Interactive Sketching for the Early Stages of User Interface Design

James A. Landay

December 19, 1996

CMU-CS-96-201

School of Computer Science
Computer Science Division
Carnegie Mellon University
Pittsburgh, PA

Also appears as CMU-HCII-96-105

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by ARPA under contract no. N66001-94-C-6037.

Keywords: User interfaces, design, sketching, storyboarding, gesture recognition, interaction techniques, pen-based computing, SILK

Abstract

Current interactive user interface construction tools are often more of a hindrance than a benefit during the early stages of interface design. These tools are hard to use and they encourage designers and evaluators to focus on the wrong issues, such as color, fonts, and alignment, at this early stage. Most designers prefer instead to sketch early interface ideas on paper. However, designing on paper also has many drawbacks. Paper-based designs are hard to edit and cannot easily be tested with users. This dissertation describes the design, implementation, and evaluation of an interactive sketching tool called SILK that overcomes these problems and combines many of the benefits of paper-based sketching with current electronic tools.

SILK allows designers to quickly sketch an interface using an electronic pad and stylus. SILK preserves the important properties of pencil and paper: a rough drawing can be produced quickly and the medium is flexible. However, unlike a paper sketch, this electronic sketch is interactive and can easily be annotated and modified using editing gestures.

SILK recognizes user interface widgets and other interface elements as they are drawn and gives the designer feedback so that these inferences can be changed. Recognized interface elements have built-in behaviors and thus these elements can be exercised in their sketchy state. For example, the "elevator" in a sketched scrollbar can be dragged up and down. Unfortunately, the behavior of individual widgets is insufficient to test a working interface. SILK's electronic storyboards allow the illustration of the dynamic behavior between interface elements, such as a dialog box appearing when a button is pressed. A designer creates a storyboard by first copying sketched screens to the storyboard window and then drawing transition arrows on the screens. The arrows specify which objects cause transitions to which screens when the end-user clicks on the objects. When the designer is satisfied with this early prototype, the system can transform the sketch into an operational interface using real widgets in a specified look-and-feel.

An evaluation of SILK, using professional and student designers, showed that it was an effective tool for both early creative design and for communicating the resulting design ideas to other members of an engineering team. By supporting the early phases of the interface design life cycle, electronic sketching can both ease the prototyping of user interfaces and improve the interfaces that are eventually produced.

This work is dedicated to Hank Wan, my friend and classmate. He lives on in the memory of his friends and family.

Acknowledgments

First, I would like to thank my advisor, Brad Myers, for his continuous support of me at CMU. I would never have gotten to the point of even starting this dissertation, let alone finishing it, without his advice and support. He was always there to meet and talk about my ideas and mark up my papers and chapters (several times). He kept me thinking when I thought I had run into a fatal problem.

I would also like to thank my co-advisor, Jim Morris, and the rest of my committee for their great comments. I would especially like to single out Tom Moran, who gave me more feedback and time out of his busy schedule than anyone next to Brad. The dissertation is that much stronger because of it. There are other members of the CMU faculty who also gave me valuable feedback, especially on my survey and evaluation. In particular, I would like to thank Al Corbett, Bonnie John, and Jane Siegel.

This last year of work on the dissertation has been one of the best periods of my life, thanks to the love and companionship of Eileen Toomey. Without her these last few months would have been very hard for me. I look to the future and see that with Eileen the rest of my life will be fulfilling no matter what I do. I would also like to thank the United States Marines for putting on a great marathon, thus giving Eileen and I another goal to work towards and accomplish this fall.

Besides my advisor, there has been one friend who is most responsible for helping me complete this challenging program. Dave Kosbie has been a great friend and mentor in both my chosen field and in the rest of my life. Without his encouragement and incredibly motivational speech a few years ago, I would have left the program without finishing. He also helped me see the importance of facing the questions in my life and answering them. Thanks Dave!

Friends are something without which I could not have made it through this program and time in my life. The best part of CMU SCS is the people who have chosen to come here. Phoebe Sengers has been my officemate for over four years and I am only saddened that I didn't know her in my first two years. She has always been fun to talk to and a great supporter of me and my work. Thanks for reading this dissertation and my papers! I would also like to thank Marc Ringuette and Jay Sipelstein. My housemates for four years, Jay and Marc were always there to talk to, proofread my papers, work through my ideas, and have fun with.

There were many others who helped me through this period and I hope to see them out West: my classmates, Jonathan Hardwick, Sue Older, and Dennis Grinberg; my lunch buddies, Anwar Ghuloum and Juan Leon. Finally, thanks to some of the others who I enjoyed time with: Scott Nettles, Mike Gleicher, Rowan Davies, Howard Gobioff, Joseph O'Sullivan, Lin Chase, Bob Harper, Bruce Maggs, Danny Sleator, Francesmary Modugno, Belinda Thom, Robbie Warner, Stewart Clamen, Bennett Yee, and the home brew crew. Finally, I would like to thank the staff here at CMU. They are an exemplary crew and make this ship sail smoothly.

Contents

.

List of Figures

.

List of Tables

CHAPTER 1
Introduction

When professional designers first start thinking about a visual interface, they often sketch rough pictures of the screen layouts. In fact, everyone who designs user interfaces seems to do this, whether or not they have a graphic design background. These sketches are used to explore the overall layout and structure of interface components, rather than to refine the detailed look-and-feel.

In addition to sketching the various screens, designers often build up storyboards from these sketches. By numbering the screens, drawing arrows on them, and attaching annotations, a designer can describe the major transitions that occur between screens when a user manipulates the interface. Figure 1-1 illustrates a simple sketched storyboard.

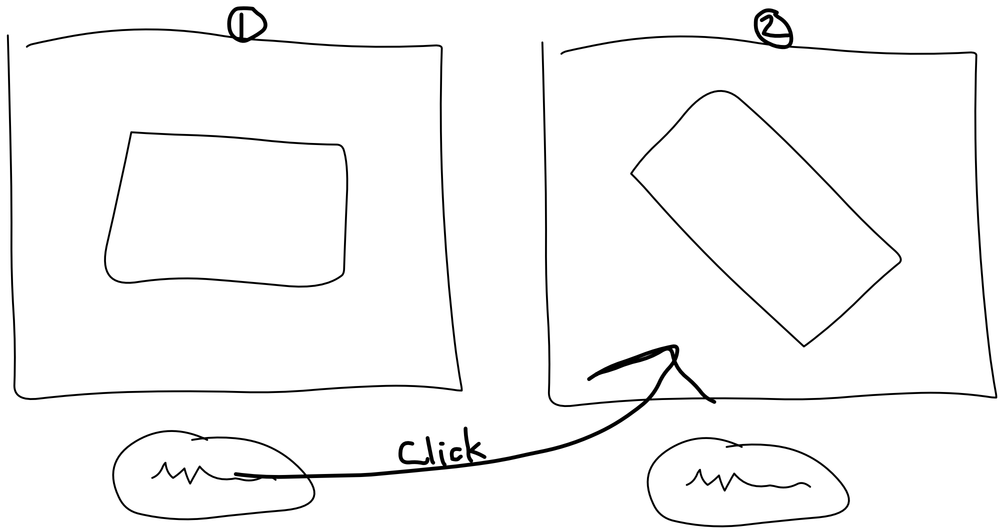


Figure 1-1. This hand-drawn storyboard illustrates that the rectangle should rotate when the button is pressed.

Designers need interactive tools that give them the freedom to sketch rough design ideas quickly, the ability to test the designs by interacting with them, and the flexibility to fill in the design details as choices are made [Wagner 1990]. Because current interactive tools do not offer this support, designers are still more comfortable sketching than using traditional palette-based interface construction tools in the early stages of design. Additionally, research indicates that designers *should not* use current interactive tools in the early stages of development since this places too much focus on design details like color and alignment rather than on the major interface design issues, such as structure and behavior [Wong 1992].

This dissertation investigates the design, implementation, and evaluation of a novel interface design tool, SILK, that is intended to combine the advantages of paper-based sketching and storyboarding with the features afforded by computer-based construction tools. This chapter first discusses the goals of the research and then motivates the creation of such a tool by describing the advantages and disadvantages of existing design tools and methodologies. This is followed by an overview of SILK and an extended example that highlights SILK's user interface. The chapter closes with a discussion of the contributions of the research and an outline of the rest of this document.

1.1 Research Goals

The primary goal of this work is to demonstrate that electronic sketches and storyboards can be successfully used to prototype user interfaces in the early stages of design. SILK [Landay 1995a], which stands for Sketching Interfaces Like Krazy, allows designers to quickly sketch an interface using an electronic stylus. SILK retains the *sketchy* look of the components. This rough representation keeps designers and evaluators from focussing on unimportant details at this stage of design, and when combined with the interactivity provided by storyboards, encourages productive design discussions with other members of the design team. Figure 1-2 illustrates a simple sketched interface screen with a window containing a rectangle and a button below the window[1].

The terms *user* and *designer* are used interchangeably in this document to refer to the user of SILK. The term *end-user* is used to refer to the user of the interface that is prototyped with SILK.

Unlike paper sketches, electronic sketches allow the designer or test subjects to try out the design before it becomes a finished interface. This is because SILK sketches include interactive behavior, both for recognized interface widgets and for

1. All interface sketches in this document were created using SILK unless noted otherwise in the figure caption.

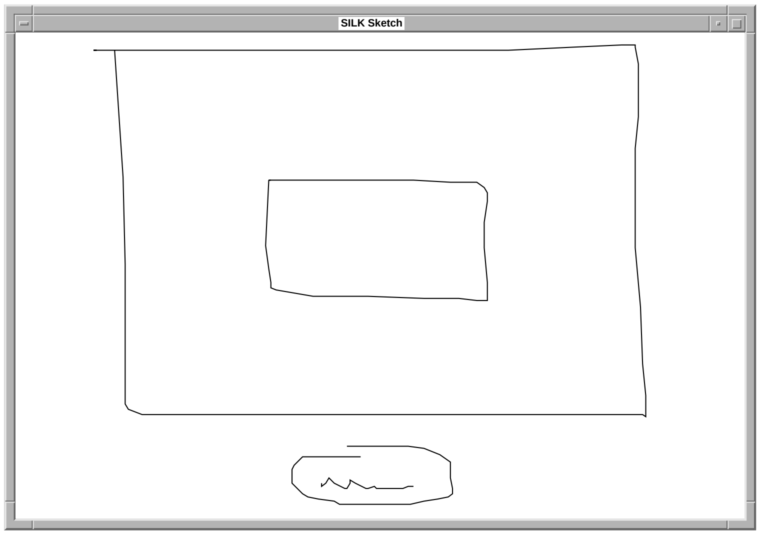


Figure 1-2. A simple sketched interface screen containing a window, a rectangle, and a button.

transitions among various screens. As a consequence, another goal of the research is to make it easy to illustrate this behavior without requiring programming or complicated dialogs. Figure 1-3 illustrates a storyboard that adds behavior to the interface illustrated in Figure 1-2. This storyboard is a SILK representation of the hand-drawn storyboard shown in Figure 1-1. At each stage of the design process the interface can be tested by manipulating it with the mouse, keyboard, or stylus.

Traditional user interface construction tools are often difficult to use and interfere with the designer's creativity. Therefore, another goal is to make SILK's user interface as unintrusive and flexible as possible. In addition to providing the ability to rapidly capture sketched user interface ideas, SILK allows a designer to edit the sketch

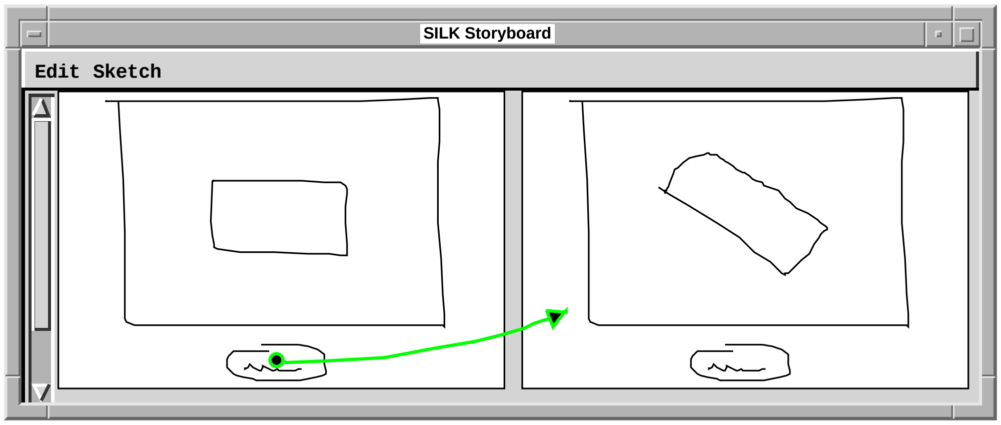


Figure 1-3. SILK storyboard that illustrates rotating the rectangle when the button is pressed.

using simple gestures. Furthermore, SILK's design memory mechanisms allow designers to review annotations made to a design over the course of a project and to reuse portions of old designs. Thus, unlike paper sketches, SILK sketches can evolve without forcing the designer to continually start over with a blank slate.

Most existing tools only support a small portion of the design process (see Figure 1-4). Therefore, a final goal of this research is to concentrate on the early stages of design, while allowing an easy transition to the tools that will be used in the later stages. SILK's widget transformation process turns interface sketches into real user interfaces for actual systems without re-implementation or programming. Thus, SILK can support the entire design cycle — from developing the initial creative design to developing the prototype, testing the prototype, and implementing the final interface. To some extent SILK will be able to replace paper sketches and prototyping tools (e.g., HyperCard, Director, and Visual Basic) for designing, constructing, and testing user interfaces in the early stages of user interface design (see Figure 1-4).

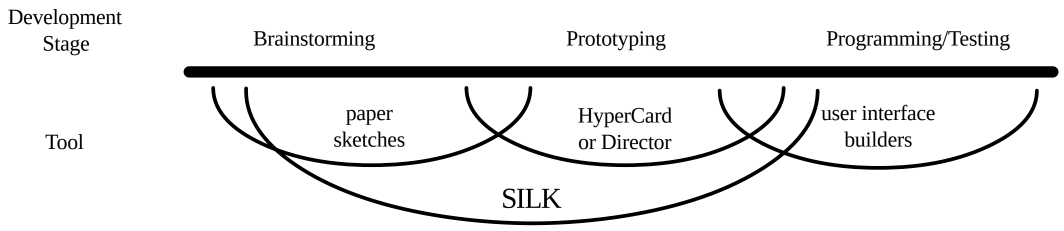


Figure 1-4. SILK can be used during all stages of user interface design, construction, and testing.

1.2 Drawbacks of Current Design Methods

User interface designers have become key members of software development groups. To get a better understanding of their design process, I visited some of them at their workplaces. In particular, I visited US West's Advanced Research Lab and Apple Computer's Advanced Technology Group. I also studied the design literature, spoke with designers at conferences, and took an electronic survey of designers (see Chapter 2, "Informal Survey of Designers"). Using insights from these different sources, I have drawn some conclusions about the problems with current design methods. Currently, designers use some combination of low-fidelity techniques, prototyping tools, and user interface builders. I describe these in turn.

Designers often use sketching and other *low-fidelity techniques* [Rettig 1994] to generate early interface designs. Low-fidelity techniques involve creating interface mock-ups using sketches, scissors, glue, and post-it notes. Designers use these mock-ups to quickly try out design ideas. Later they may use prototyping tools or user interface builders, or they may hand off the design to a programmer.

Prototyping tools, such as Macromedia's Director [Macromedia 1994], Apple's HyperCard [Apple 1993], and Microsoft's Visual Basic [Microsoft 1993], allow non-programmers to write simple application mock-ups in a fraction of the time required using traditional programming techniques. Prototyping tools allow designers to quickly illustrate examples of what the screens in a program will look like. These tools allow designers to create applications that have varying degrees of interactive behavior, but due to poor performance are rarely used to produce significant commercial applications.

Unlike user interface prototyping tools, interface builders are generally used for producing the final application. These tools, such as the NeXT Interface Builder [NeXT 1991] and UIM/X [Software 1990], have become invaluable in the creation of both commercial and in-house computer applications. They allow the designer to create the look of a user interface by simply dragging widgets from a palette and positioning them on the screen. Unlike prototyping tools, user interface builders then generate code or specifications that can be linked with the application code written in a conventional programming language. This facilitates the creation of the widget-based parts of the application user interface with little low-level programming, which allows the engineering team to concentrate on the application-specific portions of the product. Unfortunately, prototyping tools, user interface builders, and low-fidelity techniques have several drawbacks when used in the early stages of interface design.

1.2.1 Interface Tools Constrain Design

Traditional user interface tools force designers to bridge the gap between how they think about a design and the detailed specification they must create to allow the tool to reflect a specialization of that design. Much of the design [Boyarski 1994] and problem-solving [Polya 1973] literature discusses drawing rough sketches of design ideas and solutions, yet most user interface tools require the designer to specify more of the design than a rough sketch allows.

For example, the designer may decide that the interface requires a palette of tools, but she is not yet sure which tools to specify. Using SILK, a thumbnail sketch can easily be drawn with some rough illustrations to represent the tools (see Figure 1-5). This is in contrast to commercial interface tools that require or at best encourage the designer to specify unimportant details such as the size, color, finished icons, and location of the palette. This over-specification can be tedious and may also lead to a loss of spontaneity during the design process. A study using graphic designers found that "the finished appearance of screen-produced drafts shifts [the designer's] attention from fundamental structural issues" [Black 1990]. Thus, the designer may be forced to

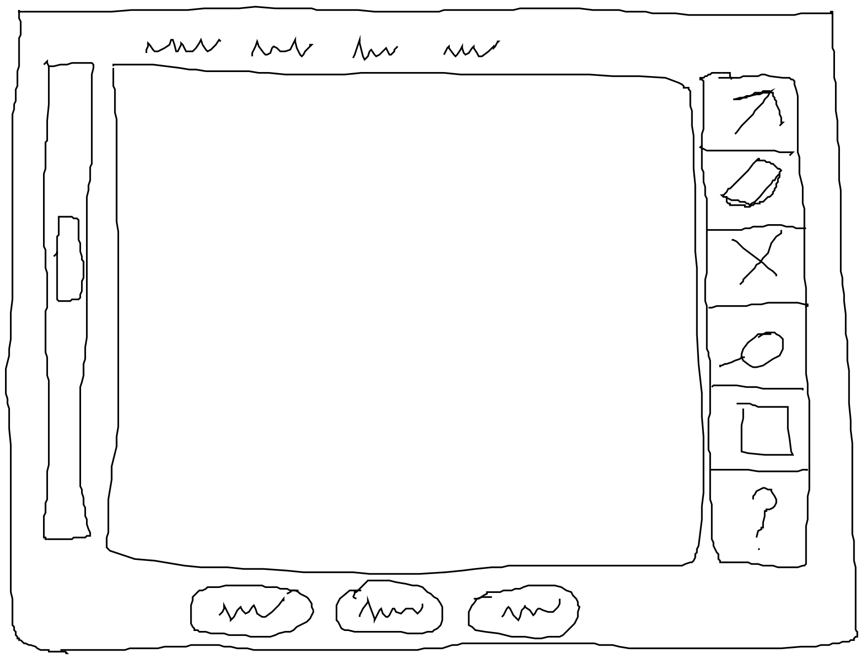


Figure 1-5. Sketched application screen for a drawing editor.

either abandon computerized tools until much later in the design process or to change design techniques in a way that is not conducive to early creative design.

One of the important lessons from the interface design literature is the value of iterative design; that is, creating a design prototype, evaluating the prototype, and then repeating the process several times [Gould 1985]. It is important to iterate quickly in the early part of the design process because that is when fundamentally different ideas can and should be generated and examined. This is another area in which current tools fail during the early design process. The ability to turn out new designs quickly is hampered by the requirement for detailed designs.

For example, Figure 1-5 illustrates a simple sketched interface screen. It has a scroll bar and a window for the scrolling data. It also has several buttons at the bottom, a palette of tools at the right, and four pulldown menus at the top. In one test using SILK, I sketched this screen using a mouse in just 70 seconds (sketched on paper it took 53 seconds). The use of a mouse accounts for the increase in time compared with using pen and paper. Creating this same interface (see Figure 1-6) with a traditional user interface builder, Gilt [Myers 1991], took 329 seconds, which is nearly five times longer. In addition, the UI builder time does not include adding representative icons to the tool palette due to the excessive time required to design or acquire them.

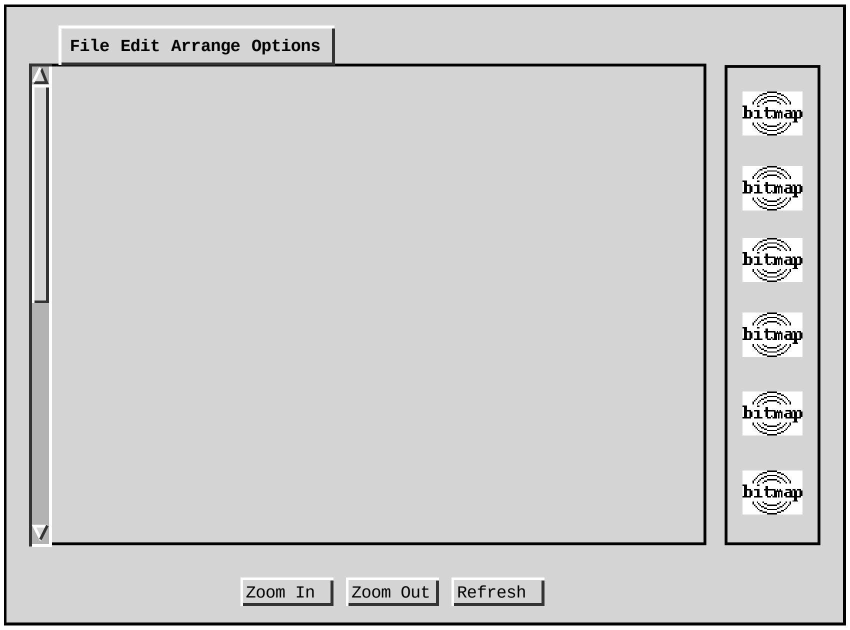


Figure 1-6. The sketched application from Figure 1-5 created with a traditional user interface builder (the Gilt tool supplied with Garnet [Myers 1990c]).

Director, HyperCard, and Visual Basic are three of the most popular prototyping tools used by designers. Though useful in the prototyping stages, these tools come up short when used either in the early design stages or for producing production-quality interfaces. Besides having many of the problems mentioned previously, these tools commonly require using a programming language since their built-in behaviors are often insufficient for the application-level interactions a designer may wish to illustrate. These full-powered programming languages are inappropriate for most interface designers, who often have a background in graphic design or art, not in programming. In addition, Director and HyperCard cannot be used for many commercial-quality applications due to their poor performance, which usually forces the development team to reimplement the user interface with a different tool.

1.2.2 Paper-based Sketching Lacks Interactivity

Because of these problems with interactive tools, designers often use paper sketches for their early creative brainstorming. Brainstorming is a process that moves quickly between fundamentally different design ideas. Sketches allow a designer to quickly preserve thoughts and design details before they are forgotten. The disadvantage of making these sketches on paper is that they are hard to modify as the design evolves. The designer must frequently redraw the common features that the design retains.

One way to avoid this repetition is to use translucent layers [Wong 1993; Gross 1994]. Another solution is to use an erasable whiteboard. Both of these approaches are clumsy at best. In order to be effective, translucent layers require forethought on the part of the designer in terms of commonality and layout of components. Whiteboards make it hard to scale and move compound objects. Neither of these solutions help with the next step when a manual translation to a computerized format is required, either with a user interface builder or by having programmers create an interface using a low-level toolkit. This translation may need to be repeated several times if the design changes.

Another problem with relying too heavily on paper sketches for user interface design is the lack of support for *design memory* [Herbsleb 1993]. Design memory is a record of the design process: what was done at each stage and why. Paper sketches may be annotated, but a designer cannot easily search these annotations in the future to find out why a particular design decision was made. In my visits with designers and in the sketches designers sent me (see Section 2.2.3), I have observed that the annotations often occupy more space in the sketches than the actual interface design. Practicing designers have found that the annotations on design sketches serve as a diary of the design process, which are often more valuable to the client than the sketches themselves [Boyarski 1994]. Sketches made on paper are also difficult to store, organize, and reuse.

One of the biggest drawbacks of using paper sketches is the lack of interaction possible between the paper-based design and an end-user, which may be one of the designers at this stage. In order to actually see what the interaction might be like, a designer needs to "play computer" and manipulate several sketches in response to an end-user's verbal or gestural actions. This low-fidelity prototyping technique does not offer a realistic depiction of how the application will eventually be used.

1.3 Advantages of Sketching for Design

Despite many problems with paper-based sketching, the process of sketching has several advantages, especially for preliminary creative design. This section first discusses the advantages of sketching and then describes the specific advantages of computer-based sketching.

1.3.1 Advantages of Paper-based Sketching

Several researchers have explored the advantages of using diagrams in reasoning [Larkin 1987], but only recently has anyone studied the process of using sketching in the design process. Sketching has two important advantages over other design methods: it is a way to quickly externalize an idea and it is inherently ambiguous.

There is only so much of a design that a designer can keep in her mind at one time. Early on in the creative process it is important to put the idea in some tangible form – a form that can be seen, thought about, and then changed. At this early stage, it is important to generate many different solutions and variations to a design problem in a short amount of time. The lack of constraints on paper-based sketching permits a "quick and intuitive representation of concepts" [Kolli 1993]. In fact, designers will become frustrated if they cannot externalize their mental images, as this externalization is important for restructuring an initial design [Verstijnen 1996].

The iterative restructuring of preliminary mental images is a key step in creative design. The ability to restructure a design and come up with different alternatives is also supported by the ambiguity in sketches. An *ambiguous* drawing is one that can be reinterpreted by the designer with a meaning that is different from the designer's original intention. This ambiguity can result from a lack of detail or the incompleteness of the sketch. For example, the sketched widget in Figure 1-7 can be interpreted by the designer as either a radio button or a check box, as the designer has left the object to the left of the text very rough – it is neither round nor square. In fact, researchers have found that ambiguities "may trigger innate recognition search mechanisms that generate a stream of imagery useful to invention" [Fish 1990].

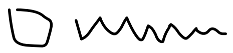


Figure 1-7. An ambiguous sketch that can be interpreted as either a radio button or a check box.

Goel noted that the ambiguity of sketches facilitates the large number of element interpretations and transformations necessary to explore alternatives at this stage of design [Goel 1995]. Goel observed several designers and found that with freehand sketching, when a new idea was generated, a number of variations were quick to follow, whereas with a drawing program (in this case, MacDraw) most subsequent effort after the initial generation was devoted to detailing and refining the same idea.

Few electronic design environments support ambiguity or incompleteness, although Henderson noted the need for it in a paper describing Trillium, one of the early user interface builders, "The environment should support incomplete specifications" [Henderson Jr 1986]. The sketches designers sent me in response to a survey (see Chapter 2, "Informal Survey of Designers") and the sketches I have seen in designer's work spaces exhibit a high degree of ambiguity. For example, these sketches contain squiggly lines to represent text and rough shapes to represent icons.

1.3.2 Advantages of Electronic Sketching

Anecdotal evidence shows that a sketchy electronic interface is much more useful in early design reviews than a more finished-looking interface. Wong found that rough electronic sketches kept her team from talking about unimportant low-level details, while finished-looking interfaces caused them to talk more about the *look* rather than interaction issues [Wong 1992]. The belief that colleagues give more useful feedback when evaluating interfaces with a sketchy look is commonly held in the design community [Rettig 1994]. In addition, as with paper-based sketches, electronic sketches keep the *designer* from focussing on the wrong issues.

Electronic sketches have most of the advantages described previously for paper sketches: they allow designers to quickly record design ideas in a tangible form. In addition, they do not require the designer to specify details that may not yet be known or important. Electronic sketches also have the advantages normally associated with computer-based tools: they are easy to edit, store, duplicate, modify, and search. Thus a computer-based tool can make the design memory embedded in the annotations even more valuable.

The final advantage of electronic sketches over traditional interface tools pertains to the background of user interface designers. A large number of user interface designers, and particularly the intended users of SILK, have a background in graphic design or art. These users have a strong sketching background and a survey (see Chapter 2, "Informal Survey of Designers") shows they often prefer to sketch out user interface ideas. An electronic stylus is similar enough to pencil and paper that most designers should be able to use one effectively with little training. In fact, the SILK usability test (see Chapter 6, "Evaluation") showed that designers could use SILK effectively with only a mouse after working through a short (approximately 45 minute) written tutorial.

1.4 Designing Interfaces with SILK

Designers need tools that give them the freedom to sketch rough design ideas quickly, the capability to specify the behavior of interface elements and the transitions between screens, the ability to test the designs by interacting with them, and the flexibility to fill in the design details as choices are made. SILK was designed with these needs in mind. A quick reference manual for SILK is given in Section D.10, though it is easier to get a feel for SILK by watching a video of the system in use [Landay 1996a]. The rest of this section reviews the major components of SILK and uses an extended example to illustrate how it is used to sketch and edit designs, specify and test interactive behavior, and finally to transform an interface to a more finished design.

The functionality of SILK is partitioned across four windows: the **`Sketch`** window, the **`Controls`** window, the **`Storyboard`** window, and the **`Finished`** window (see Figure 1-8). The designer draws interface screens in the **`Sketch`** window. These screens may include interactive widgets and static screen decorations. The **`Controls`** window is used to perform editing operations on the sketch, change modes, save/load/print designs, and give the designer feedback on SILK's inferences. At any point, the designer can copy sketches to the **`Storyboard`** window. The designer can then draw arrows representing screen transitions on top of the screens. After switching to *Run* mode, the resulting behavior can be tested in the **`Sketch`** window. When the designer is satisfied with the design, it can be transformed to an interface using real widgets. This is displayed in the **`Finished`** window. The rest of this section describes these steps in detail.

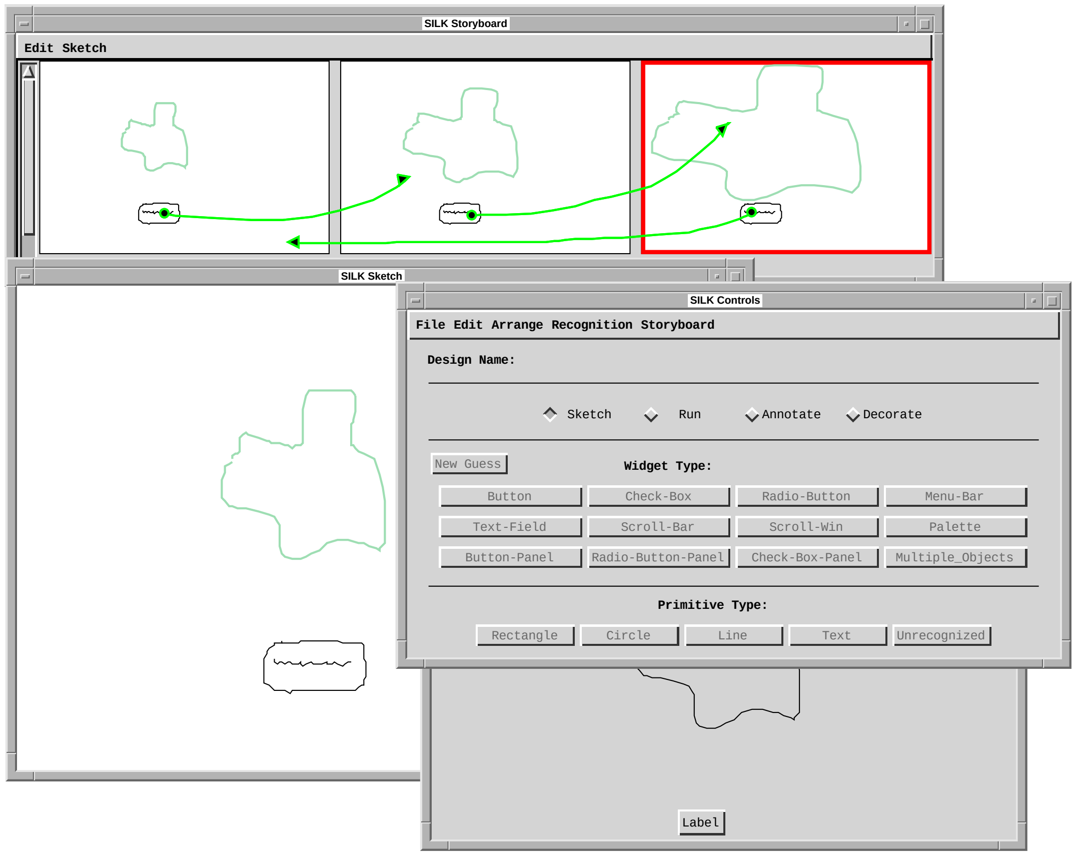


Figure 1-8. The SILK windows: Storyboard, Sketch, Controls, and Finished (top to bottom).

1.4.1 Sketching Interfaces

SILK enables the designer to move quickly through several iterations of a design by using gestures to edit and redraw portions of the sketch.

1.4.1.1 Primitive Components and Widgets

Two types of pen strokes are recognized by SILK, *primitive components* and *editing gestures* (discussed in Section 1.4.1.3). The primitive components are basic shapes that when combined together make up a widget. For example, the button in Figure 1-9 was created by sketching a rectangle and then a squiggly line inside of it (though the order in which they were sketched does not matter). The other primitive components recognized by SILK are circles and lines.

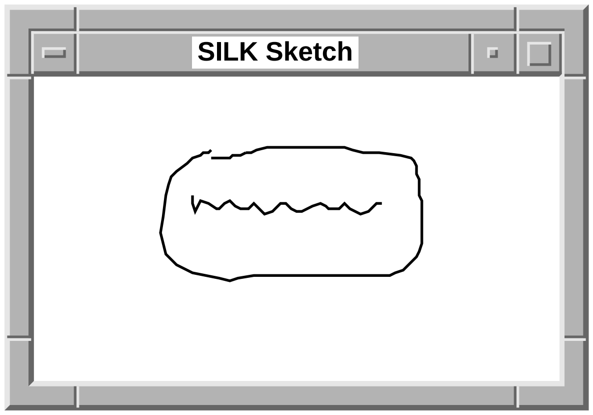


Figure 1-9. A button composed of two primitive components: a rectangle and a squiggly line.

The system tries to recognize primitive components and user interface widgets as they are drawn. Although the recognition takes place as the sketch is made, it is unintrusive and users need only be aware of the recognition results if they choose to exercise the widgets. Since SILK may offer multiple interpretations for a widget, which can be chosen from later in the design process, a SILK design can take advantage of the ambiguity in the sketch. A complete list of the widgets SILK recognizes and a discussion of the recognition algorithm is given in Chapter 3, “Widget Recognition”.

1.4.1.2 Feedback

The primitive components of the last recognized widget appear on the screen in purple to give the designer feedback about the inference process (see Figure 1-10). In addition, the type inferred for the currently selected widget is displayed in the **`Widget Type`** field and it is also selected in the panel of widget type buttons (see Figure 1-11).

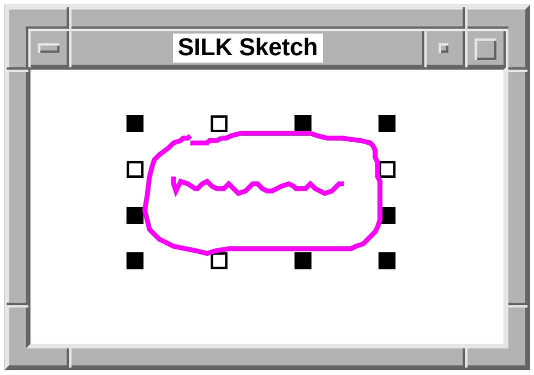


Figure 1-10. Primitive components of recognized widgets are displayed in purple when the widget is selected and are treated as a group for moving and growing.

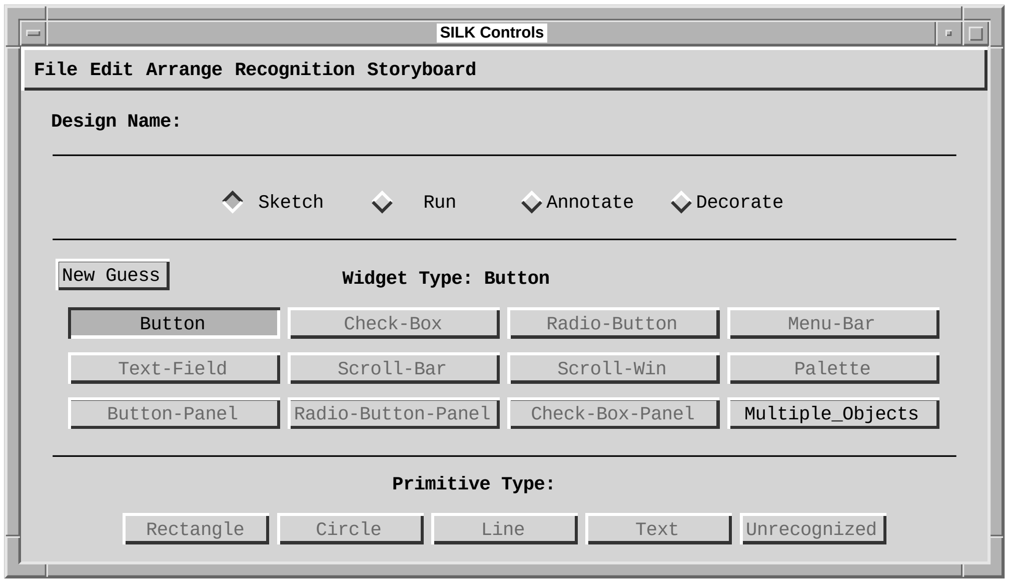


Figure 1-11. **`Widget Type`** field and panel of buttons show that SILK has recognized a `Button`.

If SILK made no inference on the components in question, the designer might group the primitive components of the widget and press the **New Guess** button (see Figure 1-12) to force the system to reconsider its inference and focus on the grouped components. If it is a widget that is not built-in to SILK's recognition system, the designer may just leave it as a grouped object.

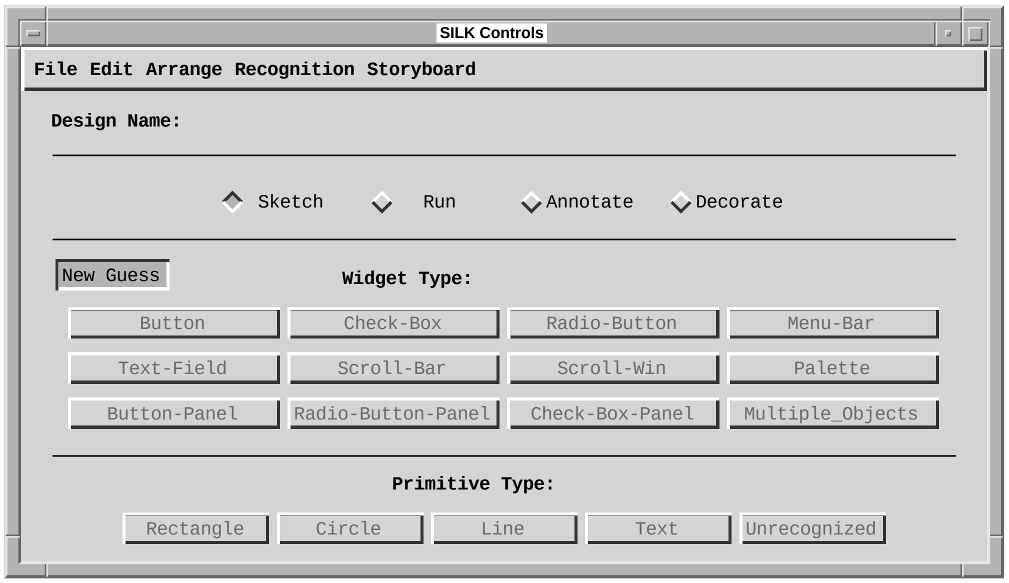


Figure 1-12. Pressing the **New Guess** button causes SILK to make a new widget inference, focusing on the selected primitive component(s) or widget(s).

1.4.1.3 Editing Sketches

One of the advantages of interactive sketches over paper sketches is the ability to quickly edit them. When the user holds down the button on the side of the stylus or the middle mouse button, SILK interprets strokes as editing gestures instead of gestures for creating new objects. These gestures, displayed in red to differentiate them from drawing strokes, are sent to a different recognizer than the one used for recognizing primitive components. The power of gestures comes from the ability to specify, with a single mark, a set of objects, an operation, and other parameters [Buxton 1986]. For example, deleting a section of the drawing is as simple as making an **X**-shaped stroke with the stylus (see Figure 1-13).

SILK supports gestures for changing inferences, deleting, grouping, and ungrouping primitive components or widgets. There is also a text editing gesture. Examples of these gestures are illustrated in Figure 1-14. Sketching the **caret** gesture on top of text squiggles allows the user to replace the squiggles with typed text. The gesture can also be used for editing existing typed text or, when drawn on the background, for creating new typed text.

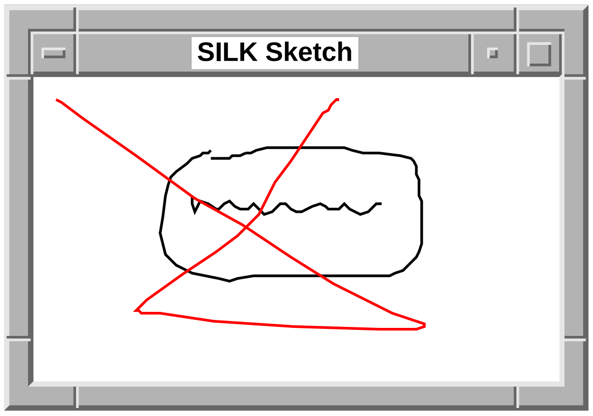


Figure 1-13. Drawing an **X**-shape with the pen button or middle mouse button held down over objects deletes them.

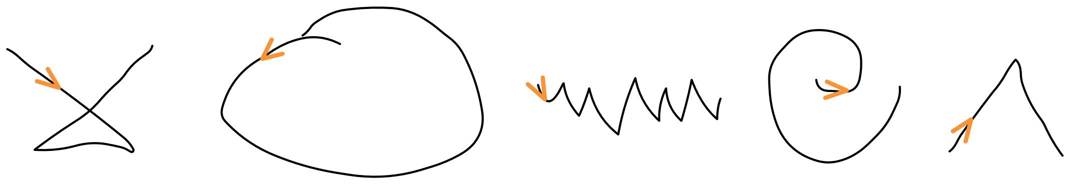


Figure 1-14. Editing gestures for delete, group, ungroup, change inference, and text editing. The arrows indicate a recommended direction to draw these gestures for best recognition performance.

SILK also supports more standard interaction methods for editing sketches. For example, an object can be selected by clicking on its outline with the left or middle mouse buttons. Once an object is selected, it can be cut, copied, cleared, or pasted by selecting from the **`Edit`** menu. These operations can be undone by selecting **`Undo`** from the same menu. In addition, the system displays black grow handles and white move handles on the selected object (see Figure 1-10). Dragging on the grow handles with the left or middle mouse button resizes the object. Similarly, dragging on the move handles with the left or middle mouse button moves the object.

1.4.1.4 Decorate Mode

In addition to the standard widgets, designs often also contain several static elements, decorations, or interactive elements with a custom look. Since SILK cannot recognize these objects, the system has a separate decoration mode for sketching them without running the widget recognition engine. The designer can switch from *Sketch* mode to *Decorate* mode by selecting the **`Decorate`** radio button in the **`SILK Controls`** window (see Figure 1-15). For example, Figure 1-16 shows a "blob" object that was drawn in decorate mode and added to the sketched interface containing the button. Decorations are rendered in a dark green to differentiate them from widgets. Decorations are another way to create ambiguous sketches which have no system-maintained interpretations, which is a desirable property in the early stages of design (see Section 1.3.1).

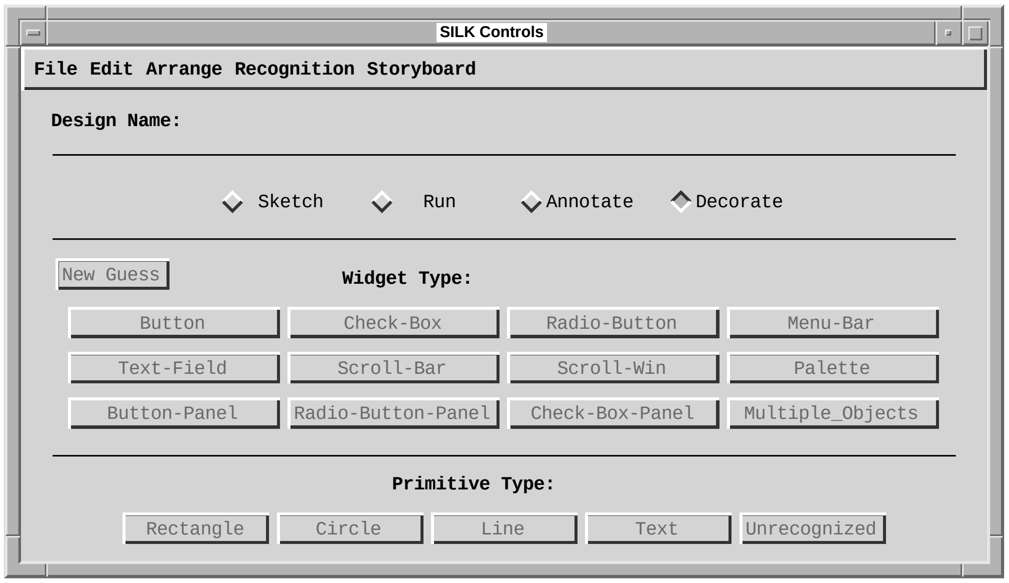


Figure 1-15. Pressing the **`Decorate`** radio button switches SILK to *Decorate* mode.

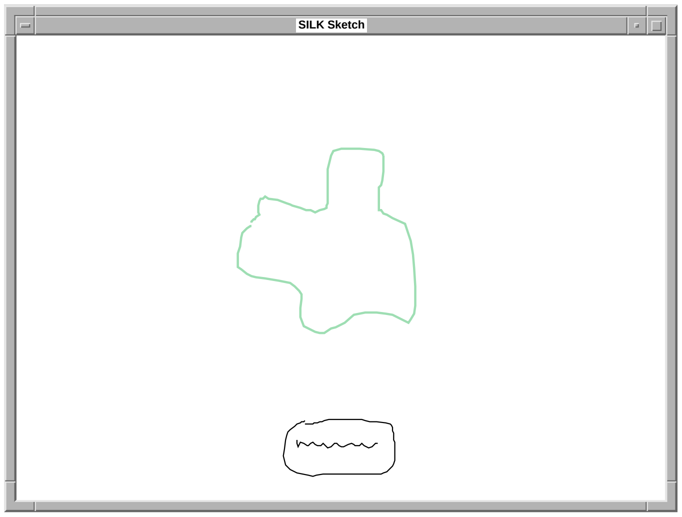
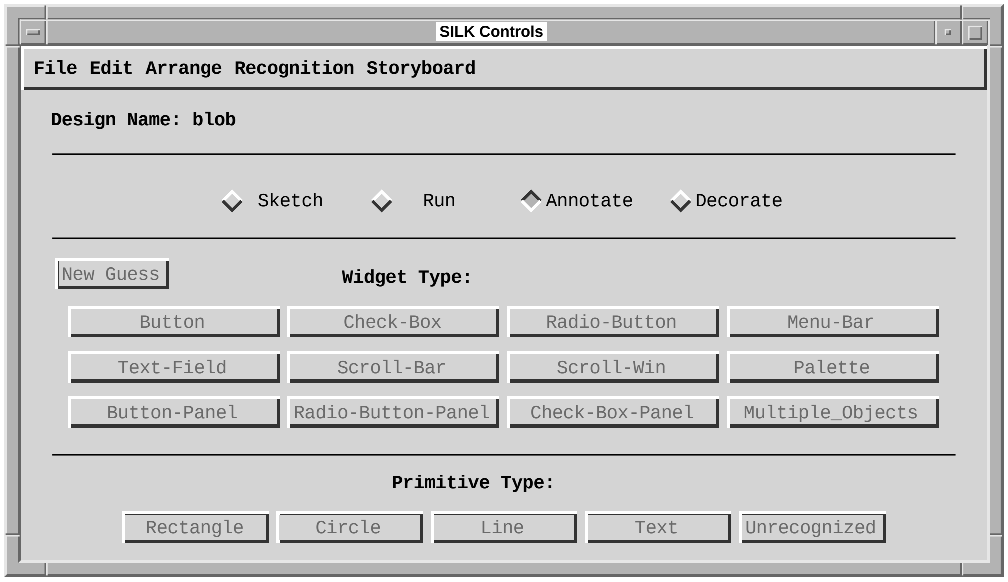


Figure 1-16. A "blob" drawn in *Decorate* mode has been added to the interface in Figure 1-9.

