Error Correction Techniques for Handwriting, Speech, and other ambiguous or error prone systems

ABSTRACT

Interfaces which support natural inputs such as handwriting and speech are becoming more prevalent and this is a desirable trend. However, these recognition-based interface techniques are error prone. Despite research efforts to improve recognition rates, a certain amount of error will never be removed. Suitable research efforts should attend to the problem of correction techniques for these error prone techniques. Humans have developed countless ways to correct errors in understanding or clarify ambiguous statements. It is time for interface designers to focus on ways for computers to do the same. We present a survey of a new area of user interface research —the design, implementation, and study of interfaces for correcting error prone input technologies. We ground the definition of this research topic in previous work by others and our own research into flexible pen-based note-taking environments.

KEYWORDS: handwriting and speech recognition, interface design, error handling

1 INTRODUCTION

1.1 Motivating the Problem

Computer interfaces which support more natural human forms of communication (e.g. handwriting, speech, and gestures) are beginning to supplement or replace elements of the GUI paradigm. These interfaces are lauded for their low learning curves and their ability to support tasks such as authoring and drawing without drastically changing their structure. Additionally, they can be used by people with disabilities that make the traditional mouse and keyboard less accessible.

Unfortunately, these new interfaces come with a new set of problems —they make mistakes. When errors occur, the initial reaction of system designers is to try to eliminate them, for example by improving recognition accuracy. This is often a difficult task —Buskirk & LaLomia (1995) found that an improvement of 5-10% is necessary before the majority of people will even notice a difference in a speech recognition system.

Worse yet, eliminating errors may not be possible. Even *humans* make mistakes when dealing with these same forms of communication. As an example, consider handwriting recognition. Even the most expert handwriting recognizers (humans) can have a recognition accuracy as low as 54% when looking at word fragments without the benefit of their context (Schomaker, 1994). Human accuracy increases to 88% for cursive handwriting (Schomaker, 1994), and 96.8% for printed handwriting (Frankish et al., 1995), but it is never perfect. This evidence all points to the conclusion that computer handwriting recognition will never be perfect.

Computer-based recognizers are even more error prone than humans. The data they start with is often less fine-grained than that which humans are able to sense. They have less processing power. And variables such as vocal fatigue can cause usage data to differ significantly from training data, causing reduced recognition accuracy over time in speech recognition systems (Frankish et al., 1992).

On the other hand, recognition accuracy is not the only determinant for user satisfaction. Both the complexity of error recovery dialogues (Zajicek & Hewitt, 1990), and the amount gained for the effort (Frankish et al., 1995), affect user satisfaction. For example, Frankish found that users were less frustrated by recognition errors when the task was to enter a command in a form than when they were writing journal entries. He suggests that this is because the pay-back for entering a single word in the case of a command is much larger than in a paragraph of a journal entry when compared with the effort of entering the word.

Error handling is not a new problem. In fact, it is endemic to the design of computer systems which attempt to mimic human abilities. Research in the area of error handling for recognition technologies must assume that errors will occur, and then answer questions about the best ways to deal with them. The goal of this paper is to present a survey of existing research in discovering and correcting errors in recognition based interfaces.

1.2 Defining The Area

Our survey has have identified five key research areas for error handling of recognition-based interfaces.

Error reduction Error reduction involves research into improving recognition technology in order to eliminate or reduce errors. It has been the focus of extensive research, and could easily be the subject of a whole paper on its own. Evidence suggests that its holy grail, the elimination of errors, is probably not achievable. And big improvements (5-10%) are required before users even notice a difference (Buskirk & LaLomia, 1995). Because of these facts, we have chosen not to address error reduction in this paper.

Error discovery Before either the system or the user can take any action related to a given error, one of them has to know that the error has occured. The system may be told of an error through user input, and can help the user to find errors through its output. In addition, system designers have used three techniques to automate error discovery —thresholding, rules, and historical statistics.

