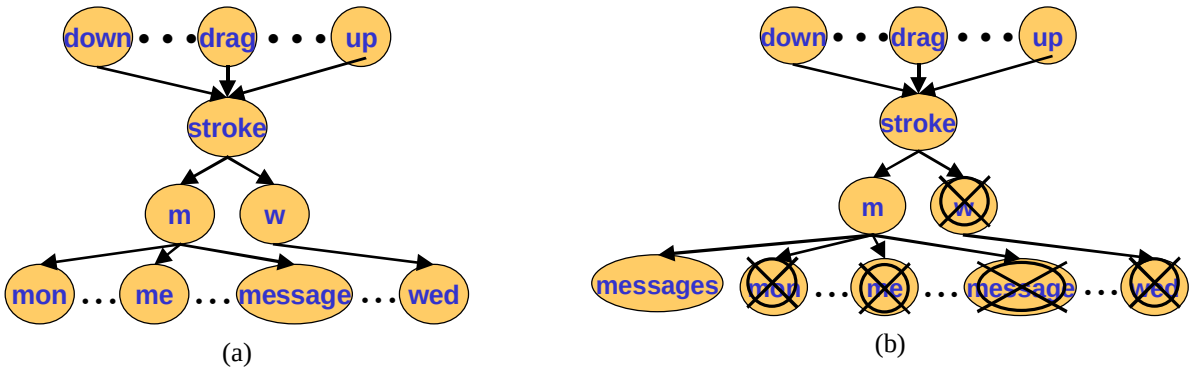


Figure 5: (a) The original event hierarchy in Figure 4 (a-c) and **(b)** the final hierarchy after mediation.

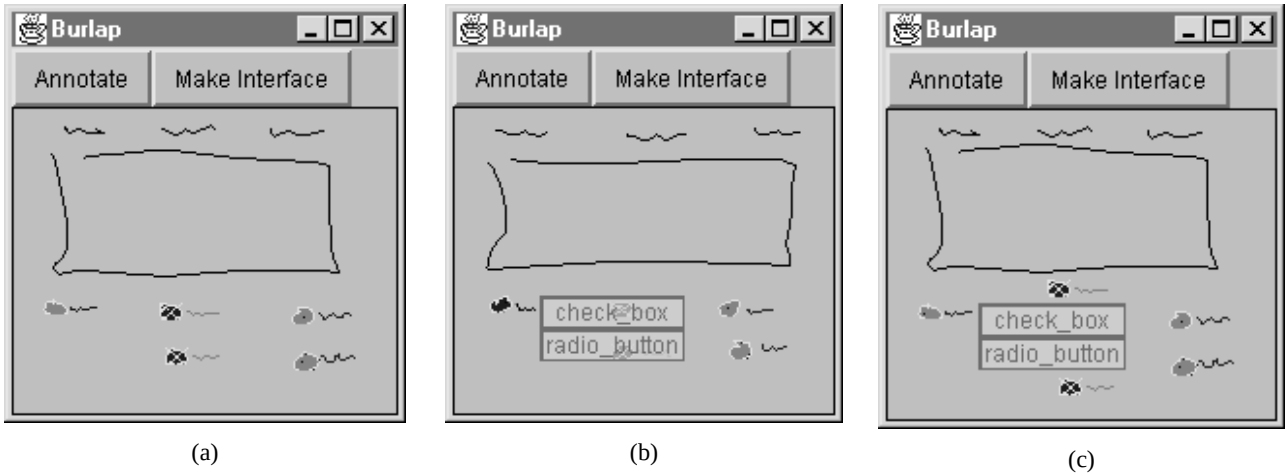


Figure 6: An example of fluid negotiation to position a mediator in the Burlap application. **(a)** The user needs to mediate whether the object on the bottom left edge is a checkbox or radiobutton. **(b)** This mediator (an *n*-best list) occludes some of the sketched interactors. **(c)** This mediator repositions all interactors that intersect with it so as to remove any occlusion.

Toolkit support for adding alternatives

This example dynamically generates new interpretations during mediation. In the example of Figure 4, a new interpretation ('`messages`') is created. All other interpretations are rejected. Figure 5 shows the original and changed hierarchy. The new event is accepted immediately since the user just specified that it was correct using the mediator. It is then dispatched, with the result that it is consumed by the text edit window. No further mediation is necessary since neither the new event nor the event hierarchy it is added to is ambiguous.

