2000

OOPS: A Toolkit Supporting Mediation Techniques for Resolving Ambiguity in Recognition-Based Interfaces

Jennifer Mankoff
Georgia Institute of Technology - Main Campus

Gregory D. Abowd
Georgia Institute of Technology - Main Campus

Scott E. Hudson
Carnegie Mellon University

OOPS: A Toolkit Supporting Mediation Techniques for Resolving Ambiguity in Recognition-Based Interfaces

Jennifer Mankoff [a] Gregory D. Abowd [a] Scott E. Hudson [b]

[a] *Georgia Institute of Technology, GVU Center, Atlanta, GA,*
{jmankoff,abowd}@cc.gatech.edu

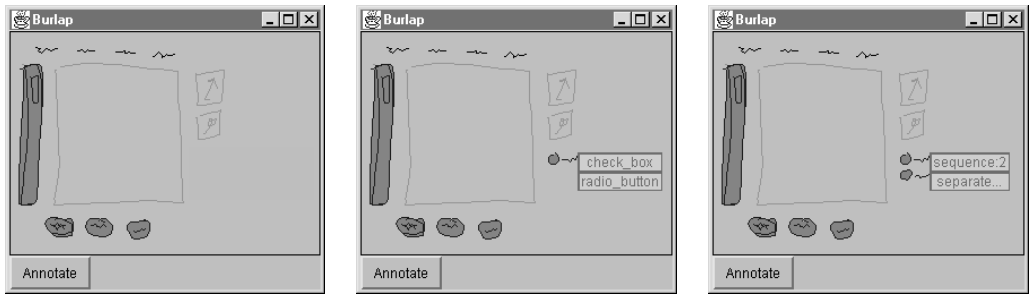
[b] *Carnegie Mellon University, HCI Institute, Pittsburgh, PA, hudson@cs.cmu.edu*

Abstract

Recognition technologies are being used extensively in both commercial and research systems. Recognizers are still error-prone however, and this results in performance problems and brittle dialogues, creating barriers to the acceptance and usefulness of recognition systems. Better interfaces to systems using recognition, which can help to reduce the burden of recognition errors, are difficult to build because of lack of knowledge about the ambiguity inherent in recognition. We present a survey of the design of correction techniques in interfaces which make use of recognizers. Based on this survey, we have created a user interface toolkit, OOPS (Organized Option Pruning System) [27]. OOPS consists of a library of reusable error correction, or *mediation*, techniques drawn from the survey, combined with necessary architectural extensions to model and to provide structured support for ambiguity at the input event level of a GUI toolkit. The resulting infrastructure makes it easier for application developers to support error handling, thus helping to reduce the negative effects of recognition errors, and allowing us to explore new types of interfaces for dealing with ambiguity.

1 INTRODUCTION

Recognition technologies such as speech, gesture, and handwriting recognition, have made great strides in recent years. By providing support for more natural forms of communication, recognition can make computers more accessible. Such "natural" interfaces are particularly useful in settings where a keyboard and mouse are not available, such as very large or very small displays, and mobile and ubiquitous computing.



(a) (b) (c)

Fig. 1. **An interface sketched in Burlap (a simplified version of Silk [24]). The user has sketched a sample interface to a drawing program. (a) The interface as it looks initially. (b) The user adds a radiobutton. A menu asks the user if they have sketched a check box or a radiobutton. (c) The user adds a second radiobutton. A menu asks the user if the two radiobuttons should be considered a sequence or separate (in a sequence, only one radiobutton at a time can turn on).**

However, recognizers are error-prone (they don't always interpret user input as the user intended), and this can confuse the user, cause performance problems, and result in brittle interaction dialogues. For example, Suhm found that even though humans speak at 120 words per minute (wpm), the speed of spoken input to computers is 40 wpm on average because of recognition errors [42].

Interfaces that use recognizers must contend with errors, and in fact can reduce the negative impact of errors. Research has shown that variations on these interfaces can reduce some of negative effects of recognition errors [42,1]. We will illustrate these techniques throughout this document with a prototype system, Burlap, the application shown in Figure 1.

Burlap, built with OOPS, is a simplified version of the sketch-based user interface builder SILK (Sketching Interfaces Like Krazy [24]). Like SILK, Burlap is a drawing tool for early sketching of a graphical user interface. It allows the user to sketch interactive components (called interactors) such as buttons, scrollbars, and checkboxes on the screen. Figure 1(a) shows a sample user interface sketched in Burlap. The user can click on the sketched buttons, move the sketched scrollbar, and so on. For this to work, an (error-prone) recognizer is converting the user's sketches into interactive components. The purpose of SILK (and, nominally, Burlap), is to allow designers to create very rough prototype interfaces without losing the flexibility and rapid action that sketching provides.

In Figure 1(b), the user has drawn a radiobutton in the lower right that is easily confused with a checkbox. Rather than simply guessing, the system has brought up a menu asking the user which interpretation of her input is correct. In this example, the user selects the radiobutton option, her intended

interpretation. She then sketches a second radiobutton, below the first, as shown in Figure 1(c). Here, the user has drawn a sequence of two radiobuttons, and the system needs to know whether they should be grouped in sequence, that is, clicking one un-sets the other and vice versa, or kept separate, so it displays a menu indicating these choices.

These menus are examples of a choice-based interface, one of two primary interfaces strategies for resolving uncertainty in recognition. Choice-based interfaces display multiple potential interpretations of the user's input, and allow her to select one. In the other major strategy, repetition-based interfaces, the user repeats her input, either in the same or in a different modality. These strategies were uncovered in a survey of interfaces making used of recognition, described in Section 3. In addition, mediation may be done automatically, without user involvement.

1.1 Defining the area

We began this work by surveying interfaces which make use of recognition. For our purposes, recognition involves looking at user input or information sensed about the user, and interpreting it. Some traditional forms of recognition include speech, handwriting, and gesture recognition. Other types of recognition include face recognition, activity recognition, and word-prediction.

Recognition is generally a difficult process that may result in errors or may be ambiguous (there may be multiple potential interpretations). Because ambiguity is inherent in human communication, the holy grail of eliminating errors is probably not achievable. In addition, when recognition technologies are used in noisy, varied environments, accuracy typically goes down. In practice, researchers try to reduce errors instead of eliminating them. Error reduction is a tough problem, and big improvements (5–10%) are needed before users even notice a difference [7], even for very inaccurate recognizers ($\leq 50\%$ accuracy).

Our approach is to assume that errors will occur and to incorporate them into the toolkit input model and deal with them in the interface. We divide this into three sets of issues, discussed in the next three sections of this paper.

Discovery (Section 2) In order to provide toolkit level support for dealing with uncertainty in recognition (ambiguity or errors), we need some way of identifying uncertainty. Good discovery can allow us to choose when to handle uncertainty, and allow recognizers to learn from their mistakes on the fly.

Mediation (Section 3) Once ambiguity or errors have been identified, they must be dealt with. We call the process of correcting (and avoiding) recognition errors *mediation*, because it generally involves some dialogue between

the system and user for correctly identifying the user's intended input. In surveying existing interfaces to recognition systems, we found a plethora of approaches to mediation.

In a broad sense, mediators allow the user to inform the system of the correct interpretation of her input. Recognizers will often generate multiple, ambiguous potential interpretations. Computer programs, not normally built to handle ambiguity, may select the wrong alternative.

With so many mediation strategies available, it is difficult for designers to know which approach to use in which setting. For example, it does not make sense to check with the user constantly if the recognizer is usually right. Unfortunately, the user's context, which affects the most appropriate mediation technique can change dynamically. By context we mean things such as recognition accuracy, or user interruptability, *Meta-mediation* techniques support dynamic selection of mediation techniques based on information about this type of context. They determine, at any given moment, which mediator will be displayed to the user. They also determine *when* mediation will occur, since timing is an important factor in minimizing the negative impacts of interrupting the user with a mediation task.