Error correction techniques Just as the user interface is the only way one party can inform the other that an error has occured, it is also the only way that the user can correct an error. We found that current error handling techniques fall into three main categories —choosing a default, encouraging less ambiguous input, and mimicking natural human correction strategies.

Validation of techniques Validation goes hand in hand with research into error correction techniques. Validation is the only way to determine the effectiveness of different designs. Our survey uncovered research into theoretical issues such as how to compare techniques, and practical results such as which techniques are effective.

Toolkit level support Toolkits provide reusable components and are most useful when a class of common, similar problems exists. Interfaces for error handling would benefit tremendously from a toolkit which could be used and re-used every time an error prone situation arose. In addition to interface widgets, a toolkit would need to support complete reversibility, and keep track of multiple potential interpretations at once.

In addition to surveying existing work, we are building a platform to test strategies for dealing with segmentation errors, handwriting recognition errors, and gesture recognition errors (see Figure 1). Our system, called PenPad, supports handwriting recognition in the context of personal note-taking. Our motivation for this application is to support note taking and document creation in situations when typing is not an option. This includes mobile settings, and users with repetitive stress injuries or other disabilities which make keyboard typing difficult.

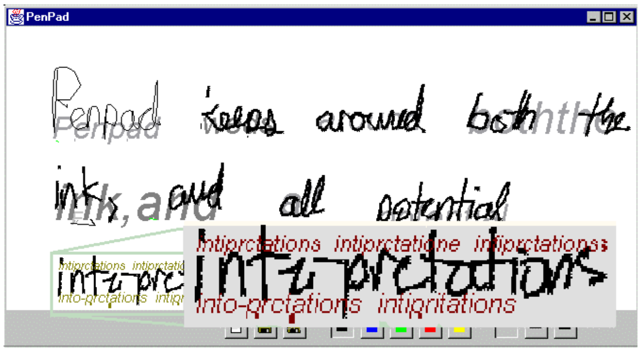

Figure 1: PenPad's user interface. The words: *Penpad; around; both the; all; potential;* were all recognized correctly. The darker the word, the surer the recognizer is of this. The word "interpretations" was recognized incorrectly. When the user moves the mouse over this word, five alternatives are displayed, shown in the blow-up. The words "ink, and" were originally incorrect, but the user was able to select them from a similar set of five potential choices.

The rest of this paper describes the results of our survey. We discuss research in each of the last four sub-areas mentioned above —error discovery, error correction techniques, validation of techniques, and toolkit level support.

2 ERROR DISCOVERY

Before the system can support error recovery in any way, or the user can handle an error, one or the other needs to know that an error has occurred. The user interface is a conduit through which the system and user can pass information. User input can notify the system of an error (and correct it, described in more detail in the next section). And it is through visual or oral feedback that the system helps the user to identify errors.

The system can also try to determine when it has made a mistake without the user's help, either through thresholding (Baber & Hone, 1993; Poon et al., 1995; Brennan & Hulteen, 1995), a rule base (Baber & Hone, 1993; Davis, 1979), or historical statistics (Marx & Schmandt, 1994).

2.1 User input to help the system find errors

In the most common approaches to notification, the user explicitly indicates the presence of an error by, for example, clicking on a word, or saying a special keyword. Many speech and handwriting recognition systems use this approach. Three well known examples are the PalmPilot^{tm}, DragonDictate^{tm}, and the Apple MessagePad^{tm}. For example, when the user clicks on a word in the Apple MessagePad^{tm}, a menu of alternative

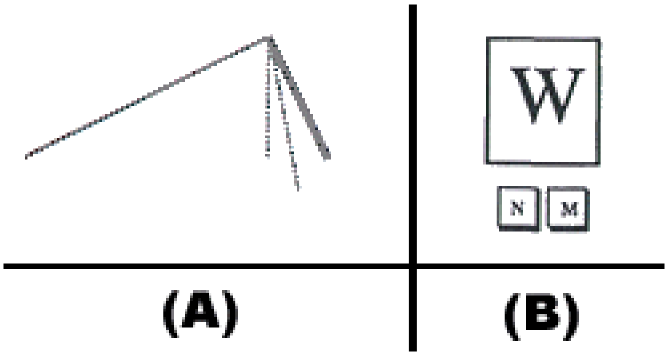

Figure 2: Pictures of two user interfaces, adapted from a paper about drawing understanding (A, left) (Goldberg & Goodisman, 1991), and pen input (B, right) (Igarashi et al, 1997)

interpretations appears.