Reusability

This mediator can be used with any recognizer that returns text, including speech recognition or handwriting recognition, since the alternatives generated by these recognizers tend to have some similarities to the intended word. For example, some of the characters in a written word may be recognized correctly while others are not. The general idea, to add interactivity supporting repetition to a selection task, can be applied to other domains as well. For example, the automatic beautifier, Pegasus, uses a choice mediator to display multiple lines [14]. Repetition could be added to this by allowing the user to drag the end point of a line around. This would also allow Pegasus to display fewer alternatives in cases where too many are generated, in which case the line could snap to hidden interpretations.

OCCLUSION IN CHOICE MEDIATORS

Choice mediators generally display several possible interpretations on the screen for the user to select from. They are fairly large, and may obscure important information needed by the user to select the correct choice. Since they are also temporary, it doesn't make sense to leave screen space open just for them. An alternative is to dynamically make space for them.

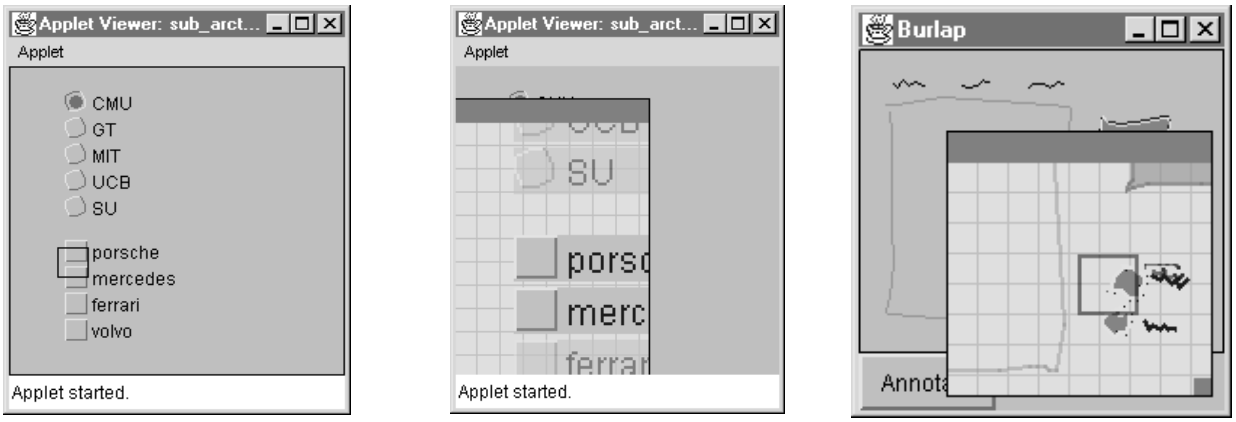
For example, consider Burlap, the application shown in Figure 6 [20]. Burlap is a drawing program for sketching user interface elements, based on SILK [16]. The user can sketch buttons, scrollbars, and so-on. These interactors are recognized and become interactive. However, recognition is error-prone. For example, checkboxes are easily confused with radiobuttons.

The *n*-best list in Figure 6(b) is obscuring two buttons. Is the leftmost sketch a checkbox or a radiobutton? This type of ambiguity is not mediated in Burlap until the user tries to interact with a button. So he may have drawn the hidden buttons some time ago. In order to be consistent, the user may need to see the buttons now in order to determine their status.

Our solution moves the sketches occluded by the mediator into a more visible location (Figure 6(c)). This approach is based on one of the interface techniques used in *fluid negotiation*, a concept that Chang *et al.* developed for handling temporary displays of information [5]. Some of the possible approaches they suggested include shrinking, fading, and call-outs. In [5], the temporary display negotiated the best approach with the underlying document. Because our mediator is used for input as well as output, and is the focus of the interaction, we have chosen a technique that only changes the underlying document (the sketched interface), not the size or shape of the mediator.

Toolkit support for dealing with occlusion

This is accomplished in a way that requires no changes to the underlying application. The only difference between the application shown in Figure 6(b) and (c) is which mediator is installed. The new mediator is based on a lens that uses the subArctic interactor tree to pick out all of the interactors that intersect its bounding box. It then uses the techniques described in [6] to modify the way they are drawn (without changing the interactors themselves). This



(a) (b) (c)

Figure 7: (a) The user is trying to click on a button. Which one? **(b)** A mediator magnifies the area in order to let the user specify this (The ambiguous choices are displayed in darker colors). **(c)** An example of the same mediator being used in Burlap.

mediator is modal, the user is required to resolve the ambiguity before continuing his interaction.