Toolkit level support (Section 4) We can provide reusable support for mediation interfaces like those described in the literature by modeling and providing access to knowledge about the ambiguity resulting from the recognition process. Existing user interface toolkits have no way to model ambiguity, much less expose it to the interface components, nor do they provide explicit support for resolving ambiguity. Instead, it is often up to the application developer to gather the information needed by the recognizer, invoke the recognizer, handle the results of recognition, and decide what to do about any ambiguity.

When ambiguity is modeled explicitly in a user interface toolkit, it becomes accessible to interface components. This means that components can give the user feedback about how their input is being interpreted before the correct interpretation has been selected. At the same time, by carefully separating and structuring different tasks within the toolkit, neither application nor interface need necessarily know anything about recognition or ambiguity in order to use recognized input, or to make use of predefined interface components for resolving ambiguity.

2 DISCOVERY

Discovery is the process of identifying when the top choice returned by a recognizer is wrong, that is, not what the user intended. Discovery does not help to figure out what the correct answer is, it simply identifies problems with recognition. The system can then use mediation techniques to decide what to

do about the problem, for example by asking the user for help. Discovery is used to figure out when the system needs to mediate.

In order for the system to mediate problems, it needs some way to represent them. We refer to these problem spots as *ambiguous* and we represent them as a set of potential interpretations. The set contains the recognizer's original top choice (which discovery tells us may be wrong) and any other possibilities which may be correct (such as the additional results returned by some recognizers).

These ambiguous sets can be generated in several ways.

- Many recognizers will return multiple possible answers when they think there is any uncertainty about how to interpret the user's input. For example, IBM's ViaVoice^{TM} and the Paragraph^{TM} handwriting recognizer both do this, as do many word-prediction systems [2,15], and Burlap's reimplementation of Landay's interactor recognizer (from SILK [24]).
- The discovery process itself may generate alternatives. One example discovery strategy is to use a *confusion matrix*. A confusion matrix is a table, usually based on historical information about recognizer performance, which shows potentially correct answers that the system may have confused with its returned answer. For example, Marx and Schmandt compiled speech data about how letters were mis-recognized into a confusion matrix, and used it to generate a list of potential alternatives for whatever the speech recognizer returned [28].

  We use a confusion matrix for similar reasons in Burlap, to supplement our gesture recognizer, which returns only the top choice guess. For example, the recognizer often confuses very small rectangles (used to sketch checkboxes) with very small circles (used to sketch radiobuttons) so we always assume that if the recognizer returns one, we should add the other to the set of ambiguous possibilities.
- The semantics of the task may result in ambiguity. For example, if a system uses multiple recognizers, the choices returned by *each* recognizer represent an ambiguous set of alternatives. In fact, ambiguity may also arise between an interactor and a recognizer. For example, how do we know if the user intended to click on a button or draw a new interactor in Burlap?

The techniques described above are limited to two classes of errors commonly called substitution errors and insertion errors. In both of these cases, problems arise because the recognizer returned some response which was wrong for some reason. It is much harder to identify errors of omission, where the user intended something to be recognized but the recognizer did not return any response at all (because for some reason it did not notice the input, or discarded it). In this case, discover is often done by the user rather than the system.

3 MEDIATION

Mediation is the process of selecting the correct interpretation of the user's input. For our purposes, correct is defined by the user's intentions. Because of this, mediation often involves the user by asking her which interpretation is correct. Good *mediators* (components or interactors representing specific mediation strategies) minimize the effort required of the user to correct recognition errors or select interpretations.

For example, the menu of choices shown earlier in Figure 1(b)&(c) were created by an interactive mediator that is asking the user to indicate whether she intended to draw a radiobutton or a checkbox. It represents a choice-based mediation strategy. Behind the scenes, an automatic mediation technique has already eliminated the possibility that one or more of those strokes was simply intended as an annotation not to be interpreted.

As this example suggests, there are a variety of ways that mediation can occur. The first, and most common, is repetition. In this mediation strategy, the user repeats her input until the system correctly interprets it. The second major strategy is choice. In this strategy, the system displays several alternatives and the user selects the correct answer from among them. The third strategy is automatic mediation. This involves choosing an interpretation without involving the user at all.

3.1 Repetition

Repetition occurs when the user in some way repeats her input. A common example of repetition occurs when the user writes or speaks a word, but an incorrect interpretation appears in her text editor. She then proceeds to delete the interpretation, and then dictate the word again. This is the extent of the support for mediation in the original PalmPilot^{TM}. Other devices provide additional support such as an alternative input mode (for example, a soft keyboard), undo of the mis-recognized input, or partial repair of letters within a mis-recognized word.

Figure 2 shows an example of repetition in Burlap. The user's first attempt at sketching a radiobutton is not recognized (a). She repeats part of the sketch by drawing the tail of the radiobutton again (b), and recognition is completed (c). There are hidden choices in this example about modality, partial repair, and undo, three dimensions in the design space of repetition based interfaces. For example, we could have required the user to first delete her original strokes (undo), and then type the name of her intended interactor (modality change). In another example, Suhm allowed users to cross out or write over

(a) (b) (c)

Fig. 2. **The user is trying to sketch a radiobutton. Due to an error of omission (the recognizer does not realize that it should recognize those strokes) she must repeat part of her input (partial repair repetition). Now the recognizer responds appropriately.**

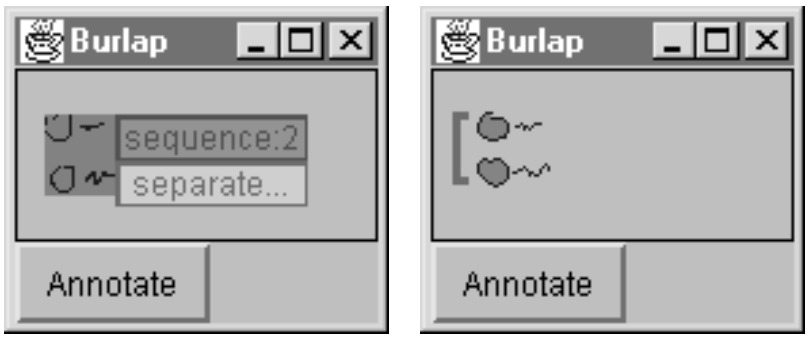
mis-recognized letters from spoken words with a pen [42]. Users do not have to undo their original spoken input. They correct the input in a new modality (pen instead of speech), and may correct small parts of the input (individual letters). All three dimensions are described in more detail below.

Modality Repetition often involves a different modality, one that is less error-prone or has orthogonal sorts of errors. For example, in the Newton MessagePad^{TM}, Microsoft Pen for Windows^{TM}, and other commercial and research applications, the user may correct mis-recognized handwriting by bringing up a soft keyboard and typing the correct interpretation.

When repetition is used without the option to switch to a less error-prone modality, the same recognition errors may happen repeatedly. In fact, research has shown that a user's input becomes harder to recognize during repetition in speech recognition systems because he modifies his speaking voice to be clearer (by human standards) and, therefore, more difficult for the recognizer to match against normal speech [12]. On the other hand, less error-prone modalities are generally awkward, slow, or otherwise inconvenient (for example, may require hands), or the user would have chosen them in the first place.

There are also examples of alternate modalities that are highly integrated with the rest of the application, and thus less awkward. In his speech dictation application, Suhm allowed users to edit the generated text directly using a pen, rather than bringing up a separate window or dialogue [42].