In cases where there is no special interface for notification or correction, user action may still help the system to discover errors. For example, if the user deletes a word and enters a new one, the system may infer that an error has occurred by matching the deleted word to the new one.

2.2 System output to help the user find errors

There is a plethora of hidden information available to the system designer which can help users to identify errors. The likelihood that something is correct, the history of values an item has had, other possible values it could have, and the user's original input are just a few of the non application-specific ones. Our survey shows that designer after designer has found it beneficial to reveal some of this hidden information to the user (Brennan & Hulteen, 1995; Davis, 1979; Goldberg & Goodisman, 1991; Igarashi et al., 1997; Kurtenbach et al., 1994; Rhodes & Starner, 1996) Two of the most common pieces of information to display are the probability of correctness (called certainty in this paper), and multiple alternatives.

An example of a system which shows information about certainty is the PenPad system. The probability of correctness is displayed through color. For example, the typewritten word *PenPad* is lighter (less certain) than the corresponding words *ink, and* in Figure 1. Figure 2 shows two example systems which display multiple alternatives. The first (Figure 2A) is a drawing understanding system designed by Igarashi et al. (1997). The bold line represents the system's current top guess. The dotted lines represent potential alternatives, and the plain line is a past accepted guess. Figure 2B shows a character recognition system designed by Goldberg & Goodisman (1991). The larger character is the system's top choice; the two smaller letters are the second and third most likely possibilities. In both systems, the user can click on an alternative to tell the system that its default choice should be changed. In both systems, if the user continues input as normal, they are implicitly accepting the default choice. Interestingly, although Igarashi had success with this approach in his drawing-understanding system, Goldberg and Goodisman found that it required too great a cognitive overhead to be effective in their character recognition system.

Both certainty and the display of multiple alternatives can also be achieved in an audio-only setting, as demonstrated by Brennan & Hulteen (1995). They base their approach on linguistic research showing that humans reveal positive and negative evidence as they converse. Positive evidence is output which confirms that the listener has heard the speaker correctly. For example, the listener may spell back a name which has just been dictated to them. Negative evidence is output which somehow reveals that the listener (in this case, the recognition system) is not sure they have understood the speaker correctly. Examples are repeating the speaker's sentence and replacing the questionable word with a pause or simply saying "Huh?" Negative evidence can also be used to display multiple alternatives, So, for example, the system may say "call *John* or *Jane*?" in response to a user's request. Brennan and Hulteen built a sophisticated response system using both techniques. They make use of positive and negative evidence, and they limit the display of alternatives based on a contextual analysis of the likelihood of correctness.

Another setting in which multiple alternatives are commonly displayed is word prediction (Alm et al., 1992; Greenberg et al., 1995). Word prediction is often used to support communication and productivity for people with disabilities which make typing, and in some cases even using a mouse, very difficult. As the user types each letter, the system retrieves a list of words which are the most likely completions of what has been typed so far. Often there are a large number of potential completions, and many are displayed at some distance from the actual input on screen.

2.3 Thresholding

Many error prone systems return some measure of the probability that each result is correct when they return the result. This probability represents the confidence of the interpretation. The resultant probabilities can be compared to a threshold. When they fall below the threshold, the system assumes an error has occurred. When they fall above it, the assumption is that no error has occurred. Most systems set this threshold to zero, meaning they never assume that there has been a mistake. Some systems may set it to one, meaning they always assume they are wrong (e.g., word prediction), and other systems try to determine a reasonable threshold based on statistics or other means (Poon et al., 1995; Brennan & Hulteen, 1995; Baber & Hone, 1993).

