Sketch Recognizers from the End-User's, the Designer's, and the Programmer's Perspective

Jason Hong, James Landay, A. Chris Long[1], Jennifer Mankoff
Group for User Interface Research
University of California at Berkeley
{jasonh, landay, jmankoff}@eecs.berkeley.edu

Abstract

Although the accuracy and robustness of sketch recognizers have been steadily improving, there are still many issues involved in using them in applications. This paper details our experiences with using sketch recognizers from three different perspectives—end-users, user interface designers, and programmers—and describes some of the open research problems in each of these areas.

Introduction

Successes in speech and handwriting recognition have allowed recognizers to move into mainstream products, such as IBM ViaVoice (IBM Corporation), Windows XP (Microsoft Corporation), and the Palm Pilot (Palm). A great deal of research has also been devoted to improving sketch recognition technologies, with the accuracy and robustness of such recognizers steadily improving. However, accuracy and robustness are two fairly low-level and scientific perspectives on recognizers. Very little has been published about making sketch recognizers work in applications in practice. What is also needed is a critical look at how recognizers are used in practice–that is, from the end-user's, the designer's, and the programmer's perspective.

Table 1 summarizes the three different roles and their concerns in the development of sketch-based applications. Starting on the bottom, we have programmers, the people that design and develop the necessary software architecture. Programmers are primarily concerned with using recognition technology, using the toolkits and APIs for enabling sketch-based applications, and debugging these technologies when things do not work as expected.

In the middle are the user interface designers, whose role is to bridge the chasm between the programmers and the software technologies, and the end-users and their needs. Designers are concerned with developing interaction techniques for sketching, feedback for helping end-users understand what the state of the application is, the overall look and feel of the interface, and prototyping tools for quickly trying out many ideas.

Role	Goals	Concerns with Recognition
End-User	Accomplish some task with the technology	Useful Usable Desirable
Designer	Create an interface, evaluate it, iteratively improve it	Interaction Techniques Feedback Look and Feel Prototyping Tools
Programmer	Write good code and debug it	Recognizers Reusable Toolkits APIs Debugging

Table 1: The roles in developing sketch-based applications and their concerns.

At the top we have end-users, the target audience for the application. End-users are primarily concerned with having applications that are useful, usable, and desirable.

As Table 1 shows, designers and programmers have very different needs in terms of the tools and techniques they use to accomplish their tasks. In the next section, we describe some of our experiences with each of these roles in the development of sketch-based tools and applications.

Overview

Most of our work has been in the general area of pen-based interfaces. In particular, our earliest work, SILK (Landay and Myers 2001), allowed people to sketch user interfaces and provided a perspective on the importance of sketches in early design. In Burlap (Mankoff, Hudson 2000), a rougher version of SILK, an interactive technique known as mediation is used to correct recognition errors. DENIM (Lin, Newman 2000), a sketch-based tool for web designers, takes an alternative approach by minimizing the amount of recognition needed while still retaining sketchy interaction. With the Quill tool (Long, Landay 2000), designers can quickly design a pen gesture set, analyze it for potential computer recognition errors, store test sets of user input, and use these test sets for regression tests on the gesture set. Finally, to address the lack of support for

[1] This work was done while at University of California at Berkeley. Chris Long is currently at the HCI Institute at CMU.