Reusability

This mediator can be used anywhere an *n*-best list might be used. For example, considering the mediators described in Table 1, this includes word-prediction, speech and handwriting recognition in GUI settings, and the Remembrance Agent [27]. More generally, similarly to the fluid negotiation work, the lens responsible for moving screen elements out from under the mediator could be combined with any mediator that is displayed on screen in a GUI. Of course, there are some situations where occlusion is appropriate, such as in the next example.

TARGET AMBIGUITY

We have encountered three major classes of ambiguity in our work. These are recognition ambiguity (which word did she write?); segmentation ambiguity (was that "sew age" or "sewage"?) and target ambiguity. In [20], we described these classes of ambiguity in more detail, and introduced a mediator of segmentation ambiguity. In almost all previous systems, mediation has only addressed recognition ambiguity. Here we demonstrate mediation of target ambiguity.

Target ambiguity arises when the target of the user's input is uncertain. For example, it is unclear if the circle around the word circle is intended to include "is" or not.

We are interested in using target ambiguity to model situations that, although they may seem ambiguous to the user, are commonly treated as straightforward by the computer. In particular, we are interested in situations where mouse motion becomes difficult. For example, people use the term “fat finger syndrome” to refer to situations in which the user's finger is larger than the button they want to press (very small cell phones, touch screens). In addition, misalignment on something like a digital white board can cause a mouse click to go to the wrong interactor. Also, certain disabilities may make it difficult to control a mouse, as can age. For example, research shows that older users have trouble selecting common GUI targets [31] as do people with disabilities such as cerebral palsy.

These problems can be addressed by treating the mouse as an area instead of a point [31]. However, the resulting area may overlap more than one interactor (an example of target ambiguity). We mediate this target ambiguity using a magnifier (Figure 7(b&c)). This magnifier only appears when there is a need due to ambiguity. For context, we include an area four times the size of the area mouse. The magnified area is interactive and users can click on interactors inside it just as they would on an unmagnified portion of the screen. As soon as the user completes his action, or the mouse leaves the magnifier, it goes away.

Toolkit support for target ambiguity

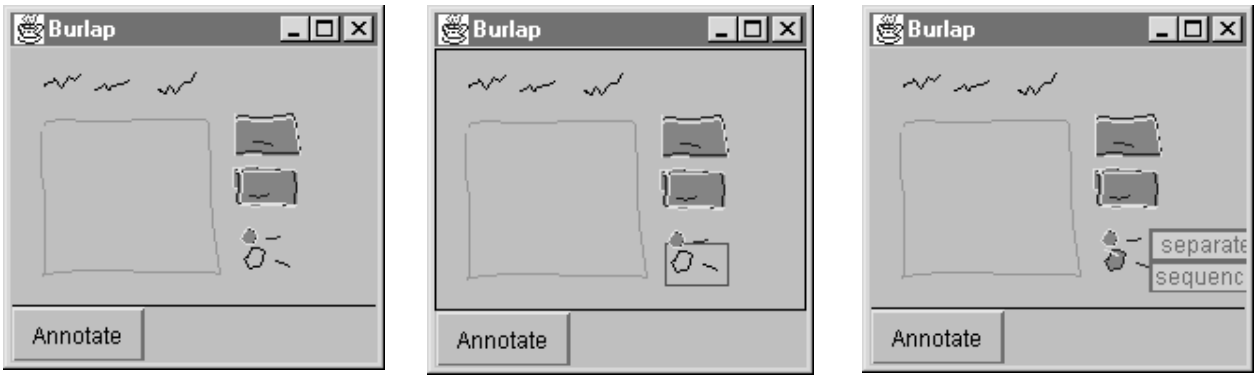
First, target ambiguity is generated by a recognizer that checks for multiple mouse targets. If only one target exists, the input is processed normally. If several targets exist, the results are passed to the mediator.

It is because of our extremely general support of recognition that this is possible. For example, when the extended mouse area (but not the mouse itself) intersects a single interactor, this recognizer creates a new mouse event over that interactor as an interpretation of the raw mouse event it gets as input. This interpretation is dispatched and consumed by the interactor, which does not even know that a recognizer was involved. As far as the interactor is concerned, the user clicked on it.

Our mediator makes use of a lens that magnifies the area under the input [6]. In addition, the lens is responsible for adjusting any positional input it gets based on the new size and position of the pixels it is magnifying.