2.4 Rules

Baber & Hone (1993) suggest using a rule base to determine when errors may have occurred.This can prove to be more sophisticated than either statistics or thresholding since it allows the use of context in determining whether an error has occurred. An example rule might be:

When the user has just written 'for (', lower the probability of correctness for any alternatives to the next word they write which are not members of the set of variable names currently in scope.

This goes beyond simple statistics because it uses knowledge about the context in which a word has been written to detect errors.

2.5 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 doing a statistical analysis of historical data about when and where the system makes mistakes. This talk itself benefits from good error discovery. A historical analysis can help to increase the accuracy of both thresholding and rules. For example, Marx & Schmandt (1994) compiled speech data about which letters were misrecognized as "*e*", with what frequencies, and used them as a list of potential alternatives whenever the speech recognizer returned "*e*". They did the same for each letter of the alphabet.

The example below shows pen data for "*e*" generated by the first author by repeating each letter of the alphabet 25 times in a PalmPilot^{tm}. The first column represents the letter that was written; the other columns show which letters the PalmPilot^{tm} Graffiti^{tm} recognizer returned. Only letters which were mistaken for "*e*" are shown.

original	top guess	other guesses
e	*e*(100%)	
k	k(72%)	l(16%), *e*(8%), s(4%)
l	l(80%)	c(17%), *e*(3%)

This sort of matrix is called a *confusion matrix* because it shows potential correct answers that the system may have confused with its returned answer. In this way, historical statistics may provide a default probability of correctness for a given answer. More sophisticated analyses can help in the creation of better rules or the choice of when to apply certain rules.

Although error discovery is a necessary component of error handling interfaces, it has a stigma associated with it: The task of error discovery is itself error prone. Rules, thresholding, and historical statistics may all be wrong. Even when the user's explicit actions are observed, the system may incorrectly infer that an error has occurred. Only when the user's action is to explicitly notify the system 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 error handling issues.

3 ERROR CORRECTION TECHNIQUES

Once a mistake has been identified, the system can take action to correct it, or ask the user's help in correcting it (through some sort of error handling interface). Alternatively the system can support error handling in an integrated fashion. For example, the interactive beautification system shown in Figure 2A displays alternatives after every stroke. The same interface also supports notification —if the user selects an alternative, the system can infer that the original default was wrong and the alternative is correct.

Most of the tasks being supported require the selection of a single correct interpretation of user input (one exception to this is search engines, which may have multiple correct responses). One important choice facing the designer of error handling techniques is how active the system should be in selecting this interpretation. Essentially, the designer must choose whether to accept the most certain choice by default, or to wait for user confirmation. The first part of this section discusses where each choice has shown up in the literature, and why. The remaining parts discuss two commonly used techniques for error handling, encouraging less ambiguous input, and mimicking natural human correction strategies.

3.1 Choosing a Default

The number of answers returned by an error prone system is often larger than the number of answers expected by the user. This leaves the interface designer with the choice of selecting none of the answers, or selecting one (or more) of the answers as "correct" by default. For example, the drawing understanding system mentioned above selects one line by default (shown bold in Figure 2A) (Igarashi et al., 1997). The interface designer should use information about the probability of correctness and the overhead for correcting a mistaken choice of default to decide when it is appropriate to choose a default. In the case of the drawing understanding system, the interface is designed so that the user does no more work when the system selects a default than when it doesn't. And if the system selects the correct choice, the user does less work (since they don't have to select it themselves before they continue drawing).

An example of a system which does well to select nothing by default is Rhodes & Starner's (1996) *remembrance agent*. The remembrance agent retrieves documents based on their relevance to the current text in an editor. Rather than immediately displaying the most relevant document, it has a small permanent window where it shows a single line from each of three potentially interesting documents. Actually selecting a document and displaying it would be far more invasive, difficult to correct, and often not what the user wants. Even if the system has found relevant documents, the user may not want to be interrupted in order to read them.