1.4.1.5 Annotate Mode

SILK also supports a mode for annotating sketches with drawn, written, or typed comments. The annotation layer, which is drawn in blue, can be displayed or hidden by toggling the **`Annotate`** radio button (see Figure 1-17) Figure 1-18 illustrates some annotations that have been added to the example illustrated in Figure 1-16.



Design Name: blob

Figure 1-17. Pressing the **`Annotate`** radio button switches SILK to *Annotate* mode.

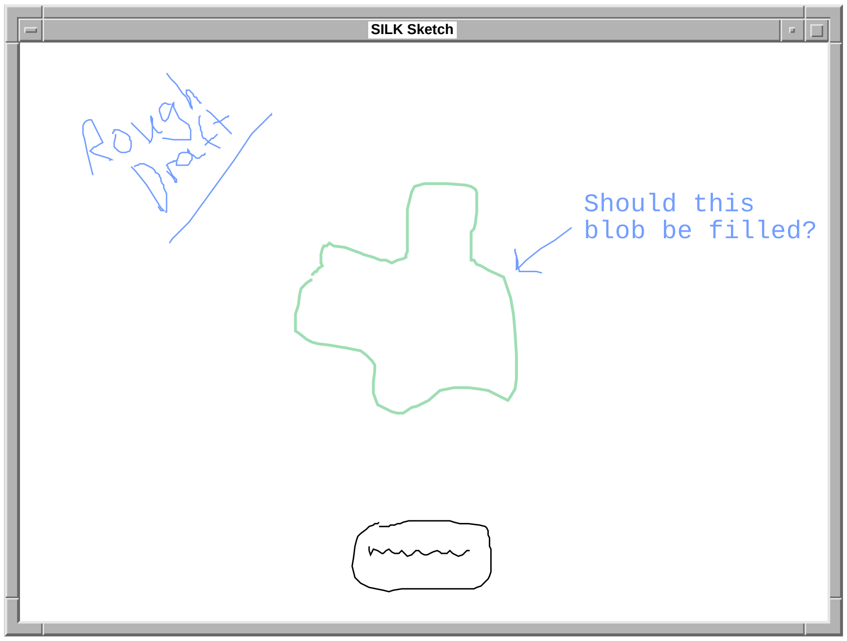


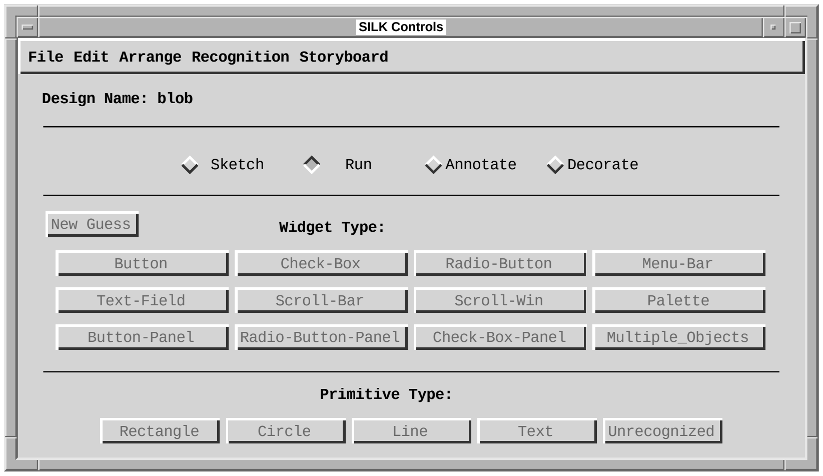
Figure 1-18. Annotations can be drawn, written, or typed and are displayed in blue.

1.4.2 Specifying Behavior

Easing the specification of the interface layout and structure solves much of the design problem, but a design is not complete until the behavior has also been specified.

1.4.2.1 Run Mode

Run mode allows a designer or end-user to test an interface design. It can be turned on at any time by selecting the **Run** radio button (see Figure 1-19). For example, as soon as SILK recognizes the button shown in Figure 1-9, the designer can switch to *Run* mode and operate the button by selecting it with the stylus or mouse. The button will highlight when held down (see Figure 1-20). The highlight goes away if the mouse is moved out of the button or if the mouse is released over it. As with button widgets included with standard interface builders, this feedback indicates that the button will perform an action if the mouse is released over it. The other widgets recognized by SILK have similar interactive behaviors (see Section 3.3 for illustrations and descriptions of these behaviors). Although only objects recognized by SILK offer this feedback, arbitrary sketched objects may still need to have interactive behaviors attached to them. This is accomplished using storyboards.



Design Name: blob

Sketch [✓ Run] Annotate Decorate

Figure 1-19. Pressing the **Run** radio button switches SILK to *Run* mode.

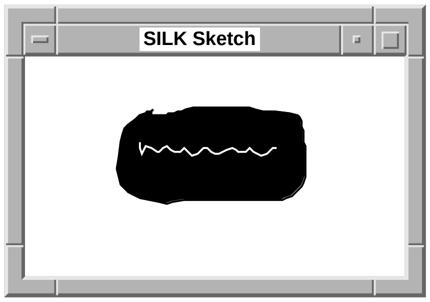


Figure 1-20. Buttons are highlighted when the mouse is held down on them in *Run* mode.

1.4.2.2 Storyboarding

Unfortunately, the behavior of individual widgets is insufficient to test a working interface. For example, SILK knows how a button operates, but it cannot know what interface action should occur *after* a user presses the button. This is specified using storyboards. Storyboards use the sequencing of screens to allow the specification of the dynamic behavior between widgets and the basic behavior of new widgets or application-specific objects, like a dialog box appearing when a button is pressed. Screen sequencing is expressed by drawing arrows in the **`Storyboard`** window from objects to screens that should appear when the object is clicked on with the mouse (see Figure 1-3). Storyboards are a natural representation, they are easy to edit, and they can easily be used to simulate functionality without worrying about how to implement it.

The interface illustrated in Figure 1-16 can be copied to the **`Storyboard`** window by selecting **`Copy Screen to Storyboard`** from the **`Storyboard`** menu. The original screen can then be modified in the **`Sketch`** window – in this example the blob is made larger. This process can be repeated until there are three screens in the **`Storyboard`** window, as illustrated in Figure 1-21. Then, arrows can be drawn from the buttons in each screen to the next screen, as shown in Figure 1-22. Now, switching to *Run* mode and clicking on the sketched button will cause SILK to make screen transitions in the **`Sketch`** window. The blob will appear to grow and then shrink back to its original size.

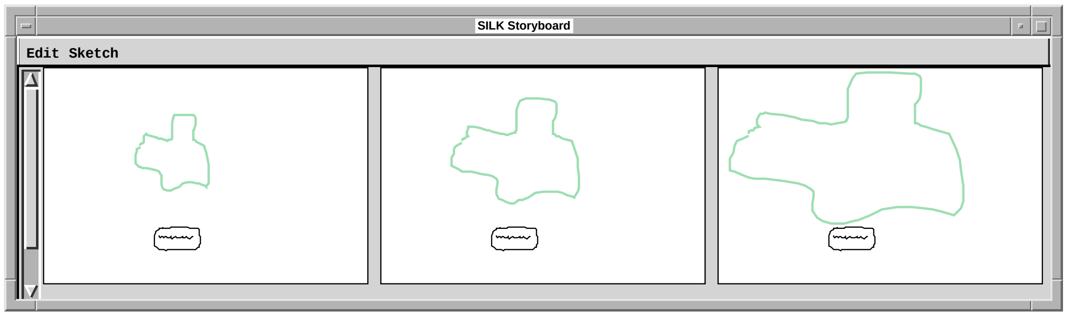


Figure 1-21. Three versions of the screen illustrated in Figure 1-16 are copied to the storyboard.

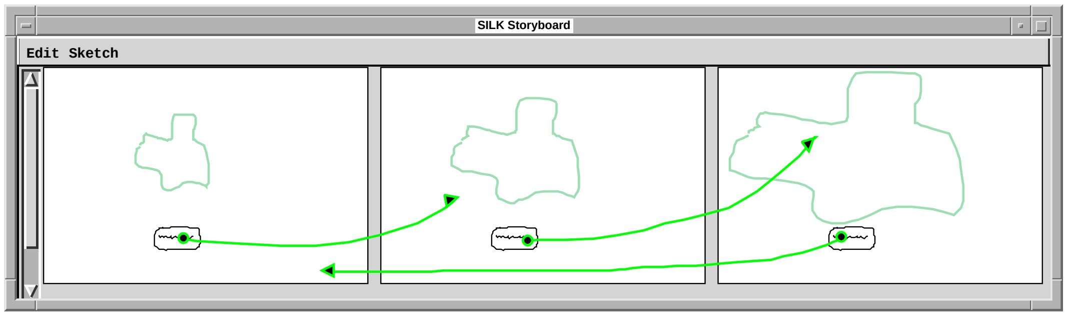


Figure 1-22. Arrows drawn on the storyboard indicate that when the button is clicked repeatedly, the blob should grow twice and then the sequence should start over.

1.4.3 Iterative Refinement

The philosophy behind SILK is that designers are never forced to give more detail than they wish to, but they can add more detail as these specifics become known. For example, the button labels shown so far have been simple text squiggles. This is only sufficient for a short time. Eventually, the designer may wish to specify handwritten labels and later, typed labels. This progression is illustrated in Figure 1-23. Note that there is no OCR done in this example; the designer first sketches squiggles, then replaces those with written labels[1], and later replaces the written labels with typed labels using the caret gesture. SILK recognizes all three versions as text labels and properly infers that they are part of a radio button panel.

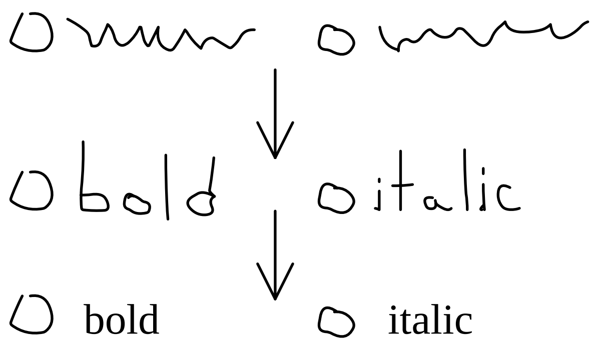


Figure 1-23. Iterative refinement of text labels: squiggle to handwritten to typed.

1.4.4 Transformation

When the designer is satisfied with the interface, SILK can create a new window that contains real widgets and graphical objects that correspond to those in the rough sketch. These objects can take on the look-and-feel of a specified standard graphical user interface – SILK currently supports the Motif look-and-feel. The transformed interface is only partially finished since the designer still needs to finalize the details of the interface (e.g., textual labels, colors, and alignment). At this point, programmers can add callbacks and constraints that include the application-specific code to complete the application. Figure 1-24 illustrates the transformed version of the third screen illustrated in Figure 1-22. Notice that the button has a generic label and the blob has been transformed as is.

1. The current version of SILK does not allow replacing squiggles with handwritten text that will be recognized as text. The code supports [- and]-shaped gestures to perform this operation, but it is not fully implemented.

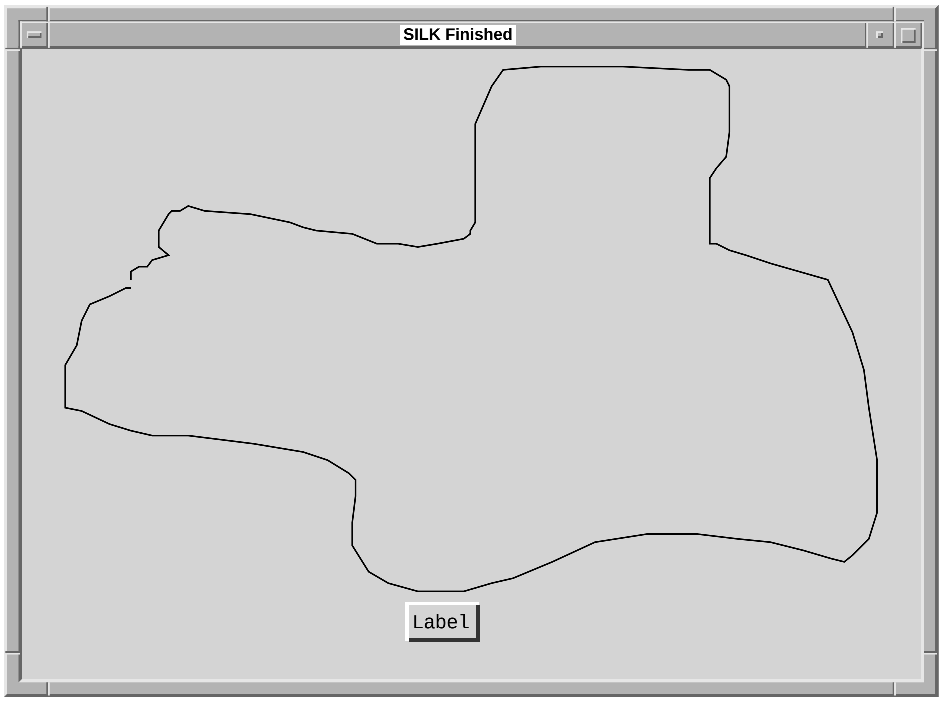


Figure 1-24. Transformed version of the third screen from the storyboard shown in Figure 1-22.

1.4.5 Mode Feedback

The proceeding overview highlighted that SILK's interface is very modal. There are four modes: *Sketch*, *Decorate*, *Annotate*, and *Run*. Knowing which mode the system is currently in does not appear to be problematic as SILK changes the drawing cursor to indicate the mode and the color of the ink is different for the three drawing modes (black for *Sketch* mode, green for *Decorate* mode, and blue for *Annotate* mode).

1.4.6 Data Model

The SILK data model is quite simple. A *screen* is a sketch that is created and edited in the **`SILK Sketch Window`**. It represents what is to be shown on the display to the end-user when the application being designed is running. A *design* is a collection of screens that are stored in the **`SILK Storyboard`** window. Designs are saved to the disk using a filename. Another *version* of a design is simply a design saved under a different filename.

1.5 Design Range

SILK is oriented towards the design of the user interfaces found in graphical editors and simple forms-based applications. Therefore, it recognizes and supports the interactive behavior of the standard UI widgets (e.g., buttons, text fields, scroll bars, and menus). SILK is not limited to this style of interface, as SILK's storyboarding component allows screen transitions to occur when arbitrary graphical objects are clicked with the mouse. A designer can use this mechanism to *simulate* interfaces that are not dominated by either the structure or behavior of the standard widgets. This technique was used by several designers during the SILK usability tests (see Chapter 6, "Evaluation") to support new widgets that were not built-in to SILK. Since SILK is mainly a tool for exploring and communicating interface ideas in the early stages of design, it is not necessary to support all possible interactive behaviors.

1.6 Implementation

SILK was written entirely in Common Lisp and runs on both UNIX workstations and the Apple Macintosh. SILK can be used with a mouse or a Wacom tablet. It is implemented using the Garnet User Interface Development Environment [Myers 1990c]. The rest of the dissertation details the implementation of the algorithms used by SILK.

1.7 Thesis and Contributions

The thesis postulated by this dissertation is that electronic sketches and storyboards can be effectively used to prototype user interfaces in the early stages of design. The contributions of the research include the following:

Concepts and Techniques:

- A new methodology, electronic sketching, for designing user interfaces.
- A method for illustrating the behavior of a user interface visually, using storyboards, without conventional programming.
- Feedback techniques for showing how the storyboard caused the current interface screen to become active and editing techniques for manipulating the storyboard screens and transitions.
- The automatic recognition of sketched user interface components.
- The transformation of sketched interface screens to operating user interfaces that use real widgets in a standard look-and-feel.

Artifacts:

- The first tool designed for use during the early phases of interface design by professional user interface designers. In addition, through screen transformations, SILK also supports the later phases of the interface design cycle. Finally, the implementation is suitable for use by designers wishing to experiment with this new methodology.
- A widget recognition algorithm that can be applied to other domains.

Experimental Results:

- A survey of designers which illustrates that designers are dissatisfied with the existing tools and methods used in the early stages of user interface design.
- An evaluation that shows that SILK is usable for quickly designing user interfaces and that designers are excited by the prospects of using electronic storyboarding instead of programming for illustrating interactive behaviors. The evaluation also showed that SILK allowed designers to effectively communicate their design ideas to engineers.
- A demonstration of successful uses for sketch-based interfaces.

1.8 Dissertation Outline

The rest of this dissertation describes how SILK functions and how it can be used effectively by user interface designers. To ensure that the system would work well for its intended users, I took an informal survey of professional user interface designers to determine the techniques they now use for interface design. The results of the survey and a discussion of how these results were used in the design of SILK are presented in Chapter 2.

The next three chapters describe SILK's technical details. Chapter 3 describes the algorithm used to recognize sketched widgets. Chapter 4 contains the implementation details necessary to support electronic storyboards. Chapter 5 discusses annotations, sketch transformations, and other supporting features for the later stages of the design process.

Chapter 6 discusses a descriptive evaluation of SILK with six designers and six engineers which showed designers could effectively use SILK to design user interfaces and that they saw a lot of potential for electronic sketching and storyboarding. The testing also illustrated that SILK allows designers to effectively communicate design ideas to engineers. In fact, the designers and engineers were able to discuss designs and make changes to them while the engineers were present.

Chapter 7 contains a review of the work related to this research. This is followed in Chapter 8 by a discussion of some new research problems that building SILK has brought to light. Finally, Chapter 9 summarizes the dissertation and reviews its contributions.

CHAPTER 2
Informal Survey of Designers

One of the basic tenets of human-computer interaction is the importance of user-centered systems design [Draper 1986]. This methodology requires knowing who the intended users of the system are, involving them in the design, keeping their needs in mind during design and implementation, and having these users test prototypes of the system throughout the design process. As mentioned earlier, SILK was designed for user interface designers, particularly those with an art or graphic design background. To determine the needs of this user population, I followed three related courses of action:

1) I read the design literature and on-line design discussion groups.
2) I spoke with designers and observed their designs and work spaces.
3) I surveyed designers with a written questionnaire.

The first two research items are discussed elsewhere (see Sections 1.2, 1.3, and 7.1). This chapter contains a description of the third item, the written questionnaire. It opens with an overview of the methodology used and is followed by some quantitative and qualitative results. The last section contains a discussion about what these results imply for a user interface design tool and how these results were used in the design of SILK.

2.1 Methodology

To help ensure that SILK would work well for its intended users, I took an informal survey of professional user interface designers to determine the techniques they now use for interface design. To carry out this survey, I took the following steps:

1) I wrote the initial questionnaire
2) I piloted the questionnaire with three designers

3) I revised the questionnaire based on the results of the pilot
4) I posted the questionnaire to the Internet's VISUAL-L mailing list[1]
5) I analyzed the returned questionnaires

Since the questionnaire was posted to a mailing list, the individuals who responded are self-selected. This is why I term the survey "informal." Nevertheless, these results were useful for giving some direction to the original design.

2.1.1 What Was Asked

The designers were asked about their background and the tools and techniques they used in the early stages of user interface design. The questionnaire asked them what they liked and disliked about paper sketching, and what they liked and disliked about electronic tools. The designers were also given a description of SILK and asked to comment on the strengths and weaknesses of such a tool. In addition, the designers were asked to estimate the percentage of time they spent sketching different types of interface elements. Finally, I asked the designers to send me sketches that they had made early in the design cycle of a user interface. The designers were assured that these sketches would be kept in confidence. The complete questionnaire is included in an appendix (see Appendix Section A.1).

2.2 Survey Results

The survey was first posted on September 4, 1994 and the last response was received on October 30, 1994. A total of 31 designers responded to the survey. There were approximately 300[2] subscribers to the list at that time. A summary of the data from the returned questionnaires is included in the appendix (see Appendix Section A.2).

2.2.1 Background of Designers

The designers surveyed have an average of about six years experience designing user interfaces. They work for companies from around the world that focus on areas such as desktop applications, multimedia software, telephony, and computer hardware manufacturing. Respondents who reported having backgrounds in architecture, art, graphic design, industrial design, or visual communications were grouped together and termed *visual designers*. Eighteen (58%) of the designers had this visual design background

1. The VISUAL-L mailing list discusses the design of visual interfaces and is maintained at Virginia Tech. University and can be accessed by mailing to the following email address: `listserv@vtvm1.cc.vt.edu`
2. This is an upper-bound estimate made by the mailing list maintainer Andrew Cohill (`cohill@bev.net`) in October of 1996: "back then my hazy memory recalls less than 300 people." Cohill estimates that there are approximately 950 subscribers as of October 1996.

and thirteen of the respondents did not. Eleven of the thirteen non-visual designers had backgrounds in engineering or psychology. Though not entirely accurate, the thirteen designers without a visual design background are termed *engineers*. All of the respondents are termed *designers*. As visual designers are the intended users of SILK, this chapter focuses on their responses to the questionnaire. The results for engineers are similar and are shown after the visual design numbers in the following sections. These numbers will only be compared in the cases where they are substantially different[3].

2.2.2 Why Designers Sketch

Almost all of the respondents (100% of the visual designers and 85% of the engineers) sketch in the early stages of user interface design, whereas only about half (56% of the visual designers and 62% of the engineers) also use software tools at this stage. The survey tried to collect some qualitative feedback both on why designers sketch at this stage and why they do not always use electronic tools.

The speed of sketching was the most commonly cited reason (by 50% of the visual designers and 62% of the engineers) for not using software tools at this stage. Both groups claimed that user interface tools, such as HyperCard or user interface builders, would waste their time during this phase. One designer stated that in the early stages of design “iteration is critical and must happen as rapidly as possible — as much as two or three times a day.” The designer said that user interface builders always slowed the design process, “especially when labels and menu item specifics are not critical.” This comment is consistent with that of another designer who claimed paper was more useful in the early stages when they “did not understand the problem well” and wanted to try several different ideas. Most of the designers also cited their familiarity with paper as a graphic designer. The use of pencil and paper was described as intuitive and natural.

The designers also noted the importance in leaving the design in a rough state. This keeps the designer from focusing on unimportant details, as illustrated by comments such as: “[sketches are] crude enough so that I don’t spend time trying to work on details that the design is not ready for” and “paper is the most versatile tool in the early stages of an interface design, when the ‘tyranny of the pixel’ is the furthest thing from the designer’s mind.” Another important motivation for keeping the design sketchy is to keep observers from getting the wrong impression of a design, as illustrated by comments such as: “there seems to be an implied commitment to something once it hits the screen,” “it is important that the client ‘sees’ the work as ideas and not

3. I only discuss these numbers when the absolute difference between them is more than 30 percentage points. I am not implying that the numbers differ in a statistically significant way. This is mainly due to the fact that the respondents to the questionnaires were not randomly selected and the sample size is rather small. Thus, statistical analysis may not be appropriate.

the final design – therefore the work should have that sketchy feel to it," and "if you use the computer you face the problem of 'over concretization' – that is, even the most flippant ideas look as if they are 'real interface.' The problem looks to be solved, when in fact it is not at all." Keeping details from getting in the way of both the designer and evaluators was cited by 22% of the visual designers and 8% of the engineers.

Other reasons cited for sketching included its ease of use and versatility (cited by 33% of the visual designers and 23% of the engineers), and the collaboration and portability afforded by sketching on paper and whiteboards (cited by 33% of the visual designers and 62% of the engineers). Finally, 39% of the visual designers and 23% of the engineers cited the perspective and ease of organization that sketches provided.

In summary, sketching was cited as more useful at this stage because it is good at "catching ideas quickly," it is inherently ambiguous, and it is portable.

2.2.2.1 Electronic Tools Designers Use

The designers gave two common reasons for using computer software in these early stages. First, software permits adding dynamic behavior that can be tested with users. Second, computer tools, specifically user interface builders, allow the designer to produce an artifact that may be used in the final design.

The designers used a wide range of software-based design tools in their work. This section tries to give a breakdown of the most common tools and shows some significant differences between those with and without a visual design background. Most of the designers surveyed (83% of the visual designers and 69% of the engineers) have used HyperCard, Director, or Visual Basic during the prototyping stage of interface design. Some also used high-powered user interface builders. The most commonly cited were TeleUSE [Thomson Software Products 1996], DevGuide [Sun 1991], NeXT Interface Builder [NeXT 1991], and X-Designer [IST 1996]. The designers listed 19 different user interface builders.

Macromedia's Director was the most common tool cited by those with a visual design background (61% of the visual designers and only 23% of the engineers), whereas Apple's HyperCard was more commonly used by engineers (69% of the engineers and only 33% of the visual designers). This difference between Director and HyperCard may be explained by their intended uses. Director was intended for building multi-media presentations, but it is fairly general and allows designers to paint, draw, and animate custom objects easily. In fact, everything is custom, as there are no built-in objects. This freedom is appealing to designers who are trying to give a flavor of an interface in the early stages of design.

In contrast, HyperCard is intended for building small applications based on button clicks. The drawing interface is fairly constrained and thus it is not as easy to create new objects, although there is a palette of standard widgets to choose from. In addition, HyperCard applications rely much more on its scripting language, HyperTalk, than Director presentations rely on its Lingo language. Designers with a background in graphic design or art typically do not have, nor do they wish to learn, sophisticated programming skills. For example, one designer claimed that Director was the "best prototyping tool, but [it] relies too heavily on code for simple prototyping."

The designers often reported that Director was only useful for "movie-like" prototyping, i.e., as a tool to illustrate the functionality of the user interface without the interaction. In addition, the designers disliked Director because it lacked a widget set, the designs could not be used again in the final product, and every control and system response had to be created from scratch. HyperCard was also cited for its lack of some necessary user interface components.

Visual Basic (used by 33% of the visual designers and 23% of the engineers) and user interface builders (used by 44% of the visual designers and 54% of the engineers) were used by both groups. It is possible that if the survey were done now, a larger percentage of designers would be using Visual Basic (VB) than HyperCard or Director, as VB seems to have become quite popular in the design community. This may be due to three reasons: 1) there are a large number of third-party widgets and controls (VBXs) available for VB, 2) it integrates a user interface builder with an interpretive language that is considered easy-to-learn compared with most traditional industrial programming languages (e.g., C), and 3) Microsoft's Windows operating system currently dominates the computer market.

The designers complemented the user interface builders, such as TeleUSE and the NeXT Interface Builder, on their complete widget sets and the fact that the designs could be used in the final product. Though, the difficulty of learning to use the user interface builders, especially those with scripting languages, was considered a drawback. In fact, "hard to learn" programming or scripting languages seemed to be a major concern for all of these systems. One designer said that interface design tools require "too much programming to do anything innovative."

2.2.3 What Designers Sketch

Besides finding out what tools designers use and why they prefer sketching to computer-based tools, it was also important to see what types of sketches designers made. In response to my request for sketches, I received four design sketches. These sketches were fairly rough and ambiguous. They contained squiggles to represent text and rough icons that acted as buttons. In addition many of the sketches were covered with

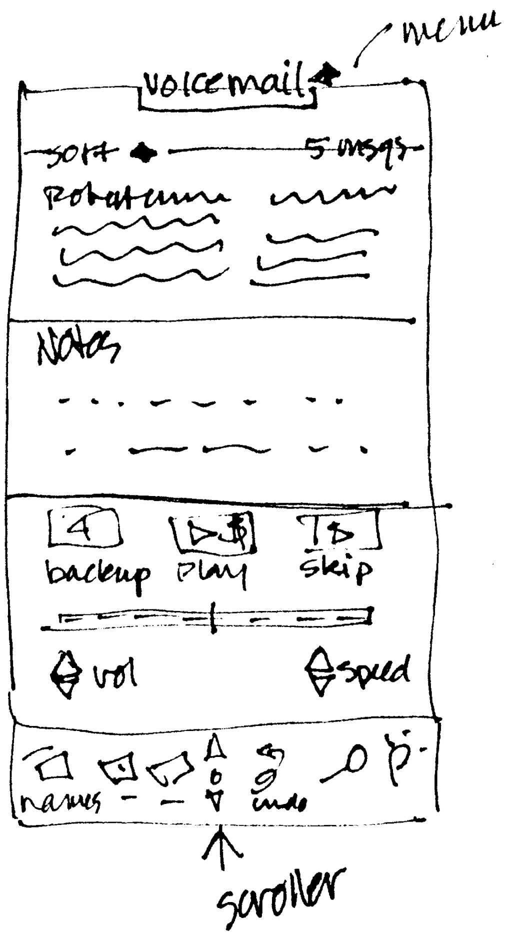


Figure 2-1. Sketch of a user interface for a voicemail application. Reproduced by permission of Annette Wagner, Sun Microsystems.

hand-written annotations. Figure 2-1 is an example of one of the sketches I received; I was given permission to reproduce this sketch. As mentioned previously, the designers were assured these sketches would be kept in confidence, so the rest are not reproduced here. The remainder of this section reports the mean percentages of time that designers said they spent sketching particular types of user interface elements.

The survey showed the designers spent much of their time sketching the standard widgets that are found in a toolkit: buttons, scroll bars, etc. (26% of visual designer's time and 30% of engineer's time). Arbitrary static graphics and decorations were the next most commonly sketched elements (25% of the work of visual designers and 15% of the work of engineers). The written comments showed that designers were unhappy with tools that did not allow them to add these decorations easily. Interactive graphical objects (4% of visual designer's work and 12% of engineer's work) and multimedia applications that use video and sound (9% of visual designer's work and 12% of engineer's work) were less commonly sketched.

The survey also showed that designers occasionally need to design controls with custom looks or behaviors (20% of the work of visual designers and 12% of the work of engineers). Often these are widgets whose behavior is analogous to that of a known widget. For example, designers frequently need to draw a new icon and specify that it should act like a button. Occasionally, designers need to sketch a widget that has an entirely new behavior. Finally, the "other" category was responsible for the remaining design time (16% of the time of visual designers and 18% of the time of engineers). This category included designing the layout of widgets and designing interactions that span multiple widgets or windows.

Several of the written comments show that designers often illustrate sequences of system responses and annotate the sketches as they are drawn. One designer claimed that an annotated sketch could be presented to management and tested with users before building a prototype. This is a simple form of storyboarding. Another said that "storyboarding is extremely important to interface design. It's a quick way to show a user/client how they would perform search, selection, and navigation on an interface/ system without having the system engineers and programmers actually implement the design into a system."

2.2.4 Interest in SILK

The designers reacted favorably to a short description given to them about SILK. Some were concerned that it was not really paper and that they might need to get accustomed to it. Others wanted assurances that the system would share the portability they currently have with paper. The comments were positive overall. The designers felt the described system would allow quick implementation of design ideas and it would also help bring the sketched and electronic versions of a design closer together. In addition, the designers were happy with the ability to quickly iterate on a design and to eventually use that design in the final product. All but three (83% of visual designers and 100% of engineers) expressed a willingness to try such a system.

2.3 Design Implications for SILK

The primary purpose of the informal design survey was to confirm the original design ideas for SILK and to focus the research on areas that were not originally anticipated. For example, until I visited designers in their work places and took the design survey, I did not even consider the support for annotations of sketches.

One of the most important results of the survey was in highlighting the importance of sketching in two different respects. First, designers believe the physical action of sketching is quicker and encourages creativity during this early design phase. Second, the roughness embodied in a sketch keeps both the designer and evaluators from focussing on details too early in the design process. This result confirmed two major design decisions: the choice of a sketch-based interface for design instead of a palette of sketchy widgets and the decision to keep the design rough rather than cleaning it up after each widget recognition.

Another important result of the survey was the fact that many of the designers seemed to use software-based tools only because of the interactivity or the re-use they provided. This confirmed that interactive behavior should be an important part of any new tool and that the entire design process could be improved if SILK had the ability to hand off the design to another tool later in the design process (see Section 5.2).

Finally, by finding out that designers spent a lot of their time sketching common toolkit widgets and static graphics, SILK could be designed to support these elements best, yet still allow less common elements to be used. Since many designers already use paper-based storyboards, need to support interaction, and occasionally design widgets with new behaviors, electronic storyboarding can be used to support these needs (see Chapter 4, “Storyboarding”).

CHAPTER 3
Widget Recognition

Allowing designers to sketch on the computer, rather than on paper, has many advantages, as discussed in Chapter 1. Several of these advantages cannot be realized without software support for recognizing the interface widgets in the sketch. Having a system that recognizes the drawn widgets gives the designer a tool that can be used for designing, testing, and eventually producing a final application interface. SILK's recognition engine identifies individual user interface components as they are drawn, rather than after an entire sketch has been completed. This way the designer can test the interface at any point without waiting for the entire sketch to be designed and recognized. This is key for a tool that supports iterative design. In fact, I observed subjects trying out new ideas and immediately testing them during usability testing (see Chapter 6, "Evaluation").

This chapter gives an overview of the widget recognition process, describes how primitive components are recognized, explains how the rules system is used to combine primitive components into widgets, and finishes with some ideas on how to learn new rules.

3.1 Recognition Overview

Working within the limited domain of common 2-D interface widgets (scroll bars, buttons, pulldown menus, etc.) facilitates the recognition process. This is in contrast to the much harder problems faced by systems that try to perform generalized sketch recognition or beautification [Pavlidis 1985]. A rule system, which contains basic knowledge of the structure and make-up of user interfaces, allows SILK to infer which widgets are included in the sketch.

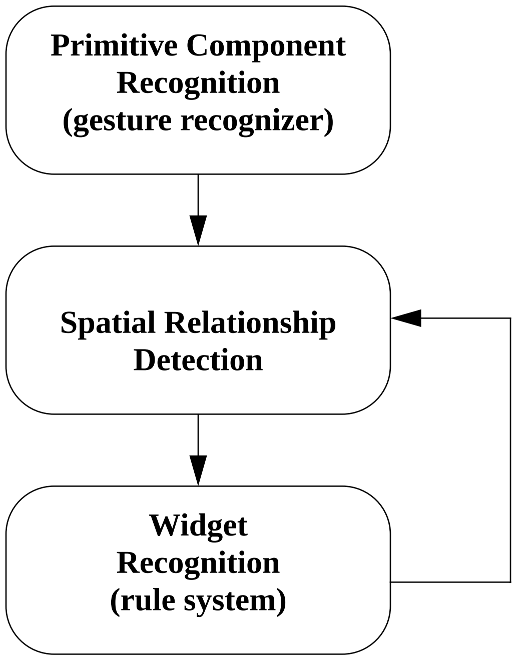

Primitive Component Recognition (gesture recognizer)
Spatial Relationship Detection
Widget Recognition (rule system)

Figure 3-1. Phases of SILK's widget recognition algorithm.

The widget recognition algorithm is broken into three phases, as illustrated in Figure 3-1. First, SILK tries to recognize the *primitive components* in the sketch as they are drawn. The primitive components are basic shapes that when combined together make up a widget. For example, the scroll bar in Figure 3-2 was created by sketching a tall, thin rectangle and then a small rectangle (though the order in which they were sketched does not matter[1]). After recognizing a primitive component, the system looks for spatial relationships between the new component and other components in the sketch. Finally, the result is passed to a rule system which tries to combine that component together with related components to form a more complex widget. The rest of this chapter describes these three recognition phases.

1. The arrows indicate a preferred starting point and direction of the stroke for each primitive.

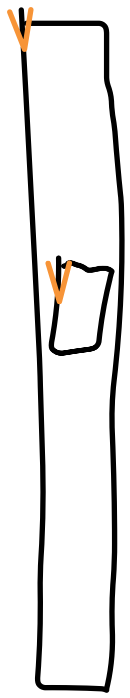


Figure 3-2. A vertical scroll bar composed of two rectangles.

3.2 Recognizing Widget Components

The recognition engine uses Rubine’s gesture recognition algorithm [Rubine 1991b; Rubine 1991c] to identify the primitive components that make up an interface widget. The primitive components recognized by SILK include rectangles, circles, lines, and squiggles (which represent text). These components are illustrated in Figure 3-3. The arrows indicate a preferred, but not required, drawing direction. Each of the primitive components of a widget are trained by example using the Agate gesture training tool [Landay 1993]. Each primitive component was trained with 15–20 examples. The examples varied in size and drawing direction.

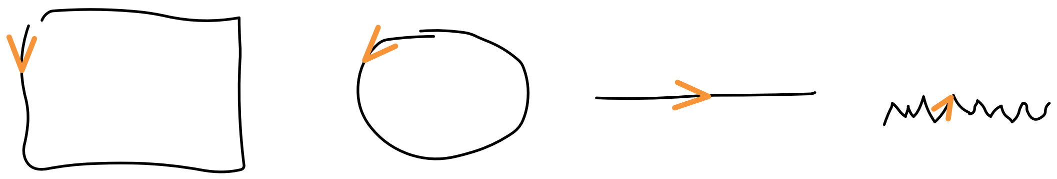


Figure 3-3. Primitive components recognized by SILK – rectangle, circle, line, and a squiggle that represents text.

3.2.1 Rubine’s Algorithm

Rubine’s algorithm uses statistical pattern recognition techniques to train classifiers and recognize gestures. These techniques are used to create a classifier based on the features extracted from several examples. In order to classify a given input gesture, the algorithm computes the distinguishing features of the gesture and returns the best match with the learned gesture classes. The following features of the gesture are used:

- cosine and sine of initial angle with respect to the x axis
- length of the bounding box diagonal
- angle of the bounding box diagonal
- distance between first and last point
- cosine and sine of angle between first and last point
- total gesture length
- total angle traversed

Rubine’s original algorithm also computed the maximum speed (squared) and the path duration, but these two temporal features were not used as Rubine found they were not very useful [Rubine 1991a].

3.2.1.1 Limitations of Rubine’s Algorithm

The algorithm currently limits SILK to single-stroke gestures for the primitive components. This means the components must be drawn with a single stroke of the pen (i.e., not lifting up the pen or mouse button during the stroke). A single stroke of the pen was used to draw each of the rectangles that comprise the scroll bar in Figure 3-2.

There are several possible solutions that overcome this limitation. The first solution takes advantage of the observation that single-strokes only appear to be a limitation for rectangles. That is, most of the other shapes are commonly drawn with only one stroke of the pen. Therefore, the low-level gesture recognizer could be trained with **L** and **U** shapes, as seen in Figure 3-4. By combining recognized line gestures together to form these shapes and also combining recognized **L**- and **U**-shaped gestures together with lines, the system could then recognize rectangles made with 1, 2, 3, or 4 strokes.

Another solution to the multi-stroke problem is to modify the low-level recognizer to support multiple strokes. One way to do this is to add a time-out mechanism to the gesture interactor supplied by Garnet [Landay 1993]. The interactor currently sends points to the recognizer as soon as the pen is lifted. Instead, it could wait until the pen has been lifted for a specified amount of time. If the user starts making new strokes before the time-out has passed, then it is assumed that the new stroke is a con-

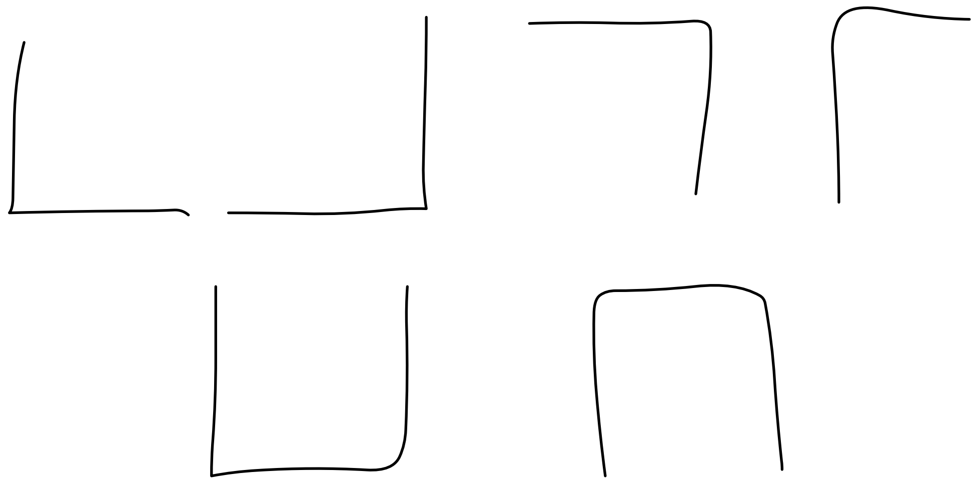


Figure 3-4. Partially completed rectangles made up of **L** and **U** shapes.

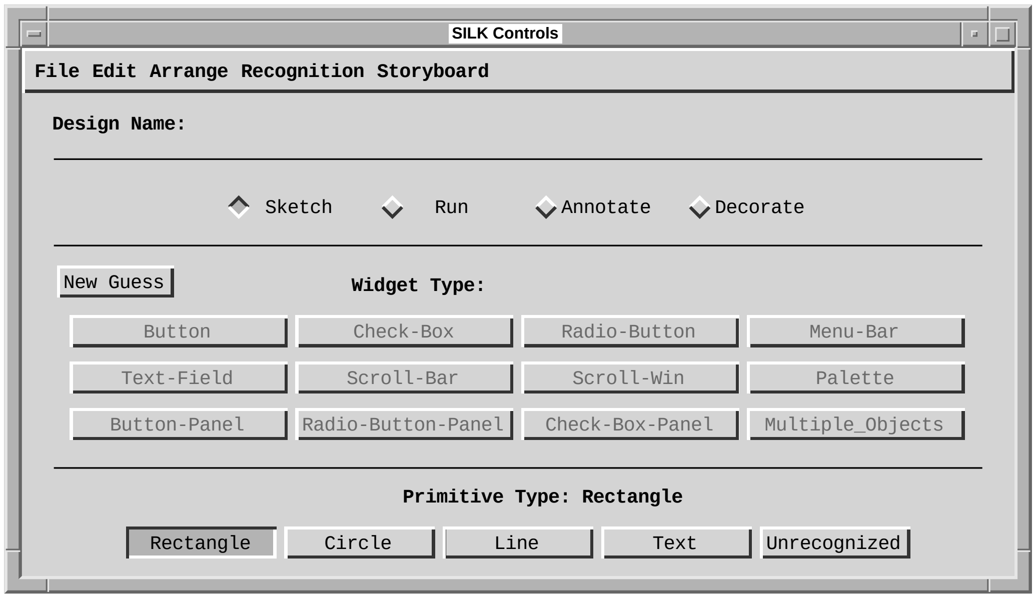
tinuation of the previous stroke. Eventually, the time-out will be passed without a new stroke. At that point, the previous set of strokes can be ordered in some logical way (for example, by finding the closest strokes to the endpoints of a given stroke) and passed as one set of points to the unmodified single-stroke gesture recognizer.

The main difficulty with a time-based scheme is in picking a time-out value that is not confusing to the user – it should not force unrelated strokes to be grouped together and likewise it should not separate related strokes. Other researchers have used a scheme in which the time-out is proportional to the size of the previous stroke. They have indicated that a more complex scheme may be necessary [Apte 1993b]. Another problem may be in ordering the points in a way that does not undermine the predictive ability of the features used by Rubine's algorithm.

The other major limitation of Rubine's algorithm is the fact that it does not account for variations in size or rotation. In SILK, rotations are only a problem for editing gestures, but the lack of support for variations in size is problematic in recognizing both editing gestures and the primitive components. One way to deal with this limitation is to train gestures of many varying sizes. The problem with this approach is that it leads to a recognizer that performs better on the average sizes, but not well on the outliers. To work around this in SILK, I broke up some of the primitive components into multiple classes. For example, rectangles are divided into `rectangle-large`, `rectangle-small`, `rectangle-very-small`, and `rectangle-thin`. All of these are mapped into the type `rectangle` as soon as they are returned from the low-level gesture recognizer. Rubine's algorithm tends to do much better when the gestures are organized in this manner.

3.2.2 Training from Corrections

One innovation in SILK is the ability to learn new examples of gestures representing the primitive components during regular use of the system. This feature was added after some early user testing revealed that when the system made a recognition error on a primitive component, the user was forced to delete the object and then redraw it. Previously, the type of the recognized component was only displayed in the **`Primitive Type`** field. In response to this problem I added a panel of buttons in the **`SILK Controls`** window (see Figure 3-5). These buttons represent the primitive components. When SILK recognizes a primitive component, the appropriate button is highlighted. If the system misrecognizes the component, the user can easily click on one of the alternative buttons and SILK will use that new type definition instead.



Primitive Type: Rectangle

Figure 3-5. Primitive type buttons (bottom) in **`SILK Controls`** window.

The next improvement in SILK's recognition mechanism was taking advantage of the user's correction to improve future inferences. After passing the points composing the corrected stroke, the correct gesture name, and the current gesture classifier to the gesture training routines, SILK produces a new classifier which will more accurately recognize the user's strokes. As implemented, the system performs this retraining transparently to the user. The new classifier takes effect immediately, but is not saved to disk until the user next saves her design. The new classifier is saved to the user's home directory and the system looks for one of these personalized classifiers each time it starts up.

3.3 SILK Widgets

The following set of figures illustrate[1] the complete set of widgets SILK recognizes and the feedback these widgets display when manipulated in *Run* mode. SILK also recognizes vertical and horizontal sequences of buttons, check boxes, and radio buttons as panels.

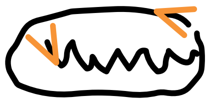


Figure 3-6. A button.

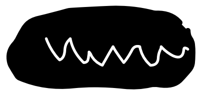


Figure 3-7. Buttons are drawn in reverse video when the mouse button is held down over them. The feedback is removed when the mouse is released.

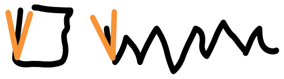


Figure 3-8. A check box.

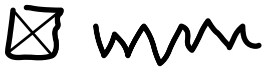


Figure 3-9. Check boxes display an X-shape in the box when they are selected with a mouse click on the box. Another click on the box will deselect it and turn the feedback off.

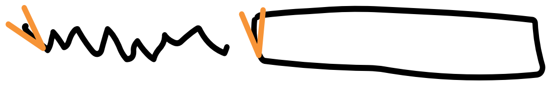


Figure 3-10. A text field. Note: the text field currently offers no *Run* mode feedback.

1. Note that these sketches were not generated with SILK to allow the easy addition of arrowheads.

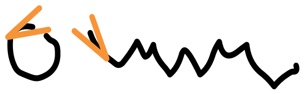


Figure 3-11. A radio button.

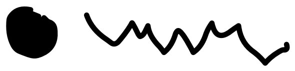


Figure 3-12. Radio buttons fill the circle when they are selected with a mouse click on the circle. Another click on the circle will deselect it and turn the feedback off. Note: if the radio button is in a panel of buttons, only one can be selected at any time. Selecting a new radio button in the same panel will deselect the currently selected radio button.

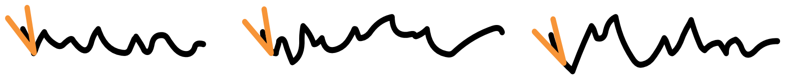


Figure 3-13. A menu bar. Note: labels must be drawn at upper left corner of the **`SILK Sketch`** window.

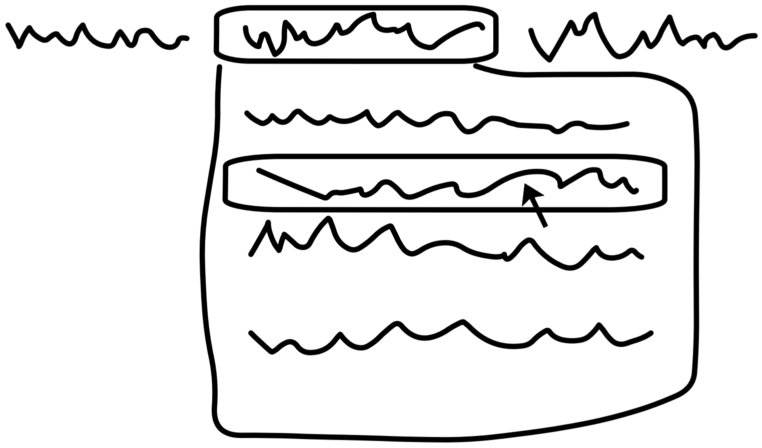


Figure 3-14. Menu bars display a generic pulldown menu when the mouse is held down over the bar items. The mouse can be moved between menu items in the resulting pulldowns. Another menu can be selected by moving the mouse to the new bar item without releasing. The currently selected bar item and menu item are enclosed by roundtangles. When the mouse is released, the pulldown disappears. Note: currently, there is no way of specifying the strings for the menu items (though they can be specified for the bar items) or the transitions from those menu items.