Partial Repair In dictation style tasks, it is often the case that only part of the user's input is incorrectly recognized. For example, a recognizer may interpret "She picked up her glasses" as "She picked up her glass." In this case, the easiest way to fix the problem is to add the missing letters, rather than redoing the whole sentence. In another example, Huerst *et Al.* note that users commonly correct messily written letters in handwriting, and they built support for applying this style of correction before passing handwriting to the recognizer to be interpreted [20]. Figure 2 demonstrates partial repair in Burlap. The user "corrects" only one stroke of a multi-stroke gesture, a sketched radiobutton.



(a) (b)

Fig. 3. **The user is trying to sketch some radiobuttons. In (a) the system uses a menu to ask the user if the buttons should be grouped. In (b) a totally different layout is used for the same purpose.**

Undo Depending upon the type of application, and the type of error, repetition may or may not involve undo. For example, repetition is the most common when the recognizer makes an error of omission (the recognizer does not make any interpretation at all of the user's input), and in this case, there is nothing for the user to undo. In contrast, in very simple approaches to mediation, the user must undo or delete her input before repeating it (*e.g.* PalmPilot^{TM}, "scratch that" in DragonDictate^{TM}). In some applications, such as Burlap, it does not matter if there are a few extraneous strokes on the screen (they actually add to the "sketched" effect), so undo is unnecessary (see Figure 2). In other situations, such as entering a command, it is essential that the result be undone if it was wrong (what if a pen gesture representing "save" were misinterpreted as the gesture for "delete?").

3.2 Choice

Choice user interface techniques give the user a choice of more than one potential interpretation of her input. One common example of this is an n-best list (a menu of potential interpretations) We have identified several dimensions of choice mediation interfaces including layout, instantiation time, additional context, interaction, and feedback [26]. Table 1 gives an overview of some commercial and research systems with graphical output that fall into this design space. Each system we reference implemented their solutions from scratch, but as Table 1 makes clear, the same design decisions show up again and again. These design decisions, which illustrate the dimensions of a choice interface, are described below.

Layout describes the position and orientation of the alternatives on the screen. The most common layout is a standard linear menu [3,9,36]. Other menu-like layouts include a pie menu [22], and a grid [2,15,43]. We also found

System	Layout	Instantiation	Context	Interaction	Feedback
MessagePad^{TM} [3]	linear menu	on click	original ink	drag-release	ASCII words
DragonDictate^{TM} [9]	linear menu	speech command	none	speech command	ASCII words
Brennan&Hulteen[6]	speech	on completion	system state	natural language	pos&neg nat. lang. evidence
Goldberg and Goodisman [13]	below top choice	on completion	none	click on choice	ASCII words
Word-prediction ([2,15,29])	bottom of screen (grid)	continuously	none	click on choice	ASCII words
Word-prediction (Netscape [8])	in place	continuously	none	return selects top keystroke, arrow requests more	ASCII words
Marking Menu [22]	pie menu	on pause	none	flick at choice	commands, ASCII letters
Beautification [21]	in place	on completion	constraints	click on choice	pictures (lines)
Remembrance Agent [36]	bottom of screen, linear menu	continuously	certainty, result excerpts	keystroke command	ASCII sentences
UIDE [43]	grid	on command	none	click on choice	thumbnails of results
QuickSet [34]	linear menu	on completion	output from multiple recognizers	click on choice	ASCII words
Lookout [17]	pop up agent, speech, ...	on completion	none	click ok	ASCII desc. of top choice

Table 1
A comparison of different systems that offer the user a choice of multiple potential interpretations of her input.

examples of text floating around a central location [13], and drawings in location that the selected sketch will appear [21].

Another variation is to display only the top choice (while supporting interactions that involve other choices) [13].

Figure 3 shows two different layouts for the same choice display in Burlap. Both are mediating ambiguity about whether two radiobuttons should be grouped (as in Figure 1(c)). One is a simple menu, while the other is a graphical representation of the idea that these buttons are linked.

Instantiation time refers to the time at which the choice display first appears, and the action that causes it to appear. Variations in when the display is originated include: on a mouse click [3] or other user action such as pause [22] or spoken command [9]; based on an automatic assessment of ambiguity [17], continuously [2,15,8,36]; or as soon as recognition is completed [13,21]. If instantiation is delayed, there is still the decision of whether to display the top recognition choice (thus indicating to the user that recognition has occurred), or simply to leave the original, unrecognized input on the screen.

For example, the mediator in Figure 1(b) only comes up when the user tries to interact with (click on) the radiobutton, while the mediator in Figure 1(c) comes up as soon as a potential sequence is detected.

Contextual information is any information related to how the alternatives were generated or how they will be used if selected. Additional context that may be displayed along with the actual alternatives includes information about their certainty [36], how they were determined [21], and the original input [3].

The menu in Figure 3(a) shows an extension of a simple n-best list mediator that highlights the associated strokes.

Interaction, or the details of how the user indicates which alternative is correct, is generally done with the mouse. For example, Goldberg and Goodisman suggest using a click to select the next most likely alternative even when it is not displayed [13]. Or systems may allow the user to implicitly confirm the indicated choice simply by continuing their task [13,21]. For example, when the user starts drawing again, the mediator in Figure 3(b) automatically accepts the displayed choice (linking the radiobuttons). In cases where recognition is highly error-prone, the user most select something to confirm, and can implicitly contradict the suggested interpretation [2,15,16]. In non-graphical settings interaction may be done through some other input mode such as speech.

Feedback is the method of displaying the interpretations. This generally correlates closely to how they will look if they are selected. Text is used most often [2,3,9,13,15,8,36], but some interpretations do not map naturally to a text-based representation. Other variations include drawings [21], commands [22], icons [43], and mixtures of these types. In addition, feedback may be auditory. For example, Brennan & Hulteen use natural language to "display" multiple alternatives [6]. For example, Figures 3(a)&(b) represent radically different feedback for the same set of choices.

Two systems that differ along almost all of the dimensions just described are the Apple MessagePad^{TM}, and the Pegasus drawing beautification system [21]. The Apple MessagePad^{TM} (Figure 4) used a menu layout, which is instantiated

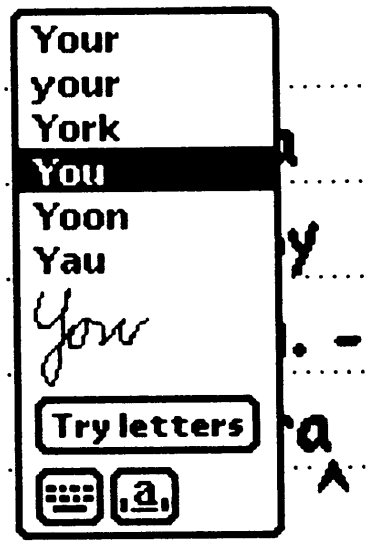


Fig. 4. **The mediator used in the Apple MessagePad**^{TM}

when the user double clicks on an incorrectly recognized word. The original handwritten input is displayed at the bottom of the menu and an alternative is selected by clicking the mouse on the choice. This system also supports repetition in an alternate modality (a soft keyboard).

In contrast, the Pegasus drawing beautification system recognizes user input as lines [21]. This allows users to more easily and rapidly sketch geometric designs. Layout is "in place," i.e. lines are simply displayed in the location they will eventually appear if selected. Instead of waiting for an error to occur, the choice display is instantiated as soon as recognition is completed. It is essentially informing the user that there is some ambiguity in interpreting her input, and asking for help resolving it. As in the previous example, the user can select an alternative by clicking on it. However, interaction differs in that the top choice will automatically be selected if the user continues to draw lines. This system also shows the constraints used to generate each choice as additional context.

By identifying this design space, we can begin to see new possibilities. For example, although Table 1 shows that the continuous instantiation style has been used in text-based prediction such as Netscape's word-prediction and the Remembrance Agent [8,36], to our knowledge it has not been used to display multiple predicted completions of a gesture in progress.