Word prediction systems also demonstrate why the designer may choose not to select a default. If, for example, the system assumes its top prediction is correct, it will insert it. But word prediction is a particularly difficult task in which the top choice is often wrong. And it will most likely take more keystrokes for the user to delete the mistake and continue typing than it would to have simply typed the whole word out in the first place, especially if similar mistakes happen automatically after every character typed.

Even when it is appropriate to choose a default for the user, this choice may be wrong, and because of this the user interface needs to support error correction. One way to support this is to display alternatives from which the user can select a correct choice. Another approach is to unobtrusively provide ways to change the default without necessarily displaying alternatives. For example, Goldberg & Goodisman (1991) suggest using a simple gesture (a tap) to select the next choice. As another example, consider the *Tivoli* system in which some inputs are interpreted as gestures and others simply as ink to be drawn on the screen (Moran et al., 1997). If a user draws a gesture which could trigger an action, such as "move", the system by default assumes that the action is intended (and not simply drawing on the screen). However, if the user doesn't follow through (by selecting an object to move in this case), Moran et al. automatically undo it, replacing it instead with its alternate interpretation as plain ink.

3.2 Encouraging Less Ambiguous Input

Certain modes of input are known to be less error prone than others (compare typing to handwriting recognition), and there are times when it is appropriate to make use of this fact. For example, Suhm found that recognition accuracy actually *decreases* by 10–65% during this sort of error repair in a speech recognition system (Suhm, 1997)). One option is for the computer to offer a less ambiguous input method as an alternative. This technique has been used effectively in the Apple MessagePad^{tm}, as well as for speech input (Marx & Schmandt, 1994), pen input (Goldberg & Goodisman, 1991), and a mixture of the two (Suhm et al., 1996b).

Alternatively, an interface designer may choose to encourage a less error prone input from the outset. For example, the designers of the PalmPilot^{tm} chose to use a unistroke alphabet (Goldberg & Richardson, 1993). It is easier to recognize unistrokes than to recognize handwriting because there is no possibility of segmentation errors since each letter is exactly one stroke (pen up to pen down). In another example, Goldberg & Goodisman (1991) suggest using on-screen marks (boxes) to reduce segmentation errors and discourage cursive handwriting.

Several researchers have made use of a human's tendency to mimic the output of whatever they are communicating with. Zoltan-Ford (1991) found that people will mimic sentence structures of the computer's responses, something that helps to make natural language processing easier. Kurtenbach et al. (1994) investigated the use of crib sheets which display gestures for a user to copy. The user can request an animation of a command by clicking on its picture on the crib sheet. Crib sheets have also been found to successfully improve recognition in a character recognition system (Wolf, 1990).

3.3 Mimicking Natural Human Correction Strategies

Although computers are a major source of errors, humans also make mistakes. Both experience and research have shown that humans already have ways of correcting mistakes. They may cross out a letter or add another letter or word to what they just wrote. When they mis-speak, they may pause, or repeat the correct word; with or without the addition of non-speech audio cues to indicate an error. These are what we call 'natural' correction strategies.

These "corrections" are so natural that users may do them even though most recognizers don't know how to interpret the corrections. Huerst et al. (1998) have experimented with a handwriting pre-processor which looks for and applies these corrections before sending handwriting to the recognizer.

Essentially what the user is doing in this strategy is correcting their original input in its original form, perhaps even in the midst of entering it. This is done with handwriting, speech, and more novel types of input. For example, we support this strategy in our unistroke keyboard (Mankoff & Abowd, 1998).

Brennan & Hulteen's (1995) work in applying linguistic research to interface design (described above in the section on Error Discovery) also demonstrate the usefulness of mimicking humans. One human strategy which they don't mimic is the pause. However, the questions of when and how long to pause have been investigated by several researchers (Aref et al., 1995; Kato & Nakagawa, 1995; Lopresti & Tomkins, 1995; Kurtenbach et al., 1994).

For example, Kurtenbach et al. (1994) use a pause to allow a user to request guidance when drawing a gesture. Each gesture is really a selection from a pie menu, which is only displayed if the user pauses (these gesture sets are called marking menus). A pause can also be used to support delayed, or lazy recognition. Schomaker

(1994) suggests echoing the input in the case of "invisible" commands such as gestures. The system could then provide a moment in which the user can act to undo a command before it becomes permanent.