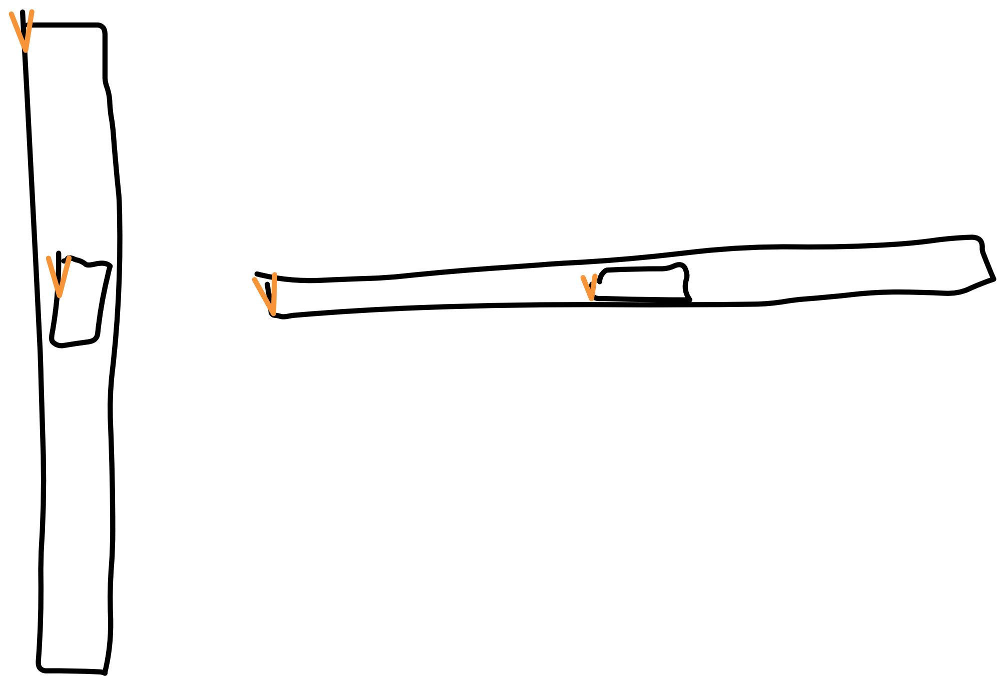


Figure 3-15. Vertical and horizontal scroll bars. The scroll bar “elevator” can be dragged within the confines of the outer rectangle.

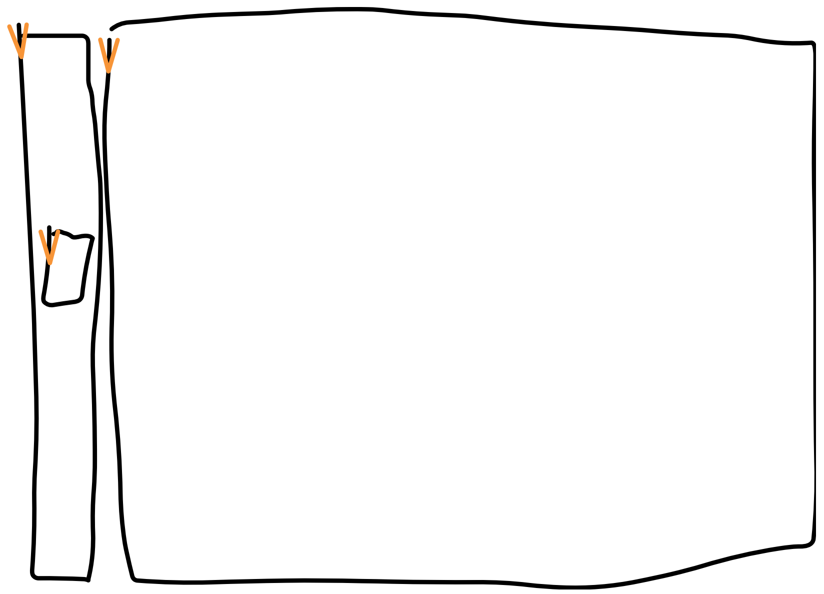


Figure 3-16. A scrolling window. Note: scrolling windows can have any combination of scroll bars at left and below. Currently, objects in the window do not scroll.

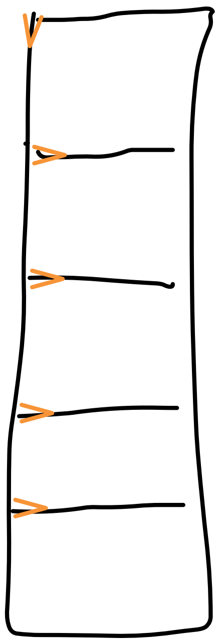


Figure 3-17. A palette. Note: there can be an arbitrary number of boxes and arbitrary graphics can be drawn in the boxes.

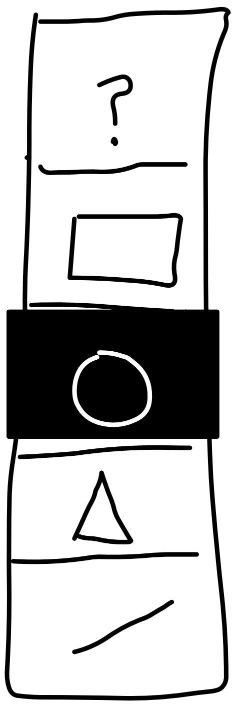


Figure 3-18. Clicking on the rectangle containing a palette item selects it. The currently selected item is displayed in inverse video. Only one item can be selected at a time.

3.4 Composing Components

In order to recognize interface widgets, SILK's algorithm must combine the results from the classification of the single-stroke gestures that make up the primitive components. The way it does this is to first find some basic spatial relationships between the new component and the other primitive or widget components in the sketch. Then these relationships are passed along with the known types of components to a rule system that tries to identify the most likely widget which includes the new component.

3.4.1 Spatial Relationships

As each component is sketched and classified it is passed to an algorithm that looks for the following spatial relationships (among both primitive and widget components):

- Does the new component contain or is it contained by another component?
- Is the new component near (left, right, above, below) another component?
- Is the new component in a vertical or horizontal sequence made up of any combination of components of the same type or sequences of that type?

The first relationship is the most important for classifying widgets. Many of the common user interface widgets can be expressed by containment relationships between more primitive components. For example, the scroll bar in Figure 3-2 is a tall, skinny rectangle that contains a smaller rectangle. The second relationship allows the algorithm to recognize widgets such as check boxes, which usually consist of a box with text next to it. The final relationship allows for groupings of related components that make up a set of widgets (e.g., a panel of radio buttons).

3.4.1.1 Overriding Spatial Relationships

After early user testing, SILK was modified to allow the user to specify hints that override the normal calculation of these spatial relationships. A prime example of this is when sketching a pulldown menu bar. A pulldown menu bar is defined as a sequence of menu items at the top left of the window. Real applications often also have some items of the same menu bar at the far right of the window (e.g., for a help menu). Unfortunately, sequences are defined as a series of the same type of object such that each item is “near” the preceding item. This rules out many cases, including the one described above.

To allow cases that violate SILK's distance constraints, the distance between objects is ignored if multiple objects are selected at the time of inference. To give this hint, the user can select multiple objects and then choose **`New Guess`**. Then, the objects will be considered near each other and objects of the same type in a horizontal or vertical line will be considered a sequence. This is also useful when the designer sketches the text of a radio button or check box too far from the circle or rectangle, respectively.

3.4.2 Rule System

After identifying the basic relationships between the new component and the other components in the sketch, SILK passes the new component and the identified relationships to a rule system that uses basic knowledge of the structure and make-up of user interfaces to infer which widget was intended. Each of the rules that matches the new component and relationships assigns a confidence value that indicates how close the match is.

3.4.2.1 Rule Structure

There is at least one rule for each widget[1] that is recognized by SILK. The "test" part of the rule checks whether the rule applies. For example, the test for a vertical scroll bar is illustrated in Figure 3-19. This test makes sure that one component is contained by the other, both components are rectangles, and the container is skinny. This rule illustrates that some simple graphical characteristics of the objects (in this case determining whether the container is skinny or not) are considered in addition to the component types and spatial relationships.

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

Figure 3-19. Test for vertical scroll bar rule.

1. The current implementation uses exactly one "rule" for each widget. This is because the rules are implemented as arbitrary Lisp functions that can each have multiple "test" / "then" pairs. A cleaner implementation would break these compound rules up into multiple rules.

The “then” part of the rule simply returns a list containing a confidence value for that match, a function that when evaluated can add the correct interactive behavior to the given sketchy components, and the widget type. For example, the “then” part for the vertical scroll bar is illustrated in Figure 3-20. At this time, the confidence values are assigned weights that I have constructed by simple experimentation. Rules for widgets that have no alternative inferences are given a confidence value of 1. Rules for widgets with more than one “test” / ”then” pair are given a range of values less than or equal to one. The pair with what seems the most likely inference is given a confidence value of 1. The complete set of rules used in SILK is given in Appendix E.

```
(list 1 ‘(scroll-bar-constructor
             ,container ,containee
             ,widget-agg :vertical)
        ‘scroll-bar)
```

Figure 3-20. “Then” part for vertical scroll bar rule.

3.4.3 Attaching Behavior

After each of the rules have been tried, the algorithm then takes the match with the highest confidence value and evaluates the constructor function given in the list returned by the rule. This function assigns the sketched components to a new aggregate object that represents the widget. The system keeps a list of all rules that matched ordered by confidence value. This way it is easy to allow the designer to revise SILK’s inference if it is incorrect (see Section 3.6). If none of the rules match, the system assumes that there is not yet enough detail to recognize the widget. Note that there is one constructor function for both horizontal and vertical scroll bars, so the rule illustrated in Figure 3-20 passes the `:vertical` tag to cause a vertical scroll bar to be constructed.

Each of the widgets that SILK recognizes has a corresponding Garnet object that use the Garnet Interactor mechanism [Myers 1990a] to support interaction and feedback. When SILK identifies a widget, it attaches an instance of an interactor object that implements the required interaction to the new aggregate object. For example, the scroll bar interactor object knows how to allow dragging of the “elevator” up and down without letting the “elevator” leave the confines of the scroll bar.

3.5 Recognition Examples

This section presents two walk-throughs of the widget recognition algorithm. In the first example, the designer initially drew a text squiggle Figure 3-21 (a). The gesture recognizer returns `text` (b), but no graphical relationships (c) and thus no rule system matches are found (d). The user then sketches a rectangle around the text (e), which is recognized by the gesture recognizer (f). This is passed to the graphical relationship detector which returns the fact that the rectangle contains some text (g). Using this, the rule system infers that the rectangle and text combine to form a button (h).

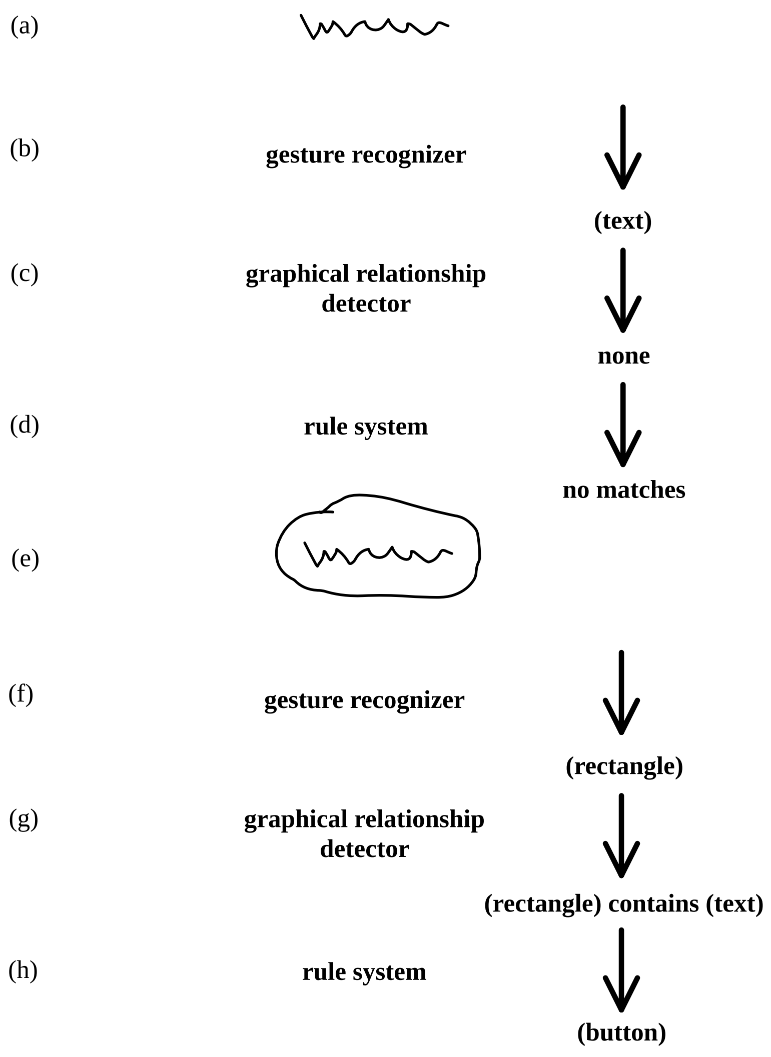


Figure 3-21. Phases performed by the widget recognition algorithm on the example drawn in (a) and (e).

The second example is shown in Figure 3-22. Assume the check box on the left and the rectangle on the right in (a) have been successfully recognized. After the squiggle on the far right is drawn, the gesture recognizer returns `text` (b). This is passed to the graphical relationship detector which returns the fact that the text is to the right and aligned[1] with the rectangle (c). This fact is then passed to the rule system which infers that the rectangle and text combine to form a check box (d). This is fed back into the graphical relationship detector (as implied by Figure 3-1), which detects a sequence of check boxes (e). Finally, this is passed again to the rule system, which returns a match for a check box panel (f). This is passed back into the graphical relationship detector, but no further relationships are found.

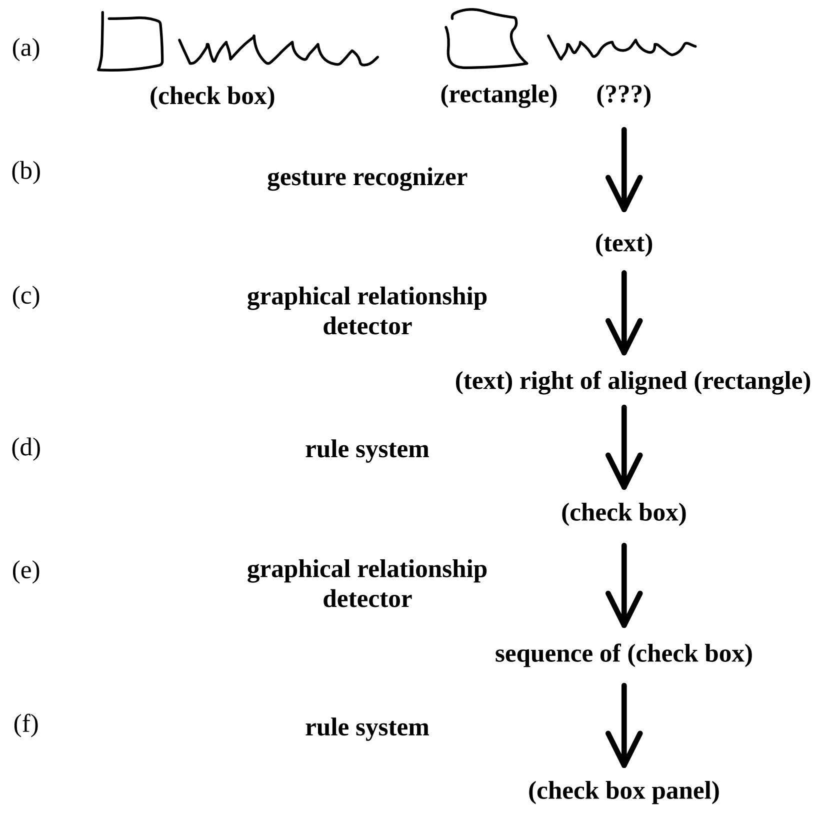


Figure 3-22. Phases performed by the widget recognition algorithm on the example drawn in (a).

1. Note that the text is too far from the check box on the left for any relationship between those two objects to be returned.

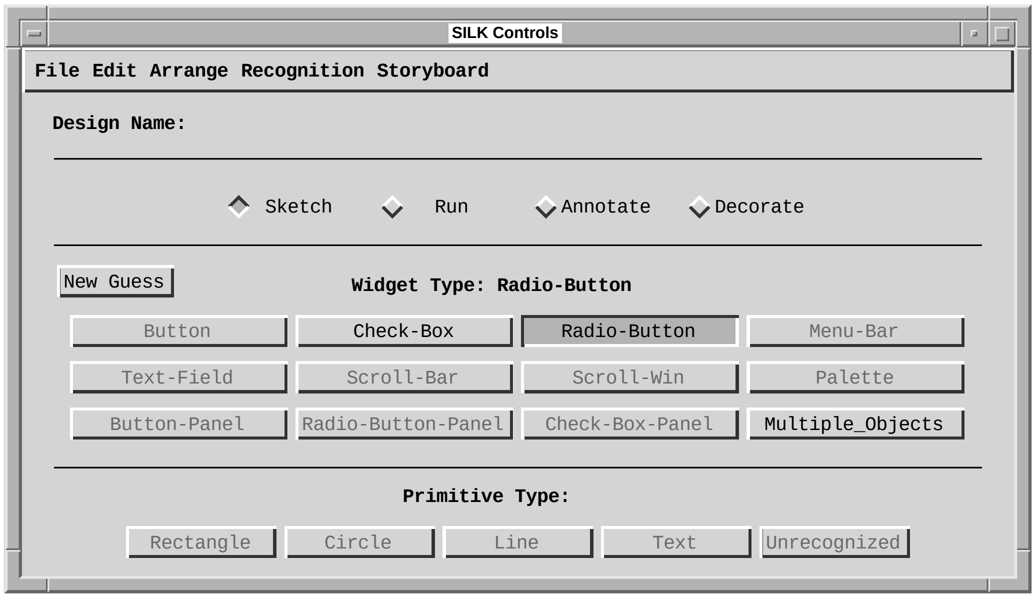
3.6 Revising Widget Inferences

In any system that does inferencing, it is important to give the designer good feedback on the inference results and make it easy to change those results when the system has erred.

3.6.1 System Feedback

SILK gives the designer feedback on two related decisions: the type of the last primitive or widget inference, and an indication of which alternative inferences are available. SILK displays primitive type inferences in the **`Primitive Type`** field and in a panel of buttons, as was described in Section 3.2.2.

Widget inference feedback is given in three different ways: using color, text fields, and button panels. After an inference has been made, the system draws the primitive components forming the proposed widget in purple to indicate they are related. In addition, the widget type is displayed in the **`Widget Type`** field of the **`SILK Controls`** window (see Figure 3-23). The type is also selected in the panel of widget buttons in the same window. The buttons representing all other possible inferences are also enabled. For example, Figure 3-23 shows that SILK has inferred a radio button, but indicates that a check box or multiple unrelated objects are also possible inferences by not graying out those buttons. The control window can be iconified if the designer



Design Name:

Widget Type: Radio-Button

Primitive Type:

Figure 3-23. SILK infers that the designer has drawn a radio button. The buttons for check box and multiple objects are also enabled, indicating those are possible alternative inferences. All other buttons are grayed out, indicating they are not possible inferences.

wishes to sketch ideas quickly without any interruptions. Later, she can go back, select each widget, and make sure they are of the correct type.

3.6.2 Automatic Reinferencing

Modifying an existing sketch should cause SILK to revise previously made widget identifications. For example, adding new components to the sketch, deleting components in a widget, or moving components should all cause the system to revise previous widget identifications. This would only occur if the changes caused the rule system to identify a different widget as more likely than its previous inference. The current implementation does *not* do this.

SILK lacks automatic reinferencing because the graphical relationship search does not check the children of any top level object or widget. This could be changed, but it is not clear that breaking up previously inferred widgets would be satisfactory without other major changes to the system. Specifically, the calculation of rule confidence values would need to account dynamically for how well a particular rule matched the given components and spatial relationships. SILK's hard-coded confidence values only allow choosing between known alternative inferences for the same set of components. For example, SILK can choose between a radio button and a check box. This is necessary because the low-level primitive component recognizer is not 100% reliable at distinguishing between small circles and small rectangles.

Figure 3-24 illustrates the problem with hard-coded confidence values. The three objects were drawn in order from left to right. The confidence values say nothing about choosing between the possible check box on the left or the possible text field on the right. Which is more likely? Should the system automatically break-up the check box to allow the text field inference? Currently, this can only be done by explicitly hitting the **`New Guess`** button while the check box (the first inference) is selected.[1]

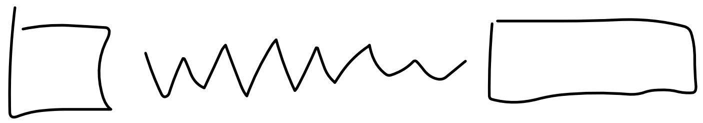


Figure 3-24. Should SILK infer a check box or a text field?

1. Note that the rectangle on the right does not need to be selected. It will be considered with the text when the check box on the left is broken up in order to generate new guesses.

3.6.3 Designer-guided Reinferencing

The designer can help the system make the proper inference when the system has made the wrong choice, no choice, or the widget that was drawn is unknown to the system. In the first case the designer might use the **`New Guess`** button or gesture to ask the system to try its next best choice. Alternatively, the designer may just click on one of the available **`Widget Type`** buttons in the **`SILK Controls`** window. These alternative inferences are only kept until the user starts sketching something new. If the **`New Guess`** button is used later, SILK runs its inference algorithm, regenerates the alternatives, and changes the type of the widget to the next best choice.

If SILK made no inference on the widget in question, the designer might select the primitive components and click on the **`New Guess`** button, thus forcing the system to reconsider its inference and focus on the components that have been selected. As discussed previously (see Section 3.4.1.1), SILK will then ignore distance requirements between the selected items when testing the widget rules. For example, Figure 3-25 shows the rule for a radio button. If the individual components of a potential radio button are selected (see Figure 3-26) and the user clicks on the **`New Guess`** button, SILK redefines the result of `near-p` to return `true` for the two objects, not just objects that are near the text string. This redefinition will cause the radio button shown in Figure 3-26 to match the radio button rule.

```
(and (circle-p comp1)
     (text-p comp2)
     (near-p comp1 comp2)
     (aligned-p comp1 comp2 :bottom)
     (horizontal-left-p comp1 comp2)
     (check-box-width-p comp1))
```

Figure 3-25. Test for radio button rule.

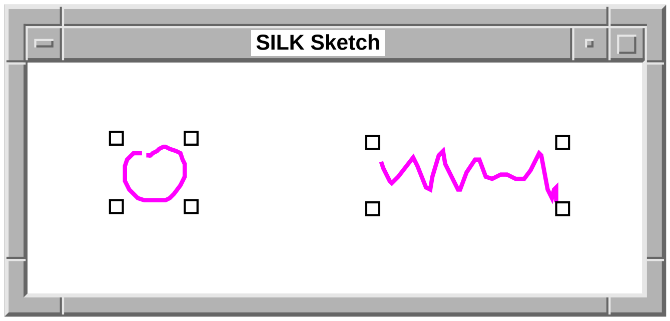


Figure 3-26. A radio button whose components have been drawn far apart.

Finally, if the designer draws a widget or graphical object of which SILK has no knowledge, the designer can group the primitive components in question so that they are not used in an alternative inference later. Currently, SILK has no way of learning the new widget. Thoughts about how SILK might learn to recognize new widgets are given in Section 3.7.

3.6.4 Reinferencing Problems

There are a few user interface problems with respect to SILK's inferencing mechanism. First, the designer is unable to edit a widget without first breaking it up, losing the inference, making the desired changes, and then forcing the system to reinfer. This could be solved with a mode that would allow edits inside of a widget and then either perform an automatic reinference or force the user to ask for new inferences.

A related, but more severe problem is the lack of feedback and the inability to change a lower level inference when the system uses that inference to make a higher level inference. The example in Figure 3-27 illustrates this problem. The designer drew the primitive components in the order shown: a radio button followed by a check box to the right of it. Unfortunately, the system may mistake the rectangle for a circle and thus the check box for a radio button. This will cause SILK to immediately group the second radio button together with the first and form a radio button panel. The designer is given no chance to correct the mistake since the system will now only offer alternative inferences for the group containing the two buttons (and there are no alternatives).

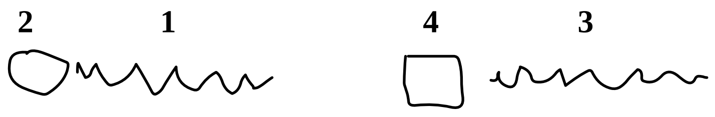


Figure 3-27. Radio button and check box potentially mistaken for a radio button panel.

To fix this mistake, the designer currently needs to break-up the panel using either **`Ungroup`** or by selecting **`Multiple_Objects`** from the panel of possible widget types. Then, the incorrect type can be changed from `radio button` to `check box` by pressing the **`New Guess`** button. A better solution would be to always show the primitive type of the last primitive drawn. Thus, the designer could change it and cause a new widget inference. Another alternative would be to allow **`New Guess`** to change the last primitive inference when there are no other available widget inferences.

3.7 Learning New Rules: A Proposal

The main problem with SILK's widget inference algorithm is the hard-coded nature of the rules. This limits the flexibility of the system in terms of recognizing new ways to draw known widgets and supporting new user-defined widgets. This section proposes three ways to try to learn new widgets during regular use of the system. All three techniques assume that the user would select the primitive components in question, give the widget a name (which may be the same as an already known widget), and then instruct the system to learn a new widget definition or modify an existing one.

3.7.1 Learning Spatial Definitions

The first technique involves trying to learn the spatial relationships found between the primitive components in a widget using the same algorithms that are used to find these relationships when recognizing widgets. These relationships can then be used to add new rules to the rule system. For example, SILK could learn the vertical scroll bar test shown in Figure 3-19 by noticing that one rectangle contained the other and that the containing rectangle was skinny. Since the example may fulfill several graphical relationships and features, it may be good to allow the user to indicate which of these relationships should be used for recognition (as is done in the Electronic Cocktail Napkin [Gross 1994; Gross 1996]). For example, in the scroll bar shown in Figure 3-2, SILK notices that in addition to the containment relationship, the two rectangles are a horizontal sequence (based on their centers). This fact is unimportant to any rule the system should learn from this example.

The SILK user study (see Chapter 6, "Evaluation") illustrated several situations in which SILK's built-in rules could not identify a widget that could easily have been recognized by learning spatial relationships. One participant tried to sketch a text field that had the field label above the type-in box (see Figure 3-28). The SILK rule for text fields requires the field label to be to the left of the type-in box (see Figure 3-29). Allowing the user to redefine this rule using the new spatial relationships would allow SILK to recognize this non-standard configuration while still supporting the desired behavior. A similar situation was encountered by two of the first three participants who tried to sketch a button by surrounding a text squiggle with a circle instead of a rectangle. The hard-coded rule for buttons did not allow this and was changed to prevent this problem from occurring with later participants. Again, this situation could have been avoided by allowing the designer to add a new rule using spatial relationships.

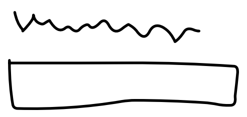


Figure 3-28. Text field that is *not* recognized by SILK.

```
(and (text-p comp1)
     (rectangle-p comp2)
     (near-p comp1 comp2)
     (aligned-p comp1 comp2 :bottom)
     (horizontal-left-p comp1 comp2))
```

Figure 3-29. Test for text field rule.

There are several difficulties in getting this type of learning algorithm to work. First, there is a problem of rule conflict. Learning a new rule could cause an older rule to become ambiguous. Imagine there were no rules for text field and the rule for check box was: `text near rectangle`. That works fine until the user attempts to add a text box. It will be misrecognized as a check box and the user will try to add a new rule. The system may give text box the same rule as check box or even more specifically: `text left of aligned rectangle`. This could cause the system to get confused when trying to recognize these two widgets. One solution is to point out such rule conflicts and force the user to specify more features to disambiguate the two cases. A related problem is making sure that there are enough built-in spatial relationships and graphical features to cover most cases, but not too many, as that will complicate the interface for picking out the important ones.

Another problem with trying to learn spatial definitions is in assigning the rule confidence values. As was described previously, these values are currently hard-coded and their assignment becomes complex as the number of recognized widgets and widget variations grow. An automated way of learning these values would be preferable to hand-picking them.

3.7.2 Bayesian Belief Networks

Bayesian belief networks [Pearl 1988; Charniak 1991] may offer an automated way to learn better confidence values for SILK's rule system. This method of reasoning uses probabilities to try to infer concepts from uncertain information. By noticing how a designer corrects the widget inferences made by SILK, the system could adjust its probabilities and infer concepts better in the future. In addition, Bayesian networks would also offer a way to have the "probability" values returned by Rubine's algorithm percolate up into the rule system. Currently, the probabilities are only used to see if the gesture is recognized above some basic cut-off value, and then they are thrown away. If these values were retained, they could offer more evidence in determining if one inference is more likely than another.

3.7.3 Neural Networks

The final technique to consider in learning widgets by example would be neural networks. The advantage of these is that the network determines the important features to distinguish the objects being classified, rather than the user or programmer. There are two major problems with using neural networks in SILK. The first is figuring out which level of data to train the networks on: the pixels making up a widget, the Rubine-style features of the objects making up a widget, or something at an even higher level.

A related and even more important problem is the large number of examples necessary to train a neural network successfully. Neural networks often require thousands of examples to classify new examples well. This can be reduced by picking the right features to train on. For example, with SILK, the Rubine-style features for each primitive in the widget may be a good choice to experiment with. The examples can then be systematically varied by size in order to multiply the number of examples the trainer sees. This technique could be successful for learning the set of built-in widgets by collecting examples from many designers, but it may prove impossible to make it work well when a single designer is trying to extend the system.

3.7.4 Analogous Widgets

The three techniques described above would allow SILK to recognize new widgets, though the system would know nothing about the widget's behavior unless the designer demonstrated it using storyboards (see Chapter 4, "Storyboarding"). Another way this problem can be solved is by allowing designers to point out the parts of new widgets that are analogous to existing widgets. The system could then try to infer the widget behavior from the differences and similarities between the old and new widgets. This proposed technique is discussed in more detail in Section 8.1.1.

3.7.5 Explicit Type Specification

It is possible that any widget recognition algorithm may be too error-prone. Automatic recognition of widgets may hurt the design process more than it helps by forcing the designer to continually pay attention to and correct the recognizer. If widget recognition cannot be improved using the learning methods described above, a more explicit widget type or behavioral specification technique may be worth looking into (as described in Section 8.1.2).

3.8 Summary

This chapter presented a widget recognition algorithm that operates in three separate stages. First, single-stroke primitive components are recognized by a gesture recognizer using Rubine's algorithm. Second, a spatial relationship detector looks for relationships between the newly recognized primitive and other components in the sketch. Finally, these relationships are passed to a rule system that tries to combine related components and form new widgets. SILK attaches the pre-defined interactive behavior of its best guess to the sketchy components. Since sketched input is inherently ambiguous, several possibilities may be proposed to the user so they can select an alternative when SILK makes an incorrect inference. This chapter also proposed several methods for learning how to recognize new types of widgets.

CHAPTER 4

Storyboarding

Rough sketches of interface screens are often tied together by simple storyboarding techniques. As in their traditional use in film and animation [Thomas 1981], storyboards are used to illustrate important sequences of the design artifact. A designer will illustrate sequences of system responses to end-user actions by annotating the sketches to indicate these relationships. Figure 4-1 shows a simple sketched storyboard. The storyboard illustrates that the rectangle in the drawing window should be rotated when the button at the bottom of the screen is clicked.

Designers often build up storyboards from their sketches by numbering the screens, drawing arrows on them, and attaching annotations. In this way a designer can describe the major transitions that occur between screens when a user manipulates the interface. A desire to iterate quickly leads designers to use hand drawn storyboards for this type of work. This technique has been shown to be a powerful tool for designers making concept sketches for early visualization [Boyarski 1994]. In fact, although not

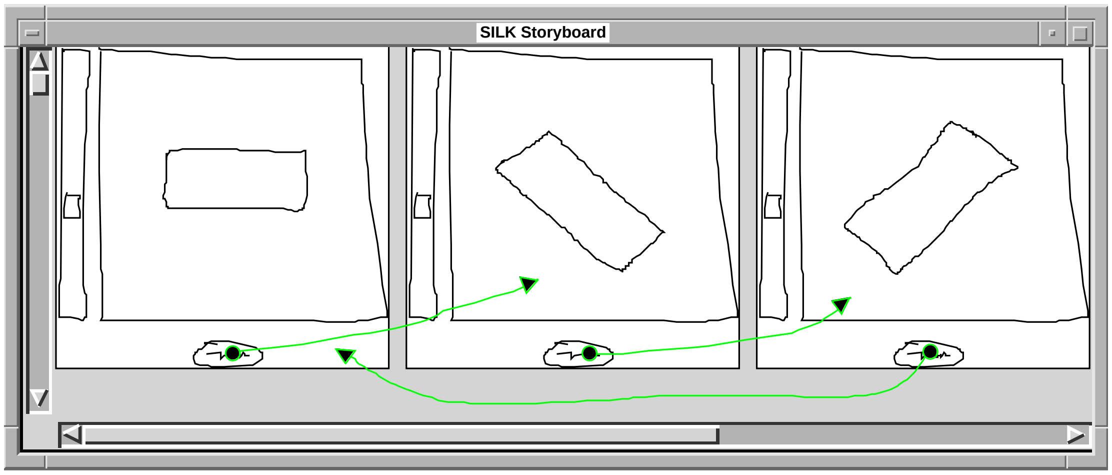


Figure 4-1. A storyboard that illustrates rotating a rectangle upon button presses.

specifically asked about storyboards, several of the designers I surveyed (see Chapter 2, “Informal Survey of Designers”) mentioned using storyboards during the early stages of user interface design. Storyboards are a natural representation, they are easy to edit, and they can easily be used to simulate functionality without worrying about how to implement it. In addition, the success of HyperCard has demonstrated that a significant amount of behavior can be constructed just by transitioning to different screens upon button presses.

SILK storyboards allow the specification of the dynamic behavior between widgets and the basic behavior of new widgets or application-specific objects. Having a dialog box appear when a button is pressed is an example of the dynamic behavior between widgets. A designer can illustrate these interface behaviors while the interfaces are still in their rough early stages. The main advantage over paper sketches is that SILK allows the storyboards to come alive and permits the designer or test subjects to exercise the interface in this early, sketchy state. In *Run* mode, buttons and other widgets are active (i.e., they give feedback when clicked), but without storyboards they cannot perform any actions.

This chapter describes how SILK can be used by interface designers to illustrate sequencing behaviors. It first describes how difficult it is to illustrate behavior with existing tools. Next, it describes the design and implementation of SILK’s storyboarding mechanism. The chapter concludes by exploring some potential extensions to the system.

4.1 Drawbacks of Existing Tools

When it comes to supporting interaction, existing tools fall short of the ideal. Most user interface builders, such as the NeXT Interface Builder and Visual Basic, require the use of programming languages in order to specify any interaction beyond that of individual widgets. Prototyping tools such as Director and HyperCard allow the sequencing of screens, and although they use direct-manipulation methods to specify these sequences, these methods lack the fluidity of paper-based storyboarding. To review or edit the screen transitions and for anything but the most simple sequences, these tools require the designer to traverse a series of complex dialog boxes or use a scripting language. Requiring the use of programming or scripting languages is not realistic for SILK’s target application and audience. Therefore, SILK allows the rapid illustration of a significant amount of interaction by sketching alone.

Due to the lack of good interactive tools, many designers use low-fidelity prototyping techniques [Rettig 1994]. One of the biggest drawbacks to using low-fidelity prototypes is the lack of interaction possible between the paper-based design and a user, who may be one of the designers at this stage. In contrast, SILK performs screen transitions automatically in response to a user's actions. This allows more realistic testing of rough interface ideas. In addition, rather than being thrown out like paper prototypes, these electronic specifications could possibly be used to automatically generate code that implements the transitions in the final system (see Section 4.7.2).

4.2 Visual Language

A simple model was chosen for SILK's storyboarding language. This visual language has two types of objects, screens and arrows. Each screen is a sketch of an interface in a particular state. For example, Figure 4-1 illustrates three screens that differ in only the orientation of the rectangle in the drawing window. Arrows connect objects contained in one screen with a second screen. The arrow indicates that when the object in the first screen is manipulated (the current model is limited to mouse clicks), the system should display the second screen instead of the first. For example, the arrows in Figure 4-1 indicate that when the user clicks on the button at the bottom of the screen, the user should see the rectangle in its new (rotated) orientation.

Using a notation of sketchy marks that are made on the interface sketches is beneficial since these marks are similar to the types of notations that one might make on a whiteboard or a piece of paper when designing an interface. In addition, the same visual language is used for both the specification of the behavior and the static representation that can be later viewed and edited by the designer. This static representation is natural and the SILK usability test showed that it is easy to use (see Chapter 6, "Evaluation"), unlike the hidden and textual representations used by other systems, such as HyperCard and Visual Basic.

4.3 Examples

SILK storyboards make it easy to illustrate several common interface behaviors. This section presents several examples. In Figure 4-1, the designer has illustrated a repeating sequence of rectangle rotations. Each time the button is clicked, the rectangle in the drawing window rotates 60 degrees. This example also shows that the transitions can loop back to the screen they started on. An important point about this example is that it shows that a designer's knowledge and creativity allows the illustration of a behavior (i.e., rotation) that the underlying tools, SILK and the Garnet toolkit it uses, do not even support.

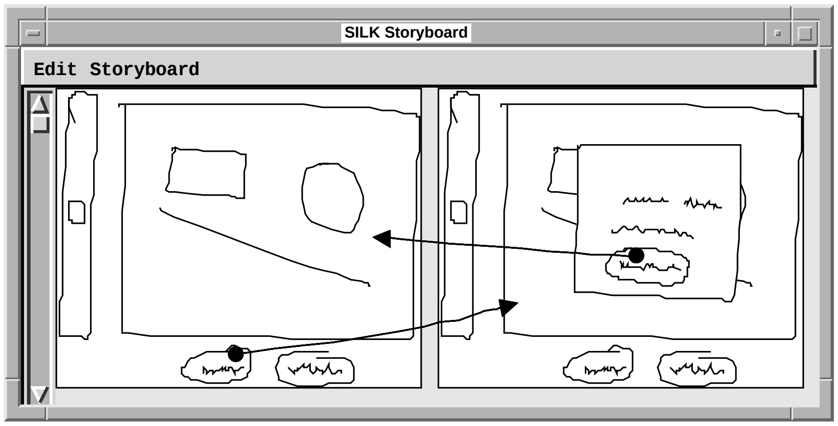


Figure 4-2. Make a dialog box appear and disappear when the appropriate button is pressed.

Figure 4-2 illustrates bringing up a dialog box on top of the sketched drawing window. This example is interesting since the designer is able make the dialog box opaque so that it hides any objects it may appear over. This is accomplished by first grouping the objects making up the dialog box and then selecting **`Make Opaque`** from the **`Arrange`** menu in the **`SILK Controls`** window[1]. This technique can also be used for illustrating pop-up menus. An alternative design I considered was to have SILK recognize these types of widgets and then set this property automatically.

Figure 4-3 illustrates a sequence in which the user can select a circle by clicking on it. The circle can then either be doubled in size or shrunk in half by clicking on the buttons at the bottom of the screen. This example also illustrates the feedback of selection handles. It is important to remember that this example does not *specify* how selection or scaling should work in general, but instead *illustrates* how it might operate.

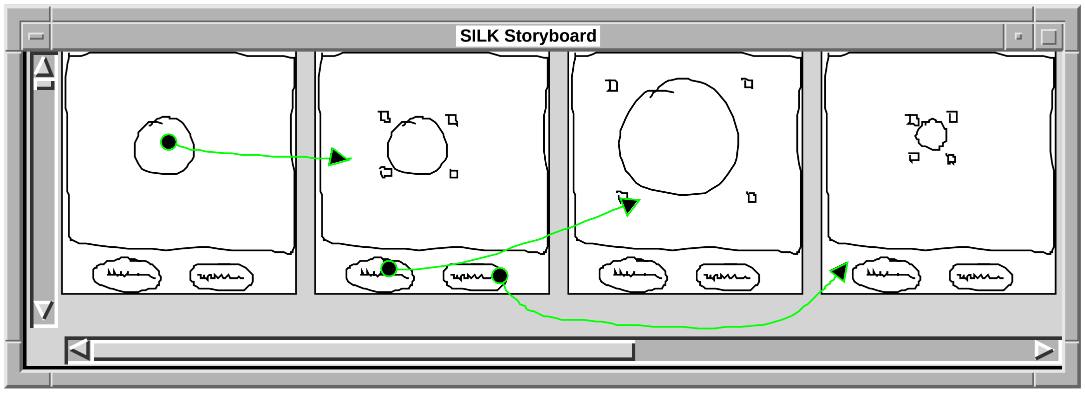


Figure 4-3. Scaling a circle with selection handles.

1. `This` is currently unimplemented. The example was created by changing properties of the dialog box using the Garnet debugger.

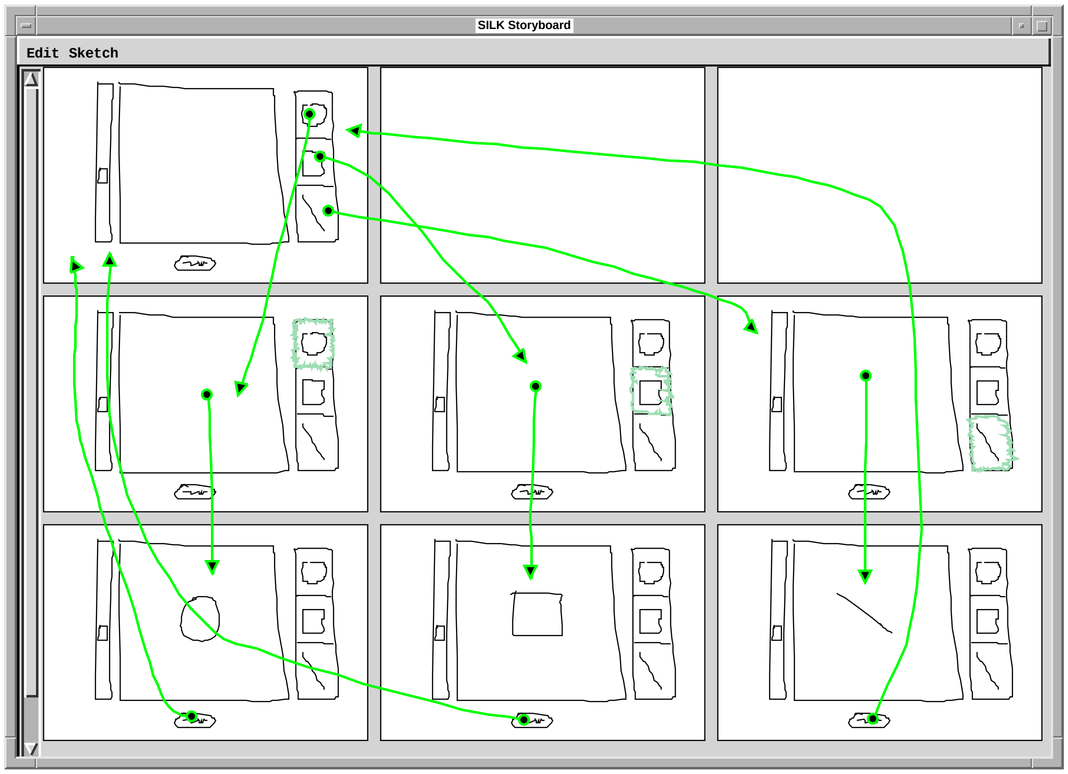


Figure 4-4. A partial screen graph for a simple drawing tool. Clicking on a palette item changes the state and then clicking on the background creates an object of the correct type. The bottom button deletes the object.

Finally, Figure 4-4 shows how a designer could illustrate the operation of an arbitrary palette of tools. In this example the user is able to create any of three objects in a drawing window by first clicking on the object in the palette and then clicking in the drawing window. The button at the bottom of the window allows the user to delete the newly created object. Note that although these examples show the arrows leaving widgets, transition arrows can start from any graphical object or even the background.

4.4 Screen Graphs

SILK's storyboarding model implies that a program can be thought of as a graph (see Figure 4-4). The nodes of the graph are the different states of the program (i.e., screens) and the arcs out of each node represent the end-user actions that cause state changes. To fully specify a program, the designer would have to specify the entire graph. However, this is not a major drawback of the model, since storyboarding is generally used for illustrating important sequences in an interface, rather than for specifying an entire interface. Section 4.7 contains a discussion of several techniques which vastly reduce the amount of work needed to specify more complete screen graphs.

The important sequences in an interface can be thought of as partial paths through the screen graph of an application. For example, the middle path in Figure 4-4 illustrates the sequence in which the user creates a rectangle and then deletes it by clicking on the delete button. By illustrating a few partial paths, the designer can specify enough of an interface for the design team to quickly consider several possible interface ideas.

4.5 Storyboard User Interface

The system was designed to make specifying sequencing paths easy. The designer constructs storyboards by sketching screens in the **`SILK Sketch`** window as described in Chapter 1. Screens can then be copied to the **`SILK Storyboard`** window using a menu command. At this point, the original screen can be modified in the sketch window to show how its state might change. After this, the new screen is also copied to the storyboard window. Now, the designer can start drawing arrows in the storyboard window that indicate screen sequencing, or more screens can be produced.

The arrows can be drawn from any widget, graphical object (e.g., primitives and decorations), or the background, to another screen. Thus, the designer can cause transitions to occur when the user clicks and releases the mouse button on any of these items. The transition is aborted if the mouse-up occurs anywhere outside of the object. Each arrow shows an anchor point on the object it was drawn from and an arrowhead on the screen it was drawn to. Unlike the arrows in many visual dataflow languages [Pictorius 1995] and the wires in CAD tools, SILK's storyboarding arrows are free-form. This unconstrained control permits the designer to avoid some of the "rats-nest" problems associated with these other systems, where lines cross at 90 degree angles and are thus hard to follow.

4.5.1 Editing Storyboards

SILK offers several operations to help in the editing of the screens and transitions specified in the storyboard. Clicking on a screen or an arrow highlights it in red and then the selected screen or arrow can be deleted. Screens can also be cut, copied, or pasted in the storyboard window. The entire storyboard can also be cleared. In addition, SILK automatically maintains the transitions and thus the legality of their semantics when screens are removed or new arrows are added.

Removing a screen can allow arrows to come from or go to a non-existent screen. Therefore, when a screen is deleted all arrows coming into or out of that screen are also deleted. In addition, removing a screen causes all of the screens after it to move in the left to right, top to bottom, screen layout. Thus, SILK redraws all arrows to or from the affected screens using straight lines.

Adding an arrow may violate the rule that only one arrow is allowed to leave a particular object. When this case arises, the old arrow is automatically deleted after the new arrow is drawn. Maintaining the original arrow until after the new one is drawn helps when the designer is replacing the straight line arrows generated by SILK.

The SILK usability test (see Section 6.2) showed that designers would like more control when editing the storyboard. For example, designers would like to position the screens wherever they like, rather than in the sequential order that SILK uses. They also wanted a tighter coupling between the **`Storyboard`** and **`Sketch`** windows.

4.5.2 Testing the Interaction

When the designer is ready to test the specified interaction, she can switch to *Run* mode. At this point, the designer must specify which screen will be the initial screen to start the interaction. This is accomplished by selecting a screen in the storyboard window and copying it back to the sketch window. Now the designer or an end-user can start interacting with the sketch and it will make the proper transitions as defined by the visual program displayed in the storyboard window. Each time the user clicks on an object that has the anchor of an arrow attached to it, the system will replace the current screen with the screen attached to the arrowhead. For example, the behavior illustrated by Figure 4-1 will show a progression of rectangle rotations when the user clicks on the button in the sketch window.

4.5.3 Feedback

In order to allow the designer to debug their storyboards, we have supplied some feedback mechanisms that are displayed while in *Run* mode. First, the currently active screen (i.e., the one being displayed in the sketch window), is always highlighted in the storyboard window. This allows the designer to know the current state of the system. Second, the object that caused the last transition to the current screen is highlighted along with the arrow leading to the current screen. A designer can use these mechanisms to help check that her visual program is working properly. Finally, widgets in the sketch window may supply their own feedback. For example, a button highlights while the mouse is held down over it.