Reusability

The magnification mediator works with any interface built in OOPS. This includes animated (moving) interactors;



(a) (b) (c)

Figure 8: (a) An interface sketched in Burlap. In the lower right is an unrecognized radiobutton. **(b)** The user has selected the radiobutton in order to indicate that re-recognition should occur. **(c)** Re-recognition is completed, and the result is further ambiguity. One of the possible interpretations generated was a radiobutton. If that is correct, should the new button be in sequence with the old one, or should it be separate?

Burlap; and any other interface which uses mouse clicks. Although it is currently limited to mouse input, in theory it could be generalized to any positional input.

ERRORS OF OMISSION

The last problem area addressed in this paper is errors of omission. An error of omission occurs when some or all of the user's input is not interpreted at all by the recognizer. For example, in Figure 8, the user has sketched a radio button in the lower right, but those strokes were not recognized. An error of omission has occurred.

The rules used for recognition in Burlap are taken from [16] and are based upon the shape and size of individual strokes drawn by the user and the graphical relationships between sets of strokes (such as distance, orientation, *etc*.). When the user draws something too big, too far apart, or so on, it is not recognized. In general, the causes of errors of omission are very recognizer-dependent. For example, an error of omission may occur in speech recognition because the user speaks too quietly.

One solution to this problem is repetition. The user can simply try the sketch again. However, research in both pen [8], and speech [7], recognition has found that re-recognition accuracy rates are no better than recognition rates

We address this with guided re-recognition. The user can initiate re-recognition by selecting the strokes that should have been recognized using the right mouse button. By doing this, the user is giving the system important new information. In Burlap's case, the new information is that the selected strokes should be interpreted as an interactor. We can use this information to eliminate options, such as interactors that have a different number of strokes.

Toolkit support for guided re-recognition

Essentially what the user is doing in this example is providing segmentation information to the recognizer. Although the unistroke gesture recognizer used in Burlap (a third party recognizer [18]) does not support guided re-recognition, the interactor recognizer does. We pass the selected strokes to the interactor recognizer using **resegment(Set, Object)** method described in the toolkit section of this paper. The recognizer generates a new set of interpretations based on those strokes. Because the recognizer now knows the number of strokes, it can quickly narrow the possible interactors and generate alternatives.

Since the recognizer generates new events during mediation, those events must be dispatched, potentially resulting in further interpretations. The new events are ambiguous and OOPS will mediate them (Figure 8(c)), and tell any currently visible mediators to update themselves to show the new interpretations. Any events that were already accepted or rejected in the hierarchy remain that way.

Reusability

The same mediator could be applied in other graphical settings with segmentation issues such as handwriting recognition. This mediator must be told which recognizer it should work with (in the case of Figure 8, the interactor recognizer). It will automatically cache any events generated by that recognizer. Alternatively, it may be given a filter that knows enough about the domain to cache the correct events. In either case, once the user specifies an area, any cached events inside that area are sent to appropriate recognizer to be re-recognized.

CONCLUSIONS AND FUTURE WORK

Recognition today is used in many applications and these applications make use of some common user-interface techniques, called mediators, for handling the recognition errors and ambiguity. However, there are problems with existing techniques. Some recognition errors, such as those caused by target ambiguity and errors of omissions, are harder to deal with. Also, there are limitations to how choice interfaces are commonly handled.

The work described in this paper addresses these problems. All of the mediators presented in this paper were enabled by the OOPS toolkit. They were built with the intent to be re-used in many situations.

We have shown that it is possible to build a variety of techniques that go beyond the current state of the art in correction strategies. Beyond our exploration of the previously known classes of mediation techniques, we have shown how principled handling of ambiguity at the input level allows for mediation in other important settings.

In the future, we wish to explore mediation strategies applicable to segmentation errors, non-GUI applications, and command recognition. All are difficult to mediate because they have no obvious representation (unlike, for example, the text generated in handwriting recognition).

We also plan to use the OOPS toolkit to support empirical work comparing the effectiveness of different mediation techniques. OOPS can allow us to build a framework for evaluating existing and new mediation technologies in a more controlled setting. Because we can build a variety of mediators and easily apply them to the same application, controlled studies to compare the effectiveness of the mediation strategies is now made much simpler.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants IRI-9703384, EIA-9806822, IRI-9500942 and IIS-9800597. Many thanks to Chris Long for the use of his recognizer [18], and to the reviewers.