4 VALIDATION OF TECHNIQUES

Designers need some basis for choosing between the huge number of possible techniques that can support error handling. User studies, and other standard HCI methods for gathering qualitative and quantitative data about user interfaces, can be a major source of guidance. A variety of results which can guide us in the design of error recovery interfaces are already present in the literature. Although many of these studies are small and limited in their representation, this only demonstrates how much we have to gain from investigating the area more deeply.

One place to begin is by observing users in situations where error correction occurs —both in everyday life (Baber & Hone, 1993; Zajicek & Hewitt, 1990), and in interactions with error prone computer programs (Nanja & Cook, 1987). For example, both Baber and Hone, and Zajicek and Hewitt, studied the effectiveness of human-like recovery strategies in the context of speech recognition. Their work verifies that linguistic theories about human conversation patterns can be used to guide error recovery techniques.

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. Some innovative work in measuring frustration quantitatively as well as qualitatively was done by Riseberg et al. (1998) in their research of affect (the measurable aspects of emotions).

In order to compare studies of different interfaces for error correction which can be used in the same application, Suhm (1997) suggests normalizing the data based on the number of errors which occur. For systems which generate ASCII, he also devised a way to relate accuracy to words per minute (Suhm et al., 1996a).

The simplest type of error correction possible is to simply repeat the input which was mistaken. Our survey uncovered several studies which compare some more sophisticated correction technique to repeat. Zajicek and Hewitt found that users prefer to repeat their input at least once before having to choose from a menu, a finding confirmed by Ainsworth & Pratt (1992). Also, in the realm of pen input, Goldberg & Goodisman (1991) found that even when alternative guesses are displayed, it takes too much cognitive effort for the user to select from them,a result that meshes with observations about input speed made in the word prediction community (Alm et al., 1992). Baber & Hone (1993) give a good overview of the pros and cons of repetition vs choice. Suhm (1997) added to this work when he found that spoken repetition is faster than choosing from a list, but something like partial word repair is better than both. Partial word repair allows users to correct part of a word when it is almost correct. This could be done either with a pen or with spoken input.

User testing can help to identify the sources of errors as well as with the design of error handling techniques. For example, Frankish et al. (1995) found that systems tend to misunderstand a subset of possible written inputs much worse than the rest, a result confirmed by Marx & Schmandt (1994)in the realm of speech recognition.

5 TOOLKIT LEVEL SUPPORT FOR ERROR HANDLING

Toolkits support the creation and use of reusable components. One of the most common application areas for toolkits is building user interfaces by combining widgets, especially graphical user interfaces. One benefit of using toolkits is that they make sophisticated interface features available to every programmer. If we can identify reusable components in the domain of error handling, perhaps we can provide toolkits which make it more likely that interface designers will include support for error handling in their interfaces.

The domain of interfaces for error handling has significant overlap with toolkits for building user interfaces. There are two key features that are needed to support error handling —complete reversibility, and support for keeping track of multiple potential interpretations at once. Without the former, the system may not easily be able to undo wrong choices which the user may have had no part in (and thus may be unable to help to correct). Without the latter, the system has to commit to a single interpretation at each stage, possibly throwing away potentially useful data. Incorporating multiple potential interpretations into the interface is also difficult without the support of the toolkit.

There is a lot of work on introducing undo/ reversibility into GUI toolkits, particularly in the object-oriented toolkits. For example, the Amulet system supports both regular and selective undo (Myers & Kosbie, 1996). A more theoretical treatment of the subject can be found in Thimbleby's (1990) book on *User Interface Design*.

In addition, both reversibility and support for multiple potential interpretations were addressed in the work of Hudson & Newell (1992) on probabilistic state machines for handling input. However, this work focused on the event handling stage of an interface toolkit and is most applicable to handling visual feedback. For example, if there is uncertainty as to whether the user is pointing at button A or button B, a probabilistic state machine would simplify the task of highlighting both buttons. However, it is not clear that the same system could keep track of the multiple potential system states that might result from potentially pressing both buttons.