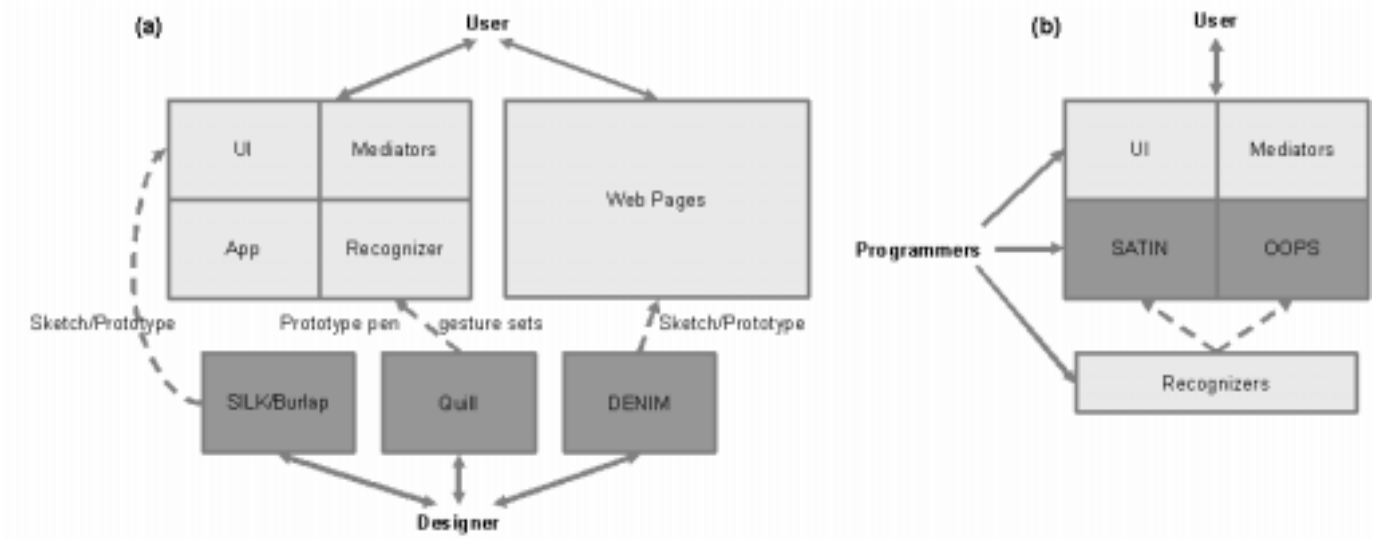

Figure 1. (a) The relationship between Designers and SILK/Burlap, Quill and DENIM. SILK/Burlap can generate a user interface (UI); Quill can generate gesture sets, and DENIM is used to sketch Web pages. (b) The relationship between Programmers, SATIN and OOPS. SATIN is used to build pen-based UIs while OOPS provides access to techniques for mediating recognition errors. Both provide APIs for dealing with recognizers.

programmers, we developed two different toolkits, SATIN (Hong and Landay 2000) and OOPS (Mankoff, Hudson 2000), which offer the APIs and algorithms necessary to support programmers who are dealing with strokes and recognizers as first class objects.

Figure 1 shows the relationships between the tools we describe in this paper and End-Users, Designers, and Programmers. Notice that the tools intend for use by designers (SILK/Burlap, Quill, and DENIM) are used to generate interfaces and web pages, things that will be used directly by end-users. They all support quick prototyping and iteration and require no programming. In contrast, the tools intended for use by programmers provide infrastructure, APIs, and library components that can be used in application and UI development. They require programming to be used and cannot support the same style of quick prototyping. This meshes with the roles and needs of developers and programmers described above.

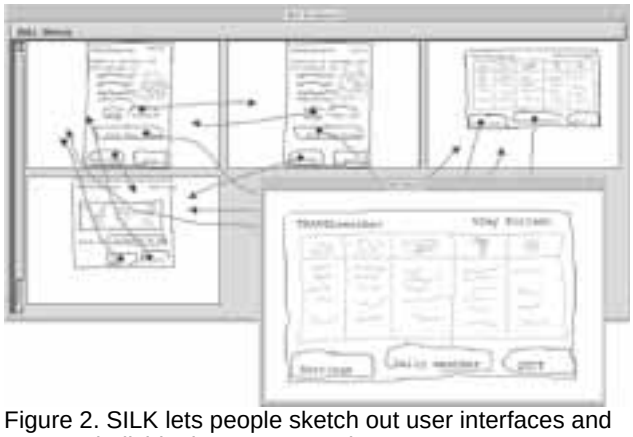

Figure 2. SILK lets people sketch out user interfaces and connect individual screens together.

We have examined the use of recognizers by all three of these groups. There are some characteristics of recognizers that impact people at each of these levels. The characteristics we have focused on include: recognition errors, the ability of strokes to be ambiguous, the complexity of strokes, and issues in displaying both strokes and the results of recognition. We summarize the lessons we have learned about recognizers with respect to these various roles, examining the strengths and weaknesses in the state of the art, and conclude with suggestions for future directions for research.