4.6 Implementation

SILK's storyboarding model maps into an implementation that is also quite straightforward. Every sketched widget and screen is given a unique ID. When a screen is copied from the sketch window to the storyboard window, the objects are copied along with their IDs.

When the system is put into *Run* mode, a screen transition table is built by examining the arrows in the storyboard along with the objects and screens they connect. For each arrow the system creates a transition entry that contains the object's ID along with the origination and destination screen IDs. When the user clicks on an object in the sketch window, the system checks whether that object has a transition defined on it by looking it up by its ID and the ID of the current screen. If a transition is defined, the system copies the screen specified by the destination ID to the sketch window. Thus a program executes upon each input event by examining the transition table and copying the specified screen.

This implementation could be made more efficient (and exhibit less screen flicker) by checking for differences (deltas) between the two screens and storing in the transition entry the necessary operators to effect the change. Instead of the outright replacement of objects that change, the transition could modify "interesting" object parameters, such as: visibility, left, top, width, height, and scale. Then when a transition fires, the system would only change parts of the screen rather than make an entirely new copy. The interfaces created by designers during the SILK usability test (see Appendix "SILK Evaluation Data") exhibit many common objects among the screens, thus supporting the use of a delta transition model.

4.7 Extending Storyboards

The current model of storyboard sequencing is sufficient for many applications, but falls short when the designer wishes to specify a more complete interface. There are several ways the basic model can be extended to increase the power of this specification style.

4.7.1 Parallel Storyboards

Currently the system supports multiple paths through an interface screen graph (as seen in Figure 4-4), but only one of these paths can be followed at a time. An obvious extension of this is to allow multiple states (i.e., screens) to be active at the same time, but only one in each *independent* path. An independent path is a sequence of screens that cannot be reached from another independent path by following the drawn transition arrows forwards or backwards. I term these independent paths, combined with the previously discussed difference checking algorithm, *parallel storyboards*.

Parallel storyboards would allow partial updates to the current screen in the sketch window by combining the *differences* specified by multiple storyboard screens. Thus, the designer could describe separate paths for manipulating different objects, displaying the results of the distinct manipulations in the sketch window at the same time. For example, the operation of the palette objects in Figure 4-4 could be described by the two paths given in Figure 4-5. Then, an end-user would be able to produce the screen shown in Figure 4-6 by creating a rectangle followed by a circle or vice-versa.

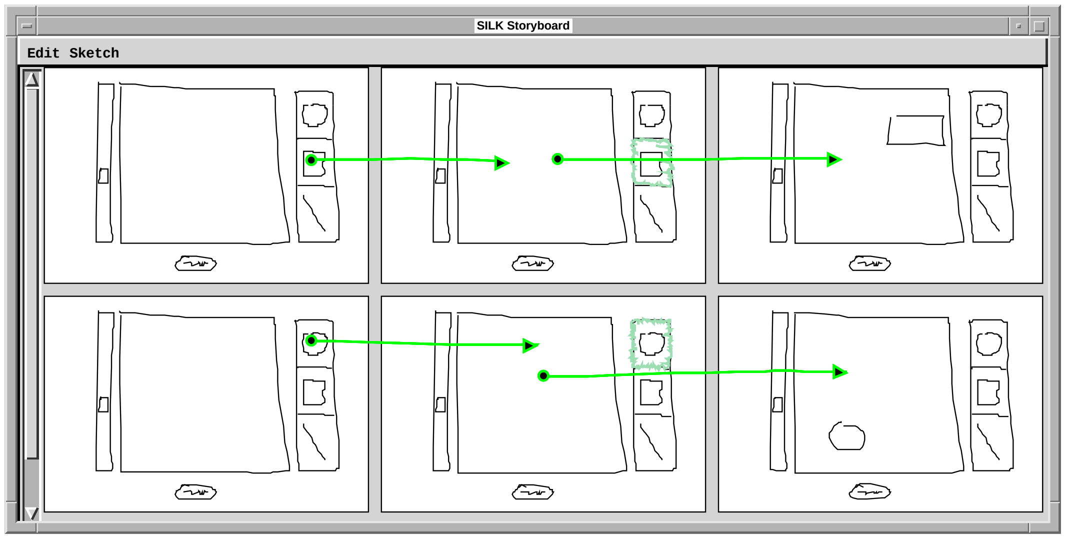


Figure 4-5. Parallel storyboards that allow the creation of circles and rectangles in any order.

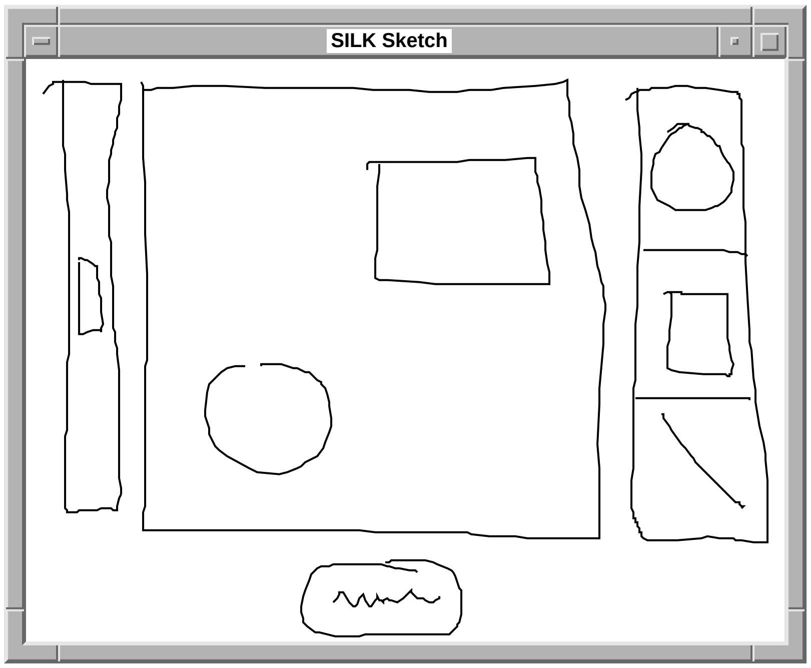


Figure 4-6. A drawing that could be created with the parallel storyboards defined in Figure 4-5.

In the current storyboarding model, the size of the specification grows exponentially. Parallel storyboards allow the size of the specification to grow as the product of the number of *active* objects (i.e., objects that start transitions) and the number of supported operations on them. This is a significant improvement. In the example illustrated in Figure 4-5 there are two active objects (the rectangle and the circle) and one operation (create object).

This design for parallel storyboards is not quite correct – it permits ambiguity. In the example illustrated in Figure 4-5, which object should be created if the user first clicks on the rectangle palette item, then the circle item, and then in the drawing area? To avoid this, the designer needs a way to indicate that the middle screens are a shared state that can be active in only one path at a time. Figure 4-7 shows that grouping the screens is one way this might be indicated. In addition, there would need to be a rule to resolve the conflict when it occurs. For example, when a second path enters the grouped states at run time, make the path that reached the state first retreat one state, undoing any screen changes it made. Now the example works as expected. When the user clicks on the circle palette item, the rectangle selection feedback will be undone to indicate that only the circle is selected.

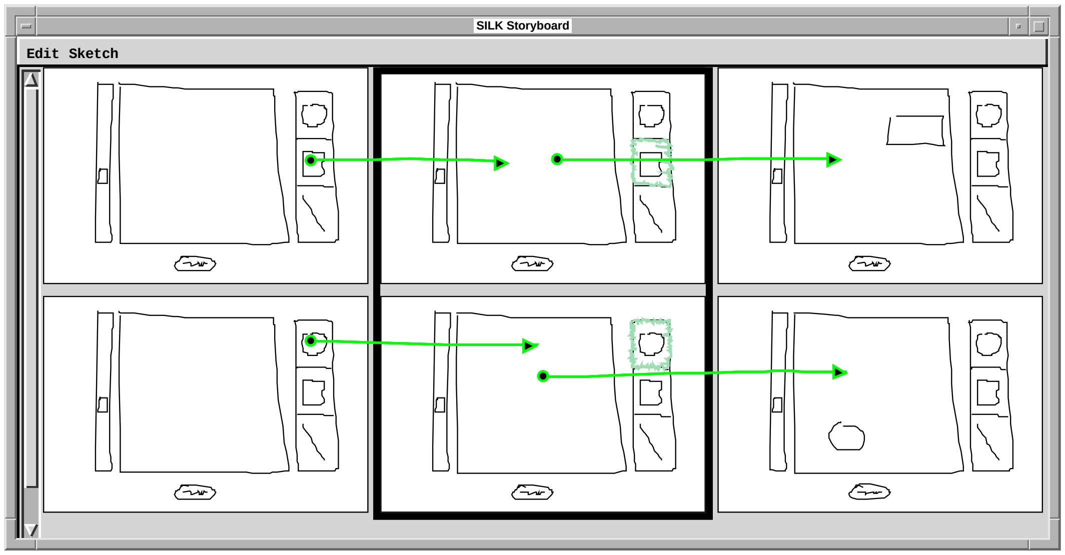


Figure 4-7. One way to remove the ambiguity in Figure 4-5 – shared state that has two arrows leaving on the same event. (Note: the grouped state feedback was added programmatically.)

4.7.2 Inferencing Techniques and Code Generation

Another way in which to make storyboards more expressive is by applying inferencing techniques to them. Programming-by-demonstration (PBD) is a technique in which one specifies a program by directly operating the user interface [Myers 1992; Cypher 1993]. The system then tries to infer a program to implement the interaction. SILK's storyboarding system is similar to a PBD system. In the sketch window the designer specifies the layout and structure of the interface, while in the storyboard window she demonstrates possible end-user actions and shows how the layout and structure should change in response.

In the current system there is no inferencing involved. All actions and responses must be specified by the designer. Consider the rotation case illustrated in Figure 4-1. A PBD system could take the two rotation steps shown and try to infer the amount of rotation to apply on any subsequent button press. The PBD system could then indicate this inference by replacing the screens in question by one compound screen in which the inference is made explicit. Double clicking on the compound screen would display the original sequence as drawn. This would save both the designer's time and lots of valuable screen space.

Another application of inferencing is to allow the system to infer that operations applied to one type of object may also be applied to other objects. For example, in order to support scaling for both rectangles and circles the designer currently needs to specify two separate sequences operating on both types of objects. If the system could infer that scaling is simply the modification of a parameter of the selected object, then one example sequence would allow scaling on all objects. Again, this would save a considerable amount of designer time and storyboard space.

A critical problem with PBD techniques is the lack of a static representation that can be later edited. Marquise [Myers 1993] and Smallstar [Halbert 1984] use a textual language (a formal programming language in the latter case) to give the user feedback about the system's inferences. In addition, scripts in these languages can then be edited by the user to change the "program." This solution is not acceptable considering that the intended users of SILK are user interface designers who generally do not have programming experience. I believe that SILK's visual notation can be extended to show the inferences that a PBD system might make. For example, simple graphical attributes could be displayed on top of the storyboard screens to indicate iteration and conditionals, as was done in Pursuit [Modugno 1995].

It may also be useful to defer much of the inferencing until transformation time, thus preserving the fluidity of the sketching and brainstorming phases. At that point, PBD might also be used to help construct a call-back skeleton from the transitions.

This skeleton could become much more useful if the inference engine was able to generalize operations, as in the previously described scaling example.

4.7.3 Storyboard Space Saving Techniques

One of the major problems with visual languages is the large amount of screen space they require as compared to textual languages. Besides PBD, there are several other space-saving techniques that may help overcome this problem.

The first major problem encountered by designers is the “rat’s nest” of arrows connecting screens. As an interface gets more and more complex, these arrows become hard to follow. This problem is solved in many CAD systems by performing automatic routing algorithms. This would be especially useful after moving or deleting screens in the storyboard. Another solution is to allow multiple views of the storyboard window. For example, a designer should be able to specify that she wishes to only see arrows coming into a particular screen, out of a particular screen, or out of a particular object. Another view might only show screens that are reachable from the current screen or selected arrow. User-controlled routing and editing of free-form arrows, along with multiple views, may be enough to solve many arrow related problems.

The second major problem involves the size of the storyboarding panels. Currently when the designer copies a screen to the storyboard window, it is displayed at 40% of its original size. It may be useful to allow the designer to vary this parameter for more control over the space. It may also be nice to automatically vary the scale among different panels according to which part of the interface the designer is working on. This technique is similar to the use of fisheye views in visualization tools [Furnas 1986].

Another technique for saving storyboarding space is to only show relevant changes to the screens, instead of the entire screens. In the rotation example (see Figure 4-1), the second storyboard screen might only show the rotated rectangle without the palette, window, or button. This is similar to some of the techniques used in Chimera [Kurlander 1993] and systems based on graphical rewrite rules. Another way to conserve storyboard space is to try to compress multiple changes into a single screen. The Pursuit [Modugno 1995] system uses such techniques. Something similar could be implemented in SILK if arrows were drawn not just to the next screen, but to the object that should be modified by the specified action. Thus, multiple arrows could arrive on distinct objects in the same screen to illustrates multiple state changes.

4.7.4 Additional Events, Animation, and Conditionals

In addition to making the storyboards more expressive and compact, there are some simple additions that can be made to support more forms of interaction. The only event the current system permits is clicking on widgets or graphical objects. SILK could allow more complete user interfaces by supporting dragging, drag and drop, double clicking, timer events, and typing. In addition, a null event arrow that would match on any user event that did not match on the current screen would make error handling easier. Allowing arrows to be annotated with event types would allow some of these additional events. One way to support this is to have a palette of arrow types in the storyboard window (a mouse icon for single click, two mice for double click, a clock for timer events, etc.) The palette would allow the designer to select the current mode for the arrows drawn in the storyboard window. Arrows of different types would be distinguished by distinct colors and possibly line styles.

The timer event is especially interesting in that it would allow designers to quickly mock-up multimedia applications that have animation or video by specifying a few key frames that SILK would automatically transition between when the timer event occurred. A number of professional designers who have seen SILK requested this feature. A simple property sheet would allow changing the time-out on individual arrows or the default time-out given by the clock icon.

Besides supporting these new event types, the current implementation of SILK should be extended to support better default behaviors for two of the existing widgets, scrolling windows and text fields. Scrolling windows should scroll objects in the window by some default amount. Adding the previously described drag event would permit a custom definition of scrolling using a scroll bar along with the associated window and objects. Text fields should allow entering text in the field box by default. Adding typing events would also allow a custom definition of this behavior.

Another limitation of the current model is that it only supports checking that an event takes place on a particular object. It would be useful to have a transition that occurs conditionally on the object in question being in a particular state. For example, the designer may want a mouse click on a check box to only cause a transition if the box was previously unchecked.

4.8 Summary

Questionnaires and site visits show that hand-drawn storyboards are a common tool used to illustrate behavior. Unlike paper sketches, SILK's electronic storyboards allow the designer or test subjects to interact with the sketch before it becomes a finalized interface. The current storyboarding model is quite simple and therefore easy to pick up and use. If more expressive power is necessary, there are several changes discussed in this chapter that could be added, but at a cost of more complexity in the SILK user interface.

CHAPTER 5
Support for Later Development Stages

Although SILK was designed to support the early conceptual stages of user interface design, a successful tool cannot ignore the later phases through which a design will progress. SILK supports the prototyping and implementation stages with two technical features: design memory mechanisms for managing different versions of a design along with the associated documentation, and screen transformations for passing the design off to another tool. This chapter contains a description of these features along with a discussion of proposed extensions that can better support the entire user interface design cycle.

5.1 Design Memory Mechanisms

One of the important features of SILK is its support for design memory. *Design memory* refers to any mechanism that allows the designer to manage the documentation about the design together with the different versions of the design. Practicing designers have found that the annotations on design sketches serve as a diary of the design process, which are often more valuable to the client than the sketches themselves [Boyarski 1994]. Such documentation may include the reasons why a particular design decision was made or possibly who made a particular design change. SILK supports making and saving this documentation via an annotation layer.

The designer can annotate a design by either typing or sketching on the annotation layer (see Figure 5-1). The annotations are rendered in blue and the layer can be displayed or hidden with the click of a button in the **`SILK Controls`** window (see Figure 1-17 on page 17). Annotations made on paper can be difficult to store, organize, search, and reuse. This is not the case with SILK's electronic annotations. For instance, annotations that were made using the keyboard can be searched later using a simple search dialog box (see Figure 5-2).

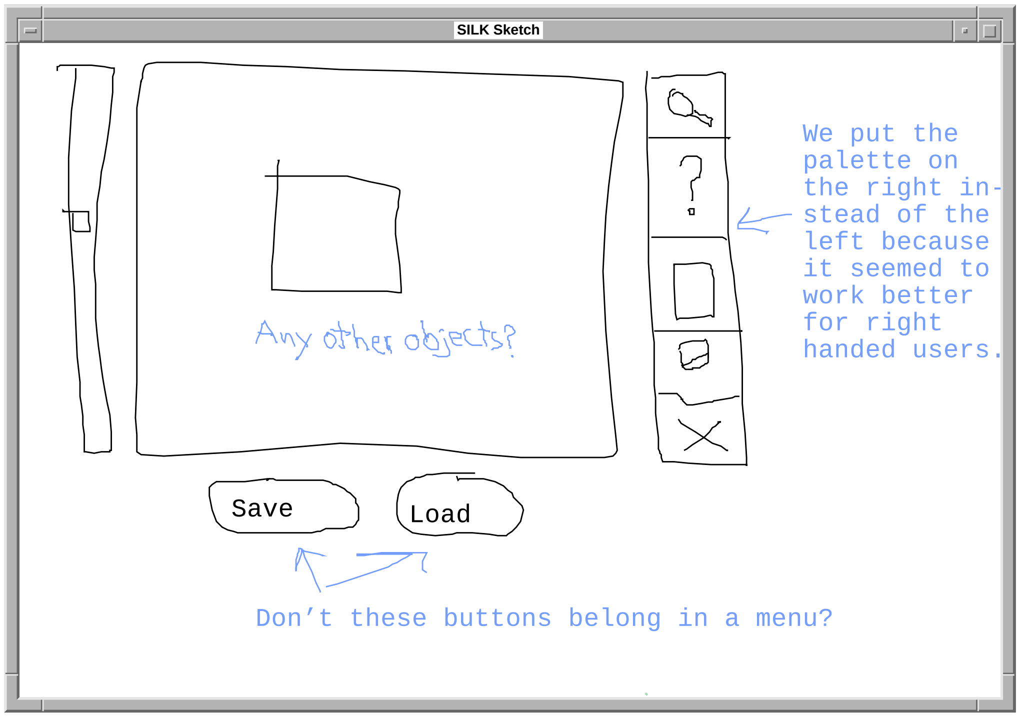
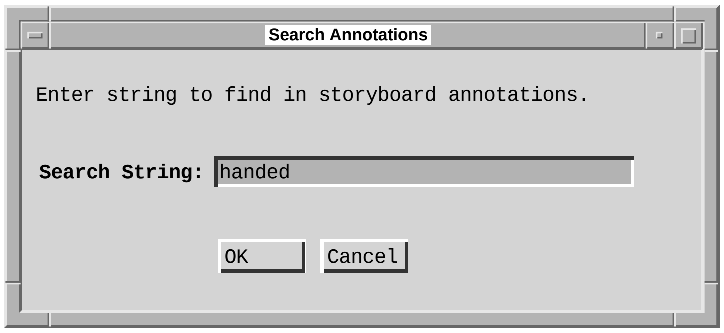

SILK Sketch
Any other objects?
Save
Load
We put the palette on the right instead of the left because it seemed to work better for right handed users.
Don't these buttons belong in a menu?

Figure 5-1. The annotation layer is displayed on top of the sketch.



Enter string to find in storyboard annotations.

Search String: handed

Figure 5-2. Dialog box for searching annotations on storyboard screens.

As in *Sketch* mode, SILK's *Annotate* mode supports the full set of editing gestures and menu commands. None of the marks drawn on the annotation layer are interpreted, though widgets that have been interpreted in the sketch can be pasted to the annotation layer. This feature can be used to document widget changes in the annotation layer by including a copy of the original object.

SILK also allows a designer to save designs or portions of designs for later use or review. The designer can reuse screens by first visually scanning the miniaturized screens in the storyboard window to find the proper screen and then copying it to the cut buffer. Then, she can open the new design and paste the screen into the new storyboard. By saving multiple versions of the same design in the one storyboard, the designer can also compare these *thumbnails* on screen.

5.1.1 Design Memory and Design Management Extensions

Since design is often a group task, SILK could easily be extended to support multiple layers, allowing different members of the design team to create personal annotations. This feature could be used to view team members' comments as the design progresses and to also review their comments at a later date.

Design management could easily be improved. First, the system could be extended to display multiple designs from separate design files (e.g., separate storyboards) at the same time. This way a designer could make side-by-side comparisons or more easily copy portions of one design into a new design. Second, SILK could display several designs in a miniaturized format that is representative of the different screens in each design so that the designer could quickly search through a directory of previously saved designs visually rather than purely by name. Another promising search method would be to support visual queries. That is, the designer could sketch a screen and query the system for any designs that contain similar screens. Such a mechanism is supported for architectural drawings in the Electronic Cocktail Napkin [Gross 1994; Gross 1996]. Finally, a more powerful versioning system that supported check-in, check-out, display of differences between versions, and roll-back, as with Unix's rcs application, would greatly enhance SILK usefulness for production quality work.

It might also be important to let designers view a graphical history of how a design progressed, both within a version and between versions. One way to do this is to use tracing layers [Wong 1993] to view different versions of screens on top of each other. Another way is to keep a history of stroke order and allow animations of the changes [Genau 1995].

5.2 Screen Transformations

In addition to supporting the management of SILK-based designs it is also important to support transferring those designs to another tool at the appropriate time in the design cycle. When the designer is happy with an interface screen, selecting **`Transform`** from the **`Recognition`** menu causes SILK to create a new window, the **`SILK Finished`** window, and to generate real widgets and graphical objects in this window. These widgets can take on the look-and-feel of a standard graphical user interface, such as Motif, Windows, or Macintosh. The current version of SILK generates Motif widgets that can be copied and then pasted into Gilt [Myers 1991], the Garnet user interface builder. Figure 5-3 illustrates the transformed version of the screen illustrated in Figure 5-1.

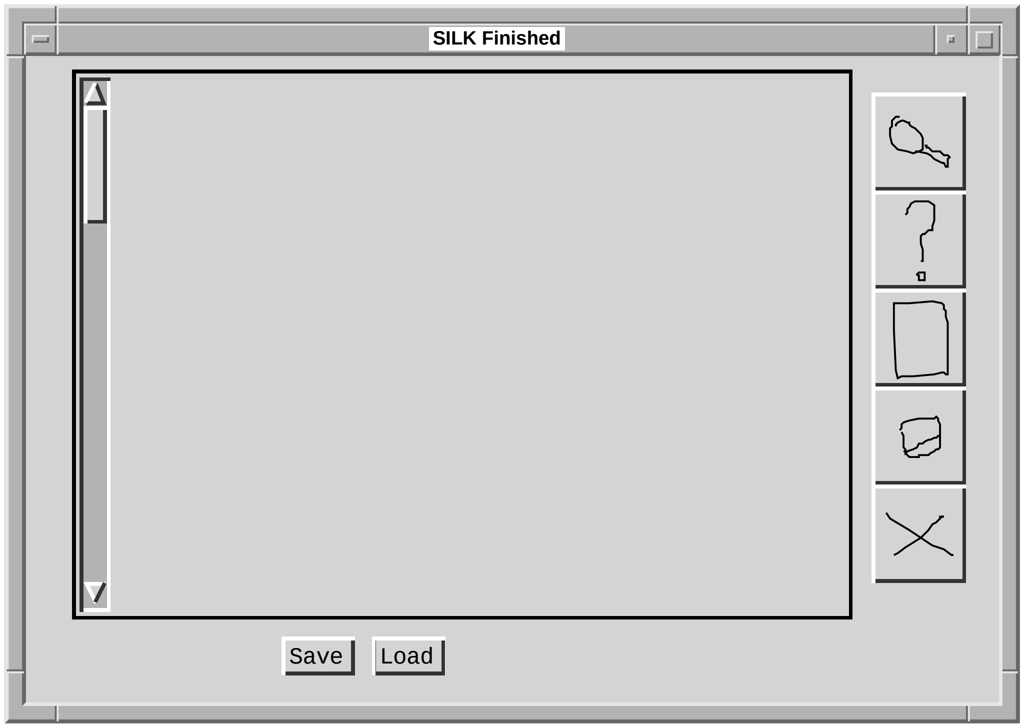


Figure 5-3. The Motif transformation of the sketched screen illustrated in Figure 5-1.

The widgets and objects are positioned in the same locations that they occupied in the original sketch. Compound widgets (i.e., pulldown menus, panels of buttons, check boxes, and radio buttons) are spaced evenly and treated as one object. Palette objects, as well as objects not making up a widget, are transferred in their existing polyline form. An interface screen that supports all of the widgets recognized by SILK (see Figure 5-4) has been transformed by SILK in Figure 5-5.

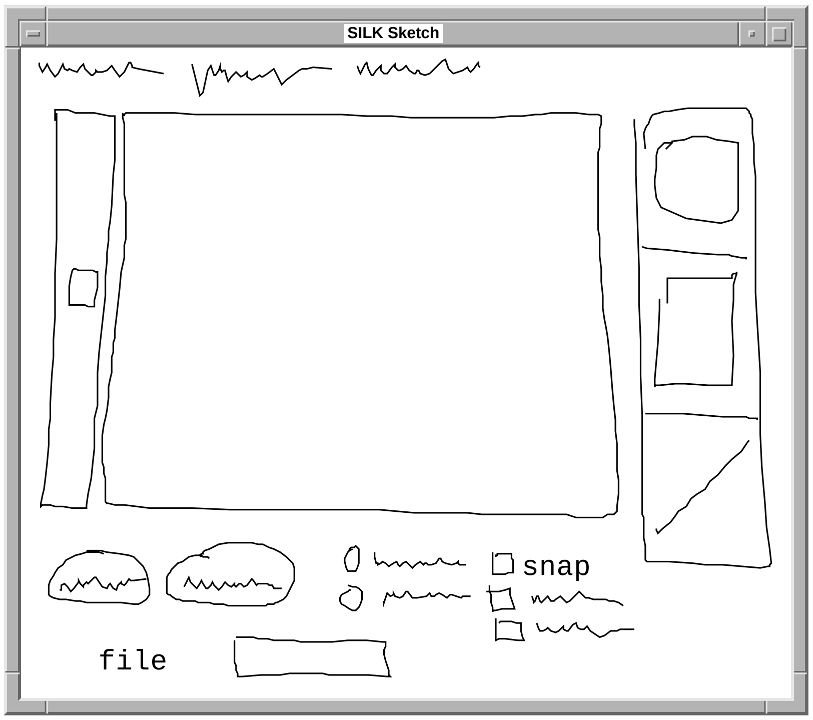


Figure 5-4. A screen that contains all the widgets recognized by SILK.

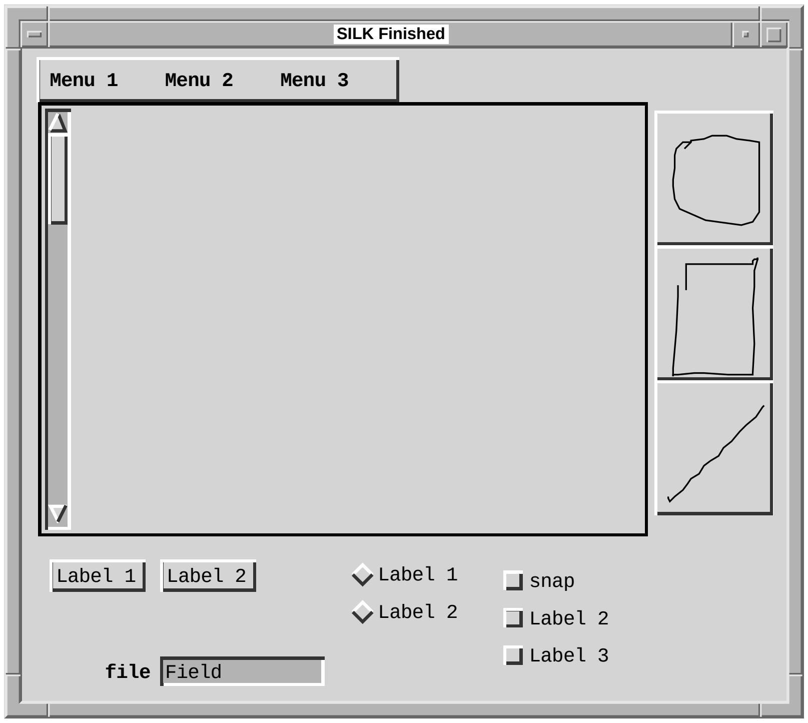


Figure 5-5. The transformation of the sketch illustrated in Figure 5-4.

The transformation process is mostly automated, but it requires some guidance by the designer to finalize the details of the interface (e.g., textual labels, colors, and alignment). If the labels were replaced in the sketch with ASCII text (using the **`caret`** gesture), this text will be transferred to the new widgets, otherwise they are transferred as generic labels. After saving the screens into a Gilt design, programmers can add callbacks that include the application-specific code to complete the application.

5.2.1 Transformation Extensions

One obvious extension to the current transformation algorithm is to transform all of the screens in the storyboard, rather than just the one that is displayed in the **`SILK Sketch`** window. In addition, rather than requiring the designer to copy and paste these transformed screens into Gilt, the system should generate the proper Gilt files automatically. These transformed files should also retain any annotations the designer has made on the design.

Since the types of the widgets are known at transformation time, the process is straightforward and is implemented as a large case statement. Instead of generating Garnet objects, this code could easily be modified to generate alternative UI widget descriptions. For instance, SILK could generate OSF's UIL [Ferguson 1993], Microsoft's Visual Basic, HTML (see Chapter 8, "Future Work"), or any other format.

The hard part about doing the screen transformation correctly comes about when the design team tries to go back and forth between the two formats. For example, the team could first use SILK, next generate Visual Basic (VB), and then using VB add new widgets and associated application code. If at that point the design team decides they need to make some changes at the sketch-based level, the new changes will be lost. One way to solve this problem is to use auxiliary data structures to keep track of new objects and code so that they can be attached later when SILK retransforms the interface. This would require modifying the interface building tool (in this case VB) or parsing the differences between the original SILK-generated VB file and the newer modified file. SILK could then use generic sketchy representations of the new VB-generated widgets. It would also be nice for SILK to use any application code that has been written. This last feature makes the problem much more difficult to solve.

Finally, the one area where the transformation process could be significantly extended is in trying to make use of the transitions that were drawn on the storyboard. Since the transitions only simulate how a program might be implemented, directly using the arrows to generate call-backs is a fairly ambiguous process. Generating such a code skeleton is a challenging area of research. The arrows could be used to generate code if the target language was HTML, which has a one-to-one mapping with SILK's transition model (see Section 8.3.1). Section 8.1.3.4 describes how the system might use the transition arrows as a demonstration that can be generalized to generate code.

5.3 Summary

Besides allowing designers to create mock-ups of user interfaces, SILK has some support for the rest of the design process. In particular, SILK supports making annotations on the sketches in order to document the design. In addition, screen transformations allow designers to move their SILK sketches to a more appropriate tool, i.e., a user interface builder, when the design ideas become more concrete.

CHAPTER 6
Evaluation

Although there is substantial evidence that an electronic sketching and storyboarding tool would be useful for early user interface design, this thesis could only be confirmed by building a system and testing it with actual designers. This chapter describes such an evaluation of SILK. The chapter starts with a description of the usability test and then presents both numerical and descriptive results. This is followed by an overview of the engineering changes made as a result of participants using the system. The chapter finishes with a discussion of the results, which confirm the thesis and show that designers see applications where SILK, and in particular electronic storyboarding, would be useful in their work.

6.1 Description of Usability Test

This section describes the objectives of the usability test, the methodology of the study, how participants were selected, the design tasks used, and finally what was measured.

6.1.1 Objectives

There were three main objectives of the usability test. The first and most important objective was to see whether designers could effectively use SILK to design user interfaces. The second objective was to see whether a tool like SILK permits designers to communicate a design idea to other members of a design team, particularly engineers. Finally, I wanted to see how well SILK performed, both in terms of its recognition algorithms and its user interface.

6.1.1.1 Design Effectiveness

The main questions to answer in terms of design effectiveness are whether designers can use SILK to produce non-trivial designs and whether SILK supports creativity. *Non-trivial* designs tend to use several screens and offer interactive behavior. Supporting *creativity*, i.e., the exploration of a wide variety of design ideas, is very important in the early stages of design. SILK will be considered to support creativity if designers are able to work on more than one idea during the session, rather than fixating on a single design and committing all of their available resources to it. Previous work has found that electronic tools encourage designers to fixate on a single design rather than exploring several ideas at the start of a design project [Black 1990; Goel 1995].

6.1.1.2 Communication Effectiveness

One of the important tasks in the early stages of design is to communicate a design idea to other members of the design team, particularly engineers. Electronic sketching and storyboards will be considered effective for communicating design ideas if engineers are able to understand a design idea after a short discussion, if these discussions concentrate on the structure and behavior of the interface, rather than the "look," or if designers are able to make design changes in real-time as a result of these discussions. An engineer will be said to show *understanding* of a design idea if they ask critical questions that are more than just reactions to what is drawn on the screen.

6.1.1.3 System Performance

System performance is a question of how well my implementation of electronic sketching works. I wanted to find any problems with the SILK user interface so that I could refine it. I also wanted to see whether designers could understand what SILK was doing, how easy the system was to learn, and how often the system performed the correct operation from the user's point of view. SILK will be considered *easy to learn* if designers are able to use the features that were taught to them in the SILK Tutorial. *Correct operation* is a question of how well the editing gesture, primitive component, and widget recognition algorithms work.

6.1.2 Task Environment and Materials

The operational conditions, user population, and task domain used were meant to represent a typical design environment. In particular, the tasks were performed in a room with two computers and a moderate noise level. The users were chosen to be non-novices with respect to user interface design (see Section 6.1.4), and the chosen task (see Section 6.1.5) was meant to be typical of the design problems a designer may need to solve when designing a user interface.

The participants used an HP Snake Workstation (HP-735) with 92 MB of RAM. The workstation ran Unix (HP-UX) and X Windows. The color monitor had a 17" diagonal effective viewing area. The computer used a round, three button, mouse that comes standard with this workstation. The system was loaded with Allegro Common Lisp and SILK running on top of the Lisp system. In addition, a video camera and VCR with digital time stamping capability were used to record the sessions. SILK was already running when the participant started the tasks, thus eliminating the need for the participant to learn how to start the tools.

6.1.3 Method

The method of evaluation was to have each designer perform a set of design tasks. I was the experimenter for the test. The test session was targeted to last around 4 hours. No time limit on the experiment was enforced, except to encourage wrap-up of the post-task engineering discussion.

For each participant, the session began by having the participant sign consent forms (see Appendix Section B.1). Next, the experimenter read aloud some basic information about the evaluation (see Appendix Section B.2). The participant was then asked to supply some basic demographic data and information about their previous use of interface design tools (see Appendix Section B.3).

The session was broken up into four major components: a short demonstration of SILK (the script for this demonstration is included in Appendix Section B.5), a practice design task that served as a tutorial for the system, a second design task from which measurements were taken, and a post-design discussion with an engineer.

The practice task was actually a written tutorial with quick reference manual (see Appendix D, “SILK Tutorial and Reference”) that led the designer through an example design task to familiarize them with SILK before the evaluation task. The practice task was not counted in the test results, although the amount of time the tutorial took to complete was recorded. The participants took a short break after the tutorial.

All of the participants were then given an identical design problem (see Section 6.1.5). After finishing this task, an “engineer” who had read a similar task description (see Appendix Section B.8) was brought in. The engineers were all CMU School of Computer Science doctoral students who did not know the designers beforehand. The designer was then asked to describe and demonstrate their designs, and the two were encouraged to discuss the designs and possibly make changes.

Afterwards, the participants were given a short questionnaire to fill out (see Appendix Section B.10 and B.11). The questionnaire, some of which is based on [Ravden 1989], tried to gauge whether the user liked the design methodology, whether the method would be useful in practice, and also collected any other comments the designers might have had. Finally, the participants were given an information sheet that explained the goals of the experiment (see Appendix Section B.12).

In order to make sure that this test plan was appropriate, three pilot sessions were run (using computer science students rather than design students). A few changes to the study were made as a result of these pilots (e.g., which variables to measure) and several bugs in SILK were fixed. The data from these pilot runs has not been counted in the results.

6.1.4 Participants

The study used design participants who were intermediate or advanced user interface designers. The criteria for deciding on the level of experience was based on whether the participant had either been a practicing professional interface designer for more than one year (advanced) or if they had taken at least one interface design course that required a significant amount of work creating actual interface designs (intermediate).

If the participant was a novice with respect to user interface design, they would have been rejected from the study. No participants met this novice design criteria. Participants who lacked computer experience (i.e., were not familiar with using a computer, keyboard, or mouse) would also have been rejected from the study. Again, no participants met this computer novice criteria.

The participants were chosen from among the students and staff at Carnegie Mellon University. The participants were found by asking members of the community to volunteer. The design participants were given $25 for their time. In addition, a prize of $100 was offered for the participant who designed the best interface as chosen by a CMU Human-Computer Interaction Institute faculty member. The engineering participants were recruited by posting to the School of Computer Science electronic bulletin board (`cmu.cs.scs`) and they were given $8 for their time. The rest of this section reviews the demographic information supplied by both the designers and the engineers who participated in the post-design evaluation.

6.1.4.1 Designer Demographics

Table 6-1 summarizes the information supplied by the designers[1] at the start of the session. The participants were evenly split along gender lines and had an average age of 30 years. Four of the six were considered advanced interface designers and two were

1. Note that the six design participants are numbered from 7 to 17 skipping evens. This is because each session used two participants (a designer and an engineer) and there were three earlier pilot experiments.

Designer Demographics

participant	7	9	11	13	15	17	mean	median	% yes
age	21	29	41	33	21	33	29.7	31.0	
sex	Male	Female	Female	Female	Male	Male			50% male
graduate program?	no	yes	yes	yes	no	N/A			60%
major	industrial design	HCI - interaction design	communication, planning, & design	HCI - interaction design	HCI	design			
years professional UI design experience	0.25	1.25	0	2	2.5	5	1.8	1.6	
advanced?	no	yes	no	yes	yes	yes			67%
taken UI design project courses?	yes	yes	yes	yes	yes	no			83%
UI tools used:									
HyperCard?	no	no	no	no	yes	yes			33%
Director?	yes	yes	yes	yes	yes	yes			100%
Visual BASIC?	yes	no	no	no	yes	no			33%
HTML?	no	no	no	no	yes	yes			33%
C++ or Java?	no	no	no	yes	yes	no			33%
used drawing/painting programs?	yes	yes	yes	yes	yes	yes			100%
read HCI books?	yes	yes	no	yes	yes	yes			83%

Table 6-1. Demographic information and previous design experience of design participants.

considered intermediate designers. Participant 17 was a practicing interface designer with a degree in design, while the remaining five participants were design students. Three of the students were enrolled in graduate design programs and the others were undergraduates taking an HCI or design major. All of the students had taken UI design courses that required a major design project as part of the course and most of the students also had some professional work experience performing interface design.

All of the participants claimed to have used Director for UI design, whereas only a third mentioned using each of HyperCard, Visual BASIC, HTML, or a programming language like C++ or Java. This seems consistent with the results of the informal design survey (see Section 2.2.2.1). All of the participants had used drawing programs (most commonly citing Photoshop and Illustrator). In addition, five of the six participants had read some HCI books. None of the design participants were familiar with SILK.

6.1.4.2 Engineer Demographics

Table 6-2 summarizes the demographic information supplied by the engineers. All but one were male and they had an average age of 27 years. They were all graduate students, four in computer science, one in robotics, and one in language technology. All but one had professional programming experience with an overall average of four years.

Engineer Demographics

participant	8	10	12	14	16	18	mean	median	% yes
age	26	33	25	27	25	25	26.8	25.5	
sex	Male	Female	Male	Male	Male	Male			83% male
graduate program?	yes	yes	yes	yes	yes	yes			100%
major	CS	robotics	language technology	CS	CS	CS			67% CS
years professional programming experience	4	9	0	2	2	7	4.0	3.0	
taken UI design project courses?	no	no	no	no	no	no			0%
UI tools used:									
HyperCard?	yes	no	no	no	no	no			17%
Director?	no	no	no	no	no	no			0%
Visual BASIC?	yes	no	yes	no	no	no			33%
Tcl/Tk?	yes	yes	no	no	no	no			33%
Other?	yes	yes	no	no	no	no			33%
read HCI books?	yes	no	no	no	no	no			17%

Table 6-2. Demographic information and previous design experience of engineering participants.

None of the engineers had taken a UI design course and only one had read an HCI book. Unlike the designers, none of the engineers had ever used Director and only one had used HyperCard. Two of the engineers had used Visual Basic and two had used Tcl/Tk [Ousterhout 1991]. Three of the engineers claimed to have never used a UI design or building tool.

6.1.5 Design Task

The task given to the designers was based on an interface design/evaluation problem given in Nielsen's *Usability Engineering* [Nielson 1993, pp. 273-275]. Two of the participants claimed to have read parts of this book, although neither had seen the exercise in the back. The participants were asked to design an interface to a weather information system designed for travelers. The system was to provide information about the weather for the current day and to give weather predictions for the next two days. The designers were told that the goal for the design task was to explore the possible design space and eventually present several good alternatives to the rest of their design team or a client (see Appendix Section B.7 for the exact task description).

6.1.6 Test Measures

In order to achieve the objectives given in Section 6.1.1, the measurements listed below were taken for each design session by observing the participant's actions, automatically logging events in SILK, video taping, and having the participants fill out a post-evaluation questionnaire (see Appendix Section B.10). The measures are listed under the objective they were meant to help determine.

Design Effectiveness:

- The number of different designs produced.
- The complexity of the designs (in terms of number of screens and transitions).
- The amount of finished vs. sketchy text in the final designs.
- Were the designs restricted to the built-in widgets?
- Was run mode used while designing the interface?
- What the designer thought about SILK as a design tool compared to other tools.
- What the designer thought were the best and worst aspects of SILK.
- The amount of time to complete the task, starting from when the task description was given to the designer and ending when the designer said they were finished.

Communication Effectiveness:

- Did the engineer ask critical questions in the post-design discussion that indicated an understanding of the design ideas?
- Did the design discussion include issues of structure and behavior?
- Did the designer make real-time changes to a design as a result of the discussion?
- Was the storyboard used for testing the interface during the discussion?
- Was the storyboard used to illustrate the overall structure during the discussion?

System Performance:

- The number of times SILK made a mistake, i.e., the system inferred a widget or gesture incorrectly. A more exact definition of this is given in Section 6.2.3.1.
- How well the designer thought the gesture recognition worked.
- Usability problems the designer had with SILK (both observed and reported).
- How long the designer took to work through the SILK tutorial.
- The designer's impression of SILK's overall performance.

As SILK is an experimental system, it occasionally crashed. For all time-based metrics, I subtracted the time it took to bring the system back up in the state it had been before the crash. The raw time data is included in Appendix Section C.3. Note that a verbal protocol was not used, but I recorded any comments/critical incidents the participants made while performing the tutorial and the design task.

6.2 Results

The results of the study are given in terms of the three major objectives discussed in Section 6.1.1. These results include numerical results (i.e., recognition accuracy and design and session characterizations)[2], descriptive observations[3], and the in-session and post-evaluation comments made by the participants[4]. These results show that SILK was an effective tool for interface design, the tool also provided an effective way of communicating design ideas to engineers, and the system was not too hard to learn, although its implementation and performance could be improved in several areas.

6.2.1 Design Effectiveness

The design sessions were successful at showing that SILK could be used effectively to design user interfaces. This section first characterizes the sketches produced during the sessions to illustrate that these designs are non-trivial and that the designers did not fixate on a single design idea. It then discusses the sessions themselves and describes how the designer's took advantage of SILK's features and what their reactions were to these features. Overall, the designers felt that SILK's "ability to prototype screen-based interactions is GREAT," but that the tool needed better support for visual effects (such as type and color) and more support for non-standard interactions. The section concludes with a summary of these missing design features.

6.2.1.1 Characterization of Designs Produced

The usability test produced information on the types of designs produced and how long it took to create them. This data is summarized in Table 6-3. The data shows that the designers each spent about 1.5 hours on the task and on average they came up with two different designs. The designs varied, both between designers and for an individual designer (although the designs for an individual designer often contained elements from a previous design iteration). Figure 6-1 illustrates two different designs by participant 7. Although vastly different, both designs use a table to display a multi-day weather forecast.

2. All of the numerical results refer to the design task, which was performed after completing the tutorial.
3. The observations are a result of notes I took during the experiment and after reviewing videotapes of the sessions. The observations cover all three phases of the experiment: the tutorial, the design task, and the post-design engineering review.
4. The complete set of answers to the post-evaluation questionnaire are reproduced in Appendix Section C.5.

Task and Design Characteristics									
participant	7	9	11	13	15	17	mean	median	stdev
task time	1:42	2:08	1:18	1:13	1:37	1:39	1:36	1:38	0:19
#designs	2	2	1	2	3	2	2.0	2.0	0.6
#screens	9	17	2	14	21	9	12.0	11.5	6.8
#transitions	20	38	0	30	81	15	30.7	25.0	27.9
screens / design	4.5	8.5	2.0	7.0	7.0	4.5	5.6	5.8	2.4
transitions / screen	2.2	2.2	0.0	2.1	3.9	1.7	2.0	2.2	1.2

Table 6-3. Task time and summary of characteristics of designs produced.

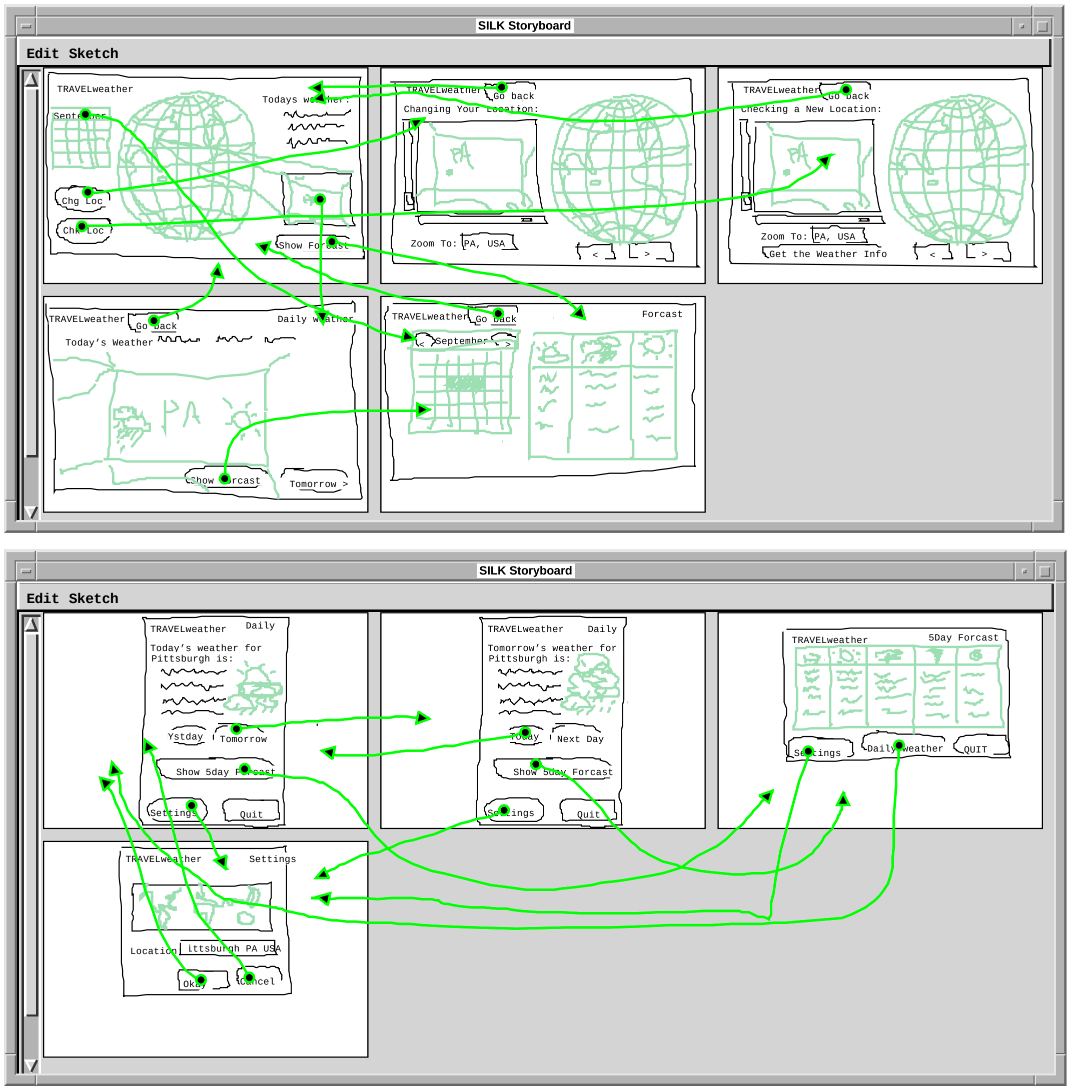


Figure 6-1. The two designs produced by participant 7 during the task.