Further research in choice displays needs to address some intrinsic problems. First, not all recognized input has an obvious representation. How do we represent multiple possible segmentations of a group of strokes? Do we represent a command by its name, or some animation of the associated action? What

about its scope, and its target? If we use an animation, how can we indicate what the other alternatives are in a way that allows the user to select from among them?

Second, what option does the user have if the correct answer is not in the list of alternatives? One possibility is to build a mediator that lets you switch between choice and repetition. Essentially it is a conglomeration of mediation techniques with all of the positives and the negatives of both. An example of this is the Apple MessagePad^{TM} (see Figure 4). Another possibility is to make choice-based mediation more interactive, thus bringing it closer to repetition. For example, an improvement to the Pegasus system would be to allow the user to edit the choices actively by moving the endpoint of a line up and down. The result would allow the user to specify any line, just like repetition would.

3.3 Automatic mediators

Automatic mediators select an interpretation of the user's input without involving the user at all. Three classes of automatic mediators are commonly found in the literature, and described below.

Thresholding Many recognizers return some measure of their confidence in each interpretation. If this can be normalized to a probability of correctness, the resulting probability can be compared to a threshold. When an interpretation falls below the threshold, the system rejects it [35,6,4].

Rules Baber and Hone suggest using a rule base to determine which result is correct [4]. This can prove to be more sophisticated than thresholding since it allows the use of context. An example rule used in Burlap is:

> When the user has drawn a very short stroke and it is over an interactor, assume that they intended to click the mouse and reject any recognition of the stroke

We use this rule in Burlap to identify "strokes" (sets of mouse clicks) that were really intended to be a single click, for example when selecting a radiobutton. This is a common problem with pen based interfaces, where often a very quick press of the pen may generate several unintended mouse drag events.

Because rules often depend upon linguistic information, they benefit from knowledge about which words are definitely correct to use as "anchors" in the parsing process.

Historical Statistics When error-prone systems do not return a measure of probability, or when the estimates of probability may be wrong, new probabilities can be generated by performing a statistical analysis of historical data about when and where the system makes mistakes. This task itself benefits from good error discovery. A historical analysis can help to increase the

accuracy of both thresholding and rules.

This approach may be used to enhance thresholding or rules. For example, a confusion matrix may be used to update certainty values before applying a threshold [28]. In general, historical statistics may provide a default probability of correctness for a given answer when a recognizer does not. More sophisticated analyses can help in the creation of better rules or the choice of when to apply certain rules.

One problem with automatic mediation is that it can lead to errors. Rules, thresholding, and historical statistics may all lead to incorrect results. Even when the user's explicit actions are observed, the system may incorrectly infer that an error has occurred (or that no error has occurred). Only when the user's action is to notify the system explicitly of an error, can we be sure that an error really has occurred, in the user's eyes. In other words, all of the approaches mentioned may create a new source of errors, leading to a cascade of errors.

3.3.1 Meta-mediation: Deciding when and how to mediate

The interactive and automatic mediators described above represent some very general mediation strategies within which there are a huge number of design choices. The decision of when and how to involve the user in mediation can be complex and application specific. User studies, and other standard HCI methods for gathering qualitative and quantitative data about user interfaces, can be a source of guidance in designing interactions with error-prone computer programs.

Although it is possible to ask the user direct questions about how they handle errors, this may miss the point since the best error handling happens with as little conscious attention as possible. An alternative is to compare task completion speeds with and without error correction support, and to test for satisfaction and frustration. This is useful but may be hard to generalize across different settings. One solution is to normalize data on task completion speeds based on the number of errors that occur during the task [40]. Normalizing the data makes it possible to compare studies of different mediation techniques that can be used in the same application. Similar methods were used by Suhm to relate recognition accuracy to words per minute (input speed) in the presence of a particular mediation technique [40]. For systems that generate ASCII, he also devised a way to relate accuracy to words per minute [41].

In addition to helping with the design of better mediators, these studies can help to inform the task of meta-mediation. Meta-mediation involves deciding whether to do automatic mediation, and how and when to interrupt the user

if automatic mediation is not possible.

The goal of meta-mediation is to minimize the impact of errors and mediation of errors on the user. Studies have shown that recognizers tend to misunderstand a subset of possible inputs much worse than the rest, both in the realm of pen input [11] and speech input [28]. For example, `u` and `v` look very similar in many user's handwriting, and because of this may be more likely to be mis-recognized. So a meta-mediator might use interactive mediation only for the error prone subset.

For example, Horvitz uses a technique called *decision theory* to provide dynamic, system-level support for meta-mediation [17]. Decision theory can take into account dynamic variables like the current task context, user interruptability, and recognition accuracy to decide whether to use interactive mediation or just to act on the top choice. Horvitz refers to this as a mixed-initiative user interfaces.

Simple heuristics can also be used in meta-mediation. Our meta-mediation strategies in Burlap choose mediators based on things like the type of recognition being mediated. For example, we only use the mediator shown in Figure 3(b) for sequences. This is done using a filtering technique that picks out instances of sequence ambiguity.

3.4 The need for toolkit support for mediation

There is a wide variety of interaction techniques for mediating recognition errors to be found in the literature. Anyone trying to design an interface that makes use of recognition has a plethora of examples from which to learn. Many of these examples are backed up by user studies that compare them to other possible approaches. Since most of the mediation strategies found in the literature can be placed in variations on one of three categories, repetition, choice, our automatic mediation, there is a lot of potential for providing reusable support for them. This suggests the need for a library of extensible, reusable techniques drawn from our survey.

However, we found few examples of toolkit-level support for these types of applications, particularly at the graphical user interface level (multi-modal toolkits, such as the Open Agent Architecture [34], do not usually deal directly with interface components). The next section describes the architecture we built in order to support mediation. The resulting library and architecture form the toolkit we call OOPS [27].

4 TOOLKIT LEVEL SUPPORT FOR MEDIATION

A key observation about the mediation strategies described in the previous section is that they can be described in terms of how they resolve ambiguity. This is important, since, as discussed in Section 2, we can model most recognition errors using ambiguity. Mediation strategies may let the user choose from multiple ambiguous interpretations of her input, or replace one interpretation with a new one. Although this does not handle errors of omission where the recognizer did not realize that the user was creating new input, it can describe the kinds of errors that happen once input gets to a recognizer.

Interfaces to error-prone systems would benefit tremendously from a toolkit providing a library of mediators that could be used and re-used, or adapted, when an error-prone situation arose.

However, it is not possible simply to add mediators to an existing user interface toolkit like one might add a new type of menu or button. In addition to mediators, a toolkit would, for example, need to keep track of ambiguous interpretations, and inform components when those interpretations were rejected or accepted. Very few toolkits today support ambiguity, and none integrate it into the toolkit input model shared by conventional on-screen components. Without this capability the system has to commit to a single interpretation at each stage, throwing away potentially useful data earlier than necessary, or requiring the system designer to build one-off solutions for retaining information about ambiguity. In addition, a toolkit that supports mediation needs to support recognized input as a first class input type similar to keyboard and mouse. In contrast, most toolkit input models assume a fixed set of input devices that are known in advance. This forces the application developer to handle the recognized input separately, as illustrated in Figure 5, and to find ways to extend existing components to handle recognized input rather than handling it through the same mechanisms as mouse and keyboard events.

Our toolkit, OOPS (Organized Option Pruning System) [27], is an extended version of an existing user interface toolkit, subArctic [19,10]. It includes the architectural solutions necessary to supporting these changes. The remainder of this section will discuss how OOPS handles ambiguity. In particular, this can be split into the task of *generating* ambiguity (Section 4.1) and *resolving* ambiguity (Section 4.2). In addition, the toolkit needs to *track* ambiguity in order to separate initial feedback from action on the final results (Section 4.3). Finally, we have a non-functional requirement of minimizing the impact of all of these changes on the application developer (Section 4.4).