In experimenting with the best way to support correction in PenPad (see Figure 1), the need for a toolkit became immediately obvious. Even attempts at a simple problem such as trying out different approaches to displaying differing numbers of alternatives around each handwritten word are hindered by the lack of toolkit level support. One of our first goals is to create a toolkit which will help to solve these problems. This infrastructure will also simplify the task of comparing and testing techniques.

6 CONCLUSION

We have introduced a new area of study: Error handling techniques for recognition, prediction, search, and other ambiguous or error prone systems. This paper surveys work which has already been done in the areas of error discovery, error handling techniques, validation, and toolkit level support. Although we have uncovered extensive work in many of these areas, significant questions remain.

Error discovery How can we improve the accuracy of error discovery? How should errors in error discovery be handled? What is the best technique for error discovery and how does this change depending on the situation?

Error correction techniques Does error handling require new types of interfaces or widgets different from other interfaces? When should error handling occur? How integrated should error handling interfaces be with the normal workflow/interface?

Validation of techniques How can we compare methods across applications? Is it possible to uncover general rules for the design of error handling interfaces?

Toolkit level support Is it possible to separate out and encapsulate interface techniques for error handling? What techniques belong in such a toolkit? Is complete reversibility possible, and if not what are the alternatives? Are there efficient ways of keeping track of increasing numbers of probabilities?

In our own research, we are developing the PenPad system as a platform for answering some of these questions. Our first task is to develop a toolkit which supports the techniques uncovered in this survey. In addition to encapsulating standard techniques, we plan to continue to work on developing innovative new techniques and investigating existing HCI techniques which could be applied to error correction.

REFERENCES

Ainsworth, W. A. & Pratt, S. R. (1992), "Feedback Strategies for Error Correction in Speech Recognition Systems", *International Journal of Man-Machine Studies* **36**(6), 833–842.

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

Aref, W. G., or Kamel, I. C. & Lopresti, D. P. (1995), "On Handling Electronic Ink", *ACM Computing Surveys* **27**(4), 564–567.

Baber, C. & Hone, K. S. (1993), "Modelling Error Recovery and Repair in Automatic Speech Recognition", *International Journal of Man-Machine Studies* **39**(3), 495–515.

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

Buskirk, R. V. & LaLomia, M. (1995), The Just Noticeable Difference of Speech Recognition Accuracy, *in Proceedings of ACM CHI'95 (Poster)*, p.95.

Davis, R. (1979), "Interactive transfer of expertise: acquisition of new inference rules", *Artificial Intelligence* **12**, pp.121–157.

Frankish, C., Hull, R. & Morgan, P. (1995), Recognition Accuracy and User Acceptance of Pen Interfaces, *in Proceedings of ACM CHI'95*, pp.503–510.

Frankish, C., Jones, D. & Hapeshi, K. (1992), "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.

Goldberg & Goodisman (1991), STYLUS User Interfaces for Manipulating Text, *in Proceedings of ACM UIST'91*, pp.127–135.

Goldberg, D. & Richardson, C. (1993), Touch-Typing with a Stylus, *in Proceedings of ACM INTERCHI'93*.

Greenberg, S., Darragh, J. J., Maulsby, D. & Witten, I. H. (1995), Predictive Interfaces: what will they think of next?, *in Extra-ordinary Human-Computer Interaction: interfaces for users with disabilities*, Cambridge University Press, pp.103–139.

Hudson, S. E. & Newell, G. L. (1992), Probabilistic State Machines: Dialog Management for Inputs with Uncertainty, *in Proceedings of ACM UIST'92*, pp.199–208.

Huerst, W., Yang, J. & Waibel, A. (1998), Interactive Error Repair for an Online Handwriting Interface, *in Proceedings of ACM CHI'98 (Poster)*, pp.353–354.