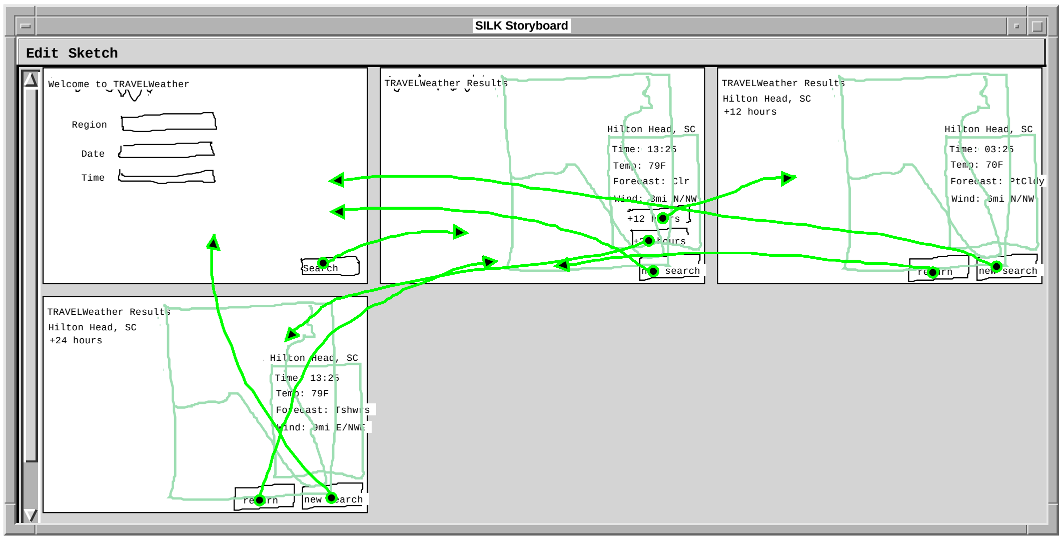


Figure 6-2. One of two designs produced by participant 13.

Figure 6-2 shows one design from participant 13. In contrast to participant 7's designs, this design uses a textual overlay on top of the map to display the weather conditions. All of the designs are shown in Appendix Section C.4. This appendix also includes the design changes made during the engineering discussions[5].

The designs used almost six different screens on average and there was an average of two transitions per screen. Two transitions per screen represents a relatively small amount of interaction per screen as compared to a real interface, but it is enough to support more than the simple linear screen flipping easily achieved with a tool like Director. The designers typically used the storyboards to illustrate a few important sequences. It is interesting to note that the tutorial used a similar amount of interaction: four screens with an average of two transitions per screen

Participant 9's designs illustrate that using a large number of screens is indeed manageable (see Figure 6-3). Participant 15's designs illustrate that too many transitions on each such screen becomes messy and hard to understand (see Figure 6-4). This particular design had so many transitions (almost four per screen) because the designer chose to perform error handling for cases where the end-user did not input a time and date or if the end-user chose geographical regions that had no weather data.

5. The numerical results presented in this chapter do not include the changes made during these discussions.

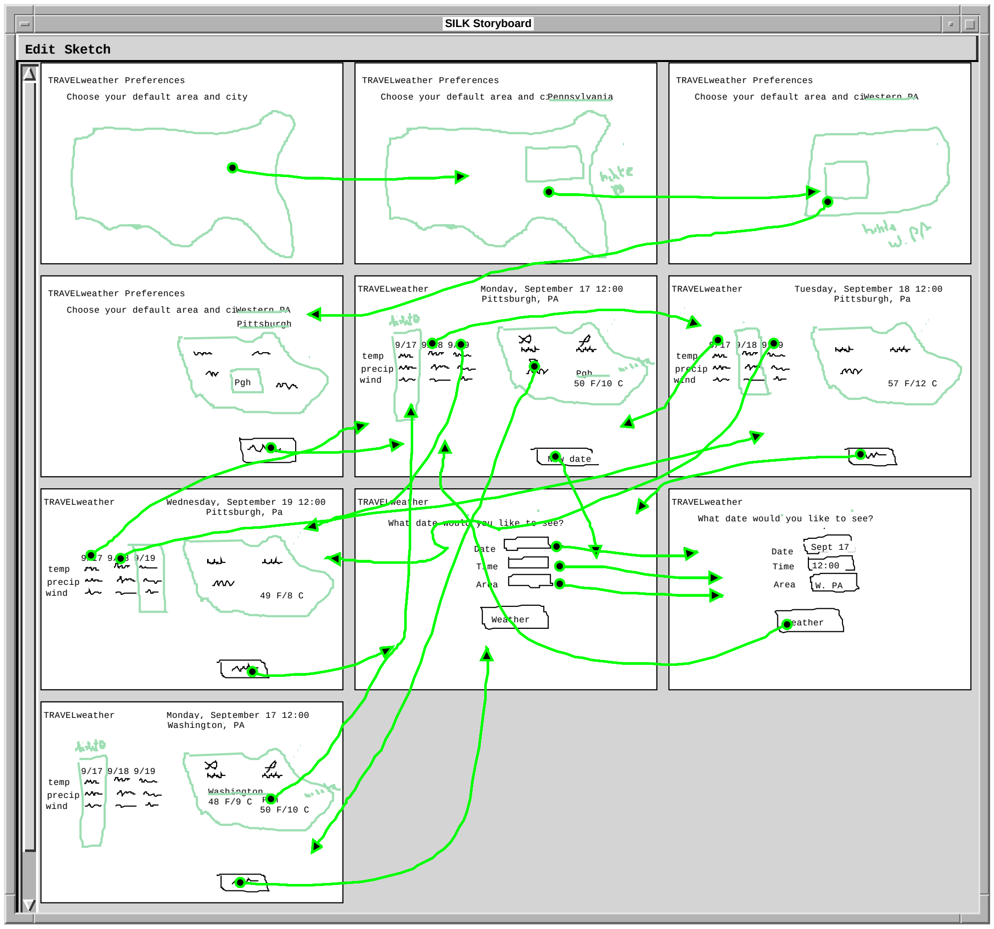

SILK Storyboard
Edit Sketch
TRAVELweather Preferences
Choose your default area and city
Choose your default area and cPennsylvania
Choose your default area and ciWestern PA
Pittsburgh
Pgh
TRAVELweather
Monday, September 17 12:00
Pittsburgh, PA
Tuesday, September 18 12:00
Wednesday, September 19 12:00
temp
precip
wind
50 F/10 C
57 F/12 C
49 F/8 C
What date would you like to see?
Date
Time
Area
Weather
Sept 17
12:00
W. PA
Monday, September 17 12:00
Washington, PA
48 F/9 C
Washington

Figure 6-3. Participant 9's second design shows that a large number of screens is manageable.

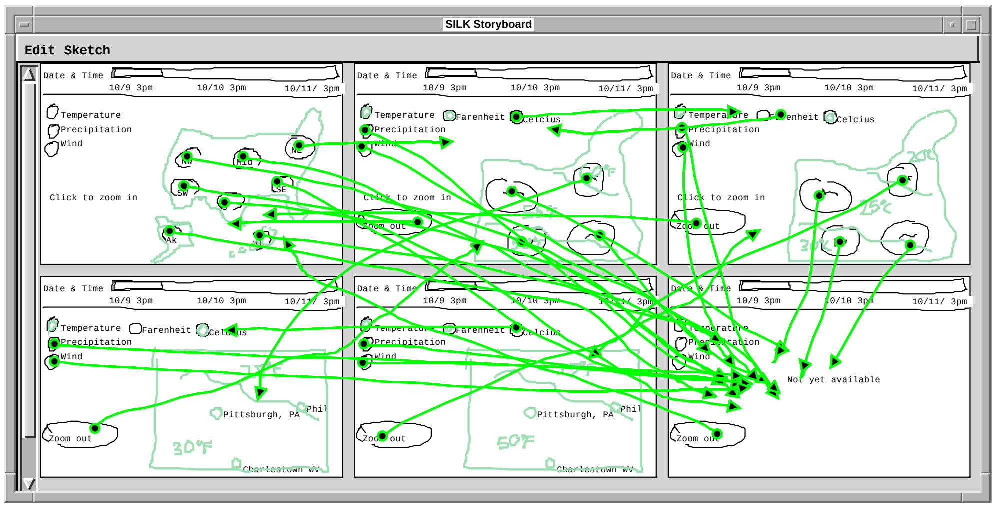


Figure 6-4. The second design by participant 15 shows that a large number of transitions per screen (in this case, almost 4 transitions/screen) can be hard to manage and understand.

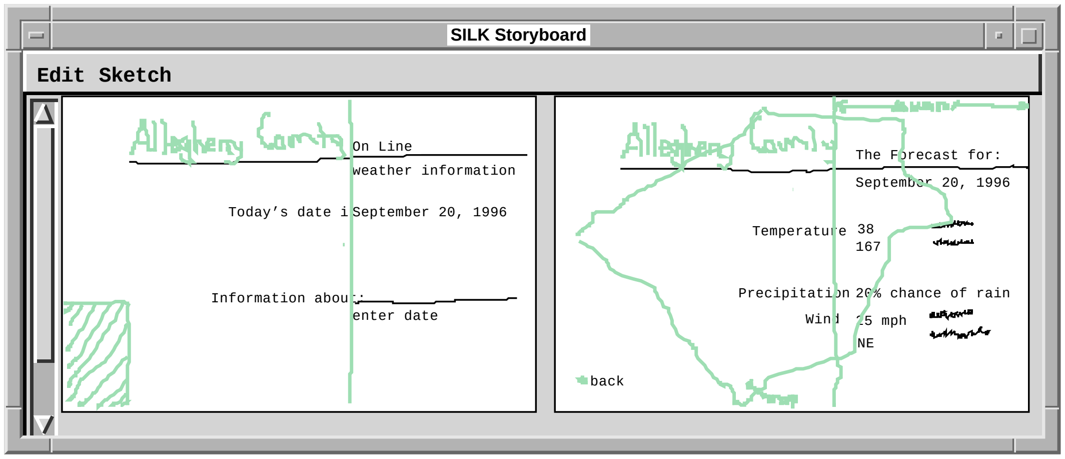


Figure 6-5. The single design produced by participant 11. Note the small number of screens and the lack of any transition arrows.

Note that participant 11, who had considerable difficulty with the SILK user interface during the tutorial, created only one design and chose to use only two screens, no interactive widgets, and no transitions between the screens (see Figure 6-5). I believe this was due to this designer's difficulty using a two-button mouse (she was used to a Macintosh) and to some problems in the interaction mechanism supplied by Garnet for moving objects (as described in Section 1.4), which requires dragging on a "drag" handle rather than on the object itself. These UI problems led to this designer's frustration and a poor understanding of SILK's storyboarding model.

Four of the six designers added interaction for objects other than the built-in widgets. This interaction was offered on custom interactive maps, tables of data, and iconic buttons, and is another sign that the designs produced are non-trivial in nature. This, together with the observation that the designs produced by each designer varied, confirms the hypothesis that SILK supports creativity in the early stages of design.

Design Rankings

The designs were ranked by a faculty member in the Human-Computer Interaction Institute at CMU. The following factors were considered: creativity, simplicity, functionality, and visual appeal. In terms of functionality, the judge looked for designs that supported simultaneous forecasts, a representation of the weather on the map, and a graphical location selection method. Table 6-4 shows the rankings for each design. The modified designs produced during the engineering discussions were not included in the judging. One interesting item to note is that for the five designers who produced more than one design, all but one produced a higher ranked design on their second attempt. The winning design, participant 13's second, is shown in Figure 6-6.

Design Rankings												
design (designer-version)	7-1	7-2	9-1	9-2	11-1	13-1	13-2	15-1	15-2	15-3	17-1	17-2
ranking	5	4	9	2	12	10	1	8	3	7	6	11

Table 6-4. Rankings of designs produced during the usability test (a ranking of 1 is best).

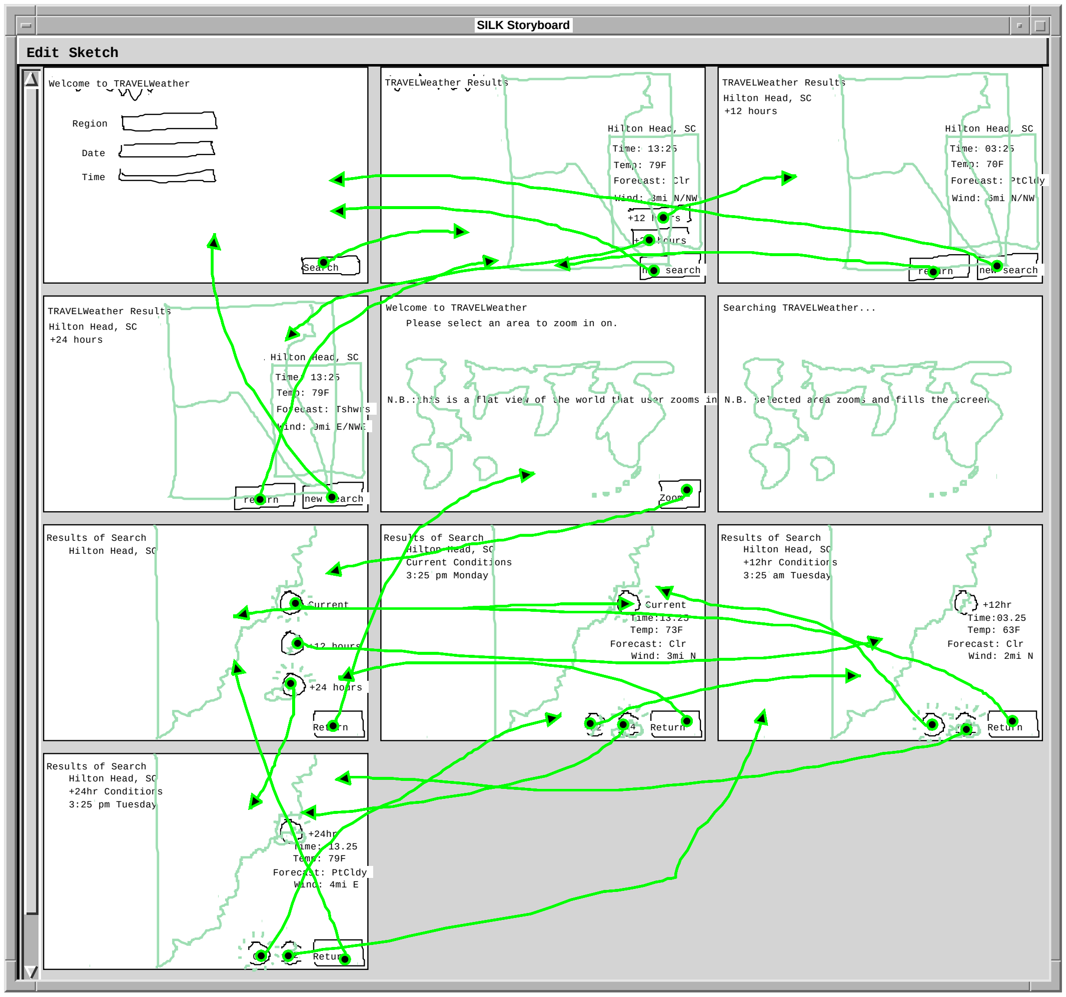


Figure 6-6. Participant 13's second design was judged to be the best of the twelve considered.

6.2.1.2 Characterization of Design Sessions

The designers took advantage of many of SILK's features to achieve this design creativity and effectiveness. In particular, the roughness of the medium successfully supported both creative brainstorming and iterative refinement. The designers also took advantage of the editing capabilities of the tool and the interaction provided by storyboards. SILK's automatic widget recognition was appreciated by some, but often got in the way. In response to this problem, the designers made a liberal use of *Decorate* mode to create both static decorations and non-standard interactive objects.

Creative Design and Iterative Refinement

Overall, the designers were fairly positive about SILK as a tool for early creative design. It was described as an "excellent 1st draft tool" that "would be good for novice interface designers" and "useful for quickly structuring information." A few of the designers still "would have liked using pencil and paper" but found that SILK was in many ways similar to paper. One designer said she "saw missing pieces [in the design] as I was working, which is the same way it works using paper. That's good!"

One important observation of the experiment was that the participants tended to leave parts of the interface in an especially rough and ambiguous state. One example of this ambiguity was that the designers did not spend much effort trying to align objects exactly. They would put things roughly where they wanted them and move on with the design. In addition, text was often left in its "squiggly" form or handwritten. These objects were left rough until the end or when more details became known. In fact, all but one of the designers had some squiggles or handwritten text in all of their final designs. Approximately 29% of the text strings in the final designs were either squiggles or handwritten text (see Table C-4 on page 175). I observed that in most cases the typed text started out as a squiggle that was later replaced with the finished text. This is a good example of iterative refinement.

The designers confirmed these observations when asked to discuss the features of SILK that made it better than HyperCard and Director. Three of the six designers said they liked the roughness or paper-like sketching that SILK provides. They appreciated the "ability to be fast and sloppy" and the "way it remains sketchy." Another said that the SILK "is great for giving the idea of a progression through a program without getting into the details of the visual design. Often with tools like Director the high level of visual detail misleads people to thinking more about the visual refinement rather than the interaction." These observations and comments confirm the hypothesis that an electronic sketching tool leads designers to focus on the overall interaction and structure rather than on the detailed look-and-feel.

Editing

A second key observation was that most of the participants used cut, copy, and paste in order to reuse widgets or portions of screens. In addition, they used these editing capabilities to revise individual designs as they went. For example, participant 9 changed a button panel to a radio button panel in the middle of a design. This editing capability is something that users expect from any computer-based tool, but is not easy to do with paper. These editing features are another win for electronic sketches over paper-based sketches.

Interaction and Structural Overview

Another important hypothesized advantage of SILK over paper is the interaction it provides for testing the design. One designer wrote that SILK was "as quick as paper sketching and provides a basis for interaction." All but one of the designers took advantage of this interaction by using transitions in their storyboards. Generally, the designers took advantage of *Run* mode to test their design towards the end of their work on a particular design. After running it, they would often notice that they had either left out necessary transitions, objects, or screens. They would then make the necessary changes and test it again. One designer said she "liked the ease with which you could test the interaction – it's a very tight loop." Another commented, "the storyboard was nice – being able to draw arrows to indicate links was very fast." The designers appreciated that the "storyboard view... is not only sequential (like movies)."

Storyboards also have the advantage of permitting designers to illustrate behavior that the underlying tools do not directly support. Participants 9 and 15 confirmed this advantage by repeatedly using *Decorate* mode to draw their own state feedback on radio buttons. Although SILK offers feedback for these widgets, the state of the feedback is not saved between screen transitions. These designers had no problem seeing how to use the storyboard transition model to implement the correct behavior. Participant 9 also implemented feedback for selecting columns from a table (i.e., she created a new interactive widget).

Besides easily illustrating behavior in the storyboard, the designers liked the ability to "see and edit the storyboard" and noted that a "view of information the users need can be explored quickly" with SILK. Even participant 11, who had extreme difficulties, saw the importance of "frame to frame" sequencing. Another wrote, "better than Director – the linking with drawings rather than Lingo is excellent. Also you can see right away what is going on."

This interaction and navigation via storyboards were the features of SILK that the designers kept referring to in their spoken and written comments. In fact, five out of the six designers mentioned storyboarding as the advantage of SILK over Director and HyperCard.

Widget Recognition

Although all but one of the designers used SILK's built-in widgets in their designs, the recognition accuracy was low enough to be a hindrance. One designer felt that the "widget detection and sketching allows for fast low level interaction," but I observed that many of the participants became frustrated when faced with repeated recognition errors. Section 6.2.3.1 discusses the recognition performance in more detail.

Decorate Mode

It was interesting to see which features the designers used that were not covered in the tutorial. For instance, the designers made liberal use of *Decorate* mode for static objects, such as the maps and weather information commonly sketched during the task. Although *Decorate* mode was shown in the pre-tutorial demonstration, it was not used in the tutorial. Almost all of the designers started using it on their own. This could indicate they found it useful for providing a second color or as a way to turn inferencing off. The participants rarely used widgets that were in the quick reference manual, but not covered in the tutorial. For example, one user simulated her own pulldown menus, but did not use the SILK defined ones. Others used a mixture of *Decorate* and *Sketch* mode to illustrate widgets not directly supported by SILK, such as scrolling list boxes (participant 9) and tables (see participant 7's designs in Figure 6-1). Similarly, two designers used *Annotate* mode to document their designs, even though this was not shown in the demonstration or tutorial.

6.2.1.3 Missing Features

Besides improving the performance of the system (see Section 6.2.3), the designers felt that SILK needed to support more dynamic affects, such as "sprites," "roll overs, perceptual cues, and movie-like transitions." In addition, several designers mentioned problems manipulating the objects, a need for multiple text sizes and colors in *Decorate* mode, and a way to visualize the links as they "got messy and were hard to follow." Two designers hinted that a storyboarding model that combined the sketch and storyboard windows and allowed drag and drop would improve SILK. One wanted a way to share objects between the screens so that one editing change could be reflected in several screens. Another commented that it would be nice if storyboards could be "ported to/edited in Director/Hypercard."

Despite missing these features, the designs produced and the techniques used during the sessions illustrate that SILK was still an effective tool for designing user interfaces in the early stages of design. One designer summed it up well by saying that SILK "works like pencil and paper; is simple, [and storyboards] show logic of navigation."

6.2.2 Communication Effectiveness

The post-design engineering review illustrated that SILK makes it easy for designers and engineers to discuss a design and demo it, while not getting caught up in the details of the look. The scaled-down size of the screens in the storyboard encouraged lots of pointing to the different screens, permitting a productive discussion of the overall design.

Table 6-5 summarizes what occurred during the post-design discussion. Three of the designers used the storyboard during the discussion in order to illustrate the overall structure of the design. All of the participants, with the exception of participant 11, used *Run* mode to exercise the interface during the design review. One of the designers permitted the engineer test the interface. Even though participant 11 had no transitions in her design, she was able to use the storyboard by repeatedly copying screens to the sketch window to describe her "quick solution."

Engineering Discussion							
question / participants	7 & 8	9 & 10	11 & 12	13 & 14	15 & 16	17 & 18	**% yes**
engineer asked critical questions?	yes	yes	no	N/A	yes	yes	67%
asked questions on structure & behavior?	yes	yes	yes	N/A	yes	yes	83%
made real-time changes?	yes	no	no	no	yes	yes	50%
storyboard used to test?	yes	yes	no	yes	yes	yes	83%
storyboard used to illustrate structure?	yes	no	yes	N/A	no	yes	50%
successful communication of idea?	yes	yes	no	N/A	yes	yes	67%

Table 6-5. Characteristics of the post-design engineering discussion. Note that the data for participants 13 & 14 is incomplete as the video tape of that discussion was not viewable.

Four of six of the engineers asked critical questions. For example, participant 8 (engineer) noted that participant 7's use of a calendar (see the top design in Figure 6-1) to allow the retrieval of old weather reports was a bad idea. He said, "you are really going out of your way to provide this weather log... It is constraining your design and making it really hard to do the common case." In the next design session, participant 10 (engineer) asked about navigation; "what if I want to pop from here to another city altogether. What is the most direct way to do it? To a city that is not on the map, say in Arizona?" These questions and comments illustrate that the engineers understood the interface designs and were looking for places where the ideas may have needed more thought. I observed that these conversations concentrated on the structure and functionality provided by the interface, rather than on the visual details.

Finally, half of the designers made real-time changes to their design during the engineering discussions. For example, participant 7 used the feedback from the engineer to rapidly modify his first design. The engineer commented that the original design (see the top design in Figure 6-1) required too many mouse clicks to get the desired weather information. The changes involved combining several screens, thus compressing a five screen design down to a three screen design (see Figure 6-7). The engineer and designer discussed which changes to make during a 15 minute conversation. The changes included deleting objects, resizing objects, moving objects between screens, designing new objects, and making new screen transitions. These revisions took under five minutes to make, so they could be done while the engineer was present. These examples show that SILK successfully achieved the objective of allowing designers to effectively communicate a design idea to other members of a design team.

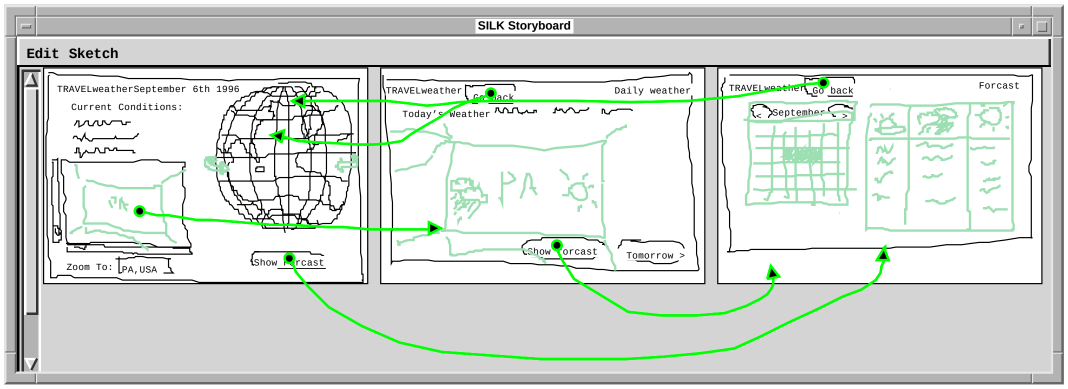


Figure 6-7. Revision of participant 7's first design made during engineering discussion.

6.2.3 System Performance

Overall, the designers were fairly positive about SILK (rating its performance at 6.2 on a scale from 0 to 10), and mainly criticized it for specifics of the implementation and the "UI of the tool itself." In addition to uncovering several bugs in the system, the usability test brought to light some problems with the recognition algorithms (for gestures, primitive components, and widgets) and with SILK's user interface in general.

6.2.3.1 Recognition Accuracy

SILK performs inferencing at three different levels: editing gesture recognition, primitive recognition, and widget recognition. This section analyzes each of these processes separately and reports the level of accuracy SILK achieved. Due to the lack of information on the test subjects intentions while drawing, all of these metrics have some amount of ambiguity, but are still reported here to give an estimation for how well the system performs in practice.

Editing Gesture Recognition

Since the current implementation of SILK does not offer the user a way to correct misrecognized editing gestures, there is no way of knowing when SILK interpreted an editing gesture incorrectly. This could be estimated by a more thorough review of the videotapes combined with an attempt at inferring the participants intent. The alternative is to use the information SILK does record. The system reports whether a gesture was classified or was found to be too ambiguous and thus unrecognized. Given these constraints, this section reports the percentage of editing gestures that were classified. This is an upper bound on the actual recognition rate, since some gestures may have been misclassified rather than unrecognized. Thus, the classification rate is given by:

$$r = \frac{e - u}{e}$$

Where e = total number of editing gestures drawn
u = number of unrecognized editing gestures

The data collected during the design task is given in Table 6-6. This table shows that the participants had a mean recognition rate of about 89%. This is in contrast to a rate of 82% achieved during the tutorial (see Table C-3 on page 174). This difference may not be significant[6] or may be attributed to the user learning how to effectively draw the gestures in order to achieve a higher recognition rate. This rate is in line with the 92% rate Rubine found in a single participant test of his single-stroke gesture recognition algorithm using a classifier that was not trained by the participant [Rubine 1991b]. If the participant were given a chance to explicitly train the recognizer or if the system were able to learn how a particular participant drew their editing gestures, this rate might improve to somewhere above 95%. As gestures are not as easy to draw with a mouse, this rate might also improve with the use of a stylus.

Editing Gesture Recognition

participant	7	9	11	13	15	17	mean	median	stdev
#editing gestures	67	139	63	119	151	78	103	99	38
#unrecognized	3	17	8	8	34	7	13	8	11
recognition rate	95.5%	87.8%	87.3%	93.3%	77.5%	91.0%	88.7%	89.4%	6.3%

Table 6-6. Editing gesture recognition data for design task.

6. The small number of data points makes it difficult to draw any statistical conclusions with a high degree of confidence.

Surprisingly, five of the six designers stated that the editing gesture recognition worked “well” or “OK”. Some did comment that the system does better if the gestures are drawn in a particular direction and that certain gestures did not work well for them. Several of the designers experienced problems with different editing gestures: some had problems with group, some with text insert (caret), and some with delete. The main problem was that the subjects drew these in directions or orientations that were different than those recommended in the tutorial (see Figure D-4 on page 203). A simple beep did not appear to give enough feedback when these gestures were unrecognized. These problems could be alleviated by having a button panel that gives feedback on the recognition results and by allowing corrections and on-the-fly learning as is done with the primitive component recognizer.

Primitive Recognition

Unlike with editing gestures, SILK *does* permit the user to correct mistakes made by the primitive component gesture recognizer. In fact, as described in Section 3.2.2, these corrections are used to improve the recognizer as the system is used. In calculating the primitive recognition rate, only user-initiated corrections are considered errors. Therefore, the primitive recognition rate for the primitive recognizer is given by:

$$r = \frac{p - c}{p}$$

Where p = total number of primitives drawn
c = number of primitive corrections

It is still possible that the recognizer will either misclassify primitives or label them `unrecognized`, but the user may not care and thus choose not to correct the system, so these are not counted as errors. These “don’t care” primitives tend to occur when the designer has drawn static graphics that are not part of widgets (without using *Decorate* mode). Many of these misrecognized and unrecognized gestures can then be considered correct inferences. Therefore, the following results represent an upper bound on how well SILK actually identified the intended primitive.

The SILK usability test produced the results given in Table 6-7. These results show that SILK correctly recognized the users’s primitive components 93% of the time. This is in contrast to the rate of 89% achieved during the tutorial (see Table C-3 on page 174). These recognition rates are quite encouraging, though they might also be improved by using a stylus and tablet. Adding a similar mechanism, i.e., visible feedback with corrections and learning, to the editing gesture recognizer seems to be a promising change to make in the system.

Primitive Component Recognition									
participant	7	9	11	13	15	17	mean	median	stdev
#primitives	120	102	40	55	123	114	92	108	36
#unrecognized	21	20	20	12	13	38	21	20	9
#corrections	9	8	3	0	15	7	7	8	5
recognition rate	92.5%	92.2%	92.5%	100.0%	87.8%	93.9%	93.1%	92.5%	3.9%

Table 6-7. Primitive component recognition data for design task.

I also attempted to measure how much better this recognizer became by learning while in use. Therefore, the results of a non-learning recognizer were recorded simultaneously. The non-learning primitive recognizer was assumed to be in error if its result did not agree with the learning recognizer. This assumption would later be reversed if the designer corrected the system and the correction agreed with the original inference of the non-learning recognizer. This measurement showed that the non-learning recognizer was correct only 72% of the time. However, these results are not very accurate. The non-learning recognizer is unfairly penalized by the assumption that the learning recognizer was right if the user did not correct it. If a “don’t care” primitive is classified differently by the learning and non-learning recognizers, this measurement technique incorrectly assumes that the non-learning recognizer was in error.

Poor feedback of primitive component inference results was responsible for most of the confusion designers experienced with SILK’s widget recognition algorithm. The designers repeatedly failed to notice that one of the primitive components they wanted grouped into a widget had earlier been misrecognized. They became confused when repeated new guess, group, and ungroup operations were not helpful. One way to offer better recognition feedback is to flash the primitive type or a rectified version of the primitive on top of the sketch. This problem may also be alleviated in actual use if the primitive recognizer continues to improve with training. One obstacle to this improvement during the experiment was the fact that some participants had difficulty remembering to correct primitive recognition errors. Thus, another potential solution is to have the **`new guess`** button cause new primitive inferences, rather than only causing new widget inferences.

Widget Recognition

The most important result, and possibly the hardest to measure, is how well SILK recognizes widgets. Some of the problems are similar to those discussed above for primitive recognition. For example, we cannot know which inferences are made on widgets that the designer does not care about. Another problem occurs when the designer forces a new inference (via the **New Guess** button) on an object that cannot possibly be one of the widgets recognized by SILK. Is that an error? Both of these problems relate to not knowing the designer's intentions.

In assessing the performance of the widget recognizer, there are three cases that must be considered:

- The designer draws the last piece of a widget and SILK proposes a widget type.
- The designer draws the last piece of a widget and SILK proposes nothing.
- The designer selects an object or multiple objects and requests a widget guess.

In the first case the designer can accept the proposed inference, click on an alternative if there are any, press the **New Guess** button to force a new inference, or break up the combined object. The object can be broken up by selecting the **Multiple-Objects** alternative or by selecting **Ungroup** from the **Arrange** menu. Therefore, each time SILK proposes a widget type, the system keeps track of which option the user takes. It is assumed that an inference was accepted if the designer did not choose one of the other options either immediately or at a later point in time.

In the second case, where SILK proposes nothing, there is no data for the system to record. If the designer really *does* desire an inference, she may select the object and click on the **New Guess** button. Therefore the second and third cases can be combined as far as measurements go. Note that the third case can result in either of the first two cases, that is, SILK may or may not make an inference. Given these possible scenarios, there are several results to report:

- The number of times SILK makes a first inference on a set of objects and how often that inference is not accepted. When it is not accepted, either immediately or later, we can report how often the designer tries to repair these inference errors[7] by selecting an alternative, clicking on **New Guess**, or breaking up the object.
- The average number of correcting operations (selecting an alternative, clicking on **New Guess**, or breaking up the object) per inference error. This gives an indication of how many operations it takes to fix an inference or how long until the user gives up.
- How often the user explicitly selects an object, asks for an inference, and either gets one or not. This gives an estimation of how often the user must force the system to give an inference. If an inference is made, how well the system did is reported by the above inference calculation.

7. Only the first correction is considered an error. Nor do repeated clicks on **New Guess** count as new inferences, unless the designer cycles past **Multiple_Objects** to a new inference.

The value for the number of times the designer breaks up an object gives an estimation of how often the system is picking the wrong objects altogether or inferencing when it should not. One common exception to this was seen in the sessions when the subject broke up text fields in order to reposition the newly typed text that had replaced the original text squiggle. These breakups were still counted as errors.

The widget recognition results are given in Table 6-8. These results show that the designers on average saw the right inference on 69% of the inferences SILK proposed. SILK was correct on 68% (98) of the 144 total inferences combined from all the participants. These results were brought down considerably by participant 15 who repeatedly broke up widgets (29 times). A large portion of this participant's breakups were for moving text inside of text fields. In addition, this designer broke up many buttons that he wanted to be unselectable to indicate that their functionality was unimplemented. This overall recognition rate is promising, but may still be too low for practical use. The rate could be improved by having the system learn new rules for inferring widgets (as discussed in Section 3.7), allowing users to move components that are inside of widgets without breaking them up first, and using a stylus and tablet instead of a mouse.

The next measurement of interest is the average number of repairs per inference error. A *repair* is an operation the designer made when attempting to correct an error. The data shows that the designer's made about one repair on average for each error and in the worse case one designer made just less than three repairs per error. This is encouraging, though it may indicate that many errors were repaired using just one operation while a few other errors generated several repair operations. In addition, we do not know whether the user got the desired results or just gave up.

Widget Recognition									
participant	7	9	11	13	15	17	**mean**	**median**	**stdev**
#widget inferences	41	17	2	14	56	14	24	16	20
#errors	10	1	1	5	26	3	8	4	10
recognition rate	75.6%	94.1%	50.0%	64.3%	53.6%	78.6%	69.4%	69.9%	16.7%
repairs:									
#new guesses	7	0	0	0	23	0	5	0	9
#changes	0	0	0	0	13	0	2	0	5
#broken	3	1	1	5	29	3	7	3	11
repairs/error	1.0	1.0	1.0	1.0	2.5	1.0	1.3	1.0	0.6
#explicit inferences	11	1	0	4	17	0	6	3	7
#explicit successes	2	1	0	1	11	0	3	1	4
explicit success rate	18.2%	100.0%		25.0%	64.7%		52.0%	44.9%	38.0%

Table 6-8. Widget recognition data for design task.

When the designer has not gotten an inference or wishes to change an inference later, an explicit inferences can be made by selecting an object or group of objects and pressing the **`New Guess`** button. The last measurement in Table 6-8 shows that almost half the time the designer explicitly requested an inference, one was given. Many of these requests may correspond to regrouping a text field after breaking it up in order to move the typed text. The data on explicit inferences is not especially reliable, however, since two of the designers never even made one explicit inference request.

Many of the participants were unable to get SILK to recognize widgets due to hard-coded size constraints in SILK's rules. The most common example of this was when the participants drew buttons that were wider than SILK's rules permitted. Some participants discovered this on their own and resized the buttons in order to force the correct recognition result. The sizes specified in the rules are fairly arbitrary and can be increased. Machine learning of widget rules would also help here (as discussed in Section 3.7). The participants also uncovered a few cases where new rules needed to be added to SILK's existing set in order to support what was drawn. For example, subject 7 drew a scroll bar to the right (rather than the left or the bottom) of a window and SILK did not recognize it as a scrolling window. Again, machine learning techniques would allow adding these rules in a fairly automatic way.

One user liked the "ability of SILK to understand widget types," although he and others were worried that "it won't understand non-standard widgets." Another designer did not like "fiddling with the recognizer" and "towards the end, [he] just wanted to grab widgets from a palette" because of continually having to "group and ungroup, change guesses, and so on." The misrecognition of primitives and having problems forcing the system to make a proper widget determination were the most common system errors the designers cited. It is possible that any widget recognition algorithm may be too error-prone and thus hurt the design process more than it helps. It is impossible to cite a recognition rate (other than 100%) that would say definitively whether or not widget recognition is worthwhile without testing such a system. My observations tell me that the current rate (69%) is not good enough. If widget recognition cannot be improved using the methods described above, a more explicit widget type or behavioral specification technique may be worth looking into (as described in Section 8.1.2).

6.2.3.2 SILK UI Problems

The designers ran into several different problems with SILK's user interface, most falling into three general areas; users misunderstood the relationship between the storyboard and the sketch, how to enter and manipulate typed text, and how to select and move objects.

Storyboarding Model

It is apparent from the study that the relationship between the **`SILK Storyboard`** and the **`SILK Sketch`** windows was unclear to many of the designers. The sketches in these two windows are distinct and must be manually synchronized by the user. Several screens were lost when designers forgot to save them to the storyboard and then started a new screen in the sketch window. In addition, the **`Undo`** menu item was not implemented for the **`Storyboard`** window. This lack of undo in the storyboard caused several screens to be lost when designers deleted them by mistake. The participants also wanted to copy previously drawn objects out of the storyboard and drag them onto the current sketch. Since the screens in the storyboard are uneditable, this required copying the current sketch to the storyboard (to save it), copying the screen with the desired elements to the sketch, selecting and copying the elements to the copy buffer, copying the saved screen back from the storyboard, and finally making the necessary changes using paste.

These storyboarding problems could be solved by going to a new storyboarding model. In this new model a single storyboard would contain all of the designer's screens, without a separate sketch window. All editing would take place on the screens in the storyboard. These screens could be scaled up or down as needed to make editing easier. For example, the currently selected screen could be automatically blown up so that its details could more easily be created and modified. This model would no longer require the designer to perform complex synchronization operations, such as repeatedly copying and pasting parts of screens.

Selecting, Grouping, and Moving Objects

Most of the other problems encountered by the designers had to do with poor user interface design choices made in both SILK and the underlying system SILK was built upon, Garnet. All of the designers had a problem selecting, grouping, and moving objects. The problems encountered with selection had to do with the mapping of mouse buttons to SILK modes. The left button is held down while sketching widgets and the middle button is held down while making editing gestures. The right mouse button is not used so as to more easily support pen-based tablets (which typically have only two buttons – one on the side of the pen and one on the pen tip).

The system interprets a single click of the left or middle button on the outline of an object as a selection operation. The designers often had problems hitting the outline. In the case of the middle button, this causes nothing undesirable, but in the case of the left button this may create a tiny object (possibly a single pixel). In addition, the designers often used the left button when purposely clicking away from an object in order to turn the selection off. Again, a spurious object was created. In both cases the designer was forced to delete this spurious object. Many of the unrecognized primitives were a result of these selection errors.

Given the design choice of using two buttons, this confusion is hard to eliminate. The alternative would have been to use a single button. Overloading a single button could be achieved by one of three options: (1) use one gesture recognizer for both editing gestures and primitive components, (2) use a time-out mechanism to decide whether the user is performing an editing operation or simply sketching, or (3) add a new mode that switches the pen between editing and sketching operations. The first option would reduce the editing and primitive gesture recognition rates. The second option may appear unpredictable to the user and thus generate more errors than the current interface. The third option would probably be both confusing and time consuming.

Another problem was with grouping. The designers often created a large number of small objects in their designs. When they wished to reuse or move these objects they had to be selected one at a time, as SILK does not support dragging out a rubber band box to select them. Although a grouping gesture (encircling the objects) is supported, this irregular gesture is often poorly recognized by Rubine's algorithm. This problem may be solved by writing a special recognizer for this gesture.

By far the most common editing problem occurred when designers tried to move objects. Most programs allow dragging an object by mousing down anywhere on the object and moving the mouse. Garnet, as described in Section 1.4.1.3, instead supplies white *move* handles (similar to the standard black grow handles) that appear on the object when it is selected (see Figure 1-10 on page 13). The user is forced to move the object by dragging on these handles with the left or middle mouse button. The designers regularly forgot this or missed the handles. Again, this led to lots of unrecognized editing gestures when the middle button was used and lots of spurious single-pixel objects when the left button was used. Participant 11 repeated this error dozens of times throughout the tutorial and design task.

Creating and Editing Typed Text

Another problem the participants commonly experienced was with creating and manipulating typed text. Creating new typed text, replacing squiggly text, or editing existing text are all accomplished by making a caret gesture at the text location (see Figure 6-8). Unfortunately, the current implementation requires that the peak of the caret gesture intersects with the bounding box of text to be edited or else a new text object is created. At least three of the participants were unaware of this and repeatedly

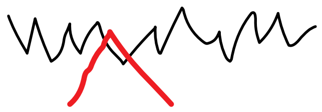


Figure 6-8. Caret gesture for creating and editing typed text.

had problems getting the peak to intersect, thus creating a new text object where none was wanted. This can be solved by having the caret gesture edit any text that the gesture intersects with, not just the peak.

In addition, the participants struggled with the size of the text. Besides only offering one font size, the usability test used larger than normal fonts in order to make the resulting video more useful. Because of this, the participants often had problems reproducing the screens in the tutorial (which used a smaller font) without having text overflow the edges of their widgets. One participant (13) avoided this problem by using the tutorial as only a guide and chose her own layout and text strings. Supporting multi-line text labels, multiple font sizes, and scalable text would alleviate some of these problems.

The final problem with typed text had to do with alignment. The previous observation about a lack of alignment activity was most commonly violated when the designer was converting squiggly text to typed text. For example, replacing squiggly text in text fields or buttons often led to the repositioning of the text object since its size changed after the text replacement. Almost all of the participants ungrouped text fields and buttons to reposition the labels. This indicates that it should be easier to reposition objects contained inside of another object without having to use ungroup and that SILK should automatically keep typed text to the left of the text field box, as illustrated in Figure 6-9. Automatic alignment would have to use heuristics, as the desired alignment often depends on the widget type (e.g., right-aligned text for text fields, left-aligned text for radio buttons, and centered text for buttons).

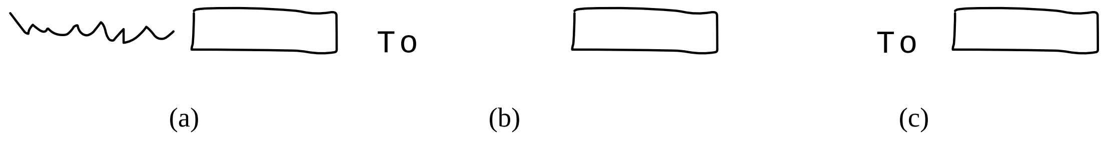


Figure 6-9. Replacing the squiggly text in (a) currently results in something like (b). A proposed change would keep the inserted text right aligned with the text field box as in (c).

Most of the problems discussed here can be solved by offering better feedback to the user, using machine learning techniques to improve the recognition of editing gestures and widgets, and making some basic changes in SILK's user interface.

6.2.3.3 Learning and Agility

The participants worked through the entire SILK tutorial in 54 minutes on average. This, coupled with the fact that five of the six designers were able to use the major features of SILK taught to them during the tutorial (the recognized widgets, editing commands and gestures, specifying behavior in the storyboard, and testing the design in *Run* mode), indicates electronic sketching and storyboarding do not take much effort to learn. One designer noted that the “tools are few and simple – it’s easy to remember all possible combinations of things.” Addressing the recognition and user interface problems discussed above may make the system even easier to learn.

In addition, improving the agility of the system may also improve its effectiveness. Several of the designers noted the difficulty of sketching with a mouse and commented that a pen and tablet would work better. SILK was designed to be used with a *display tablet*[8], but I was unable to obtain one. In addition, SILK really slowed down as the participants got near the end of the task. This was due to two reasons: as the design sizes grew, file saves took a long time and there is a memory leak in the Lisp environment the experiment used which led to frequent garbage collections near the end of the task. The designers commented on both of these problems, reaffirming the importance of fast tools for the early design phase.

The participants showed that SILK, with its current performance, will not replace paper entirely in the early stages of user interface design. Participant 11, who had the most problems, spent the first 13 minutes of the task designing on paper. This was mainly due to the problems she experienced using SILK during the tutorial. Participant 9 said she was “fighting the urge to go to my piece of paper” – she did so momentarily later in the experiment when waiting for a long system save. Still, participant 13 stated during the engineering discussion that SILK was “like pencil and paper” and a “good quick sketching tool to get ideas down.”

6.2.4 Summary of Results

In summary, the usability evaluation showed that SILK achieved two major goals: designers are effective at using SILK for creating early user interfaces designs and then communicating those ideas to other members of a design team. The designers achieved these results by using SILK to quickly focus on the important design details, edit the design, add interactivity to the design and test it, and finally demonstrate the design to an engineer and modify it as needed.

6.3 Engineering Changes

As SILK is an experimental system, many bugs were found during the pilot and the actual tests described in this chapter. This section discusses how decisions were made on what bugs to fix, describes the type of bugs that were fixed, and gives an idea of what known bugs remain.

8. I use the term *display tablet* for a unified stylus, tablet, and LCD display.

6.3.1 Bug Fix Policy

My policy was to first fix any bugs found during the pilot that would not allow a participant to complete the tutorial task. Then, any other bugs that had either an easy fix or no obvious work-around were tackled. Finally, during the SILK usability test, the policy was to only fix bugs that could keep participants from performing operations required to complete the tutorial. Bugs that crashed the system were of the highest priority. There was one exception to this policy, which will be described below.

6.3.2 Bugs Fixed

There were at least 32 bug fixes made as a result of the SILK usability test. Most of these fixes were made during the pilot. The majority of the bug fixes made before the actual experiment had to do with fixing selections, printing, and getting editing operations (e.g., cut, copy, paste, and clear) to work properly on storyboard screens.

Only five non-cosmetic fixes were made after the actual experiment started. Four of the bugs had to do with storyboard transition arrows. The first two bug fixes were made after the first participant's (7) session. This participant was unable to delete storyboard arrows (a side effect of an earlier bug fix). This was corrected before the next participant was run. The second bug fix corrected a problem in detecting the object a storyboard arrow was drawn from. A similar problem was found after the second participant's (9) session. SILK was choosing the wrong object when arrows were drawn from text fields. The participant worked around this by drawing a small object next to the text field and drawing arrows from the new object instead. This bug was fixed after participant 9's session. The final bug caused SILK to crash in rare situations when deleting arrows. This bug was fixed just before the last participant's (17) session.

There was one bug fix that violated the bug fix policy stated above. Two of the first three designers tried to draw buttons that used circles (rather than rectangles) as the container. The non-recognition of these buttons caused considerable confusion, so a new rule was quickly added after the third participant's (11) session and this problem went away. At the same time, the rule was modified to allow wider buttons, as the participants had also run into this problem. These fixes may have improved the widget recognition statistics for the last three participants, but if so it would only be a small improvement since the first three users all learned of the original limitation and a work-around during the tutorial.