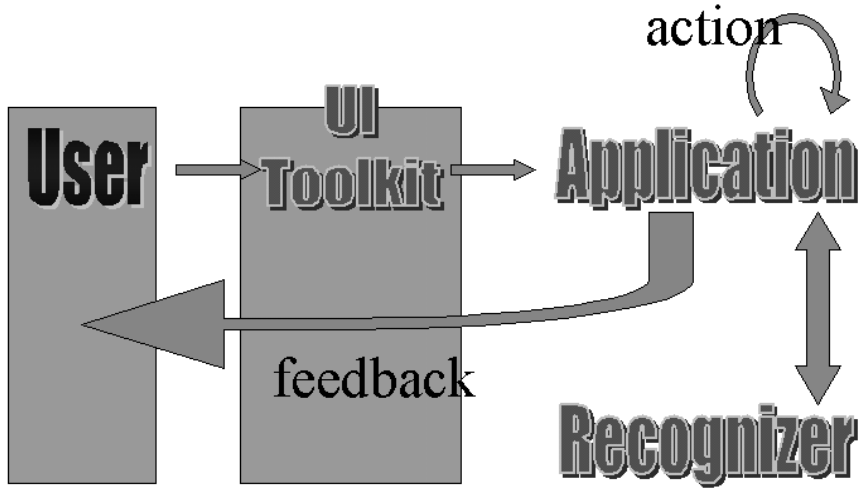


Fig. 5. A simple diagram of how recognizers, applications, and toolkits currently interact. Input (such as mouse and keyboard events) are delivered to the application by the toolkit. Meanwhile, the application talks directly to the recognizer passing it data to be recognized if necessary (such as the mouse events), and receiving recognition results from it. The application must take information from both sources and decide how to display it on the screen and what actions to take.

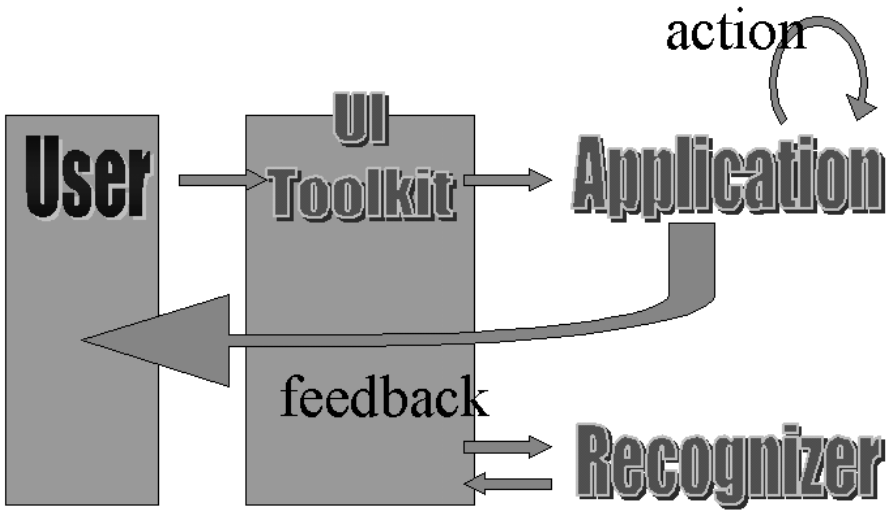


Fig. 6. An extension of Figure 5 in which the recognizer talks to the toolkit instead of the application. Recognition results are returned to the toolkit in the form of events, that are then delivered to the application just like events coming directly from the user such as mouse and keyboard events.

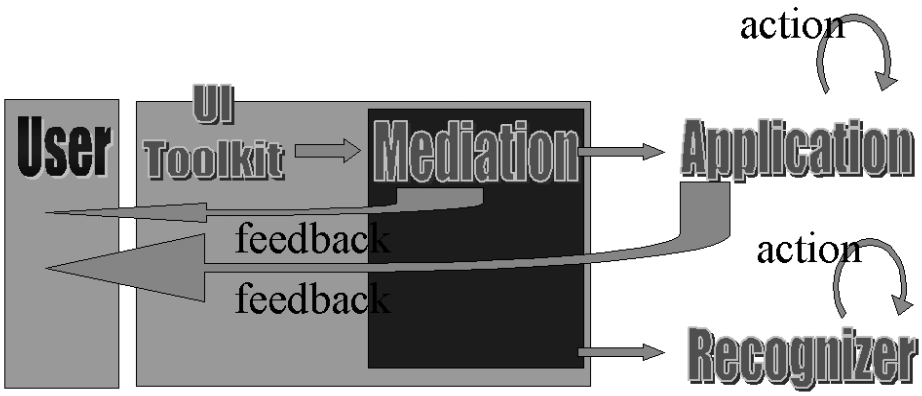
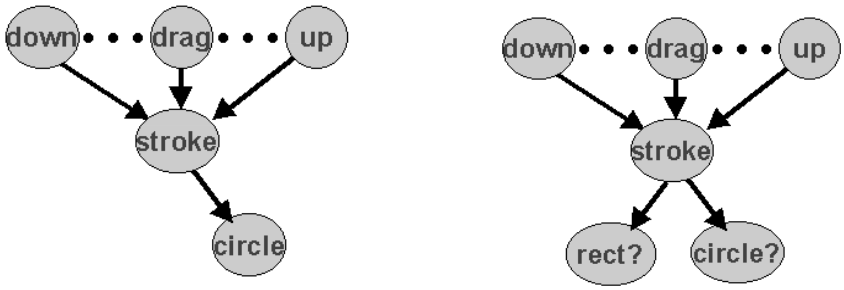


Fig. 7. **The mediation subsystem handles mediation for the toolkit. As events are accepted or rejected, the necessary components in application and recognizers are notified.**

4.1 Ambiguity generation

The most common input handling model in user interface toolkits today is the event-based model. In this model, user input is seen as changes in the state of input devices such as mouse motion and key presses. This input is split into a series of discrete events. These events are generally delivered to the application in two ways: through callbacks, or through some sort of input handler such as a state machine [16,31]. The latter approach is used in OOPS. Input handlers deliver semantic information as well as input events in this approach. For example, as the user clicks, drags, and releases the mouse, a `drag_handling` input handler may send the application messages like `drag_start`, `drag_feedback`, and `drag_end`.

Recent work in this area has extended the event model by creating a hierarchy of higher-level events from lower-level events [32]. The higher-level events are essentially the syntactic or semantic "meaning" of the low-level events [14]. For example, one interpretation of a `mouse-press` followed by a series of `mouse-drag`s and a `mouse-release` is a `stroke`. A recognizer might then further interpret that `stroke` as a circle (See Figure 8(a)). By making use of hierarchical events, we can treat the recognizer like any other event consumer, and its output like any other event. The application and recognizer can now speak to each other *through the toolkit* (see Figure 6). For example, Figure 9(a) depicts an application that uses text events. We have added a word predictor (a recognizer that tries to guess what word you are typing after each key press) to this application. The application, which consumes text events, is automatically informed about word prediction results because they are in text format (Figure 9(c)). It does not need to know that these results were produced by a



(a) (b)

Fig. 8. **A hierarchical event graph representing mouse events that have been interpreted as a stroke that has been further interpreted by a recognizer. (a) In an unambiguous world, the recognizer returns only one result. (b) In an ambiguous world, the recognizer returns two potential results, a circle or a rectangle. Both are represented in the graph until the ambiguity is resolved.**

(a) (b) (c)

Fig. 9. **This is an application that shows text typed by the user. We have hooked a word predictor up to the toolkit and now the application also shows word prediction results. (a) The word predictor before the user types the letter `f`. (b) A mediator asks the user which word prediction is correct. (c) The result**

recognizer instead of typed by the user on a keyboard.