Tools for Designers

As stated, SILK supported designers in the task of quickly prototyping user interfaces by sketching them (see Figure 2). Informal observations and discussions with designers revealed that they liked to sketch out ideas because the freeform nature allowed them to be more creative and exploratory than if they were using a computer. Sketching also made it easier to rapidly prototype and iterate on user interfaces. These findings have been confirmed in other studies (Black 1990, Goel 1995, Gross and Do 1996, Newman and Landay 2000, Wagner 1990, Wong 1992).

In SILK, each ink stroke was processed immediately as it was drawn. Although the ink was kept “as is” without transforming its presentation, designers had to correct any recognition errors in a separate window, since the system always had to have the “correct” state for it to work

properly. This recognition feedback and correction would often be distracting from the task at hand, namely user interface design.

We investigated two approaches to dealing with this. In Burlap (Mankoff, Hudson 2000), a rougher version of SILK, an interactive technique known as mediation is used to correct recognition errors (see Figure 3). Designers are not required to correct any recognition errors until they need to interact with a sketched component, for example in preparation for a demo of the sketched system. At that point, designers can select the correct choice from a list displayed dynamically near the problem sketch, or redraw the sketch if necessary. This was an attempt to minimize the amount of recognition needed while still retaining sketchy interaction.

DENIM (Lin, Newman 2000), a sketch-based tool for web designers, pushed this approach even further by removing recognition almost entirely from the design process while retaining sketchy interaction (see Figure 4). Sketched objects within a page are grouped by spatial and temporal proximity, and sketched objects between two pages are classified as arrows. DENIM was designed to be used more for documenting ideas and letting humans recognize and interpret things as opposed to the computer. Feedback from end-users during usability tests and from people that have downloaded DENIM has shown that this strategy is surprisingly effective.

However, even though there is little recognition in DENIM, we still observed designers encountering a number of difficulties. One problem stems from the fact that sketch-based applications often have "invisible" interfaces. It is not always clear what actions are and are not possible at any given time. A related problem is with feedback. Sometimes people were not sure if the system had "done the right thing" after a stroke had been drawn, even though we tried to minimize the amount of recognition involved.

Another problem is simply the individual variation between people. Some people drew objects slightly skewed or a little larger than what we had expected. This had little effect on the ink, other than occasional unexpected grouping, but had a serious impact on recognition of gestures such as cut, pan, undo, and redo. This is important because gestures are a common part of many sketch-based interfaces, and often rely on the same kind of recognizers used for understanding sketches.

Another broad problem faced by designers is in defining what sketched objects can be recognized. When SILK was developed, the only interface to the underlying Rubine recognizer (Rubine, 1991) was a programmatic API. For example, Figure 5 shows the definition of a horizontal scrollbar in SILK. This approach made it difficult to experiment with choices for the appearance of recognized gestures, or to see if they were effective from an end-user

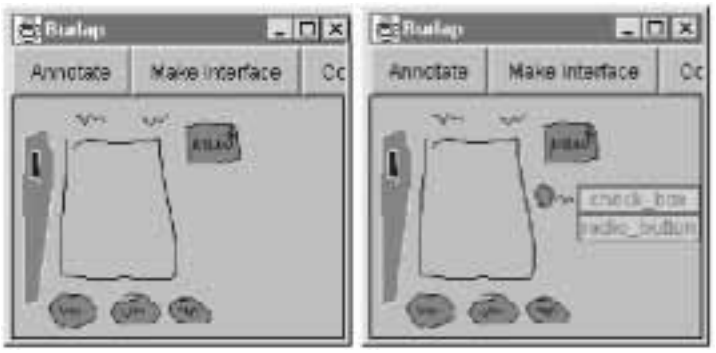

Figure 3. Burlap is a rougher version of SILK that demonstrates how mediators can be used to interactively deal with recognition errors.