Igarashi, T., Matsuoka, S., Kawachiya, S. & Tanaka, H. (1997), Interactive Beautification: A Technique for Rapid Geometric Design, *in Proceedings of ACM UIST'97*, pp.105–114.

Kato, N. & Nakagawa, M. (1995), The Design of a Pen-based Interface 'SHOSAI' for Creative Work, *in* Y. anzai, K. Ogawa & H. Mori (eds.), *HCI International '95: Symbiosis of Human and Artifact*, Elsevier Science.

Kurtenbach, G., Moran, T. P. & Buxton, W. (1994), "Contextual Animation of Gestural Commands", *Computer Graphics Forum* **13**(5), pp.305–314.

Lopresti, D. & Tomkins, A. (1995), Computing in the Ink Domain, *in* Y. anzai, K. Ogawa & H. Mori (eds.), *HCI International '95: Symbiosis of Human and Artifact*, Elsevier Science.

Mankoff, J. & Abowd, G. D. (1998), Cirrin: A word-level unistroke keyboard for pen input, *in Proceedings of ACM UIST'98*, pp.213–214.

Marx, M. & Schmandt, C. (1994), Putting People First: Specifying proper names in speech interfaces, *in Proceedings of ACM UIST'94*, pp.30–37.

Moran, T. P., Chiu, P. & van Melle, W. (1997), Pen-Based Interaction Techniques for Organizing Material on an Electronic Whiteboard, *in Proceedings of ACM UIST '97*, pp.45–54.

Myers, B. A. & Kosbie, D. S. (1996), Reusable Hierarchical Command Objects, *in Proceedings of ACM CHI'90*.

Nanja, M. & Cook, C. R. (1987), An Analysis of the On-Line Debugging Process, *in Empirical Studies of Programmers: Second Workshop*, pp.172–184.

Poon, A., Weber, K. & Cass, T. (1995), Scribbler: A Tool for Searching Digital Ink, *in Proceedings of ACM CHI'95*, pp.252–253.

Rhodes, B. J. & Starner, T. (1996), Remembrance Agent, *in The Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology (PAAM '96)*, pp.487–495.

Riseberg, J., Klein, J., Fernandez, R. & Picard, R. W. (1998), Frustrating the user on purpose: using biosignals in a pilot study to detect the user's emotional state, *in Proceedings of ACM CHI'98*, pp.227–228.

Schomaker, L. R. B. (1994), User-interface Aspects in Recognizing Connected-Cursive Handwriting, *in Proceedings of the IEE Colloquium on handwriting and Pen-based input*, number 1994/065, The institution of Electrical Engineers, London.

Suhm, B. (1997), Empirical Evaluation of Interactive Multimodal Error Correction, *in IEEE Workshop on Speech recognition and understanding*, IEEE, Santa Barbara.

Suhm, B., Myers, B. & Waibel, A. (1996a), Designing Interactive error Recovery Methods for Speech Interfaces, *in Proceedings of ACM CHI'96*, Workshop on Designing the User interface for Speech Recognition applications.

Suhm, B., Myers, B. & Waibel, A. (1996b), Interactive Recovery from Speech Recognition Errors in Speech User Interfaces, *in Proc. ICSLP '96*, Vol. 2, Philadelphia, PA, pp.865–868.

Thimbleby, H. (1990), *User Interface Design*, Addison-Wesley Publishing Co. (ACM Press), Reading, MA. 1990.

Wolf, C. G. (1990), Understanding Handwriting Recognition from the User's Perspective, *in Proceedings of the Human Factors Society 34th Annual Meeting*, Vol. 1 of *Computer Systems: Modeling*, pp.249–253.

Zajicek, M. & Hewitt, J. (1990), An Investigation into the Use of Error Recovery Dialogues in a User Interface Management System for Speech Recognition, *in Proceedings of IFIP INTERACT'90*, pp.755–760.

Zoltan-Ford, E. (1991), "How to Get People to Say and Type What Computers Can Understand", *International Journal of Man-Machine Studies* **34**(4), 527–547.