6.3.3 Bugs Remaining

As might be expected, there are many known bugs remaining in the system. These bugs are distributed over a range of system functionality. The bugs include the following: corruption in file saves, error dialogs over the wrong windows, cut/paste on storyboard arrows crashes the system, and clicks on objects in *Run* mode are not detected properly in a few rare cases. Given enough time, all of these bugs are fixable and the corrections should be made before the system is released.

6.4 Discussion

The usability test results, numerical and descriptive, suggest three important conclusions: electronic storyboarding is a major win for the early stages of design, electronic sketching is fairly easy to learn, and several engineering and tutorial changes would make SILK much more usable.

The value of storyboarding was brought to light mainly in the descriptive results. The designers repeatedly mentioned storyboarding as the feature they really liked and found useful. In addition, the observations taken during the engineering discussion showed that storyboarding, through both the interactivity and design overview it provides, helped designers to effectively convey their design ideas to engineers. Engineers and designers were able to discuss changes and make these changes while the engineer was present.

The designers found that storyboarding was fast and easy to learn. The fact that several of the designers found that SILK was similar to pencil and paper made the tool itself fairly easy to learn. Most of the designers were effective after a simple demonstration and a tutorial that took under an hour to complete. This was despite many bugs and user interface problems with SILK itself.

Many of the problems the participants encountered with recognition errors can be improved by offering better recognition feedback, using machine learning to improve the default rules and recognizers, and improving the tutorial. The tutorial needs to stress the importance of correcting inferences, rather than redrawing primitive components, so that the recognizer will learn the user's style of sketching.

The main goal of this evaluation was to see whether designers could effectively use SILK to design user interfaces. The usability test showed that this goal was met – five of the six participating designers were effective using SILK. In addition, the designers on average produced two different designs in about one and a half hours. After the designs were finished, the designers were able to use SILK to demonstrate the designs, have a productive discussion about the design with an engineer, and make real-time changes to the designs. Finally, the designers uncovered several problems with the current SILK user interface and recognition algorithms. Fortunately, most of these problems have solutions.

CHAPTER 7

Related Work

Sketching on paper has always been used to explore early design ideas. Yet only recently have the advantages of paper-based sketching over rigid electronic tools for conceptual design been more formally explained. My application of sketching to the electronic domain tries to take advantage of these results.

Similarly, though electronic design tools have been available for a number of years, only recently have they become a standard day-to-day tool for designers working on the later stages of a design idea. This may have to do with the fact that many of the early tools were research prototypes and that current commercial tools are often targeted at producing either prototypes or finished interfaces, but do not offer support for both tasks.

This chapter first looks at the previous work examining sketching in the design process. It then looks at the important electronic tools that have been produced: including prototyping tools, user interface builders, and other electronic sketching tools. Finally, it reviews the work on informal and pen-based user interfaces, recognition technologies, techniques to specify behavior, and the inference of graphical constraints.

7.1 Sketching on Paper

The work on paper-based sketching can be broken up into two main areas: the advantages gained by sketching in terms of how others evaluate a given design and the advantages gained in terms of improved design on the part of the designer.

7.1.1 Sketching Improves Design Evaluations

Wong's work on scanning in hand-drawn interfaces was the major impetus for my work in this area [Wong 1992]. This research, done within Apple's Advanced Technology Group, was an outgrowth of Wong's frustration with her colleagues mistaking early prototypes for more finished designs. This resulted in early design reviews that focussed on alignment, colors, and fonts rather than on the overall structure of the interaction. In order to allow her colleagues to focus on the important parts of the interface, Wong sketched her interfaces on paper, scanned them in, and then added behaviors to the sketchy interfaces by using Director [Macromedia 1994]. Since the interfaces looked rough, her colleagues no longer mistook them for finished designs.

Wong had to go through the tedious process of scanning-in sketches. SILK allows designers to create both the look and behavior of these interfaces directly with the computer. In addition, adding the behavior in Director was difficult since it often required writing code in Director's programming language, Lingo. SILK allows designers, who generally do not know how to program, to demonstrate behaviors more directly by simply drawing arrows between screens in the storyboard.

Another interesting observation about design evaluation comes from architectural design. Strothotte reports that architects often sketch over printouts produced by CAD tools before showing works in progress to clients [Strothotte 1994]. The reasoning is that clients who are shown precise drawings feel that the work is nearly finished and that they have little say in how it might be changed at that stage. In fact, this group has produced a system that can render precise architectural drawings in a sketchy look. More recently, they have done a survey of architects and have found that architects prefer sketches for a first draft, since they arouse interest in their designs and promote more discussion of a given design [Schumann 1996]. This work lends further evidence to the assertion that a different level of feedback is obtained from a sketchy drawing.

7.1.2 Sketching Encourages More Design Ideas

In addition to the observations about the positive effect sketches induce on design evaluations, others have found that sketches allow the *designer* to focus on the proper design issues. Goel's cognitive science thesis [Goel 1995] examined how sketching differed from structured drawing in order to argue that current versions of the computational theory of mind [Fodor 1975] fail to address sketching. Goel observed designers who were asked to solve design problems by either sketching on paper or using a computer-based drawing program.

Goel breaks design up into four distinct phases: problem structuring, preliminary design, refinement, and detailing. He characterizes preliminary design "as a process of creative, ill-structured problem solving, in which generating and exploring alternatives is facilitated through a coarseness of detail, a low commitment to ideas, and a large number of lateral transformations.” These transformations are facilitated by the ambiguity of sketches. Goel claims that a “notational symbol system, such as drafting, which differs from sketching in being nondense and unambiguous, will hamper lateral transformations.”

Thus the ambiguity of marks in an early design is important, “because one does not want to crystallize ideas too early and freeze design development.” Goel observed that in freehand sketching, when a new idea was generated, a number of variations were quick to follow. In contrast to this, with a drawing program (in this case, MacDraw) most subsequent effort after the initial generation was devoted to detailing and refining the same idea.

These results are very similar to earlier work done by researchers studying graphic design students. Black’s user study found that “the finished appearance of screen-produced drafts shifts [a designer’s] attention from fundamental structural issues” and caused designers to push computer-based designs further and explore fewer alternatives than they would using paper-based designs [Black 1990].

7.2 User Interface Software Tools

There have been many user interface software tools produced by both the research community and commercial concerns. This section reviews the earliest work and gives an overview of some more recent tools. A more complete overview of user interface software tools can be found in [Myers 1995]. The types of tools discussed have been further divided into prototyping tools and user interface builders. Unlike user interface prototyping tools, interface builders are generally used for producing the final application. Prototyping tools allow designers to quickly illustrate examples of what the screens in a program will look like.

7.2.1 User Interface Prototyping Tools

A survey of designers (see Chapter 2) showed that designers use prototyping tools more often than user interface builders. The main reason for this is that current prototyping tools are aimed at an earlier stage of design than interface builders. In addition, prototyping tools allow some end-user scripting, while not requiring programming in a complex environment that includes compiling, linking, etc.

Trillium [Henderson Jr 1986] was one of the early research prototyping tools. This Xerox PARC-produced tool was used for designing the physical user interfaces for machines (such as Xerox copiers and printers). Trillium structured the interface as a collection of frames (equivalent to screens in SILK). The system permitted the simulation of the prototype hardware user interface using a run-time interpreter rather than requiring compiling and linking to an application program. Trillium allowed some end-user extension of the look of objects, but most new objects or behavioral changes required programming. The major weakness of Trillium is that it offered little support for frame to frame transitions, since this was unnecessary for hardware user interface panels.

Dan Bricklin’s Demo [Lifeboat 1995] was one of the first commercially successful prototyping tools. It allowed designers to mock-up the ASCII menu structure and text input/output screens of MS-DOS programs. This simple tool allowed the design team to prototype a user interface before the actual application code had been completed.

Prototyping tools are now commonly used by interface designers. Of the designers I surveyed (see Appendix Section A.2), 48% used HyperCard, 45% used Director, and 29% used Visual Basic. These were the three most popular user interface prototyping tools used by these designers (at the end of 1994). Though useful in the prototyping stages, all three tools come up short when used either in the early design stages or for producing production-quality interfaces.

Apple’s HyperCard [Apple 1993] was one of the first popular UI prototyping tools for graphical applications. HyperCard’s “programming” metaphor is based on changing “cards” upon button presses. HyperCard shares many of the drawbacks of traditional user interface builders: it requires designers to specify more design detail than is desired, and often it must be extended with an event-based programming language (HyperTalk) when the card metaphor is not powerful enough. In addition, HyperCard cannot be used for most commercial-quality applications due to its poor performance and lack of ability to pass off the design to another tool, which usually forces the development team to reimplement the user interface with a different tool. HyperCard’s in-place card transitions were a major influence on SILK’s storyboarding transition scheme. Unlike HyperCard, however, SILK’s event transitions are visible, and several can be viewed at once. Thus, SILK storyboards may be easier to understand and edit.

Director [Macromedia 1994] was designed primarily as a media integration tool. Its strength is the ability to combine video, animation, audio, pictures, and text. This ability, along with its powerful scripting language, Lingo, has made it the choice of multimedia designers. These strengths, however, lead to weaknesses when used as a general interface design tool. It is very hard to master the many intricate effects that Director allows. In addition, it lacks support for creating standard user interface widgets (e.g., scroll bars, menus, and buttons) and specifying their behavior in a straightforward manner. Finally, Lingo, a full-powered programming language, is inappropriate for non-programmers.

Design tools such as Director and HyperCard allow the sequencing of screens, and although they use direct-manipulation methods to specify these sequences, these methods lack the fluidity of paper-based storyboarding. For anything but the most simple sequences, these tools require the use of scripting languages.

This is also the major drawback to using Visual Basic [Microsoft 1993], which is becoming increasingly popular for interface prototyping due to its complete widget set and third-party support. There are literally hundreds of third-party VBXs available for Visual Basic that add on new features, such as database engines, new widgets, and spreadsheets. Visual Basic includes a standard drag-and-drop user interface builder, yet it is considered a prototyping tool since the language it is based upon, Basic, is not usually considered a production language, nor does it require the tedious compile/link/test cycle of a hard to learn language such as C or C++ (the typical languages supported by user interface builders).

7.2.2 User Interface Builders

Unlike prototyping tools, user interface builders create the actual code for the application user interface. They generally allow the designer to select widgets, such as dialog boxes, menus, and windows from a predefined palette, and place them on the screen with a mouse. They also allow the designer to set properties of the widgets, such as size, position, color, and textual labels. Since user interface builders are used for producing a finished application, these systems may generate code that is compiled and linked with the rest of the application or they may generate some type of internal representation that a User Interface Management System's (UIMS) run-time system can use to implement the UI when the target application is executed.

MENULAY [Buxton 1983] was one of the first interactive UI builders and was part of an early UIMS. It was used for laying out networks of menus that could be structured in a hierarchic manner. It allowed the designer to assign function names to be called on menu selections and to set object properties, such as position, color, and size. The system produced a table that was converted to linkable C code via the MAKEMENU system. MENULAY was quite limited in that it was only for laying out textual or graphical objects that, when selected, would cause a new menu to appear.

Cardelli's Dialog Editor [Cardelli 1987; Cardelli 1988] was another of the early user interface builders. It allowed the creation of graphical UIs that feature many of the common widgets in use today (such as scroll bars, pulldown menus, text fields, buttons, etc.). In addition, it supported the graphical specification of stretching parameters for the widgets, thus allowing the user to specify how the objects should change when the windows containing them were resized. The system produced code that was then attached to Modula2+ event code.

The NeXT Interface Builder [NeXT 1991] was the first major commercial UI builder. It allowed the standard layout of UI objects, but also supported connecting actions to these objects by graphically “wiring” objects together. This style of visual programming permitted much more of the UI to be specified in the UI builder, but it still required programming (in Objective C or C++) to finish the interface. A NeXT financed study claimed that applications using the NeXT environment (including the Interface Builder) required 83% fewer lines of code and took one-half the time of applications written with less advanced tools [Hamilton 1992]. Other popular Unix-based interface builders include UIM/X by Visual Edge Software Ltd. [Software 1990] and Devguide from Sun Microsystems [Sun 1991].

There are now dozens of user interface builders for Microsoft Windows. One of the more interesting tools is Borland’s Delphi [Borland 1996]. Delphi includes an interface builder, programming language (based on Object Pascal), and debugger in a nice interactive development environment that includes fast incremental linking.

There are several problems with the interface builders discussed in this section. These tools are inappropriate for designers, since most designers do not know how to program and the tools take too much time to learn and use in the early stages of user interface design. Another major problem with interface builders is that they offer little support for the “insides” of an application, that is, the graphics area in a drawing program, CAD tool, or other style of visual editor. With SILK or one of the prototyping tools described above, the designer can at least attempt to give a flavor of how these graphics areas will look and work.

7.3 Informal Interfaces

Several researchers have recognized the benefits sketches provide (as discussed in Section 7.1) in terms of communicating a level of informality or indicating the draft quality of a design. What distinguishes the work described in this section is a reliance on standard mouse-based direct manipulation techniques for input, while using fancy rendering schemes to make the output appear to be “sketchy”.

As mentioned previously, researchers have produced a non-photorealistic renderer [Strothotte 1994] for an architectural CAD system. This system uses cubic curves to represent lines; the control points of the curves are modified to make the lines appear hand drawn. The editor allows setting parameters that affect line quality on an object or group of objects. This allows the architect to focus the attention of a viewer and elicit feedback on a portion of the design that is under consideration. One drawback to such a system is that the design is still created using a precise representation, leading the designer to potentially focus on unimportant details. A similar system is now a commercial product for PCs running Windows. Squiggle [Insight 1995] transforms CAD files, in the standard HPGL plotter format, to sketchy drawings.

The EtchaPad system [Meyer 1996] is a drawing program based on PadDraw, the drawing program supplied with Pad++ [Bederson 1994]. EtchaPad allows an end-user to draw a small number of recognized widgets that can then perform PadDraw functions that are normally accessed through menus. For example, the user can create a new button that saves a file when the button is pressed. Other interesting widgets include property widgets that allow the user to test and set values of object properties in a very direct way and tool widgets that allow graphical operations, such as aligning objects. The key idea is that these user interfaces are very light weight – a user might create them during a work session and then throw them away when done. The system uses Perlin's noise function [Perlin 1985] to give the drawings an informal look. The amount of noise can be varied in order to change the level of informality. Unlike in SILK, the widget recognition in EtchaPad is quite simple since there is a very limited set of widgets, they are composed of unambiguous shapes, and widgets must have a special diagonal "tick" line drawn in their upper left corner.

J-Sketch [Rieman 1996] is a novel attempt to make the appearance of mouse-based electronic sketches resemble that of paper-based sketches, without sacrificing the speed advantage often enjoyed by paper for rough sketches. Like the architectural CAD system mentioned above, it uses cubic curves (specifically, Bezier curves) to draw multiple overlapping smooth lines when the user draws a single line. An experiment showed that users could produce rough sketches faster with J-Sketch than with computer-based drawing applications and at about the same speed as with pencil and paper. This same type of algorithm could be added to a system like SILK to make the sketches look more like paper-based sketches.

In addition to the work of those exploring human-computer interaction issues, after 30 years of pursuing the goal of photorealistic rendering there is a growing interest in non-photorealistic rendering among computer graphics researchers. For example, non-photorealistic rendering is one of the key features of the SKETCH system described in Section 7.5. Researchers at the University of Washington have experimented with several systems for rendering [Winkenbach 1994] and interactively creating [Salisbury 1994] drawings that look like "pen-and-ink illustrations". Commercially, Fractal Design has produced a set of tools that are popular among graphic artists because these tool produce effects that previously could only be made using physical media. Painter [Design 1996] is a 2-D naturalistic painting system and Sketcher [Design 1993] is a sketching tool that can also transform photos to a more handmade look. Both tools use randomness to emulate real-world media such as charcoal, ink, oil paint, and pencil. Finally, Lansdown and Schofield give a review of the non-photorealistic rendering techniques used in computer graphics [Lansdown 1995].

7.4 Pen-based User Interfaces

The major problem with the systems discussed in Section 7.3 is that they ignore the advantages that pen-based input affords. Specifically, pen-based user interfaces offer a form of interaction that can be even more direct than the pulldown-menu oriented interaction traditionally provided with graphical user interfaces. These traditional interfaces require the user to first select an object with the mouse, then move the mouse to a pulldown menu or palette, and finally select an operation to perform on the selected object. Pen-based user interfaces can easily take advantage of gestures (see Section 7.6.4) to perform the selection and command with one stroke of the pen. In addition, pen-based user interfaces naturally support informal sketches, while also permitting more formal documents.

Sutherland's Sketchpad [Sutherland 1963] was the first system to explore pen-based user interfaces (see Section 7.5 for a description of Sketchpad). This groundbreaking effort was followed by pen-based application research at the Rand Corporation [Davis 1964]. Most of the Rand work was focused on handwriting and shape recognition.

In the mid-80s MCC's Interactive Worksurface Project developed a sketch-based interactive tablet intended for experimenting with pen-based user interfaces [Martin 1990]. The system used an AI blackboard system to support recognition of different types of data by several different recognition engines. The system supported handwritten numbers, letters, shapes, editing marks, and math symbols. The recognition engines used neural networks, which unfortunately require large sets of training samples. This technique seemed inappropriate for SILK since designers seem to draw UI components with their own look – SILK supports learning how an individual draws the basic shapes that compose widgets.

IBM had a similar research project that focussed on using gestural interfaces to support pen-based editing and markup of documents [Wolf 1989]. These user interfaces were termed "paper-like" to distinguish them from notepad computers that relied on keyboard and mouse interaction. The paper-like applications were run on *display tablets*[1] that were attached to workstations running a modified version of the X-Window system [Rhyne 1991]. The research included work on producing recognizers for

1. I use the term *display tablet* for a unified stylus, tablet, and LCD display.

run-on (i.e., overlapping) handprinted characters [Fujisaki 1990] and a dialog interpreter [Rhyne 1987], which defined the language of symbols that could be combined and the associated actions to be taken upon recognition. Some of the sample applications they explored included a cooperative meeting application with a shared electronic whiteboard, gestural spreadsheet editing, data-base inquiry, a sketching program, a musical score editor, and a two-dimensional mathematical expression formatter. A user study of editing tasks using the Lotus 1-2-3 spreadsheet application found that users completed the tasks in about 70% of the time required by keyboard users. In addition, the subjects preferred the gestural tablet interface since the operations could be carried out quickly and the gestural commands were easier to remember [Wolf 1988]. The sketching application differs distinctly from SILK in that the IBM editor regularized the form of objects (i.e., “cleaned” them up), although it also allowed freehand sketching, but without gestural editing.

The NPL electronic paper project [Brocklehurst 1991] was a British effort that was very similar to the concurrent work at IBM. This display tablet-based system offered a familiar model, i.e., paper and pencil, in an attempt to ease the learning of the system. The work focussed on the recognition of gestures and handwritten cursive text. The system used the standard proof readers’ symbols as editing gestures. The cursive text recognition was based on learning from the samples of 250 individuals combined with learning the standard styles taught to school children. The sample applications they built included a text editor (for previously typed or recognized text), a script editor for manipulating handwritten text prior to the deferred recognition, a table construction tool, a graph/chart production tool, and a diagram editor. These tools differ from the philosophy of SILK in that they try to clean up and recognize sketches (for the tables, graphs, charts, and diagrams) after they are drawn.

The research work in pen-based interfaces was followed by attempts to make pen-based computing a successful commercial venture. Go Corp. introduced a pen-based operating system, PenPoint [Carr 1991; GO 1992], in the early 90’s. The goal was to become the standard operating systems vendor for pen-based tablet computers. In addition to its focus on pen-based input, PenPoint was a fully functional object-oriented operating system that supported application embedding, preemptive multi-tasking, and other attractive features. The only major commercial product that used PenPoint was the Personal Communicator from EO.

The two main selling points for the EO machines had to do with mobility and the social rules of meetings. First, a pen-based computer could be useful to the large number of mobile professionals (salespeople, medical personal, etc.) who were not using computers in their daily work because of the difficulty of using keyboards while standing up. Second, typing on a laptop computer is socially unacceptable in most meeting situations, whereas writing on a pad is not as intrusive. EO was eventually bought by

AT&T and after disappointing sales the product was discontinued. Momenta Corp. was another early pen-based computer start-up that failed in the pen-computing arena. It appears that the EO and Momenta machines were not successful because they were too expensive, too slow, and in the case of EO did not run user's existing software.

Microsoft tried to attack that last problem with Windows for Pen Computing, a modification of its standard Windows operating system that offered pen support and allowed pen-based computers to run standard Windows applications. Manufacturers built dedicated pen-based machines or pen/keyboard hybrids, such as Compaq's popular Concerto, to take advantage of this OS. While not a runaway success, Microsoft's product seems to be the safest choice for software developers and the customers purchasing the hardware. Microsoft's support for pens changed with the release of Windows 95. Now instead of integrating the pen support into the OS, Microsoft licenses the pen services component to OEM tablet manufacturers, who can then create the drivers to allow their hardware to work properly with Windows 95 applications.

Apple's Newton personal digital assistant (PDA) has certainly been the most successful pen-based user interface in terms of units sold, although the first major version faced a considerable amount of criticism and derision [Burgess 1993; Fehr-Snyder 1995] in the months after it was released (even in a popular comic strip). The main problem was that the handwriting recognizer, although one of the best available at the time, was not accurate enough. Unfortunately, both Apple's marketing and the early overviews emphasized the ability of the device to recognize the user's handwriting – as if handwriting recognition was considered the most important feature of this small device. This is in contrast to the approach of GO, who emphasized the importance of using a small set of gestures for editing and only using handwriting recognition in fields that could be constrained by a data type (which would improve the recognition accuracy). Since that time, Apple has released a major update of the Newton OS that includes a much better handwriting recognizer. The reviews have been positive [Lu 1996; Seiter 1996], but it is unclear how much damage Apple did to the entire pen-based market with their early Newton blunders.

Concurrent with the commercialization of pen-computing, researchers have continued to work on pen-based user interfaces. There has been several years of research exploring interfaces for the LiveBoard [Elrod 1992], which is a whiteboard-sized, pen-based, interactive display designed for use in conference rooms and classrooms. This device is the one commercial outgrowth of Xerox PARC's work in ubiquitious computing [Weiser 1993], which envisioned a future of inexpensive pad-based computers, whiteboard-sized pen-based tablets, and numerous small devices the size of post-it notes all connected together via wired and wireless networks.

Tivoli [Pedersen 1993], which is supplied with the LiveBoard, is a stroke-based meeting support tool that allows participants to make meeting notes, diagrams, or display prepared presentations in front of a group. The tool uses a combination of static interface widgets and gestures for editing the meeting notes. Tivoli supports remote collaboration (shared drawing), multiple pages, imported images, and printing. Later additions to Tivoli allow users to add structure to unstructured notes when needed, so that structure does not get in the way of normal usage [Moran 1995]. The system temporarily perceives the "implicit structures" in the sketch (as suggested by the Per-Sketch system described in Section 7.6.5), such as lists, text, tables, and outlines. Operations can then be carried out according to the expected behavior of the current implicit structure, such as moving items around in an outline.

7.5 Sketching Tools

There has been a recent spurt of work on electronic sketching, some of it inspired by early reports of SILK [Landay 1995a; Landay 1995b; Landay 1996a; Landay 1996b]. Like many areas in user interfaces and computer graphics, this area of research can trace its roots back to Sutherland's original Sketchpad [Sutherland 1963]. Sutherland used a light-pen to draw structured diagrams on one of the first graphics displays. Like Sketchpad, many of the systems described below try to make sketching on the computer easier and more powerful than paper in terms of editing. By adding interactive behaviors, SILK takes this further and tries to embody Sutherland's comment made over 30 years ago, "It is only worthwhile to make drawings on the computer if you get something more out of the drawing than just a drawing" [Sutherland 1963, p. 17].

The Electronic Cocktail Napkin [Gross 1994; Gross 1996] allows architects to sketch their designs on an electronic pad similar to the one used with SILK. Like SILK, this gesture-based tool attempts to recognize the common graphic elements in the application domain — architectural drawings. SILK differs in that it allows the specification and testing of the behavior of the design, whereas the architectural drawing is fairly static. The Electronic Cocktail Napkin also supports sketch-based queries that are used to search for similar architectural designs either from a library or from the past work of the designer.

Kramer's sketching system [Kramer 1994] is similar to SILK in its goal of supporting a very fluid free-form design environment. This conceptual design system is based on structuring a sketch as a set of translucent, non-rectangular patches. SILK concentrates on user interfaces and allows the sketches to behave, whereas Kramer's system allows attaching many different "dynamic interpretations" to patches, but supports only a limited set of actual interpreters. There have been proposals [Kolli 1993] for building dedicated hardware sketch pads for industrial designers who desire a tool for conceptual design with the fluidity achieved in Kramer's system but without the structure imposed by a system like SILK.

Pugh's work using interactive sketch interpretation for designing solid three-dimensional objects [Pugh 1993] is relevant in that this system, called Viking, also tries to bring the advantages of paper-based sketching to a computer-based tool. The system used a constraint based approach to generate 3-D geometry from 2-D sketches. Pugh's work differs in that it is for designing solid models and his sketches are really precise line-drawings of the objects rather than the rough paper-like sketches supported by SILK and the systems described above.

The SKETCH system [Zeleznik 1996] from the Brown University Graphics Group is, like Viking, designed for building three-dimensional scenes using a two-dimensional interface. Unlike Pugh, the developers of SKETCH used an "approach that is very similar to Landay and Myers' use of sketching" in SILK. That is, the end-user sketches rough 2-D gestures that the system tries to group together in order to infer 3-D objects. Since SKETCH is intended for early conceptual design of 3-D models, the tool renders the objects in a non-photorealistic style. SKETCH also uses gestures for editing the scenes and supports simple constraints for object manipulation. SKETCH differs from SILK in that the gestures used to create objects correspond to partial drawings of important visual features – generally edges – of the intended 3-D primitive, whereas in the easier 2-D domain SILK can use gestures that are quite similar to the intended object. Other than simple graphical constraints, SKETCH does not address any issues of interactivity in the design.

7.6 Recognition Technologies

SILK's gesture recognition and widget inference algorithms are related to many different recognition technologies. My work was not intended to expand on these technologies, but it is important to get an overview of the area to understand how SILK might change with a different recognition system. In this section I give an overview of the recognition process and then give a summary of the important systems that recognize handwriting, sketches, gestures, shapes, and visual languages.

7.6.1 General Recognition Issues

Recognition-based user interfaces translate input in a form similar to that used for human-human communication (e.g., speech, handwriting, drawings, and gestures) to a format that is machine readable (e.g., ASCII text, point vectors, or commands). The major problem with this style of input is that it is inherently ambiguous. Many of the design issues deal with resolving this ambiguity and/or informing the user of it and allowing the user to correct it.

Recognition proceeds in two phases: the recognition system is first trained to associate input data with symbols and then, during actual use, uses the learned associations to translate a given input to symbols. The training and recognition phases often use knowledge that is statistical, linguistic, or deductive in nature to produce the associations. The symbols resulting from recognition may or may not be correct. Recognition errors fall into three categories: a failure to produce a result, an incorrect result in terms of the symbols produced, or an improper grouping of input data. Since errors are bound to occur, it is important to consider the accuracy requirements of the task (see a discussion for handwriting recognition at the end of Section 7.6.2).

A complete treatment of the issues involved in designing recognition-based user interfaces is found in [Rhyne 1993]. In the following sections we concentrate on recognition-based user interfaces for stylus input and/or sketches/diagrams.

7.6.2 Handwriting Recognition

Handwriting recognition has been an area of computer science research since Dimond described his “stylator” in the late 1950s (cited in [Suen 1980]). There has been a flurry of recent activity due to the predicted economic viability of pen-based computers, though such predictions have yet to come true.

Previous work in this area can be divided between on-line and off-line handwriting recognition systems. Off-line recognition, similar to optical character recognition (OCR), recognizes writing that has been scanned from paper. On-line systems recognize handwriting that is transmitted by an electronic tablet and stylus in real-time. Off-line recognition has the limitation of not having access to data concerning the ordering and timing of the strokes and the pen up/down events. On the other hand, unlike the on-line case, off-line recognition is not limited by the constraints imposed by real-time interactive responses and, if running on a portable pen-based computer, smaller memories and slower processing speeds. A good review of handwriting recognition, both on-line and off-line, is given in [Senior 1992]. More complete reviews of handwriting recognition can be found in [Suen 1980; Tappert 1990].

Most of the on-line work is based on calculating features found in an imaginary rectangle, called a frame, that circumscribes the character under consideration. The algorithms then differ on which features they calculate, how they recognize characters based on those features, and how the characters are combined to recognize words (e.g., use a dictionary for context or treat the characters separately). Some of the early work was based on zoning [Hussain 1972]. This technique divides the frame into several zones and calculates a feature based on the densities of points in these different regions and the order in which the zones were crossed by the pen stroke. This same technique is used for the architectural sketch and gesture recognition used by the Electronic Cocktail Napkin [Gross 1994] described in Section 7.5.

Handwriting recognizers have finally gotten to the point where decent recognizers are being sold either to manufacturers to include in their products or directly to the end-user to add on to an existing pen-based system. For example, the original cursive handwriting recognition engine for the Apple Newton was developed by a Russian firm, ParaGraph [Bylinksy 1993]. More recently, the Motorola Lexicus cursive handwriting recognition system, which is based on neural networks, has become available as an add-on for several different stylus-based platforms [Hutheesing 1996]. Finally, several shorthand recognizers are also available (see Section 7.6.2.1).

The relationship between the accuracy of handwriting recognizers and user satisfaction has been shown to be highly task dependent [Frankish 1995]. For example, if handwriting recognition gives a significant benefit in completing a task, users will tolerate errors. In addition, users demand a higher level of accuracy for work that will be passed on to a superior than they would for personal notes or passing information to an assistant or colleague [LaLomia 1994].

7.6.2.1 Shorthand

Due to the problems with handwriting recognition accuracy, there have been several attempts at using new alphabets that are faster to use and more easily recognizable by the computer. Unistrokes [Goldberg 1993], a research prototype from Xerox PARC, uses an alphabet in which the letters have been reduced to symbols that can be easily drawn and recognized with a single stroke of the pen. In addition, the more common letters (based on English language usage) have been mapped to single line strokes, which can be drawn very quickly. Unistrokes is based on the same principle as a standard typewriter keyboard: after learning the strokes (keyboard layout), an expert user can enter text very quickly, but by looking at a cheat sheet (looking at the key caps) novice users can also use the system, but at a much slower rate. Another research system was Apple's T-Cube [Venolia 1994], which used single stroke gestures and pie-menus, which addresses the user learning question ignored by Unistrokes, to enter characters with a stylus.

A commercial product, Graffiti [Fitzgerald 1994; Lee 1994], was quick to follow the research prototypes. It is much easier to learn than Unistrokes since it uses strokes that are more similar to the standard English alphabet, but it is not as fast to use nor as accurate. It is not as fast since it does not do as much optimization of the strokes for the most common characters. It is not as accurate since it does not use an alphabet that was designed for easy machine recognition. Unlike T-Cube, both Graffiti and Unistrokes can be used without looking at the screen, thus speeding up transcription or allowing the user to pay attention to a speaker. Graffiti is now available for most of the popular PDAs. The single stroke gestures used by SILK can be considered a shorthand for their more complicated multi-stroke counterparts.

7.6.3 Sketch Recognition

Negoponte defined sketch recognition as:

> the step by step resolution of the mismatch between the user's intentions (of which he himself may not be aware) and his graphical articulations. In a design context, the convergence to a match between the meaning and the graphical statement of that meaning is complicated by continually changing intentions that result from the user's viewing his own graphical statements. [Negroponte 1971] cited in [Negroponte 1973]

Negroponte's HUNCH [Negroponte 1973] was one of the early attempts at using inferences about a sketch to transform it into a more finished design. This architectural design system recorded stylus data from a pressure sensitive tablet. The system tried to make inferences of the user's intention at three hierarchical levels: what was meant graphically, in 2-D, what was meant physically, in 3-D, and what was meant architecturally. The system used heuristics to make some simple 2-D inferences, such as finding lines and corners. The rate of drawing a shape was used as a heuristic to determine whether a shape should be "cleaned-up" (such as making a square regular), left as is, or possibly used as a gesture. The system would not interfere with the designer until asked or triggered by recognized conflicts. HUNCH was meant to be "compatible with any degree of formalization of the user's own thoughts."

Another early sketch recognition system tried to rely on batch processing after the sketch was completed [Pavlidis 1985]. This work in scene *beautification* tried to "clean up" a completed drawing that had been roughly sketched by satisfying the inferred graphical constraints in the scene. Often, this method does not infer all of the proper relationships on the first attempt and the operation of beautifying is not idempotent, thus leaving designers in the dark on how many times they need to execute the beautifier to achieve the desired results. Trying to infer all possible relationships in a scene at once is much harder than trying to infer only the relationships between one object and the rest of the scene. In order to make this problem more tractable, more recent work has tried to make inferences as the sketch was drawn [Karsenty 1992]. SILK takes the same approach: by getting user assistance as each object is drawn, SILK further improves the speed and accuracy at which it can infer the correct widgets.

There has been a considerable amount of sketch recognition work done in the field of computer graphics. Like Negroponte's HUNCH, these systems are often meant to allow designers to create 3-D objects using a 2-D sketching interface. Examples include the SKETCH [Zeleznik 1996] system described previously (see Section 7.5), the IDeS system [Branco 1994], and the early work described in [Hosaka 1977]. A general review of the area is given by [Wang 1993].

7.6.4 Gesture Recognition

Buxton performed much of the important work in gesture recognition [Buxton 1986]. More recent work by Rubine at CMU [Rubine 1991b; Rubine 1991c] has been used in SILK. Rubine's algorithm uses statistical pattern recognition techniques in order to train classifiers and recognize gestures. These techniques are used to create a classifier based on the features extracted from several examples. In order to classify a given input gesture, the algorithm computes the distinguishing features for the gesture, and returns the best match with the learned gesture classes. This algorithm is limited to single-stroke gestures.

Unless a gesture set has a paper-based counterpart that the user is already familiar with, such as copy-editor marks, gestures can be hard to learn. Kurtenbach's marking menus are a novel attempt at making gestures self-revealing [Kurtenbach 1991a; Kurtenbach 1991b]. Marking menus combine gesture recognition with pop-up pie-menus [Callahan 1988]. Beginning users select a command from a series of pop-up pie-menus, while more advanced users can make a fast gesture that requires the same physical movements as required to traverse the pie-menus. The difference is that the advanced user performs the movements fast enough so that the menus never appear.

Gesture recognition is an important component of many of the pen-based interfaces discussed previously (see Section 7.4). The selection of a useful gesture set was one of the main topics of many of the listed research projects. In addition, I observed a considerable amount of user-testing of the gesture set for PenPoint when I interned with the GO Corporation in the summer of 1990.

7.6.5 Diagram/Shape/Image Recognition

Diagram editing is one of the common applications envisioned for pen-based computers and this area is quite similar to the sketch editing done with SILK. There have been several attempts at building diagram editors based on either gesture recognition (see Section 7.6.4) or visual languages (see Section 7.6.6). The thesis work of Zhao was an attempt at integrating both approaches [Zhao 1993a; Zhao 1993b]. The key idea is to use the diagram syntax to improve the low-level recognition. Unlike most visual language parsers which consider a complete picture as input, Zhao's system uses an incremental approach and also recognizes shapes that are made up of multiple-stroke gestures, independent of pauses between strokes. The system is based on the cooperation of two components. First, a low-level recognizer determines the class of symbols drawn [Zhao 1992]. Then, if certain spatial constraints are met, a high-level recognizer transforms the symbols into editing commands, thus allowing the system to work with an editor that is ignorant of the sketch recognition being performed. The incremental approach and the partitioning of the recognition into two pieces is very similar to what is done in SILK, although Zhao's system regularizes the shapes as they are recognized.

A multi-stroke shape recognition algorithm is also used in the Graphic Diagram Editor (GDE) [Apte 1993b]. Unlike Zhao's system, this shape recognizer uses a variable length time-out after each stroke, proportional to the size of the preceding stroke, to identify which strokes belong to which objects. A time-out solution is easy to implement, but it is unclear how well this works in practice. In addition, they compute the convex hull of the strokes in order to close objects in which the strokes are not connected. The recognizer uses a set of weighted filters to recognize six basic geometrical shapes – thus, their recognizer is not very flexible, nor easily extendable. GDE was implemented on top of the PenPoint operating system [Carr 1991; GO 1992]. The algorithm achieved a recognition rate of 98% on the six supported shapes when ten subjects were each asked to draw twenty examples. Again, like Zhao's system, GDE regularizes the shapes as they are recognized.

The PerSketch system [Saund 1994] is a novel attempt at exploring WYPIWYG (What You Perceive Is What You Get) image editors. That is, they adopt methods from computer vision to recognize several alternative image structures, just as a human may perceive an image in several different ways. For example, four separate lines forming a rectangle may be perceived as four lines or as a rectangle. Both inferences are valid and the PerSketch system would allow the user to manipulate the image with operations that make sense for either inference. Since the system tries to mimic how people would perceive the drawing, it would even allow manipulating the individual sides of a rectangle that was originally drawn with one stroke of the pen. "Pose matching" enables users to select among several interpretations by using a gesture to indicate the location, orientation, and elongation of the intended object.

Finally, there has been a considerable amount of work exploring visual languages and pen-based UIs at the University of Colorado. In addition to the work on recognizing architectural sketches (see Section 7.5), researchers at Colorado have explored diagram recognition for the capture and editing of visual languages on pen-based computers [Citrin 1995; Citrin 1996]. Informal experiments suggest that pen-based UIs are better than traditional palette-based interfaces for this domain [Apte 1993a; Citrin 1993]. The Colorado group had the novel idea of using inexpensive, early-model Apple Newtons as a pad-based input mechanism for a diagram editor with a recognizer running remotely on a desktop computer. The PDA performs low-level shape recognition, while the back-end computer performs higher level semantic recognition of the diagram using the Electronic Cocktail Napkin (see Section 7.5) as a recognition engine.

7.6.6 Visual Languages

Visual languages are another technique that can be used to recognize sketched input. Visual programming systems support languages that allow the user to specify a program in a two-(or more)-dimensional fashion [Myers 1990b]. Most visual programming systems are grammar-based and generate a parser from the definition of the grammar, which may be textual. This section gives an overview of the important work in the area and also discusses vmacs, a design system that is supported by visual language parsers.

Lakin's vmacs [Lakin 1986; Lakin 1989] was intended as an electronic design notebook for engineers. The system has been under development for many years and is currently offered as a commercial product. Like Kramer's system and Kolli's proposed Ideator (see Section 7.5), vmacs focuses on conceptual design. It allows the user to write or draw whatever they want. Later, the user can choose the appropriate visual language parser to add interpretation to different sections of the design. This is very similar to Kramer's dynamic interpreters, but lacks the sketch-based input and instead relies on mouse and keyboard. The system tries to be very flexible ("every graphic act which is not prohibited is permitted") and agile ("agility... requires touch-typing").

One way to parse a visual language is to allow the user to produce a picture using a standard visual editor. The picture and a description of the visual language are then fed into a "spatial parser," producing a structure analogous to a parse tree for textual languages [Golin 1990]. This was the approach taken by the Visual Programmer's Workbench [Rubin 1990], an attempt to develop a set of tools for visual languages analogous to the editors, lexical analyzers, parsers, compilers, and debuggers for text-based programming languages.

Since visual languages typically express their semantics using relative relationships, such as relative position, containment, or connectivity, language grammars that include graphical constraints [Helm 1991] appear to be a promising way to build visual language parsers. Most of the rules for SILK's widget recognition can be defined in terms of graphical constraints. It is possible that these rules could have been made declarative and then a constrained set grammar used to produce a parser. Unfortunately, these parser generation tools were not available for the programming environment I chose to use, Common Lisp and Garnet.

Commercial systems employing visual languages include Serius (reviewed in [Landay 1991]), LabVIEW [Instruments 1989], and Prograph [Pictorius 1995]. More complete surveys of visual languages can be found in [Chang 1986; Chang 1987; Shu 1988; Myers 1990b].

7.7 Behavioral Specifications

SILK's storyboarding mechanism is based on specifying screen shots from before and after an end-user action. Chimera [Kurlander 1993] and Pursuit [Modugno 1993; Modugno 1995] are both based on the before-and-after cartoon strip metaphor. SILK differs in that it allows the designer to specify what these actions are, rather than trying to infer this information from examples, as is done in Chimera and Pursuit. These systems are successful doing inferencing in their domains, graphical editing and graphical shell file operations, respectively.

SILK storyboards are similar to finite state transition diagrams. These diagrams have been used for UI specifications in the past, but have fallen into relative disuse because of problems with the exponential blowup of the specification and an inability to handle the dynamic nature of direct manipulation user interfaces. Statecharts [Harel 1987] overcome many of the problems of finite state transitions diagrams by adding hierarchy, concurrency, and communication. Wellner's Statemaster [Wellner 1989] is a UIMS that uses Statecharts for dialog specification. The proposed parallel storyboard extension to SILK tries to attack the state explosion problem in a way that is similar to Jacob's use of independent embedded state diagrams per object [Jacob 1986]. Again, SILK is not intended for creating the entire specification, but instead to illustrate the key sequences in the interface under consideration.

Two other relevant end-user programming systems are Agentsheets [Repenning 1992; Repenning 1993] and KidSim [Cypher 1995] (now called Cocoa). Both of these systems use graphical rewrite rules to allow the creation of dynamic simulations. The rewrite rules specify a graphical pre-condition that must be met. When it is met by the state of the screen, the screen state is changed by the graphical action specified in the rewrite rule. These systems focus on animated simulations, whereas SILK concentrates on end-user actions and the changes to the UI state that should occur upon those actions.

7.8 Inferring Constraints

Peridot [Myers 1986; Myers 1988] was one of the first tools to apply artificial intelligence (AI) techniques to aid in the construction of user interfaces. The user first draws widgets using a drawing tool that is similar to a standard structured drawing editor. Peridot then allows the user to specify much of an application's widget behavior by demonstrating how the widgets behave in response to end-user input. The system uses a simple rule system to make its inferences. SILK is similar in that it uses a rule system to infer the widgets, which have previously defined behaviors. Other demonstrational user interface tools that have followed in the Peridot tradition include Lapidary [Myers 1989], Druid [Singh 1990], Demo [Wolber 1991; Fisher 1992], and Marquise [Myers 1993].

One of the major complaints about Peridot was that it confirmed its inferences by asking questions. One of the principles behind SILK's design is that the inferences of the system should be kept out of the designer's way until they have a need to deal with the actual widget behavior. SILK indicates its inferences by both enabling a radio button that indicates the type in the **`SILK Controls`** window and by allowing the widget to behave when the designer manipulates it. Marks or symbols layered on top of the interface were used for feedback indicating graphical constraints in Briar [Gleicher 1994] and Rockit [Karsenty 1992]. In Rockit, the marks kept the user informed of the current inference of the system. This approach may be worth trying with SILK.

7.9 Summary

This chapter reviewed the large body of research related to electronic sketching and storyboarding. This body of work was focussed in areas relevant to SILK, such as sketching, user interface tools, informal interfaces, pen-based user interfaces, recognition technologies, and behavioral specifications. My research is unique in that it combines these technologies, which permits designers to benefit from the advantages of both sketching and electronic tools in the early stages of user interface design.

CHAPTER 8
Future Work

Previous chapters have outlined potential engineering extensions to SILK that would improve the usability and functionality of the tool. While most of the previous discussion considered additions that are simply a matter of implementation, this chapter contains a discussion of those extensions that require solving difficult research problems. The chapter closes with a few ideas on how to extend the ideas embodied by SILK to other domains.

8.1 Extensions to SILK

Most of the potential improvements to SILK involve easing the specification of new interactive behaviors. The other proposed improvements would allow SILK to fit in more closely with the way people currently work: collaboratively and on paper. This section discusses these improvements in more detail.

8.1.1 Specifying New Widget Behaviors

Recognizing new widgets refers to two related problems that are not yet solved in SILK: recognizing known widgets that are sketched in a manner that differs from the possibilities incorporated by SILK's rules and learning to recognize new widgets of the user's own design. Machine learning is a possible answer to both of these problems, and a few techniques for learning new rules were discussed in Section 3.7. The Electronic Cocktail Napkin [Gross 1996] shows that it is possible to build a system where the designer guides the system and thus permits the learning of new relationships and rules.

In a system that depends on the interactivity of the recognized objects, as SILK does, simply learning to recognize new objects is insufficient. The system must also be instructed on how the new widgets will behave. The informal design survey (see Section 2.2.3) showed that designers often want to draw new widgets whose behavior is *analogous* to that of a known widget. For example, designers frequently need to draw a new icon and specify that it should act like a button.

There are two interesting ways to make use of analogous behaviors. First, a user could train SILK to recognize a new widget and then inform the system that its behavior is analogous to that of an existing widget. The system could then try to infer which components of the two widgets serve the same purpose and thus infer how the individual components of the new widget behave. For example, the user may inform the system that the slider in Figure 8-1 is similar to a scroll bar. The system would then infer that the triangle is analogous to the scroll bar's elevator, free to move up and down from the top to the bottom of the rectangle.

The second solution relies less on inferencing and requires more direct involvement by the designer. After pointing out the analogous widget, the designer would select (using direct manipulation) the mappings between the components in the two widgets. The system would again try to infer how to adapt the existing behavior to the new widget. The interesting research lies in how to successfully perform these inferences and in finding a good interface for selecting analogous widgets and components.

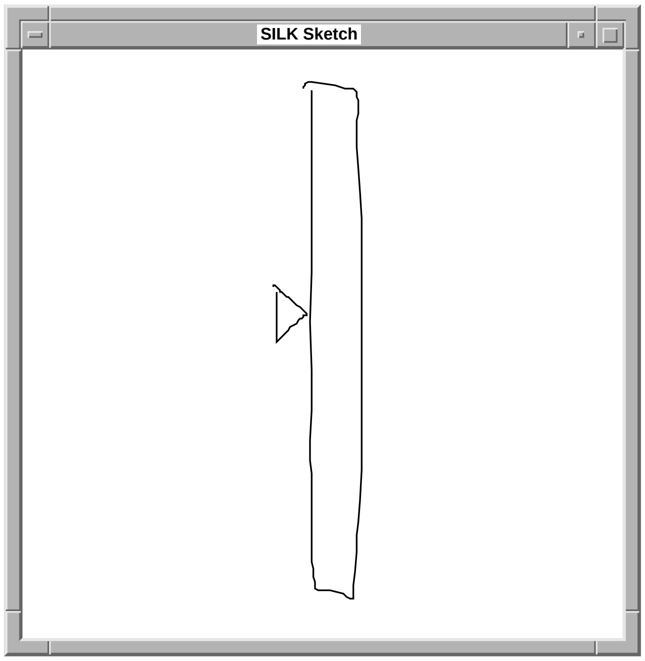


Figure 8-1. A user-defined slider - the triangle can slide up and down next to the rectangle.

8.1.2 Explicit Widget Type / Behavior Specification

There are two promising alternatives to recognizing user interface widgets. The first is to have the designer explicitly specify the widget type for a group of sketched objects. The second technique involves having the user specify the abstract behaviors that a group of objects should support. These methods are discussed in turn.

Requiring the designer to explicitly specify the type for all the interactive widgets in a sketch is a tempting alternative to a possibly error-prone recognition process. Explicit type specification would eliminate the problem of incorrect widget types, it would allow the designer to concentrate on the sketch without paying attention to SILK's inferences, and after sketching it only requires the designer's intervention on the set of objects that need interactive behaviors, alleviating the need for decoration mode. The problem with this technique is that it introduces a new inference problem: inferring how the given objects operate given the specified type. For example, if the designer draws an icon with a text label below it and calls it a button, the system must infer what to highlight when the mouse is held down on this button. This inference problem may or may not be easier than having to infer the type of sketched widgets and learning how to recognize new widgets. Another problem is that the designer may now have to do more work than with a recognition system that rarely errs.

The second technique, explicitly specifying abstract behaviors, trades-off the inference problem with more work for the designer. Using this method, a designer would attach low-level behaviors to individual widget components. For example, the designer may attach "highlight on mouse-down" to the icon part of the new button described above. Other supported behaviors may include slide (e.g., scroll bar) and pop-up (e.g., menu). Supporting the state encoded in radio buttons may be a bit harder. Again, the main drawback of this method is the designer has to spend a considerable amount of time attaching low-level behaviors to widgets that may be easily recognized and have known behaviors. A compromise may be to combine the two methods together for use in different situations, i.e., specify widget types for standard-looking widgets and specify low-level behaviors for custom-looking widgets.