REFERENCES

1. Apple Computer, Inc. (1997) *MessagePad 2000 User's Manual*.
2. Alm, N. *et al.* (1992) Prediction and conversational momentum in an augmentative communication system. *Communications of the ACM*, **35**(5), pp. 46–57.
3. Baumgarten *et al*. (2000) *IBM® ViaVoice™ QuickTutorial®*. South-Western Educational Publishing.
4. Brennan, S. and E. A. Hulteen. (1995) Interaction and feedback in a spoken language system: A theoretical framework. *Knowledge-Based Systems*, **8**(2–3):143–151.
5. Chang, B. *et al.* (1998) A negotiation architecture for fluid documents. In *Proc. of UIST'98.* pp.123–132.
6. Edwards, K. *et al.* (1997) Systematic output modification in a 2D UI toolkit. In *Proc. of UIST'97.* pp. 151–158.
7. Frankish, C. *et al.* (1992) Decline in accuracy of automatic speech recognition as function of time on task: Fatigue of voice drift? *International Journal of Man-Machine Studies* **36**(6):797–816.
8. Frankish, C. *et al*. (1995) Recognition Accuracy and User Acceptance of Pen Interfaces. In *Proc. of CHI'95.* pp. 503-510.
9. Goldberg, D. and A. Goodisman (1991) Stylus user interfaces for manipulating text. In *Proc. of UIST'91,* pp.127–135.
10. Greenberg, S. *et al*. (1995) Predictive interfaces: What will they think of next? In *Extra-ordinary human-computer interaction: Interfaces for users with disabilities*, A.D.N. Edwards, editor, pp. 103–139.
11. Halverson, C. *et al.* (1999) The beauty of errors: Patterns of error correction in desktop speech systems. In *Proc. of INTERACT'99.*
12. Horvitz, E. (1999) Principles of mixed-initiative user interfaces. In *Proc. of CHI'99*, pp. 159–166.
13. Henry, T. *et al.* (1990) Integrating Gesture and Snapping into a User Interface Toolkit Interaction Techniques. In *Proc. of UIST'90*, pp.112–122.
14. Igarashi, T. *et al.* (1997) Interactive beautification: A technique for rapid geometric design. In *Proc. of UIST'97*, pp. 105–114.
15. Kurtenbach, G. *et al.* (1994) User learning and performance with marking menus. In *Proc. of CHI'94*, pp. 258–264.
16. Landay, J. A. *et al.* (1995) Interactive sketching for the early stages of user interface design. In *Proc. of CHI'95*. pp.43–50.
17. Landay, J.A. and B.A. Myers. (1993) Extending an existing user interface toolkit to support gesture recognition. In *Proc. INTERCHI'93*. pp. 91-92.
18. Long, A.C. *et al.* (1999) Implications for a gesture design tool. In *Proc. of CHI'99.* pp. 40–47.
19. Mankoff, J. *et al*. (2000) Techniques for handling ambiguity in recognition-based input. *Computes and Graphics*, special issue on calligraphic interfaces. Elsevier. To appear.
20. Mankoff, J. *et al.* (2000) Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proc. of CHI'2000*, pp. 368–375.
21. Masui, T. An efficient text input method for pen-based computers. In *Proc. of CHI''98,* pp.328–335
22. Marx, M. *et al*. (1994) Putting people first: Specifying proper names in speech interfaces. In *Proc. of UIST'94*. pp. 29-37.
23. McGee, D. R. *et al*. (1998) Confirmation in multimodal systems. In *Proc. of COLING-ACL '98*, Montreal, Canada.
24. Myers, B.A. and D.S. Kosbie. (1996) Reusable hierarchical command objects. In *Proc. of CHI'96,* pp. 260–267.
25. Netscape Communications, Corp. Netscape Navigator. http://www.netscape.com.
26. Nigay, l. and Coutaz, J. A Generic platform addressing the multimodal challenge. In *Proc. of CHI'95*. pp.98–105.
27. Rhodes, B.J. and T. Starner. (1996) Remembrance Agent. In *Proc. of PAAM'96*, pp. 487–495.
28. Rudnicky, A.I. and A.G. Hauptmann (1991). Models for evaluating interaction protocols in speech recognition. In *Proc. of CHI'91,* pp. 285-291.
29. Sukaviriya, P. *et al.* (1993) A second-generation user interface design environment: The model and the runtime architecture. In *Proc. of INTERCHI'93*, pp. 375–382.
30. Suhm, B. *et al*. (1999) Model-based and empirical evaluation of multimodal interactive error correction. In *Proc. of CHI'99,* May, 1999, pp. 584-591.
31. Worden, A. *et al*. (1997) Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons. In *Proc. of CHI'97*, pp. 266–271.