Up to this point, nothing we have described particularly supports *ambiguous* recognized input. But support for ambiguity is a necessary piece of our infrastructure. The mediator shown in Figure 9(b) would have nothing to mediate if the infrastructure did not support ambiguity. Our approach is to extend the concept of hierarchical events to include ambiguity. This is done by allowing recognizers to create multiple, *conflicting* interpretations. So, for example, the recognizer could interpret the `stroke` in Figure 8(a) as either a rectangle *or* a circle (See Figure 8(b)). The resulting hierarchy is ambiguous because there are two, conflicting interpretations of the stroke. In order to resolve this ambiguity, one of those interpretations will need to be rejected as wrong and the other accepted as right, but a key observation is that *this does not need to*

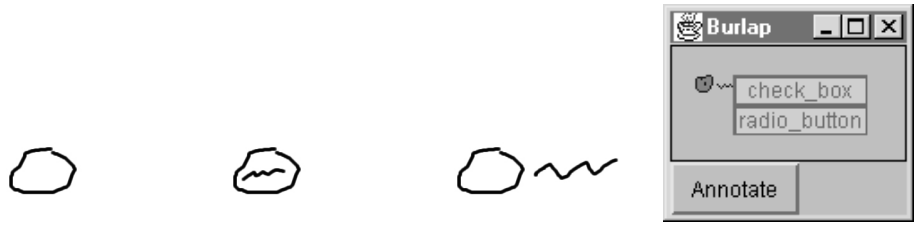


Fig. 10. **(a) The user draws a stroke that could either be a rectangle or a circle (b) The user draws a second stroke inside the first. This can only be a button, so it doesn't matter which interpretation of the first stroke was correct. (c) Alternatively, the user may draw the second stroke to the right of the first. This could be a radiobutton or a checkbox depending on which interpretation of the first stroke was correct. (d) When mediation occurs, we do not ask about the first stroke, we ask about the results of interpreting both the square and the rectangle.**

happen immediately.

For example, in Burlap there is no reason to make an immediate decision about whether the user drew a rectangle or circle (Figure 10(a)) until we see what the user does next. If they draw a squiggly line (representing text) inside the circle/rectangle 10(b), then we know they are drawing a button, and we no longer care whether or not the first stroke was rectangular. On the other hand, if they draw the squiggly line to the right of the circle/rectangle 10(c), then we have to decide if they intended to draw a radiobutton (circle) or a checkbox (square). Still, we never need to directly ask the user about the circle/rectangle. Instead, we can ask them whether the two combined strokes are a radiobutton or a checkbox (See Figure 10(d)).

In fact, we can put the decision off even longer. A radiobutton and checkbox look very similar (one reason they are easily confused). The main difference between them is how they act. Hence, we may choose to wait until the user actually clicks on the radiobutton/checkbox to ask them which interpretation is correct. If they never click on it, they probably didn't need to resolve this ambiguity in order to work out the interface design they were developing.

In order to support the example described above, and implemented in Burlap, we must propagate ambiguity throughout the input system. During the event delivery phase common in most toolkits, any interactor or recognizer can add additional interpretations to the hierarchy.

4.2 Mediation & Meta-mediation: Resolving ambiguity

Once every interested component has had an opportunity to add interpretations to the hierarchy, we can begin to prune it by rejecting or accepting conflicting events. This process is what we refer to as *mediation*.

Within this framework, mediation can be carried out using any of the techniques described in our survey in Section 3. In fact, system designers can choose from a diverse, extensible set of mediators. In order to allow this choice to be dynamic, we provide a *meta-mediation* system that selects mediators as ambiguity arises. For example, some mediators may only be interested in events that occured within a certain interactor, while others may be only interested in mouse events, and so on. The meta-mediation system handles these kinds of decisions.

The meta-mediation system consists of a series of meta-mediators that encode different policies for deciding between mediators, such as a linear priority policy, a positional policy, or a filter policy (based on event type, for example). These policies repeatedly select mediators until all ambiguity is resolved.

Mediators have the option to `pass`, `resolve` or `pause` mediation on ambiguous input given them by a meta-mediator. A mediator may `pass` if it cannot mediate something. Automatic mediators may `resolve` a portion of the hierarchy by accepting and rejecting events. They may also `pause` mediation if they need additional information (say further input events) to make a decision. For example, an automatic mediator is responsible for pausing mediation until the user clicks on the radiobutton/checkbox described above. Interactive mediators always `pause` since they depend upon user input to make their decision. For example, Figure 1(b) shows the interactive mediator that comes up once the user clicks on the radiobutton.

4.3 Maintaining and tracking ambiguity

As described above, the hierarchy is built as different components interpret events. Interpretations are *not certain*; they may be rejected. Because of this, it is important that no irreversible actions be taken based on those interpretations.

We handle this case by providing feedback only about uncertain interpretations. Action is put off until ambiguity is resolved. Essentially, what we are doing is separating the feedback/action loop shown in Figure 6 in time. During mediation (Figure 7), if an interpretation is accepted, the appropriate components will be notified that that interpretation was chosen. At this point,

they may act on the event. If its action is rejected, the components are also informed.

This differs from normal input handling because we separate feedback about events from action on those events. Because mediation may involve lengthy delays (for example, to acquire additional user input), it is important that interactors have the opportunity to provide early feedback about events.

4.4 Hiding ambiguity

The previous section describes necessary architectural changes in order to handle ambiguity. However, we do not require any input consumer to support these changes. We built the OOPS toolkit with the goal of impacting the user interface implementer as little as possible while still supporting the standard mediators described in Section 3. Because of this, we allow interactors to receive recognized input in a fashion that, on the surface, is identical to the way events were originally delivered in subArctic. By making changes *below* the input handling API, we ensured that we did not have to rewrite any of the interactors in the subArctic library. User interface implementors have the choice to write interactors in this older style, even if those interactors make use of recognized input and mediation.

These old-style interactors work by either "grabbing" events before they have been interpreted (that is, before any ambiguity is present), or "waiting" for events until after all ambiguity has been resolved through mediation. Designers can choose to make use of additional functionality if they wish to provide feedback about ambiguous events before mediation. However, since interactive mediation provides feedback to the user about ambiguous events even in the presence of old-style interactors, it is not necessary. Figure 9 shows an example of an application that was ported from subArctic to OOPS. Only two lines of code were changed during the port. First, a line was added to call the recognizer (a word predictor). Second, the `text_edit` interactor that receives the text was added to the list of interactors that "wait" for events. The default interactive mediator automatically comes up when the word predictor returns ambiguous results, and once the user resolves the ambiguity by selecting the correct interpretation, the chosen text is delivered to the `text_edit` interactor.

4.5 Comparison to other toolkits

Most UI toolkits today limit input to standard keyboard/ mouse-based input. However, some toolkit designers have taken the step of providing explicit

support for recognizing input. In addition to supporting recognition, some existing toolkits provide support for mediation.

Very few toolkits provide a principled model for dealing with ambiguity, and none integrate this into the toolkit input model shared by on-screen components. Lack of support for ambiguity forces designers to resolve ambiguity earlier than necessary, or to build one-off solutions for retaining information about ambiguity. This lack of support, combined with recognition results not being integrated into the input model, makes it difficult to build re-usable solutions for mediation.

We will split this discussion into a discussion of unimodal and multi-modal toolkit architectures. In general, unimodal architectures have focussed more on the issue of how to handle recognition seamlessly at an input level, while multi-modal toolkits focus on the problem of handling a variety of inputs and dealing with ambiguity.

Unimodal toolkits One of the earliest GUI toolkits to support gesture was the Arizona Retargetable Toolkit (Artkit) [16], a precursor to subArctic [19,10]. Artkit grouped related mouse events, sent them to a recognition object, and then sent them to components or other relevant objects. Artkit took the important step of integrating the task of calling the gesture recognition engine, into the normal input cycle, an innovation repeated later in Amulet [23]. In addition, objects wishing to receive recognition results did this through the same mechanism as other input results. However, Artkit did not support ambiguity, the recognizer was expected to return a single result.