Both of these explicit specification methods could be combined with the analogous behavior method described in Section 8.1.1 for specifying the new behaviors. Another possibility is to simply pick the types/behaviors from a list or palette. Determining if these techniques are better than widget recognition depends on how well the recognition engine works in practice. If the recognition rate is so low that it generally gets in the way of designers, one of these alternatives may work better. The recognition achieved during the SILK usability test (see Chapter 6, "Evaluation") is certainly too low, but it remains to be seen how well this could be made to work using some of the learning techniques described in Section 3.7.

8.1.3 Extending Storyboards

Another way to specify new behaviors is to storyboard them. The storyboarding examples given previously were for showing dynamic behavior between different widgets and screens, but storyboarding can also be used to specify the behavior of a single widget. For example, Figure 8-2 shows the screens and transitions necessary to define the behavior of a two-item radio button panel. The behavior of new widgets can be added in a similar same way.

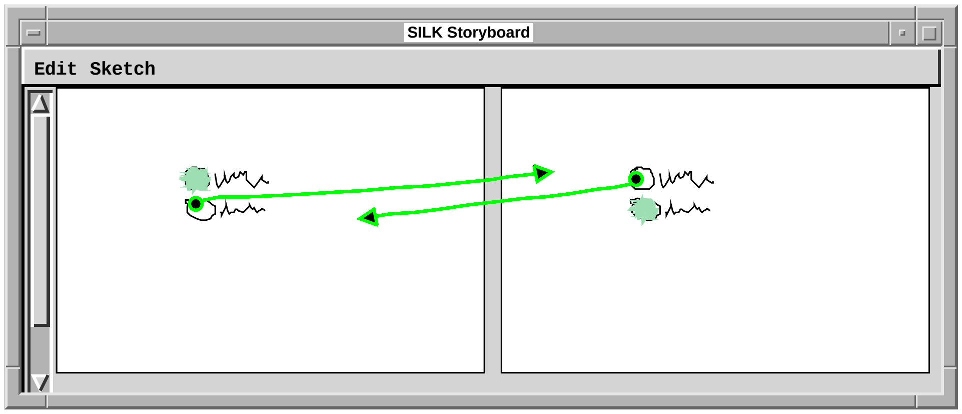


Figure 8-2. Storyboard to illustrate the behavior of a pair of user-defined radio buttons. Note that SILK's built-in radio buttons have a similar behavior by default.

8.1.3.1 Hierarchical Storyboards

The problem with this approach is that it would lead to a large blow-up in the number of screens for any storyboard containing a widget defined this way. One solution is to support hierarchical storyboards. The designer would first attach a storyboard to a region of a given screen. For example, to use the radio button illustrated in Figure 8-2, the designer would attach the storyboard to the region containing the radio button. Then, any use of the radio button in another (higher-level) storyboard would transparently include the attached definition for the radio button's behavior. When the resulting interface is tested, any events that occur within the bounding box of the radio button will be handled by the radio button's associated storyboard and the screen transitions would only affect its portion of the screen. This is a generalization of the technique used by participants 9 and 15 during the SILK usability test (see Section 6.2.1.2).

The same hierarchical design could be used to implement a palette of tools. The harder research problems occur when the designer wishes to make a screen transition that depends on the state of one of these newly defined widgets. For example, imagine the user clicks on the drawing area in the screen illustrated in Figure 4-6 on page 65. The designer would like to create either a rectangle or a circle depending on the state of the palette. Currently, this cannot be specified for even SILK's built-in widgets – finding a way to specify this easily is an open research question.

8.1.3.2 Drag Events

A previous discussion (see Section 4.7.4) centered on ways to extend SILK's storyboarding model to support other input events (e.g., double click and timer). The necessary changes to the storyboarding model needed for these events are quite straightforward: add new arrows types for different kinds of events.

Drag events are not easily supported by this proposed extension. For drags that are constrained to follow a path, the designer could simply draw a new *drag arrow* from the object in question along the path the object must follow. This would work for defining the elevator in a scroll bar (as seen in Figure 8-3). It is unclear what the designer could use to indicate that the object may be dragged anywhere in a specified region, rather than constrained to a single path.

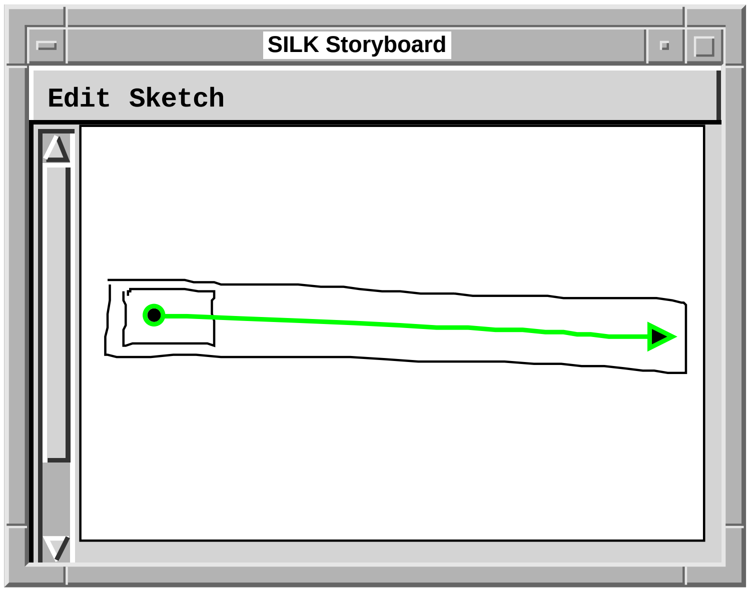


Figure 8-3. Proposed drag arrow specifies dragging a scroll bar elevator within the scroll bar.

Similarly, for drag and drop, the end-point of a *drag and drop arrow* could indicate the target object that can be dropped on. A *connected* arrow from the target object could indicate the transition that should take place after the original object has been dragged and dropped on the target object. For example, Figure 8-4 shows how a designer might illustrate filling in the region of a blob by dragging and dropping a paint can on the blob. This definition of drag and drop has a similar problem to that described previously for drag. How does the designer indicate that the object can only be dragged within a specified region? It may be enough to simply parameterize drag &

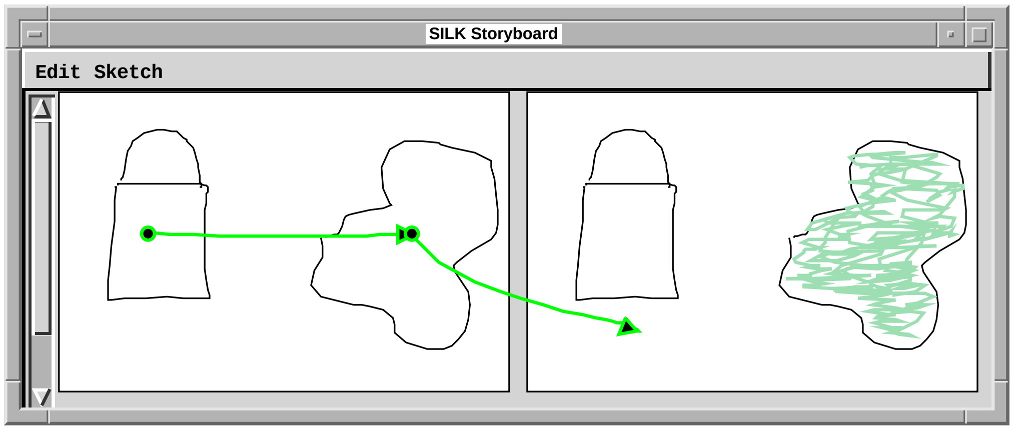


Figure 8-4. Proposed drag and drop arrow (left) indicates that the paint pail can be dropped on the blob to the right of it. Connected arrow (right) indicates that a screen transition should be made after the drop has occurred – in this case filling the blob with paint.

drop arrows such that the dragged object must either follow the path of the arrow or be permitted to move anywhere on the screen (but only dropped on the target object).

The difficult research problem is in determining which interactive behaviors are necessary for prototyping in the early stages of design. The goal of SILK was to make rough storyboarding quick and easy. This is why only single clicks are currently supported. If the system were to support all of the common interactive behaviors (e.g., the six interactive behaviors included in the Garnet interactor model [Myers 1990a]), storyboarding could become overly complex – thus hard to learn and time-consuming to use. If well-designed, it is possible that a more powerful storyboarding language could be learned incrementally. This is an exciting area for new research.

8.1.3.3 Multimedia Support

The timer event (proposed in Section 4.7.4) can help a designer mock-up a video application by representing the video as a sequence of rough sketches that automatically transition after a time-out. Representing audio output in an interface would also be easy if SILK included an interface for recording sound from a mike. In the spirit of SILK's rough prototyping model, the designer would use speech to simulate non-speech audio cues (e.g., saying "boom" to represent the sound of an explosion). Drawing arrows to icons representing the recorded sounds (stored below the associated screen) would indicate which event should cause the sound to be played back. Finally, researchers in speech recognition have expressed an interest in using SILK storyboards to prototype the dialog control in speech recognition applications.

8.1.3.4 Generating Program Structure from Arrows

One open-ended research problem is how to take better advantage of the program control indicated by the arrows in storyboards. That is, can program structure be derived from the transitions used by a designer in a SILK-based prototype? Solutions to this problem have two different applications. First, can the arrows be used to infer more general behaviors at storyboarding time and thus reduce the complexity of the storyboard needed to illustrate a given behavior? An example of this (as discussed in Section 4.7.2) is to infer that the rectangle in Figure 4-1 on page 57 should be rotated 60 degrees each time the button is pressed. It would be challenging to build a system that can both make these inferences and give good feedback to the designer about the results.

The second area that may be helped by analyzing arrow transitions is in improving the output of SILK's transformation process (see Section 5.2). Can the arrows be used to generate the source code for callbacks in the final application? This may be easy for dialog boxes and simple screen changes, but quite hard for more sophisticated behaviors. The question of how do this or deciding whether the transitions even mimic part of the callback structure are both open issues.

8.1.4 Collaborative Design Support

In addition to making SILK more powerful for the individual designer, it may be fruitful to consider that interface design is generally a group task. I have seen this in the design literature, in the comments collected during the designer survey (see Chapter 2, "Informal Survey of Designers"), and by speaking with designers. Many of these designers have pointed out the need for using design tools collaboratively. For instance, one designer wrote, "The entire engineering team must be able to see it and work with the sketch." As mentioned earlier (see Section 5.1), one way to support group work is by having annotation layers for different members of a design team. If these team members are sharing the same design file, then access and revision control mechanisms are necessary.

As a design becomes more complex, simple drawn and written annotations may not be enough. Members of the design team may wish to attach multimedia annotations, such as speech, video, and animations. It is important to study what types of annotations are needed and how they can be easily created, maintained, and viewed by all members of the team. It may also be important to make viewing a context-sensitive operation so that individual team members are initially shown only the annotations and level of design detail that are particularly relevant to them. Determining how to easily specify and infer such filtering properties is an open area of research.

In some cases this support for asynchronous design may be enough. But, often group design occurs synchronously. There are two cases that must be considered: supporting design teams that are collocated (that is, working in the same room) and supporting design teams that are distributed at remote locations. In the first case, SILK could be extended to support multiple input devices (e.g., two pens) working at the same time. Another avenue of research is the use of speech recognition to improve the inference engine results. The hypothesis is that if multiple designers are working together, one may speak about what she is designing while sketching. Doing simple keyword spotting on this speech may permit SILK to make its inferences with a higher level of confidence.

As in the collocation case, multiple remote designers may wish to work on the same design at the same time. Shared whiteboard tools have shown that this basic functionality is not too hard to support. The difficulty comes in managing the work. For example, which designer controls the floor at any one time? If the designers were in the same room, they would use voice and other physical cues to enforce this control. The important research issue is finding a proper floor control mechanism that does not constrain this highly creative task.

SILK's storyboarding mechanism could prove useful for communicating a design idea to a remote collaborator, much in the way that I have shown it was useful in the case of collocated designers and engineers. The existing feedback mechanisms (i.e., highlighting screens, objects, and transitions in the storyboard) would help inform the remote user on what was causing the screen transitions and the visual layout of the storyboard would help when giving overviews of the organization and structure of a design idea.

There is a considerable amount of previous work in the field of collaborative work, especially for supporting meetings. Finding techniques from this work that are general enough for use with interface design would be the best approach for adding collaborative support to SILK. SILK may change considerably, as it seems that when multiple users are added to a design task, the problems have less to do with the design tool itself and more to do with supporting an environment for communicating design ideas and for storing and retrieving previous ideas and comments in an easily accessible design repository.

8.1.5 Paper-based Sketch Support

Another possibility for better supporting collaborative design situations is to integrate SILK with existing paper-based sketches. This way designers can get the advantages of paper in meetings: paper is very portable and easy to share. Later, the paper can be scanned in, line following algorithms can be used to transform the scanned bitmaps to strokes composed of point vectors, and then SILK's inferencing algorithms can be performed on the strokes. The design can now be treated as if it was drawn on-line. The designer can check the widgets and perform any necessary corrections. At this stage, storyboard transitions can be added and the designer now has an *interactive* prototype. What would be lost though is the interaction with the designer which helps to improve the inference capabilities of the system.

8.2 Additional User Studies

Although paper-based sketching clearly has some advantages over computer-based tools, the SILK usability test (see Chapter 6, "Evaluation") showed that designers could use SILK for designing real user interfaces. In addition, the test showed that the designers liked the roughness afforded by electronic sketching as well as the interactivity achieved using storyboards. Further user studies could help demonstrate more firmly which components of SILK are most important for effective interface design.

For example, a two condition study, one with storyboards and one without, could help determine how important electronic storyboarding is to the design and communication of interfaces ideas. Another study might compare versions of SILK with and without widget recognition. This would help to more accurately determine whether or not the existing recognizer is helpful. Similarly, a Wizard of Oz study could be performed to determine how well a perfect recognizer would work. That is, one of the experimenters would supply SILK with the correct recognition inferences in real-time so that it appeared as if the system made no inference errors. Finally, comparing SILK to a version that cleaned-up the sketch would help to determine how important the roughness of the sketch is for both the designer and eventual design reviews.

In addition, as SILK is made more robust, it could be directly compared to an existing design tool (e.g., Director). One such experimental design is to have two groups of designers perform a design task, half using SILK and half using the commercial tool. A comparison of design time, number of different designs produced, complexity of the designs, quality of the final designs, and subjective ratings by the designers could help establish whether SILK is a better tool for early stage user interface design. A similar study could be performed to compare SILK with paper-based sketching. Finally, a hybrid study would compare SILK and paper to a suite of traditional tools (e.g., paper, Illustrator, and Director).

8.3 Domains for Informal Communications

In addition to SILK's use in the early stages of user interface design, the same core techniques can be used in a range of other applications. This section contains a discussion of two such potential applications. Like SILK, these applications work well as tools for informal communications. The first application, Web page design, is very similar to user interface design. The second application, interactive dynamic presentations, is quite distinct from UI design.

8.3.1 Web Page Design

Web page design is quite similar to user interface design. Both web page design and UI design are about communicating information through an interactive medium. Both mediums take advantage of interactive widgets (e.g., scroll bars and buttons) and both must be designed with the visual presentation in mind. The main difference is that web pages tend to be more document-oriented than conventional application software and that the primary model for moving between web pages is the single click of a hyperlink (e.g., a button).

The fact that web page design is related to document design has led to a strong market for graphics design professionals in this new on-line realm. As discussed previously, these designers are trained to make rough sketches when designing documents to get a good idea of what the overall structure of the document will be before focusing on the details. This is exactly the same design model that SILK supports. In addition, the web hyperlinks correspond directly to SILK's storyboard arrows. The only major change necessary to support web page design would be transforming the SILK storyboard to HTML (the language for specifying web pages). Additional web-specific widgets could be added as necessary. For example, Figures 8-5 and 8-6 illustrate a rough version of the CNN Interactive page (`http://www.cnn.com/`) from October 19, 1996. No changes to SILK were necessary to mock-up these web pages.

8.3.2 Informal Dynamic Presentations

The attractive features of SILK – speedy creation, rough presentation, and ease of illustrating interactive behaviors – are useful for domains other than interface design. A new application, which I term informal dynamic presentations (IDPs), involves using computer-based media as a visual aid in lectures, meetings, and computer-based tutoring. Unlike existing presentation applications, IDP tools would be oriented towards allowing portions of the presentation to have dynamic behavior. For example, a presenter may be able to demonstrate a working simulation of a mechanical part *inside* of a slide to illustrate how the part works. A user of an IDP tool would be able to manipulate other common presentation objects (e.g., graphs, flow charts, and organizational charts) on the fly.

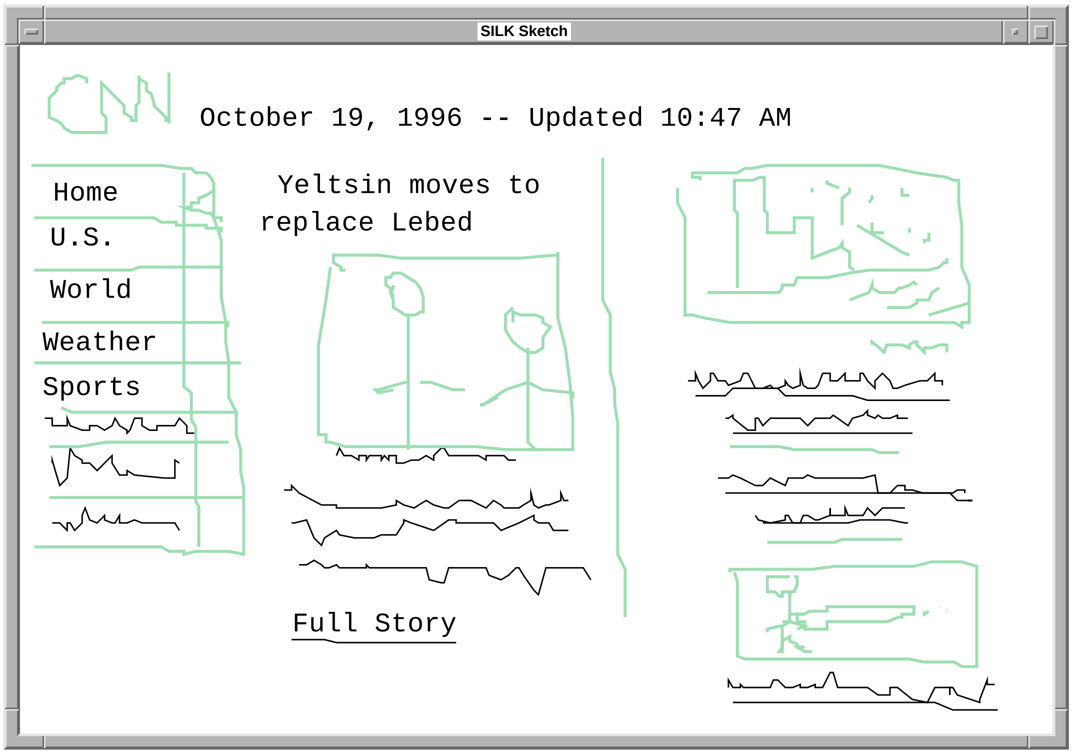


Figure 8-5. First screen for rough mock-up of CNN's web page.

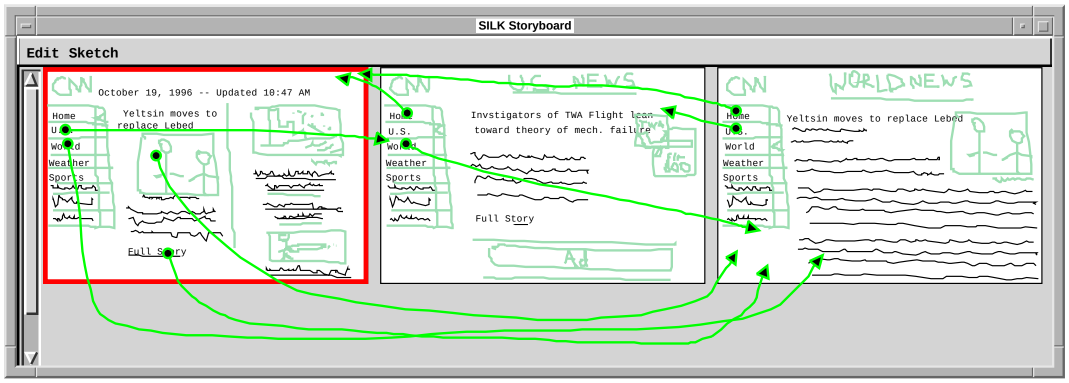

SILK Storyboard
Edit Sketch
CNN
October 19, 1996 -- Updated 10:47 AM
Home
U.S.
World
Weather
Sports
Yeltsin moves to
replace Lebed
Full Story
US NEWS
Invstigators of TWA Flight lean
toward theory of mech. failure
Ad
WORLD NEWS
Yeltsin moves to replace Lebed

Figure 8-6. Storyboard for rough mock-up of CNN's web page.

The main difference between this proposed tool and existing tools would be an orientation towards design and manipulation at presentation time. Since most people can sketch rough drawings faster than building finished-looking versions with a drawing program, the necessary diagrams could be sketched very quickly. In addition, storyboards could be used to rapidly illustrate behaviors. Thus, the user could make rough sketches for the dynamic models and specify their behavior interactively – before or possibly during the presentation. In addition, these models, like user interfaces, are intended to support several different behaviors on any one slide, depending on the dynamic needs of the talk. This is in contrast to existing presentation tools which promote a linear presentation of slides.

SILK's rule system could be adapted for recognizing common presentation diagrams (e.g., charts and graphs), so as to immediately support some common presentation-time behaviors. For example, clicking on a node of a graph might enlarge it and thus focus attention on it. In addition, the storyboarding mechanism could be used to illustrate other manipulations as needed. Lightweight dynamic presentation tools could open up a new area of interesting research.

8.4 Summary

There are several interesting, yet challenging, extensions that can be made to SILK. These range from improving SILK's ability to illustrate behavior to using more of the information embedded in storyboards when transitioning to another tool later in the design process. This chapter has made a brief attempt to describe some of the problems that must be solved for these extensions to be viable as well as describing other new domains for SILK's informal communications model.

CHAPTER 9
Conclusions

This chapter summarizes the problems and successes of SILK's approach to user interface design. In addition, the contributions of the research are reviewed.

9.1 Benefits of SILK's Approach

The design of SILK was based on discussions with the intended users of the system. Through questionnaires and site visits, hand-drawn sketches and storyboards were found to be common user interface design tools. Designers reported that current user interface construction tools are a hindrance during the early stages of interface design.

SILK overcomes these problems by allowing designers to quickly sketch an interface using an electronic stylus. Unlike paper sketches, SILK's storyboards allow the designer or test subjects to interact with the sketch before it becomes a finalized interface. The SILK usability test showed that designers saw a lot of promise in electronic sketching and storyboarding – they were able to investigate multiple interface designs quickly. The usability test also illustrated that electronic sketches and storyboards encourage a useful dialog between interface designers and software engineers. Together, they can smoothly view, test, discuss, and modify early design ideas. An interactive sketching tool that supports the entire interface design cycle has the potential to enable designers to produce better quality interfaces in a shorter amount of time than current tools.

9.2 Limitations of SILK's Approach

SILK has a number of limitations, some that were not within the scope of this research and some that were. This section first delineates some of the areas in which this work was not intended to contribute and then discusses the inherent limitations of the approach.

9.2.1 Outside of Research Scope

There are simply too many important facets of a design tool to study all of them within the confines of a Ph.D. thesis. However, many of these issues are essential to a functioning system. Thus, a commercial system must provide some solution to most of these problems. First of all, there are no successful algorithms for multi-stroke gesture recognition available in the public domain. SILK used Rubine's single-stroke algorithm. This turns out to be primarily a limitation for recognizing rectangles. The dissertation discussed several possible solutions to this limitation (see Section 3.2.1.1).

The thesis did not introduce any new artificial intelligence techniques for inferring widget types. Instead, a simple rule system was used. As discussed in Section 3.7, machine learning techniques would have been helpful for both inferring widgets from SILK's built-in set and for training the system to recognize new widgets that are drawn by designers. A better inference mechanism combined with a more robust gesture recognizer would allow the designer to pay less attention to the results of the recognition process and instead concentrate solely on their design.

Finally, SILK has no support for creating interfaces that include 3-D graphics, speech recognition, and audio/video output. Such multimedia interfaces have become more prominent since the start of this work. Other than 3-D graphics, SILK has the potential to help design interfaces that contain any of these other input and output modalities (see Section 8.1.3.3).

9.2.2 Inappropriate Domains for SILK

Although SILK was designed as a tool for early, creative, user interface design, there are some interface design tasks for which SILK's sketch-based approach is clearly inappropriate. For example, SILK would not be very useful for designing small changes to the interface of an existing product. The design team would be better served by making the changes to the existing product and trying the resulting system out with users. There are also specific application domains where SILK would not be helpful. These domains include non-visual interfaces, dynamic interfaces, and textual interfaces. The rest of this subsection treats each of these in turn.

First, interfaces that are non-visual in nature, such as telephone interfaces or interfaces for the blind that rely primarily on speech recognition and audio feedback, will not get much help from a sketching tool. SILK could be used to help the designer structure their thoughts at a systems level, but the tool would be of no help when designing the actual speech/audio user interface.

Second, while SILK's storyboarding model could be extended to support animation, dragging, and other dynamic events, it is not clear that this model is appropriate for designing application interfaces that have few static elements and are primarily made up of time-varying information. Such applications include flight simulators, video players, or complex computer games.

Finally, rather than supporting free-form text editing, SILK is oriented towards the interfaces found in graphical editors and simple forms-based applications. Thus, SILK would not be an appropriate tool for designing the interaction found in the work window of a word processor or spreadsheet. Fortunately, there are few examples of dynamic or text-based interfaces that do not include standard widgets around them. SILK could still be used for designing those parts of the user interface.

9.3 Contributions

SILK demonstrates that electronic sketches and storyboards can be successfully used to prototype user interfaces in the early stages of design. This section groups the specific contributions of the research into concepts and techniques, artifacts, and experimental results.

9.3.1 Concepts and Techniques

To reach its goals, this dissertation introduced several new concepts and techniques. First, I developed a new methodology, electronic sketching, for designing user interfaces. In addition, SILK's electronic storyboarding is a novel method for illustrating the behavior of a user interface without requiring conventional programming. SILK also includes feedback and editing techniques for manipulating these behavioral specifications, which are sorely lacking in current demonstrational and direct-manipulation systems. Finally, the sketch recognition system shows that in the domain of 2-D user interfaces it is possible to recognize common user interface widgets. The complete set of concepts and techniques are listed below.

Concepts and Techniques:

- electronic sketching for UI design
  - gesture recognition for recognizing and editing primitive components
  - automatic recognition of sketched widgets
  - automatically adding basic interactive behavior to sketched widgets
- electronic storyboarding
  - a way to add dynamic behavior to interface screens
  - a visual representation for storing related screens
  - using arrows for indicating screen transitions
  - graphical feedback at run-time to indicate what caused the last transition
- design memory mechanisms
  - sketched, typed, and written annotations on designs
  - ability to search typed design annotations
- screen transformations to real widgets in a standard look-and-feel

9.3.2 Artifacts

The research also produced two artifacts, the system itself and a widget recognition algorithm. SILK is the first tool designed for use during the early phases of interface design by professional user interface designers. By supporting the transformation of SILK sketches to a standard widget format SILK can assist all phases of the interface design cycle. The research produced a complete enough implementation for use by designers wishing to experiment with this new methodology. In addition, the dissertation includes a widget recognition algorithm that can be applied to other domains.

9.3.3 Experimental Results

The dissertation also describes several experimental results. First, the informal survey of designers illustrates that designers are dissatisfied with the existing tools and methods used in the early stages of user interface design. Usability testing shows that SILK is indeed usable for quickly generating several interface designs in these early stages. Several of the designers were even able to use it to create new interactive widgets that were not built-in to the system. The testing also illustrates that SILK allows designers to effectively communicate design ideas to engineers. In fact, the designers and engineers were able to discuss the designs and make changes to them while the engineers were present. The evaluation also shows that designers are excited by the prospects of using electronic storyboarding instead of programming languages for illustrating interactive behaviors. Finally, SILK demonstrates a domain where sketching is appropriate as the primary interface to a system.

9.4 Final Remarks

This thesis set out to investigate a way of enabling user interface designers to take advantage of computer-based sketching tools in the early stages of user interface design. The research met its goal. This dissertation discusses interesting solutions to many problems. For example, to keep designers and evaluators from focussing on the wrong details in the early stages of design, I developed the idea of sketching widgets, recognizing their types, and leaving them rough. To address the problem of illustrating interactive behavior, SILK introduced electronic storyboards. Finally, to support the later stages of design, screen transformations were added. Computer-based sketching is a key step to a future in which much of a user interface will be illustrated, created, and tested by a user interface designer rather than by programmers writing the code. Finally, this work shows the promise of using informal communications techniques, such as sketching, in several other domains.

APPENDIX A
Design Survey Materials and Data

This appendix includes the materials and data used for the design survey described in Chapter 2. Section A.1 contains the original design questionnaire e-mailed to approximately 300 subscribers of the visual design mailing list. Section A.2 includes tables that summarize the answers of the 31 designers who responded. Note that the answers shown for question 3b were derived by performing a simple content analysis on the written answers to that question.

A.1 Questionnaire

I'm a Ph.D. student at Carnegie Mellon and I am developing a UI design tool. I'd appreciate it if any professional designers out there would take 10-15 minutes to respond to the following survey concerning your use of interface design tools.

James Landay

1. What is your background?

__ graphic design __ art __ other (please specify)

2. How many years have you been designing user interfaces? __

3a. When you begin to design an interface, what tools do you use? (check all that apply)

__ pencil & paper __ whiteboard and markers __ software

3b. Why? What are the strengths and weaknesses at this stage of design?

4. Have you ever used interface builders or prototyping tools? Please list which ones along with their strengths and weaknesses.

5. I'm working on an interactive tool that allows designers to quickly sketch an interface using an electronic pad and pen. Unlike a paper sketch, this electronic "sketch" will respond to user input to allow testing of the interactions, and will also support editing of the sketched interfaces. When the designer is satisfied with this early prototype, the system will transform the sketch into a more finished interface in the specified look-and-feel of a standard graphic user interface, such as Motif or the Macintosh. This transformation takes place with the guidance of the designer.

a) What do you think would be the strengths and weaknesses of this tool?

b) Would you be interested in using such a tool?

6. What kinds of interfaces do you typically sketch? Please estimate what percent of your work is in designing interfaces of the following types:

__ Widgets from a toolkit (buttons, scroll bars, etc.)

__ Custom widgets you designed (new kinds of buttons, etc.)

__ Static graphics and decorations (icons, etc.)

__ Interactive graphical objects (rectangles, circles, or icons that can be edited)

__ Multimedia (video, sound, CD-ROM, etc.)

__ Other (please describe)

7. I'd appreciate it if you could send me copies of actual sketches drawn early in the design process for a user interface. We can use these to see what types of widgets people sketch and how they draw them. This will allow us to tune our system to recognized these widgets. All sketches sent will be kept confidential (if you can't send new ones, sketches from older products or designs are also welcome).

Table A-1. Survey data for designers with a visual design background.

Table A-2. Survey data for designers without a visual design background.

All Respondents			
Question	**Overall Average or %**	**Visual Average or %**	**Non-Visual Average or %**
1. Visual designer?	58%	count = 18	count = 13
2. Years experience	5.8	6.1	5.3
3a. In early stages of design,			
sketch?	94%	100%	85%
use software?	58%	56%	62%
3b. Why sketch?			
faster?	55%	50%	62%
portable/collaborative?	45%	33%	62%
perspective/organization?	32%	39%	23%
easier/versatile?	29%	33%	23%
details don't get in way?	16%	22%	8%
allows non-standard UIs?	6%	11%	0%
4. UI Tools Used:			
HyperCard (HC)?	48%	33%	69%
Director (D)?	45%	61%	23%
Visual Basic (VB)?	29%	33%	23%
HC, D, or VB?	77%	83%	69%
UI builder?	48%	44%	54%
5b. Interested in SILK?	90%	83%	100%
6. Elements Sketched:			
Standard widgets	28%	26%	30%
Custom widgets	17%	20%	12%
Static graphics	21%	25%	15%
Interactive objects	7%	4%	12%
Multimedia	10%	9%	12%
Other	17%	16%	18%

Table A-3. Mean values for survey question results.

The following items were listed for the “Other” category in question 6: window layout, interactions spanning multiple widgets, physical product user interfaces, splash screen art, static dialogs, animated icons, text layout, and UI structures / architectures.

APPENDIX B
Evaluation Materials

The following sections contain all the forms that were given to the participants before, during, and after the SILK usability test. See Chapter 6, “Evaluation”, for a description of the experiment itself.

B.1 Consent Forms

This section contains the consent forms given to the designers (Section B.1.1) and engineers (Section B.1.2).

B.1.1 Designer Consent Form

Carnegie Mellon University
Consent Form

Project Title: SILK
Conducted By: James A. Landay, Computer Science Department

I agree to participate in the observational research conducted by students or staff under the supervision of Dr. Brad Myers. I understand that the proposed research has been reviewed by the University’s Institutional Review Board and that to the best of their ability they have determined that the observations involve no invasion of my rights of privacy, nor do they incorporate any procedure or requirements which may be found morally or ethically objectionable. I understand that my participation is voluntary and that if at any time I wish to terminate my participation in this study I have the right to do so without penalty. I understand that I will be paid $25 for my participation when I have completed the experiment.

If you have any questions about this study, you should feel free to ask them now or anytime throughout the study by contacting:

Dr. Brad Myers
HCI Institute, School of Computer Science
412-268-5150
bam@cs.cmu.edu

You may report any objections to the study, either orally or in writing to:

Susan Burkett
Associate Provost
Carnegie Mellon University
412-268-8746

Purpose of the Study: I understand that I will be using some interface design tools. I know that the researchers are studying how people perform user interface design. I realize that in the experiment I will create some user interface designs over 2-4 hours. I am aware that I will be videotaped during the experiment so that the researchers can examine how the interface design tools supported the task.

I understand that the following procedure will be used to maintain my anonymity in analysis and publication/presentation of any results. Each participant will be assigned a number. The researchers will save the data and videotape files by participant number, **not** by name. Only members of the research group will view the tapes in detail. No other researchers will have access to these tapes.

I understand that in signing this consent form, I give Dr. Brad Myers, and his associates permission to present the results of this work in written/oral form, without further permission from me.

______________________________ ______________________________
Name (please print) Signature

______________________________ ____________ ____________
Telephone Date Subject #

Optional Permission: I understand that the researchers may want to use a short portion of videotape for illustrative reasons in presentations of this work. I give my permission to do so provided that my name will not appear.

_________ YES _________ NO (Please initial here __________)

B.1.2 Engineer Consent Form

Carnegie Mellon University
Consent Form

Project Title: SILK
Conducted By: James A. Landay, Computer Science Department

I agree to participate in the observational research conducted by students or staff under the supervision of Dr. Brad Myers. I understand that the proposed research has been reviewed by the University's Institutional Review Board and that to the best of their ability they have determined that the observations involve no invasion of my rights of privacy, nor do they incorporate any procedure or requirements which may be found morally or ethically objectionable. I understand that my participation is voluntary and that if at any time I wish to terminate my participation in this study I have the right to do so without penalty. I understand that I will be paid $8 for my participation when I have completed the experiment.

If you have any questions about this study, you should feel free to ask them now or anytime throughout the study by contacting:

Dr. Brad Myers
HCI Institute, School of Computer Science
412-268-5150
bam@cs.cmu.edu

You may report any objections to the study, either orally or in writing to:

Susan Burkett
Associate Provost
Carnegie Mellon University
412-268-8746

Purpose of the Study: I understand that I will be using/observing the use of some interface design tools. I know that the researchers are studying how people perform user interface design. I realize that in the experiment I will discuss some user interface designs over 1-2 hours. I am aware that I will be videotaped during the experiment so that the researchers can examine how the interface design tools supported the task.

I understand that the following procedure will be used to maintain my anonymity in analysis and publication/presentation of any results. Each participant will be assigned a number. The researchers will save the data and videotape files by participant number, **not** by name. Only members of the research group will view the tapes in detail. No other researchers will have access to these tapes.

I understand that in signing this consent form, I give Dr. Brad Myers, and his associates permission to present the results of this work in written/oral form, without further permission from me.

Name (please print)	Signature	
Telephone	Date	Subject #

Optional Permission: I understand that the researchers may want to use a short portion of videotape for illustrative reasons in presentations of this work. I give my permission to do so provided that my name will not appear.

__________ YES __________ NO (Please initial here ___________)

B.2 Initial Information Read to Participants

Initial Information Read to Participants

The basic instructions below are based on a instructions written by Kathleen Gomoll [Gomoll 1990].

Description of the observation (in general terms):

• You are helping me by trying out a user interface design system.

• I’m testing the system; I’m not testing you.

• I’m looking for places where the system may be difficult to use.

• If you have trouble with some of the tasks, it’s the system’s fault, not yours. Do not feel bad; that’s exactly what I’m looking for.

• If I can locate trouble spots, then the developers can go back and improve the system.

• Remember, this is totally voluntary. Although I don’t know of any reason for this to happen, if you become uncomfortable or find this objectionable in any way, feel free to quit at any time.

Demonstration of equipment:

• You will be using this workstation/PC to work on a user interface design. Are you familiar with how to use this equipment? The mouse? The keyboard?

• Please do not move or resize the windows. They are positioned to be seen by the video camera.

B.3 Designer Demographic Data Sheet

Designer Demographic Questions

Subject No.:_____

Age: _____

Sex:

_____ Male

_____ Female

Occupation:

_____ Student (Major: _________________________________)

_____ Professional Designer

_____ Other (please specify)

How many years experience have you had in doing user interface design in a professional capacity (*i.e.*, you were paid for it)?

Have you taken a course in user interface design in which you spent a significant amount of time producing actual interface designs?

If yes, which courses were these?

Which user interface design tools have you used (*e.g.*, HyperCard, Director, Visual BASIC, Antisexist, Toolbook, *etc.*)?

Which drawing or painting programs have you used? (*e.g.*, MacDraw, Corel Draw, *etc*.)?

Which HCI books have you read?

Have you read *Usability Engineering* by Jakob Nielsen?

B.4 Engineer Demographic Data Sheet

Engineer Demographic Questions

Subject No.:_____

Age: _____

Sex:

_____ Male

_____ Female

Occupation:

_____ Student (Major: ________________________________)

_____ Other (please specify)

How many years experience have you had programming in a professional capacity (*i.e.*, you were paid for it)?

Have you taken a course in user interface design in which you spent a significant amount of time producing actual interface designs?

If yes, which courses were these?

Which user interface design tools have you used (*e.g.*, HyperCard, Director, Visual BASIC, NeXTStep, Toolbook, *etc.*)?

Which drawing or painting programs have you used? (*e.g.*, MacDraw, Corel Draw, *etc.*)?

Which HCI books have you read?

Have you read *Usability Engineering* by Jakob Nielsen?

B.5 SILK Demonstration Script

Before giving the participants the SILK Tutorial, I read the following information to them while performing the actions indicated in italics.

SILK Demo Script

Overview

The idea of the system is that the designer sketches a user interface using the mouse and then leaves it in that sketchy state. The system tries to infer what the different types of user interface elements are in the sketch and they become active. The designer can then attach actions to them using storyboards.

Basic Primitives

There four are basic primitives that are recognized. For example, rectangles.
draw rectangle and show feedback

You can only draw that with a single stroke (meaning you hold down the left mouse button throughout the shape.) So you can not draw the rectangle with 4 lines.

It also recognizes circles, lines, and text.
draw circle
draw line
draw squiggly for text

Corrections/Learning

At any point the system may get one of these primitives wrong. For example you may draw a small circle and the system may have inferred it was a rectangle.
draw a small circle
click on button to correct

The system will learn from these corrections and learn how you draw your shapes. When the system makes a mistake you should correct it instead of deleting it and starting over.

Gestures

The system supports gestures that are drawn with the MIDDLE button held down. For example, you can draw an X-shape with a single stroke through objects to delete them.
draw a delete gesture to delete some items

Widgets

The system takes the basic primitives we previously drew and combines them together to get user interface widgets. For example if you draw a small circle with some text to the right of it, the system says it is a radio button.

draw a radio button

Another alternative inference for that is a check-box since a check-box is defined as a square with some text next to it. Since these two widgets are similar, the system reports both inferences.

draw a check-box
click on the radio button and click new guess

If you meant check box, we can click on new guess and we will get check-box. You can see it also gives the other alternative inferences: radio button or multiple selected objects. You can change the inference by selecting on one of those alternatives.

Selection

You can move or resize any of the selected objects by dragging on the handles. The white handles are for moving and the black handles are for resizing.

demonstrate moving and resizing

You can select multiple objects at once by clicking on the object and then while holding down the shift key clicking on a second (or more) object.

demonstrate multiple selections

Sometimes you can use these selections to give SILK a hint as to what primitives it should group together in a new widget.

draw radio button with text too far from label
multi-select
new-guess button

If you don't wish to give SILK that hint, you might just move the items closer together.

draw radio button with text too far from label
move label closer
new-guess button (with only one item selected)

Decorations

Sometimes it is important to draw static graphics that you do not want to be recognized as a UI widget. You can do this by switching to decorate mode.

switch to decorate mode and draw some things

All editing (e.g., to delete it) of these graphics needs to occur in decorate mode, but otherwise you can use them anywhere.

Storyboarding

What you can then do is attach actions to the widgets. For example, we can have a window with some buttons below it.

draw a window
draw 2 buttons below it

Now you can copy these screens to the storyboard window by selecting “Copy Screen to Storyboard”.

copy screen to storyboard

Now we can change the original.

draw a 1 in the window
copy screen to the storyboard
use backspace to delete the 1 and draw a 2
copy screen to the storyboard

We now have these screens and we would like to tell the system what actions cause screen transitions. So we draw arrows between objects and screens.

draw an arrow from 1st button to screen 1
draw an arrow from 2nd button to screen 2
draw arrows from all other buttons back to 1st screen

We then select the first screen and copy it back to the sketch window. We now switch to run mode.

copy 1st screen to sketch
switch to run mode
demonstrate running storyboards

B.6 Practice Design Task

The following practice task was given to the design participants on a paper sheet along with the SILK Tutorial and Reference (see Appendix D).

Designer Practice Task

Design an interface that would allow potential home buyers to view houses on the market through their home computer. The service should allow the user to specify various search criteria such as price and location, after which the system would show a series of photos for each of the available houses and allow the customer to set up an appointment to see a house for real.

You can assume that the computer has some type of network connection and your interface does not need to deal with making the connection with the remote home market data.

This document will lead you through building two radically different, but high quality versions of this interface. You should follow the steps in the document and try to build the two interfaces in order to become familiar with the tools for the next design task.

Sometimes the system will make a mistake. That is, the system did not do what you expected it to. Please inform us when this happens so that we can take note of it.

B.7 Measured Design Task

The following task was given to the design participants on a paper sheet. They were permitted to keep their copy of the SILK Tutorial and Reference (see Appendix D).

Designer Task 1

Design multiple interfaces for a system to provide weather information to travelers. *TRAVELweather* can provide information about the weather for the current day and can give predictions for the next two days. The interface is to be used on a window-based graphical personal computer with a mouse.

The user should be able to enter the date and time they wish a forecast for. The system should display a map of the region in question with the weather data overlaid on the map. The types of data that the system should be able to display are temperature (in either Fahrenheit or Celsius), precipitation, and wind speed/direction.

You should try to **come up with as many *radically different* design ideas** as you can. Your goal for this task is to explore the possible design space and eventually present several good alternatives to the rest of your design team or a client. You should save or label the different designs that you come up with so that we can keep them separate from each other. Remember, the designer who produces the best interface (as chosen by our panel of judges) will win $100.

Sometimes the system will make a mistake. That is, the system did not do what you expected it to. Please inform us when this happens so that we can take note of it.

B.8 Engineering Task Description

The following task description was given to the engineering participants on a paper sheet.

Engineer Task

You will be shown some preliminary designs for a user interface to a weather information system. The person showing you the designs is a user interface designer and has just spent the last few hours working on these early designs. You are to pretend that you are a member of the engineering team that will build the ensuing product. Try to give the designer feedback on the designs as you might in a real work situation. You should be concerned with the appropriateness of the user interface design (i.e., how well it will work for the users), not with how hard it will be to implement.

The description of the problem given to the designer is as follows:

> Design multiple interfaces for a system to provide weather information to travelers. *TRAVELweather* can provide information about the weather for the current day and can give predictions for the next two days. The interface is to be used on a window-based graphical personal computer with a mouse.
>
> The user should be able to enter the date and time they wish a forecast for. The system should display a map of the region in question with the weather data overlaid on the map. The types of data that the system should be able to display are temperature (in either Fahrenheit or Celsius), precipitation, and wind speed/direction.

B.9 Engineering Design Review Script

The following description of the design review was read by me to both the engineer and designer at the start of the engineering design review.

Design Review Script

[Designer name] is going to show you what he/she has done and try to run you through it. [Engineer name] is acting as an engineer that you as a designer are trying to show the design idea to. He/she knows what the design problem is from reading the problem statement. He/she might have feedback on what you did and you are free to try these changes. You are trying to show what you have done and walk him/her through your ideas. When you feel that you have fully gone over it, tell me that you are ready to end.

B.10 Designer Post-Evaluation Questionnaire

Answers to the following questions are given in Appendix Section C.5.

Designer Post-Evaluation Questionnaire

Subject number: _____

1. Assuming that the SILK system was significantly faster and less buggy than the prototype, do you think it would be a better tool for interface design than existing tools (i.e., paper and prototyping tools such as HyperCard, Director, and Visual BASIC)?

Why or why not?

Usability Problems

Did you experience problems with any of the following:

2. Understanding how to carry out the tasks (check one):

____ no problems ____ minor problems ____ major problems

Please explain:

3. Knowing what to do next to complete your task (check one):

____ no problems ____ minor problems ____ major problems

Please explain:

4. How well did the gesture recognition work?

5. What are the best aspects of SILK for the user?

6. What are the worst aspects of SILK for the user?

7. What were the most common mistakes the system made?

8. How was the overall performance of the system? (circle one)

very poor 0 1 2 3 4 5 6 7 8 9 10 very good

9. What does a system like SILK offer you that Director or HyperCard does not?

Please write any other recommendations and comments you might have on SILK here and on the back:

B.11 Engineer Post-Evaluation Questionnaire

Engineer Post-Evaluation Questionnaire

Subject number: _____

1. Assuming that the SILK system was significantly faster and less buggy than the prototype, do you think it would be a better tool for interface design than existing tools (i.e., paper and prototyping tools such as HyperCard, Director, and Visual BASIC)?

Why or why not?

Usability Problems

Did you notice the designer having problems with any of the following:

2. Understanding how to carry out the tasks (check one):

____ no problems ____ minor problems ____ major problems

Please explain:

3. Knowing what to do next to complete their task (check one):

____ no problems ____ minor problems ____ major problems

Please explain:

4. What are the best aspects of SILK for the user?

5. What are the worst aspects of SILK for the user?

6. What were the most common mistakes the system made?

Please write any other recommendations and comments you might have on SILK here and on the back:

B.12 Post-Experiment Information

The following information was given to each participants on a sheet of paper at the end of the session.

Post-Experiment Information

Thank you for participating in our experiment. The purpose of the experiment was to see whether user interface designers can use a new electronic sketching tool, called SILK, to design user interfaces. In addition, we would like to see if user's can perform design tasks more quickly with SILK than with existing design tools (such as Hyper-Card), for both the overall process and for each iteration. We would also like to find out whether SILK encourages the exploration of a wider variety of design ideas. We would also like to find out whether designers using SILK produce higher quality user interfaces. Finally, we would like to find any problems with the SILK user interface so that we can refine it.

.

APPENDIX C
Evaluation Data

This appendix contains all of the data collected during the SILK usability test. See Chapter 6, “Evaluation”, for a description of the experiment and an analysis of the data.

C.1 Designer Demographic Information

Table C-1 contains the demographic information that was given by the design participants prior to the experiment.