Figure 4. Two screenshots from DENIM, a sketch-based Web site design tool for the early stages of design.

```
(and (contains-p container containee)
     (rectangle-p container)
     (rectangle-p containee)
     (skinny-p container :horizontal)
```

Figure 5. Definition of a horizontal scrollbar in SILK.

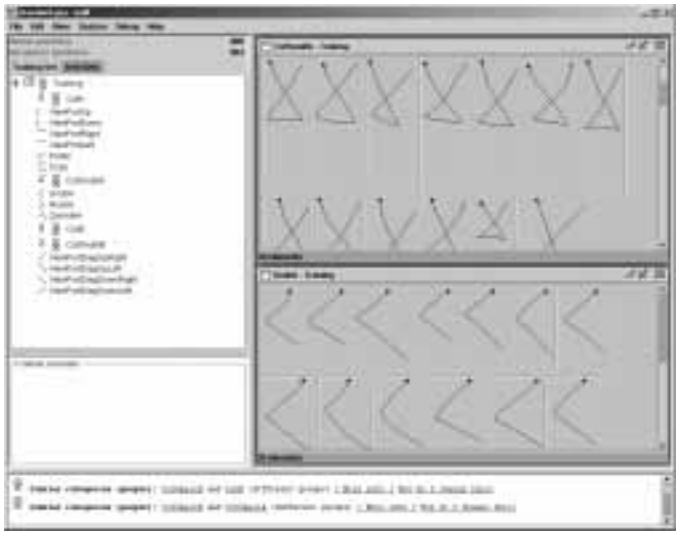

Figure 6. Quill is a tool that lets designers quickly create and test pen gestures.

perspective. Another problem with this limited definition is robustness. Sketches that were slightly off would not be correctly recognized, even if end-users perceived it as such.

Thus, our experiences as developers of SILK showed us the need for a tool to help support designers in the creation of gesture sets to be recognized. Here our focus was on supporting the creation of a gesture set that is both accessible to users and easy for the recognizer to accurately support. The result was the Quill tool (Long, Landay 2000), shown in Figure 6. With this tool, designers can quickly design a gesture set, analyze it for potential computer recognition errors, store test sets of user input, and use these test sets for regression tests on the gesture set.

Tools for Programmers

Our experiences as developers of SILK, Burlap, DENIM, and Quill led to an understanding of a multitude of problems relating to the creation of pen and recognition-based user interfaces. In particular, there was a clear need for APIs and algorithms that allow the use of strokes and recognizers as first class objects. Here we highlight some of those problems and then discuss SATIN (Hong and Landay 2000) and OOPS (Mankoff, Hudson 2000), two toolkits intend to address them.

One focus in these toolkits was on pluggability and re-usability in the context of existing applications and widgets. For example, in both SATIN and OOPS, a programmer can easily switch the recognizer in use without modifying other aspects of her application. SATIN also gives programmers the ability to easily change the behavior that takes place when a stroke is recognized, while OOPS treats the recognition results as first class input events, allowing them to be sent to widgets or further recognized.

For example, in many sketch-based systems, strokes tend to be processed only once, as they are drawn. Instead, OOPS allows ambiguous states. OOPS can defer resolution of that ambiguity, making automatic corrections as more sketches and other contextual information comes along. If it is necessary to involve the end-user in correction, OOPS provides pluggable, extensible interface elements, called mediators, for this purpose. A programmer may select mediators for use in his application, and switch from one to another without modifying any other aspect of his application. Some examples of the mediators provided in OOPS include a choice mediator, which displays a list of options, and a repetition mediator, which lets end-users repeat the input and try recognition again.

Because SILK and OOPS were only intended to support programmers, there is only minimal support for quickly designing and testing a sketch-based interface, meaning that a large portion of the interface itself has to be programmed and developed before it can be evaluated with end-users.

Summary and Future Directions

From an end-user perspective, our research group has developed two new ways in which recognizers are used. The first is with informal user interfaces, which are interfaces that support natural human input, such as speech and writing, while minimizing recognition and transformation of the input. These interfaces, which document rather than transform, better support a user's flow state. The second is realizing that handling errors are an intrinsic and essential part of recognizers, and that low-level programming support should be provided to make it easier to defer mediation to end-users.