Support for ambiguity was addressed in the work of Hudson and Newell on probabilistic state machines for handling input [18]. This work is most applicable to handling visual feedback. This approach is intended to be integrated into the event handlers that translate input events from lexical into semantic events. However, it does not address how ambiguity should be passed between event handlers, nor how mediation should be supported, both of which are dealt with by our toolkit.

Multi-modal toolkits In addition to toolkit support for building recognition-based interfaces, it is useful to consider toolkit support for handling multi-modal input [34,33]. Multi-modal input generally combines input from a speech recognizer and one or more other input modes. The additional modes may or may not involve recognition. Bolt used a combination of speech and pointing with a mouse in his seminal paper on multi-modal input [5]. In contrast, the Quickset system, which uses the Open Agent Architecture, combines speech with gesture recognition and natural language understanding [34].

These toolkits focus on providing support for combining input from multiple diverse sources of input. Unlike the unimodal toolkits described above,

the result is generally passed to the application to handle directly rather than integrated into the same input model as mouse and keyboard input as in our toolkit.

Explicit support for ambiguity was addressed in the Open Agent Architecture. Oviatt's work in mutual disambiguation tries to use knowledge about the complementary qualities of different modalities to reduce ambiguity [34]. In addition, McGee, Cohen and Oviatt experimented with different confirmation strategies (essentially interactive mediation) [30]. Although this work includes support for ambiguity, this support is not integrated into a GUI toolkit input model in a transparent fashion. As a result, the application must resolve ambiguity at certain fixed times or explicitly track the relationship between ambiguity and any actions. This makes it hard to build flexible, re-usable mediators.

5 USING THE ARCHITECTURE

This section discusses applications and mediators we have built and plan to build in order to validate the effectiveness of our toolkit architecture. Our first goal on completing OOPS was to populate it with a library of mediators based on the catalogue discussed in Section 3, where we showed that certain types of mediators (specifically choice and repetition) are used in a broad range of tasks. This suggests that we should provide defaults that support those types of mediation when the designer makes use of ambiguous input sources.

5.1 A library of mediators

We have designed a re-usable choice display that provides support for individually varying the dimensions described in the review of choice-based mediation. We are also working on a mixed repetition/choice based mediator. Currently, this mediator will be limited to text. However, since text is a fairly common representation for recognized input we feel that this will be a useful addition to our library.

Repetition shows up in two ways in our architecture. First, as in most repetition systems, errors of omission may easily be handled by the user by repeating her input. This really does not involve the toolkit architecture at all. Second, we have built an example mediator that allows the user to type in the correct interpretation of her input if the interpretation that appears on the screen is wrong.

In addition, we have created a set of meta mediation policies for deciding

which mediators to use. In particular, these include filter policies and priority policies.

Filter policy A filter policy consists of a prioritized set of filters. Each filter has an associated mediator. A filter is a class that selects portions of an ambiguous hierarchy which its mediator can mediate. For example, the radiobutton sequence mediator shown in Figure 3(b) can only mediate that one particular type of ambiguity.

Priority policy A priority policy consists of a prioritized set of mediators. Each is given the opportunity to mediate in turn.

5.2 Mediation examples, demonstrated in Burlap

In addition to many small applications, we have built one larger application to date, Burlap. Burlap includes input: from speech recognition (IBM ViaVoice^{TM}); gesture recognition (from Long's work [25]); and context identity (from work by Salber, Dey and Abowd [38]). In past work with a previous version of our toolkit, we built a simple application that supported drawing beautification [21].

As described throughout this paper, Burlap illustrates many of the dimensions covered in our survey of mediation techniques.

Error discovery Our model of recognition as ambiguity allows us to identify possible substitution errors. Because one of our 3rd-party recognizers only returns one interpretation for each stroke it recognizes, we use a confusion matrix to supplement this with other probably interpretations.

Thanks to our toolkit architecture, Burlap can handle between recognizer, as well as within recognizer, ambiguity. So, for example, when the user draws a stroke starting on a sketched radiobutton, this could be interpreted either as a click (selecting the radiobutton) or as the first stroke drawing a new interactor.

Repetition We handle errors of omission in Burlap by allowing the user to redraw any part of the interactor that was not recognized. Figure 2 shows an example of this.

Choice We have built a re-usable mediator that supports n-best list style choice mediation. Figure 3 shows an two different layouts that were created using this mediator as a base class. Figure 9(b) shows this same mediator being used in a different context.

Automatic mediation We use a form of rules in Burlap to identify "strokes" (sets of mouse clicks) that were really intended to be a single click (e.g. for selecting a radiobutton). This is a common problem with pen based interfaces, where a very quick press of the pen may generate several mouse

events. When our mediator finds a "stroke" (really a point or click) like this, any recognition of the stroke as a sketch is rejected.

Meta-mediation We use some simple meta-mediation strategies in Burlap which choose mediators based on things like the type of recognition being mediated.

Toolkit Overall, the design of Burlap benefited significantly from both our survey and toolkit. For example, because the toolkit maintains information about ambiguity and separates feedback from action, we can provide feedback to users about how their strokes are being interpreted without throwing away ambiguous information that may help us later to correctly translate their strokes into interactors. Figure 10 demonstrates an excellent example of this.

In general, the toolkit allows us to delay mediation as long as necessary where appropriate and then to choose the best style of interactive mediation once mediation is necessary. In many cases in this application, mediation never becomes an issue.

5.3 Additional Toolkit Benefits

One side-effect of our architecture is that recognizers can learn on the fly. Normally, recognizers must learn from a special training data set that has been annotated with information about the correct answer. This is because they need to know when they are wrong in order to make appropriate changes. Because OOPS knows when the user accepts or rejects an interpretation (through its mediators), and because OOPS automatically delivers this information to the recognizer (along with any other producer of interpretations), the recognizer now has access to information about its mistakes, and can learn from them.

Another beneficial side-effect is the ability to add recognition to applications not intended to use it, without changing those applications. As long as the interpretations created by a recognizer are of the same type as the input data an application was built to use, that application will receive those interpretations. Because the toolkit automatically detects ambiguity, these recognition results may event be mediated before they are passed on to the application. Figure 9 shows an example of an application that "uses" recognition in this fashion. The application was not written with recognition in mind, and only two lines of code were changed in order to add a recognizer to it.

6 CONCLUSIONS AND FUTURE WORK

We have shown the need for a OOPS, a UI toolkit consisting of a library of mediation techniques and the architecture needed to support them. We have populated OOPS's library with mediators and meta-mediators drawn from the literature and the needs of our demonstration application, Burlap. We have made necessary architectural changes to standard GUI input in order to support ambiguity, including the use of ambiguous hierarchical events. Ambiguity resolution, also called mediation, is handled by a special subsystem for mediation within OOPS. This subsystem is automatically invoked when ambiguity is present.

We have demonstrated OOPS with several applications, including Burlap and a simple text entry application combined with word-prediction. One of the purposes for producing this infrastructure is to provide support for future investigations of issues such as which mediation techniques work best with which types of errors. We also hope to explore increased architectural support for automatic mediation.

In future work, we plan to further fill out the library of mediators provided by default. In addition, we are investigating more novel mediation, including both new techniques and new settings for mediation. Some new settings include situations not normally associated with ambiguity such as context-aware computing [39], and uncertainties caused directly by user disabilities. In particular, we are building mediation of ambiguous sensor data into a context-aware In Out board, and we are providing mediation for users who have difficulty with fine motor control in the task of mouse clicking.