From a designer perspective, our research group has created tools for developing and testing gesture sets, for sketching user interfaces, and for sketching web pages. Here we learned that sketching was an essential prototyping tool in part because it encouraged a willingness to make large changes to proposed designs and an ability to make them easily and quickly.

From a programmer perspective, our research group has developed two toolkits focusing on re-usability of behavior and mediation. Here we looked at APIs for recognition and dealing with strokes, and created libraries of mediators and widgets for strokes.

Of course, each of the applications and tools discussed above address only some of the issues faced by end-users, designers, and programmers. A great deal of research is still needed in other areas. For example, although OOPS

supports ambiguity and mediation, there are still many issues involved in making this technique scale to hundreds or even thousands of strokes. Additionally, although strokes are often used to give commands to a system, most mediators were developed to help support recognition of data (for example letters and words).

Furthermore, it is still extremely difficult to build systems with sophisticated and robust multistroke capabilities. In SILK, multistrokes were defined fairly rigidly in the programming language. More support is needed for graphically defining multistrokes, either through grammars or by example.

Also, there are still only a few interfaces to sketch recognizers beyond programming APIs. Quill represents a step forward in helping designers create and test single-stroke languages for recognizers, but there is still a long ways to go before sketch recognition is made generally accessible to all designers. Designers need prototyping tools for letting them quickly create sketch-based applications for desktops and PDAs, letting them easily define what objects to recognize and what interactions are allowed.

Lastly, two key usability issues are invisibility and feedback. More techniques need to be developed to make sketching more "self-revealing," while at the same time ensuring that it does not provide too much feedback that distracts from the task at hand.

References

1. Black, A., Visible Planning on Paper and on Screen: The Impact of Working Medium on Decision-making by Novice Graphic Designers. *Behaviour & Information Technology*, 1990. 9(4): 283-296.
2. Goel, V. 1995. *Sketches of Thought*. Cambridge, MA: The MIT Press.
3. Gross, M.D. and Do, E.Y.-L. 1996. Ambiguous Intentions: A Paper-like Interface for Creative Design. In Proceedings of *ACM Symposium on User Interface Software and Technology: UIST '96*, 183-192. Seattle, WA: ACM Press.
4. Hong, J. and Landay, J.A. 2000. SATIN: A Toolkit for Informal In-based Applications. In Proceedings of *User Interfaces and Software Technology: UIST 2000*, pp. 63-72. San Diego, CA: ACM Press.
5. IBM Corporation, IBM Voice Systems. http://www.ibm.com/software/speech
6. Landay, J.A. and Myers, B.A., Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer*, 2001. 34(3): 56-64.
7. Lin, J.; Newman, M.W.; Hong, J.I.; and Landay, J.A. 2000. DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design. In Proceedings of *Human Factors in Computing Systems: CHI 2000*, 510-517. The Hague, The Netherlands: ACM Press.
8. Long, A.C.; Landay, J.A.; and Rowe, L.A., Visual Similarity of Pen Gestures. *CHI Letters*, 2000. 2(1): 360-367.
9. Mankoff, J.; Hudson, S.E.; and Abowd, G.D. 2000. Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces. In Proceedings of *Human Factors in Computing Systems: CHI 2000*, 368-375. The Hague, The Netherlands: ACM Press.
10. Microsoft Corporation, Microsoft Windows XP. http://www.microsoft.com/windowsxp
11. Newman, M.W. and Landay, J.A. 2000. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In Proceedings of *Designing Interactive Systems, DIS 2000*, 263-274. New York City: ACM Press.
12. Palm, I. http://www.palm.com
13. Wagner, A. 1990. Prototyping: A Day in the Life of an Interface Designer, in *The Art of Human-Computer Interface Design*, B. Laurel, Editor. Addison-Wesley: Reading, MA. p. 79-84.
14. Wong, Y.Y. 1992. Rough and Ready Prototypes: Lessons From Graphic Design. In Proceedings of *CHI '92: Human Factors in Computing Systems Short Talks*, 83-84. Monterey, CA: ACM Press.