Acknowledgments

This work was supported in part by the National Science Foundation under grants IRI-9703384, EIA-9806822, IRI-9500942 and IIS-9800597. The Java-based unistroke gesture recognizer used for demonstration in this paper, GDT, was provided by Chris Long from UC Berkeley as a port of Dean Rubine's original work [37]. Takeo Igarashi provided the drawing beautifier recognizer, Pegasus, that was also used for demonstrational purposes [21].

References

[1] W. A. Ainsworth and S. R. Pratt. Feedback strategies for error correction in speech recognition systems. *International Journal of Man-Machine Studies*, 36(6):833–842, 1992.

[2] N. Alm, J. L. Arnott, and A. F. Newell. Prediction and conversational momentum in an augmentative communication system. *Communications of the ACM*, 35(5):46–57, May 1992.

[3] I. Apple Computer. The Newton MessagePad^{TM}.

[4] C. Baber and K. S. Hone. Modelling error recovery and repair in automatic speech recognition. *International Journal of Man-Machine Studies*, 39(3):495–515, 1993.

[5] R. A. Bolt. "Put-That-There": Voice and gesture at the graphics interface. *Computer Graphics*, 14(3):262–270, July 1980.

[6] S. E. Brennan and E. A. Hulteen. Interaction and feedback in a spoken language system: A theoretical framework. *Knowledge-Based Systems*, 8(2-3):143–151, 1995.

[7] R. V. Buskirk and M. LaLomia. The just noticeable difference of speech recognition accuracy. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 of *Interactive Posters*, page 95, 1995.

[8] N. C. Corporation. http://www.netscape.com. Web Page.

[9] DragonDictate. http://www.dragonsystems.com/products/dragondictate. Product Web page.

[10] W. K. Edwards, S. E. Hudson, J. Marinacci, R. Rodenstein, I. Smith, and T. Rodrigues. Systematic output modification in a 2D UI toolkit. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST-97)*, pages 151–158, New York, Oct. 14–17 1997. ACM Press.

[11] C. Frankish, R. Hull, and P. Morgan. Recognition accuracy and user acceptance of pen interfaces. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 of *Papers: Pen Interfaces*, pages 503–510, 1995.

[12] C. Frankish, D. Jones, and K. Hapeshi. Decline in accuracy of automatic speech recognition as function of time on task: Fatigue or voice drift? *International Journal of Man-Machine Studies*, 36(6):797–816, 1992.

[13] D. Goldberg and A. Goodisman. STYLUS user interfaces for manipulating text. In *Proceedings of the ACM Symposium on User Interface Software and Technology — UIST'91*, pages 127–135. ACM Press, 1991.

[14] M. Green. A survey of three dialogue models. *ACM Transactions on Graphics*, 5(3):244–275, July 1986.

[15] S. Greenberg, J. J. Darragh, D. Maulsby, and I. H. Witten. *Extra-ordinary Human-Computer Interaction: interfaces for users with disabilities*, chapter Predictive Interfaces: what will they think of next?, pages 103–139. Cambridge series on human-computer interaction. Cambridge University Press, New York, 1995.

[16] T. R. Henry, S. E. Hudson, and G. L. Newell. Integrating gesture and snapping into a user interface toolkit. In ACM, editor, *UIST. Third Annual Symposium on User Interface Software and Technology. Proceedings of the ACM SIGGRAPH Symposium, Snowbird, Utah, USA, October 3–5, 1990*, pages 112–122, New York, NY 10036, USA, Oct. 1990. ACM Press.

[17] E. Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1, pages 159–166, 1999.

[18] S. E. Hudson and G. L. Newell. Probabilistic state machines: Dialog management for inputs with uncertainty. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Toolkits, pages 199–208, 1992.

[19] S. E. Hudson and I. Smith. Supporting dynamic downloadable appearances in an extensible UI toolkit. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST-97)*, pages 159–168, New York, Oct. 14–17 1997. ACM Press.

[20] W. Huerst, J. Yang, and A. Waibel. Interactive error repair for an online handwriting interface. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems (Summary)*, volume 2 of *Student Posters: Interaction Techniques*, pages 353–354, 1998.

[21] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Interactive beautification: A technique for rapid geometric design. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Constraints, pages 105–114, 1997.

[22] G. Kurtenbach and W. Buxton. User learning and performance with marking menus. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1 of *Pen Input*, pages 258–264, 1994.

[23] J. A. Landay and B. A. Myers. Extending an existing user interface toolkit to support gesture recognition. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems – Adjunct Proceedings*, Short Papers (Posters): Interaction Techniques I, pages 91–92, 1993.

[24] J. A. Landay and B. A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 of *Papers: Programming by Example*, pages 43–50, 1995.

[25] A. C. Long, Jr., J. A. Landay, and L. A. Rowe. Implications for a gesture design tool. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1 of *Alternatives to QWERTY*, pages 40–47, 1999.

[26] J. Mankoff, G. D. Abowd, and S. E. Hudson. Interacting with multiple alternatives generated by recognition technologies. Technical Report GIT-GVU-99-26, Georgia Institute of Technology GVU Center, 1999.

[27] J. Mankoff, S. E. Hudson, and G. D. Abowd. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proceedings of ACM CHI 2000 Conference on Human Factors in Computing Systems*, pages 368–375, 2000.

[28] M. Marx and C. Schmandt. Putting people first: Specifying proper names in speech interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology — UIST'94*, pages 30–37. ACM Press, 1994.

[29] T. Masui. An efficient text input method for pen-based computers. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1 of *In Touch with Interfaces*, pages 328–335, 1998.

[30] D. R. McGee, P. R. Cohen, and S. Oviatt. Confirmation in multimodal systems. In *Proceedings of the International Joint Conference of the Association for Computational Linguistics and the International Committee on Computational Linguistics (COLING-ACL)*, Montreal, Quebec, Canada, 1998.

[31] B. A. Myers. A new model for handling input. *ACM Transactions on Information Systems*, 8(3):289–320, July 1990.

[32] B. A. Myers and D. S. Kosbie. Reusable hierarchical command objects. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Development Tools*, pages 260–267, 1996.

[33] L. Nigay and J. Coutaz. A generic platform for addressing the multimodal challenge. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 of *Papers: Multimodal Interfaces*, pages 98–105, 1995.

[34] S. Oviatt. Mutual disambiguation of recognition errors in a multimodal architecture. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1 of *Speech and Multimodal Interfaces*, pages 576–583, 1999.

[35] A. Poon, K. Weber, and T. Cass. Scribbler: A tool for searching digital ink. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 of *Short Papers: Pens and Touchpads*, pages 252–253, 1995.

[36] B. J. Rhodes and T. Starner. Remembrance agent. In *The Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology (PAAM '96)*, pages 487–495, 1996.

[37] D. Rubine. Specifying gestures by example. *Computer Graphics*, 25(4):329–337, July 1991.

[38] D. Salber, A. K. Day, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of ACM CHI 99*

Conference on Human Factors in Computing Systems, volume 1 of *Tools for Building Interfaces and Applications*, pages 434–441, 1999.

[39] B. N. Schilit, N. I. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994. IEEE Computer Society.

[40] B. Suhm. Empirical evaluation of interactive multimodal error correction. In *IEEE Workshop on Speech recognition and understanding*, Santa Barbara (USA), Dec 1997. IEEE.

[41] B. Suhm, B. Myers, and A. Waibel. Designing interactive error recovery methods for speech interfaces. In *CHI 96 Workshop on Designing the User interface for Speech Recognition applications*. SIGCHI, 1996.

[42] B. Suhm, A. Waibel, and B. Myers. Model-based and empirical evaluation of multimodal interactive error correction. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1 of *Speech and Multimodal Interfaces*, pages 584–591, 1999.

[43] P. N. Sukaviriya, J. D. Foley, and T. Griffith. A second generation user interface design environment: The model and the runtime architecture. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, Model-Based UI Development Systems, pages 375–382, 1993.