Designer Demographics

participant	7	9	11	13	15	17	mean	median	% yes
age	21	29	41	33	21	33	29.7	31.0	
sex	Male	Female	Female	Female	Male	Male			50% male
graduate program?	no	yes	yes	yes	no	N/A			60%
major	industrial design	HCI - interaction design	communication, planning, & design	HCI - interaction design	HCI	design			
years professional UI design experience	0.25	1.25	0	2	2.5	5	1.8	1.6	
advanced?	no	yes	no	yes	yes	yes			67%
taken UI design project courses?	yes	yes	yes	yes	yes	no			83%
UI tools used:									
HyperCard?	no	no	no	no	yes	yes			33%
Director?	yes	yes	yes	yes	yes	yes			100%
Visual BASIC?	yes	no	no	no	yes	no			33%
HTML?	no	no	no	no	yes	yes			33%
C++ or Java?	no	no	no	yes	yes	no			33%
Drawing/painting programs used:									
Canvas	no	no	no	no	yes	no			17%
Color It!	no	no	no	no	yes	no			17%
Corel Draw	no	no	yes	no	no	yes			33%
Freehand	no	no	no	yes	yes	yes			50%
Illustrator	no	yes	yes	yes	yes	yes			83%
MacDraw	no	no	no	no	yes	yes			33%
MacPaint	no	no	no	no	yes	no			17%
Photoshop	yes	yes	yes	yes	yes	yes			100%
Superpaint	no	no	no	no	yes	no			17%
at least one?	yes	yes	yes	yes	yes	yes			100%
read HCI books?	yes	yes	no	yes	yes	yes			83%
Nielsen's *Usability Engineering* ?	no	yes	no	no	no	yes			33%

Table C-1. Complete demographic information supplied by design participants.

C.2 Engineer Demographic Information

Table C-2 contains the demographic information that was given by the engineering participants prior to the experiment.

Engineer Demographics

participant	8	10	12	14	16	18	mean	median	% yes
age	26	33	25	27	25	25	26.8	25.5	
sex	Male	Female	Male	Male	Male	Male			83% male
graduate program?	yes	yes	yes	yes	yes	yes			100%
major	CS	robotics	language technology	CS	CS	CS			67% CS
years professional programming experience	4	9	0	2	2	7	4.0	3.0	
UI tools used:									
HyperCard?	yes	no	no	no	no	no			17%
Director?	no	no	no	no	no	no			0%
Visual BASIC?	yes	no	yes	no	no	no			33%
Tcl/Tk?	yes	yes	no	no	no	no			33%
Other?	yes	yes	no	no	no	no			33%
at least one?	yes	yes	yes	no	no	no			50%
Other:									
Builder?	yes	no	no	no	no	no			17%
Lotus Notes?	yes	no	no	no	no	no			17%
X/Motif?	yes	no	no	no	no	no			17%
Xcessory?	yes	no	no	no	no	no			17%
Drawing/painting programs used:									
Canvas	no	no	no	no	no	no			0%
Color It!	no	no	no	no	no	no			0%
Corel Draw	no	no	yes	no	no	yes			33%
Cricket Draw	yes	no	no	no	no	no			17%
Freehand	no	no	no	no	no	no			0%
Illustrator	yes	no	no	no	no	no			17%
MacDraw	yes	yes	no	no	no	yes			50%
MacPaint	no	yes	no	no	no	no			17%
Paintbrush	no	no	no	yes	no	no			17%
Photoshop	yes	no	no	no	no	no			17%
Powerpoint	yes	no	no	no	no	no			17%
Superpaint	no	no	no	no	no	no			0%
Visio	yes	no	no	no	no	no			17%
Xfig	no	yes	no	yes	yes	no			50%
at least one?	yes	yes	yes	yes	yes	yes			100%
read HCI books?	yes	no	no	no	no	no			17%
Nielsen's *Usability Engineering* ?	yes	no	no	no	no	no			17%

Table C-2. Complete demographic information supplied by engineering participants.

C.3 Experimental Data

This section includes the data collected during the SILK usability test. Table C-3 contains data on the recognition results and Table C-4 contains data that characterizes the sessions and the resulting designs (each of which are included in Section C.4).

Recognition Information											
		participant	7	9	11	13	15	17	mean	median	stdev
tutorial	editing gestures	#drawn	52	40	78	56	81	55	60	56	16
		#unrecognized	10	9	2	10	30	5	11	10	10
		recognition rate	80.8%	77.5%	97.4%	82.1%	63.0%	90.9%	82.0%	81.5%	11.8%
	primitive components	#drawn	88	103	174	54	83	96	100	92	40
		#unrecognized	12	15	55	4	10	15	19	14	18
		#corrections	11	21	27	1	8	6	12	10	10
		recognition rate	87.5%	79.6%	84.5%	98.1%	90.4%	93.8%	89.0%	88.9%	6.6%
	widget inferences	#made	49	24	40	15	25	30	31	28	12
		#errors	9	5	22	2	9	7	9	8	7
		recognition rate	81.6%	79.2%	45.0%	86.7%	64.0%	76.7%	72.2%	77.9%	15.3%
		repairs: #new guesses	44	9	3	2	1	5	11	4	17
		#changes	1	0	5	1	1	5	2	1	2
		#broken	N/A	N/A	19	1	8	6	9	7	8
		repairs/error	5.0	1.8	1.2	2.0	1.1	2.3	2.2	1.9	1.4
		#explicit inferences	N/A	N/A	27	2	14	10	13	12	10
		#explicit successes	N/A	N/A	12	1	9	3	6	6	5
		explicit success rate	N/A	N/A	44%	50%	64%	30%	47%	47%	14%
weather task	editing gestures	#drawn	67	139	63	119	151	78	103	99	38
		#unrecognized	3	17	8	8	34	7	13	8	11
		recognition rate	95.5%	87.8%	87.3%	93.3%	77.5%	91.0%	88.7%	89.4%	6.3%
	primitive components	#drawn	120	102	40	55	123	114	92	108	36
		#unrecognized	21	20	20	12	13	38	21	20	9
		#corrections	9	8	3	0	15	7	7	7	5
		recognition rate	92.5%	92.2%	92.5%	100.0%	87.8%	93.9%	93.1%	92.5%	3.9%
		#non-learning errors	N/A	33	16	0	48	32	26	32	18
		non-learning recognition rate	N/A	67.6%	60.0%	100.0%	61.0%	71.9%	72.1%	67.6%	16.3%
	widget inferences	#made	41	17	2	14	56	14	24	16	20
		#errors	10	1	1	5	26	3	8	4	10
		recognition rate	75.6%	94.1%	50.0%	64.3%	53.6%	78.6%	69.4%	69.9%	16.7%
		repairs: #new guesses	8	0	0	0	23	0	5	0	9
		#changes	0	0	0	0	13	0	2	0	5
		#broken	3	1	1	5	29	3	7	3	11
		repairs/error	1.1	1.0	1.0	1.0	2.5	1.0	1.3	1.0	0.6
		#explicit inferences	11	1	0	4	17	0	6	3	7
		#explicit successes	2	1	0	1	11	0	3	1	4
		explicit success rate	18%	100%		25%	65%		52%	45%	38%

Table C-3. Recognition information for editing gestures, primitive components, and widget inferences for both the tutorial and measured task.

Tutorial, Task, Design, and Discussion Characteristics

	participant	7	9	11	13	15	17	mean	median	stdev
	total time	3:30	3:55	4:15	3:00	4:10	4:30	3:53	4:02	0:33
tutorial	**time**	0:44	0:42	1:39	0:30	0:45	1:18	0:56	0:44	0:26
	#crashes	0	0	0	0	0	2	0.3	0.0	0.8
	crash recovery or assistance time	0:00	0:00	0:05	0:00	0:00	0:06	0:01	0:00	0:02
	adjusted tutorial time	0:44	0:42	1:34	0:30	0:45	1:12	0:54	0:44	0:23
weather	**time**	2:07	2:08	1:18	1:13	1:47	1:43	1:42	1:45	0:23
task	**#crashes**	2	0	0	0	3	1	1.0	0.5	1.3
	crash recovery time	0:25	0:00	0:00	0:00	0:10	0:04	0:06	0:02	0:09
	adjusted task time	1:42	2:08	1:18	1:13	1:37	1:39	1:36	1:38	0:19
iteration 1	**time**	1:32	0:47	1:18	0:26	0:46	1:19	1:01	1:02	0:25
	#screens	5	7	2	4	8	7	5.5	6.0	2.3
	#transitions	10	18	0	8	30	13	13.2	11.5	10.2
iteration 2	**time**	0:38	1:21		0:48	0:40	0:23	0:46	0:40	0:21
	#screens	4	10		10	6	2	6.4	6.0	3.6
	#transitions	10	20		22	32	2	17.2	20.0	11.5
iteration 3	**time**					0:21		0:21	0:21	
	#screens					7		7.0	7.0	
	#transitions					19		19.0	19.0	
	#designs	2	2	1	2	3	2	2.0	2.0	0.6
	average time / iteration	0:51	1:04	1:18	0:36	0:32	0:49	0:51	0:50	0:17
	total screens	9	17	2	14	21	9	12.0	11.5	6.8
	total transitions	20	38	0	30	81	15	30.7	25.0	27.9
	screens / design	4.5	8.5	2	7	7	4.5	5.6	5.8	2.4
	transitions / design	10	19	0	15	27	7.5	13.1	12.5	9.4
	transitions / screen	2.2	2.2	0	2.1	3.9	1.7	2.0	2.2	1.2
	total strings	118	220	25	145	238	157	151	151	77
	sketchy strings	56	69	8	0	45	73	42	51	31
	% sketchy strings	47%	31%	32%	0%	19%	46%	29%	32%	18%
discussion	**time**	0:31	0:13	0:23	0:16	0:28	0:29	0:23	0:25	0:07
	engineer asked critical questions?	yes	yes	no	N/A	yes	yes	67%		
	asked questions on structure & behavior?	yes	yes	yes	N/A	yes	yes	83%		
	made real-time changes?	yes	no	no	no	yes	yes	50%		
	storyboard used to test?	yes	yes	no	yes	yes	yes	83%		
	storyboard used to illustrate structure?	yes	no	yes	N/A	no	yes	50%		
	successful communication?	yes	yes	no	N/A	yes	yes	67%		

Table C-4. Tutorial, task, and discussion times along with characteristics of designs produced

C.4 Designs Produced

This section contains the storyboards and representative screens of each of the designs produced during the SILK usability test.

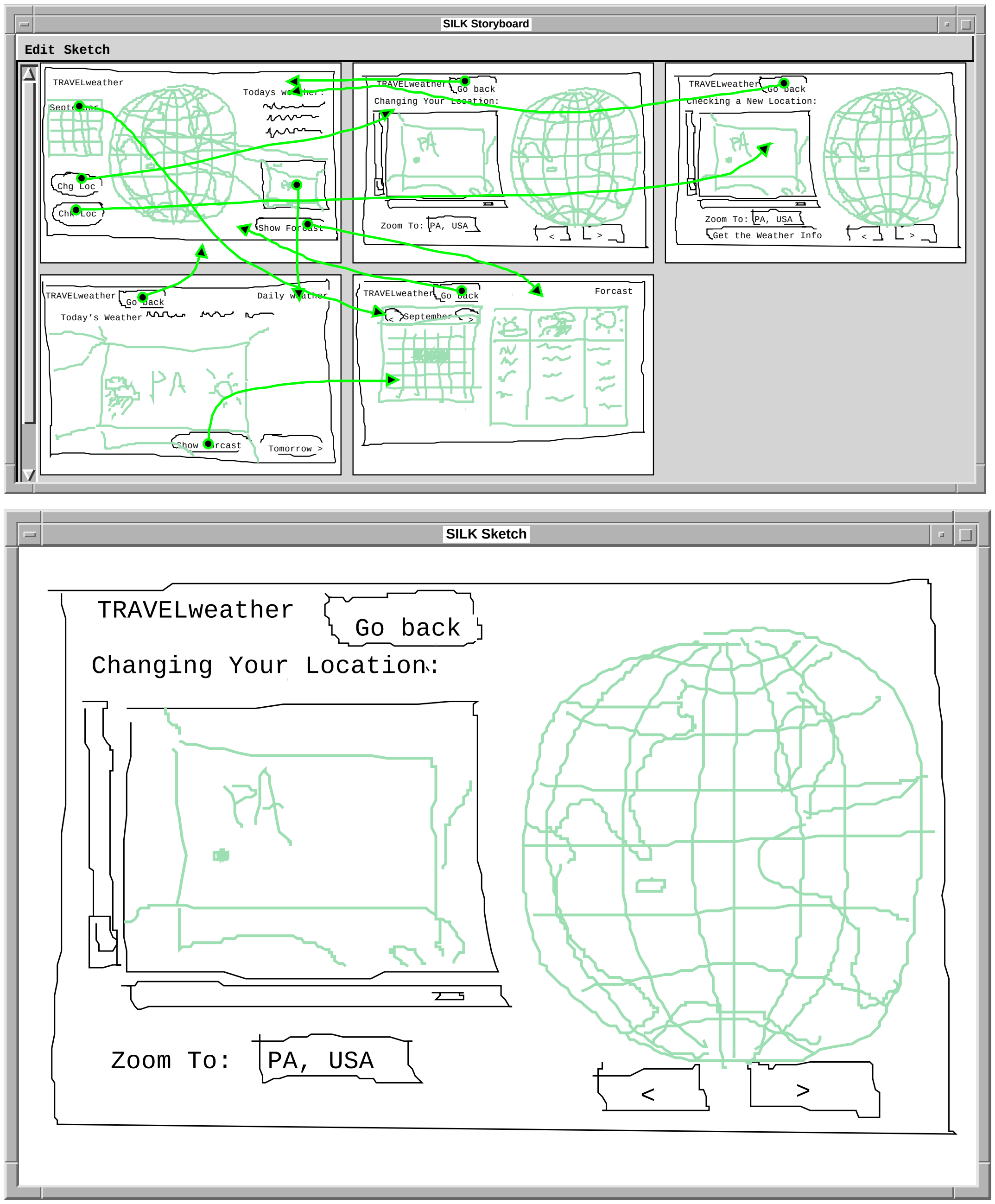


Figure C-1. First design of participant 7.

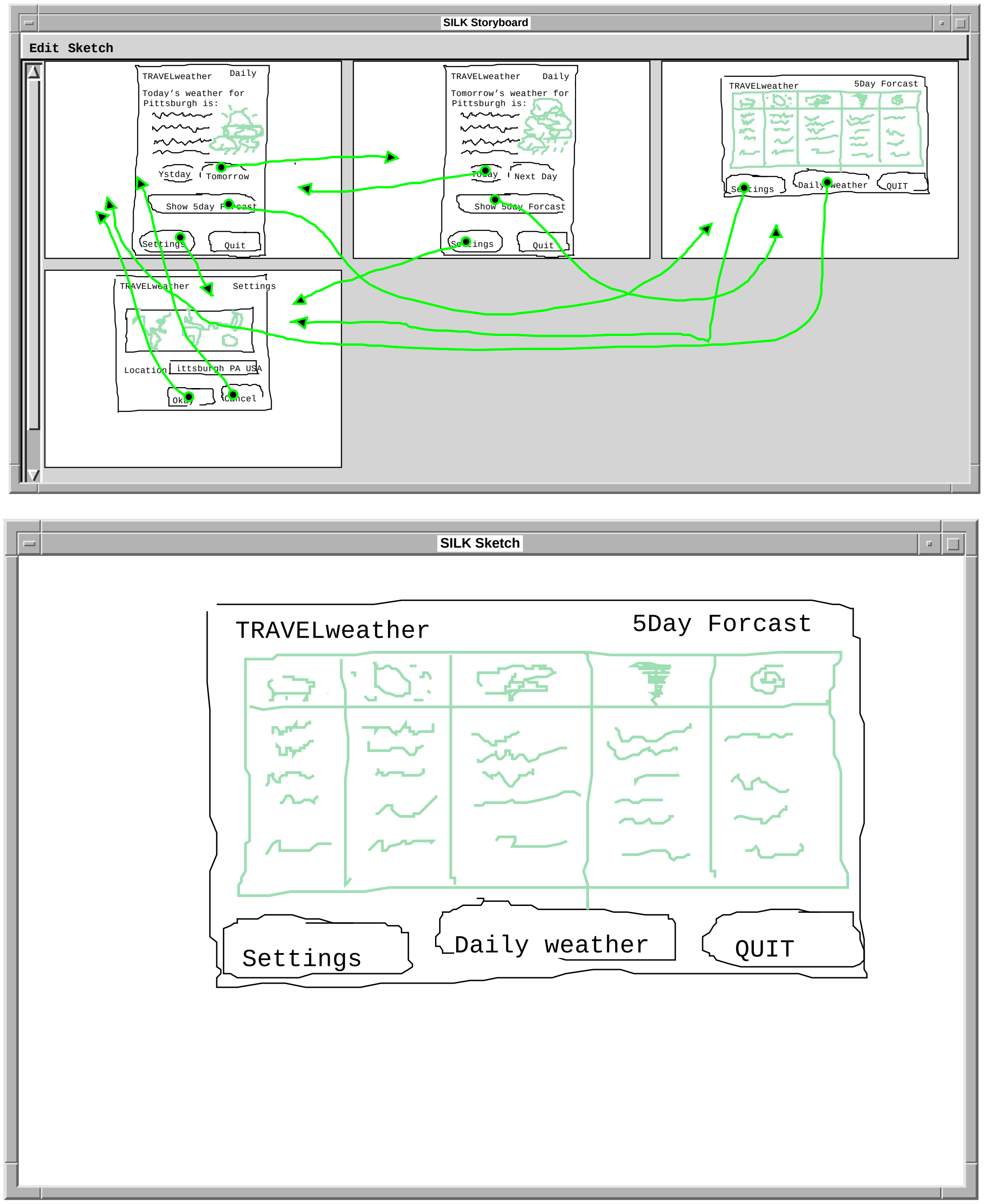
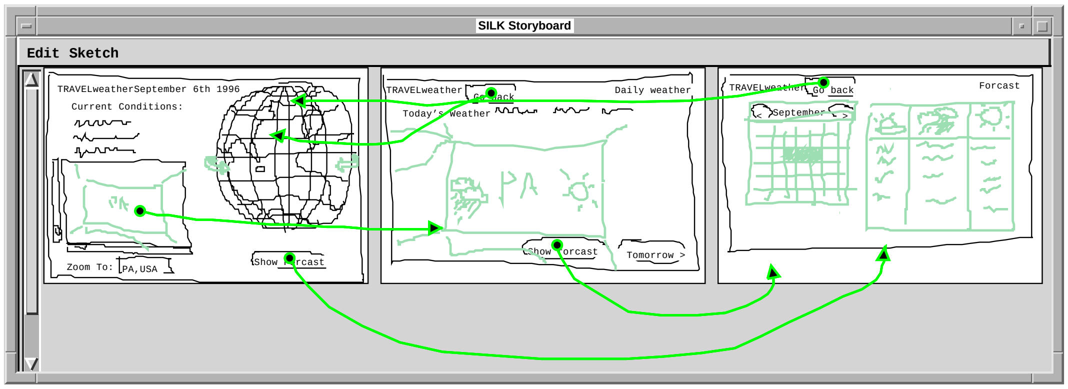


Figure C-2. Second design of participant 7.



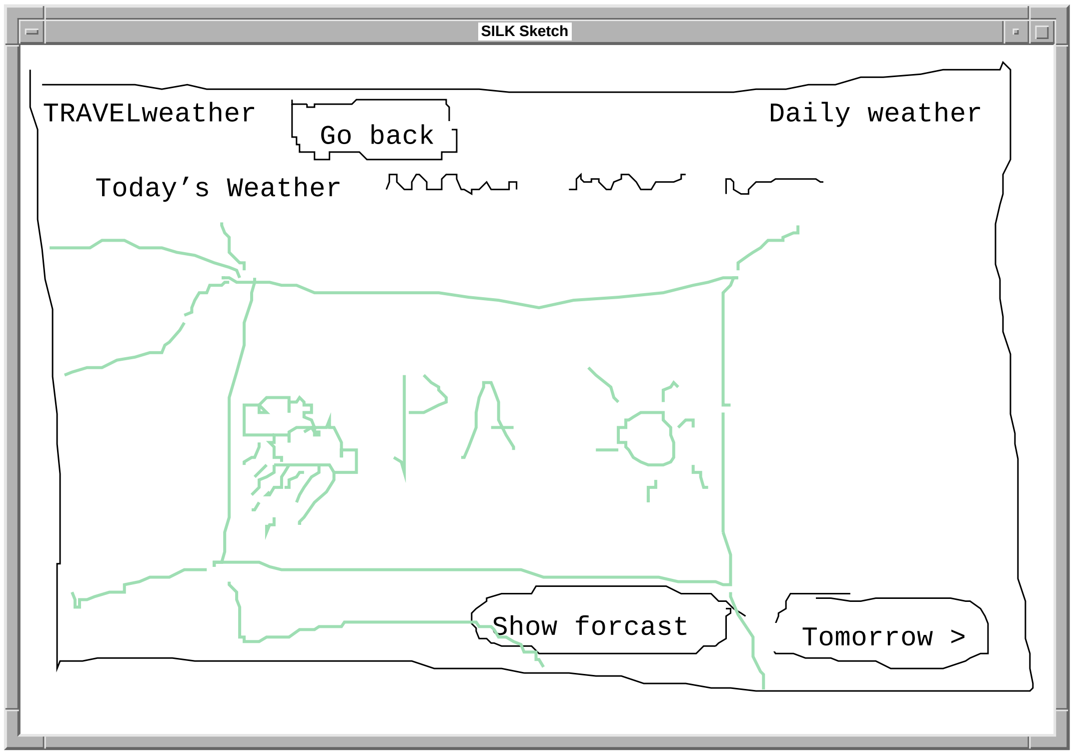


Figure C-3. Third design of participant 7. This design, a modification of participant 7's first design, was made during the engineering discussion.

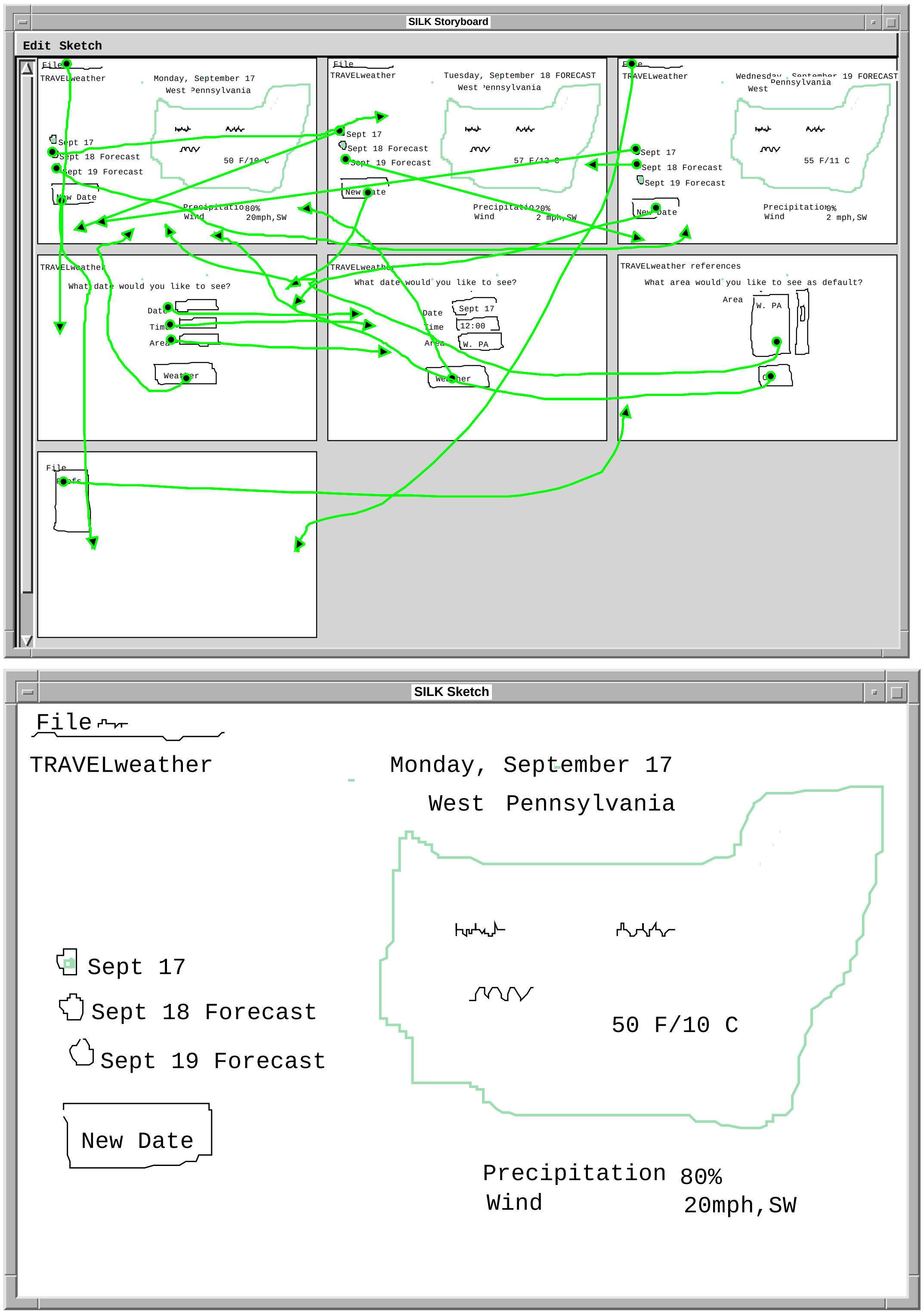
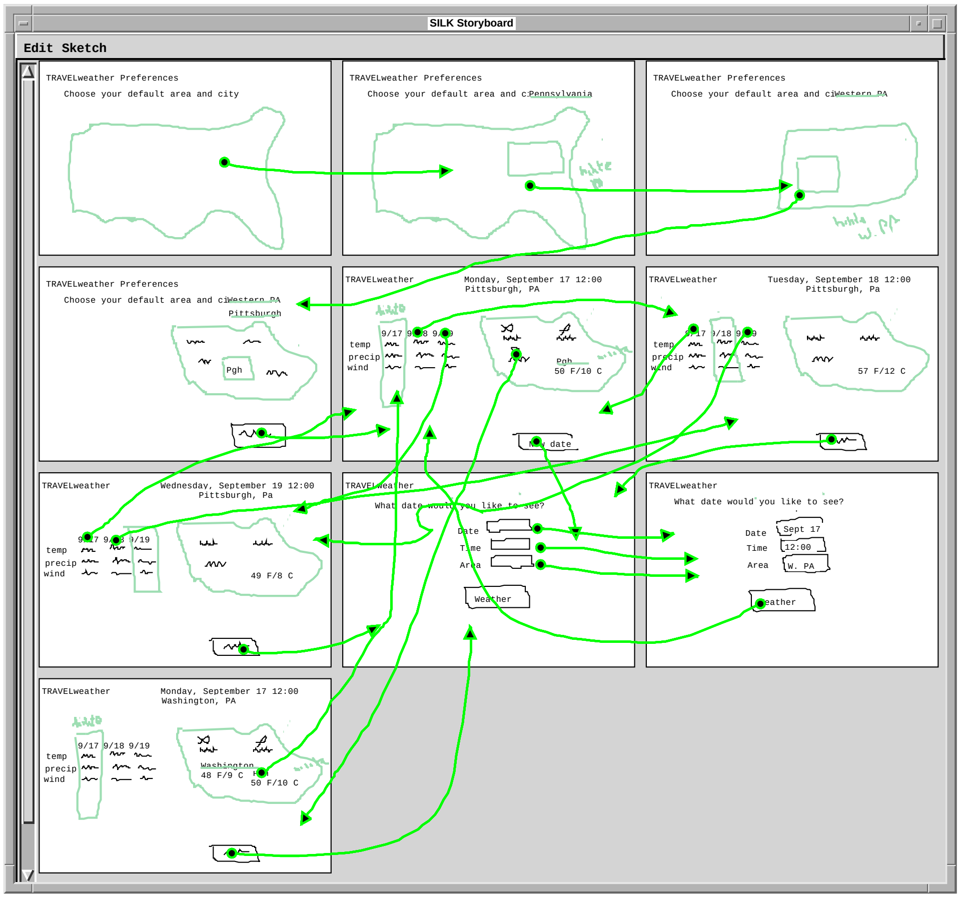


Figure C-4. First design of participant 9.


SILK Storyboard
Edit Sketch
TRAVELweather Preferences
Choose your default area and city
Choose your default area and c Pennsylvania
Choose your default area and c Western PA
Pittsburgh
Pgh
TRAVELweather
Monday, September 17 12:00
Pittsburgh, PA
Tuesday, September 18 12:00
Wednesday, September 19 12:00
9/17 9/18 9/19
temp
precip
wind
50 F/10 C
57 F/12 C
49 F/8 C
New date
What date would you like to see?
Date
Time
Area
Sept 17
12:00
W. PA
Weather
Washington, PA
Washington
48 F/9 C

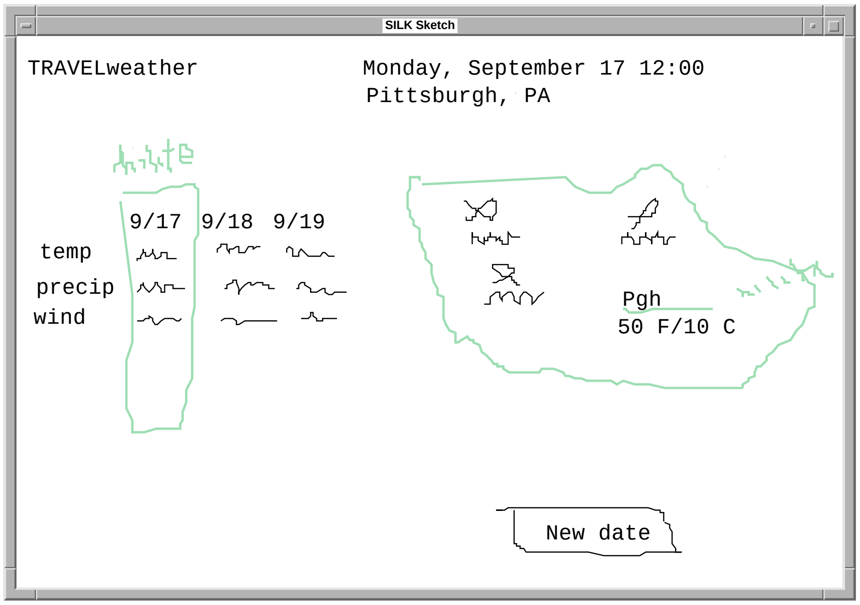
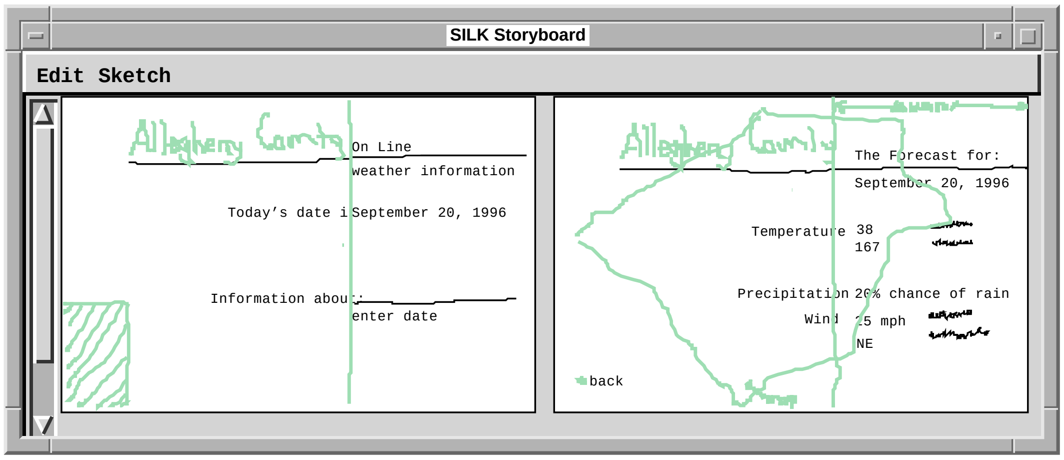


Figure C-5. Second design of participant 9.



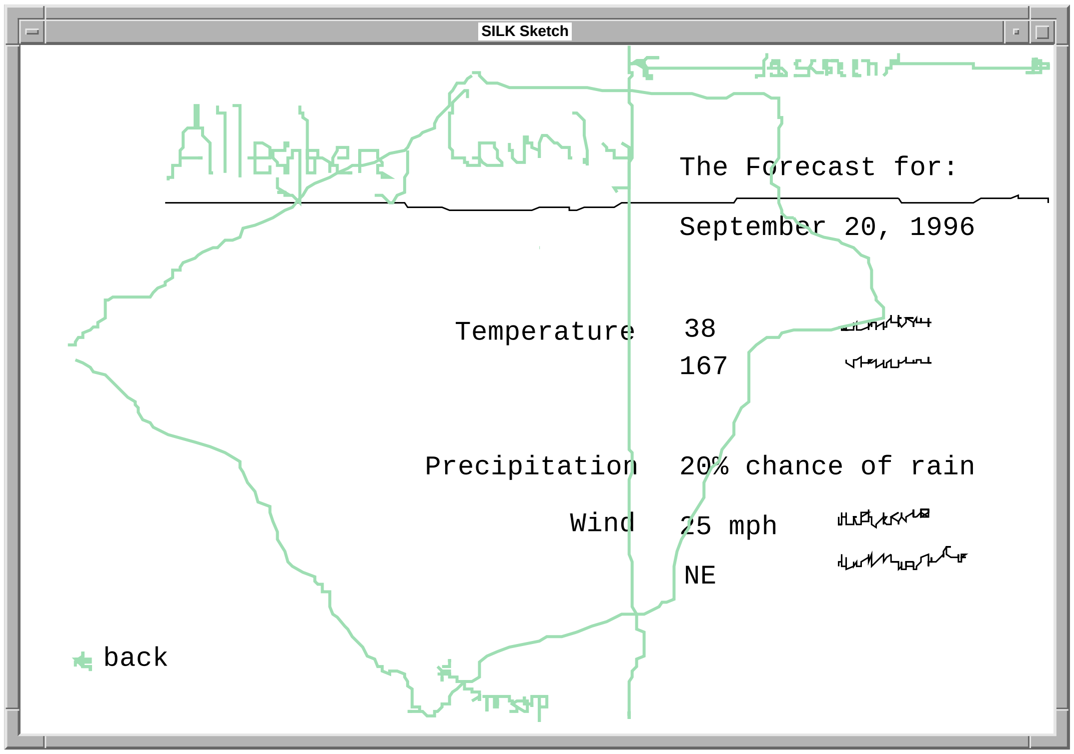
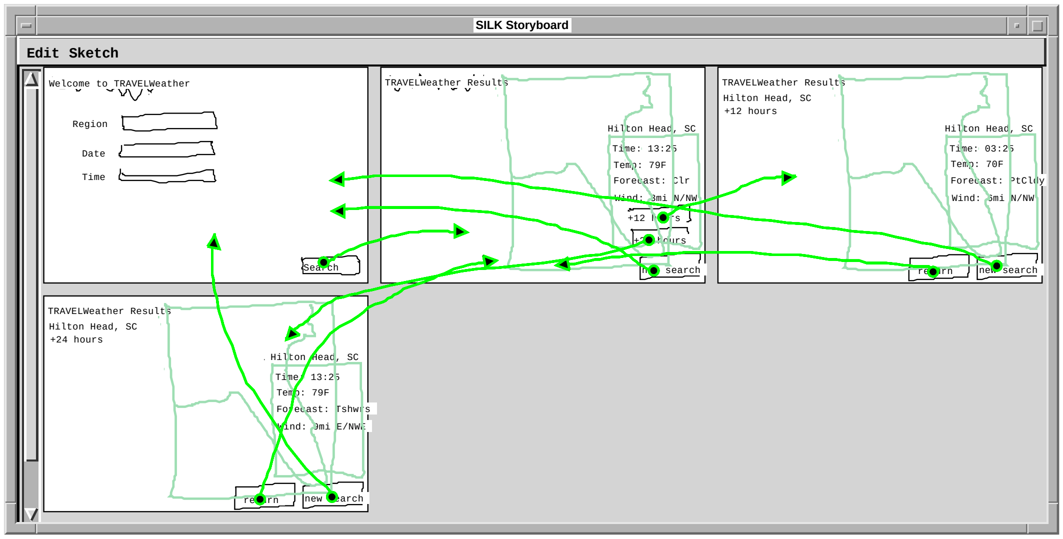


Figure C-6. Sole design of participant 11.



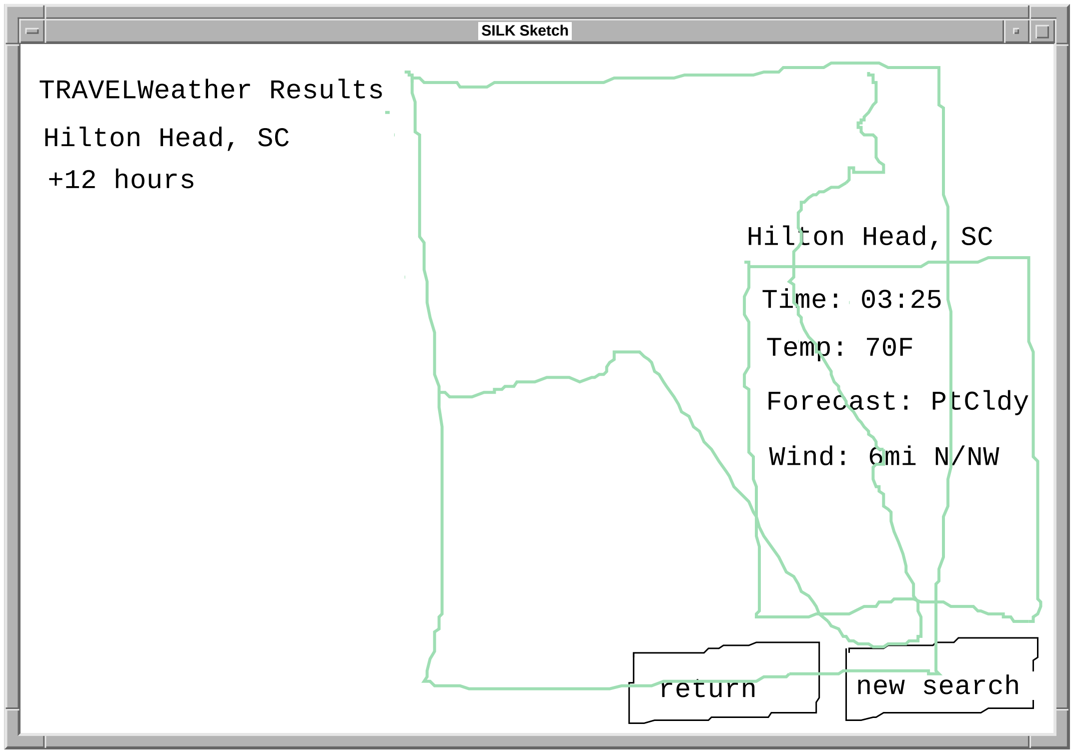
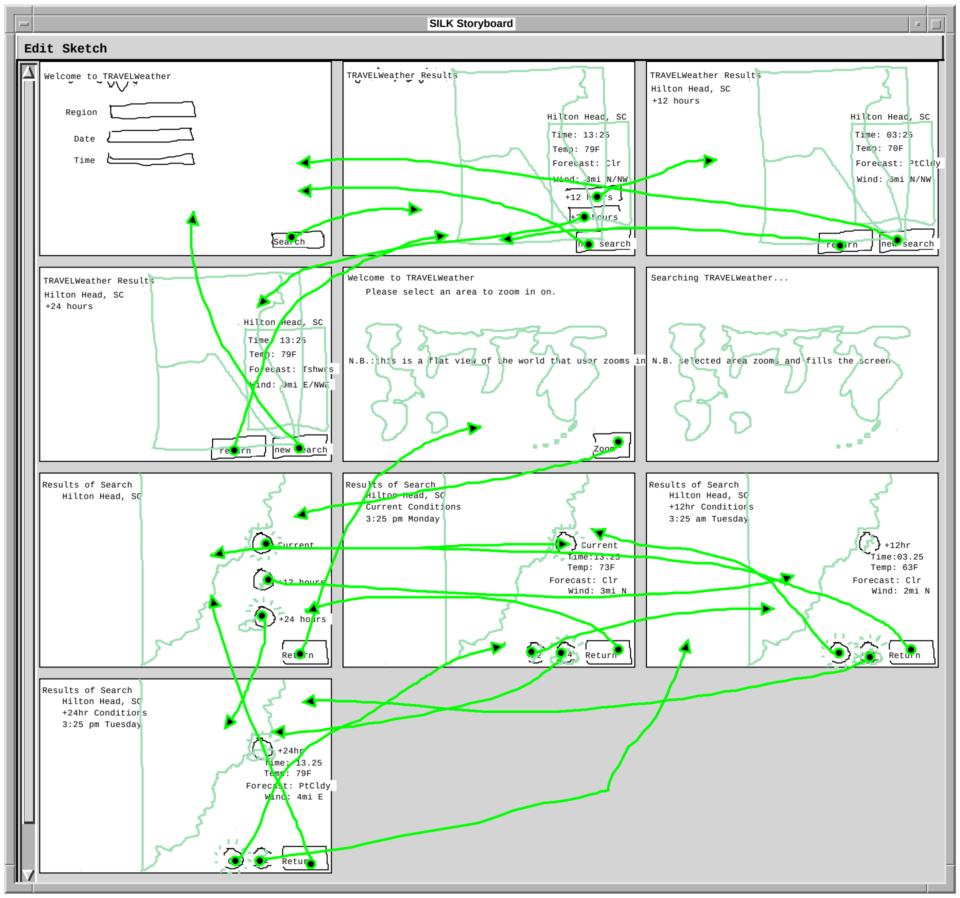


Figure C-7. First design of participant 13.



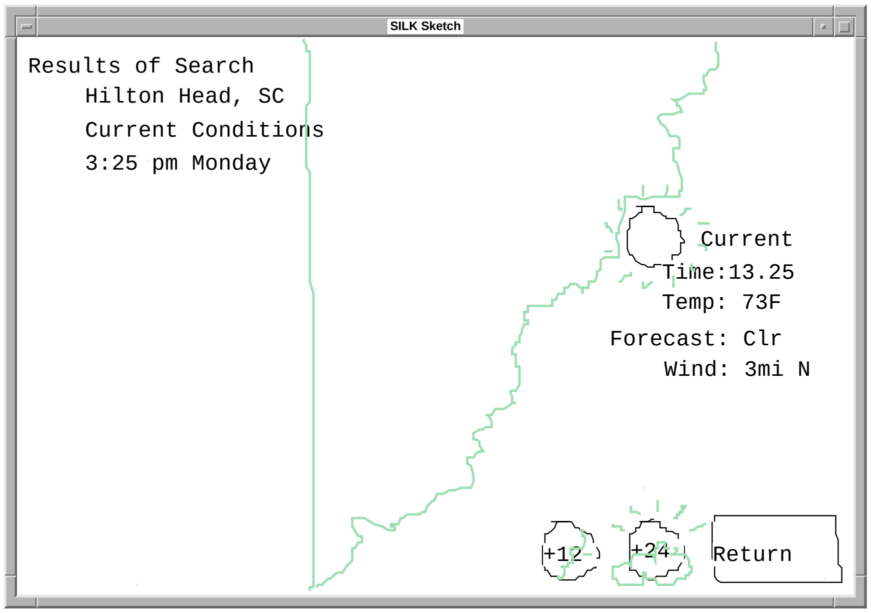
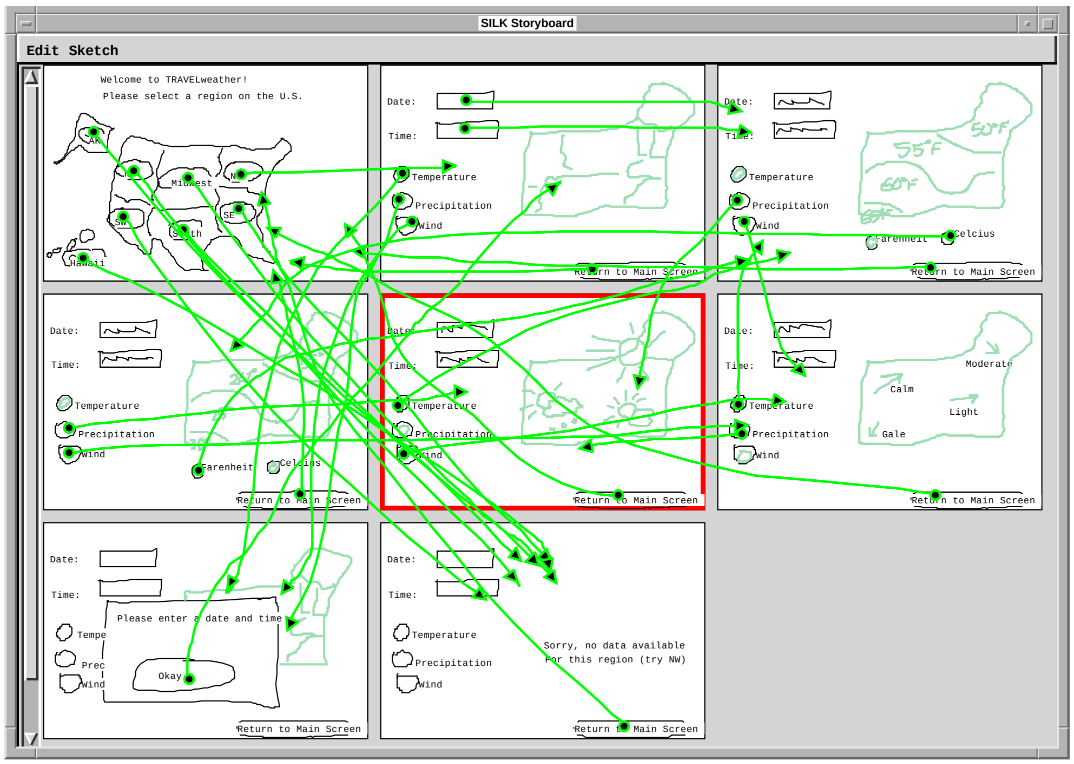


Figure C-8. Second design of participant 13. This was judged to be the best design produced.



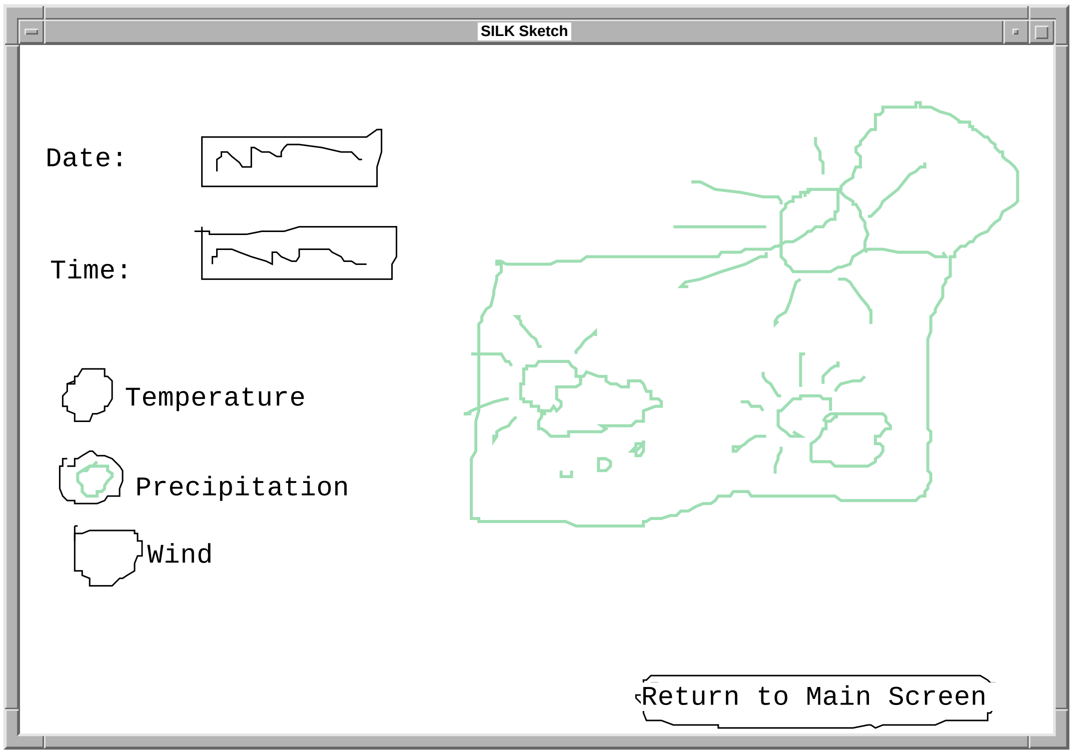
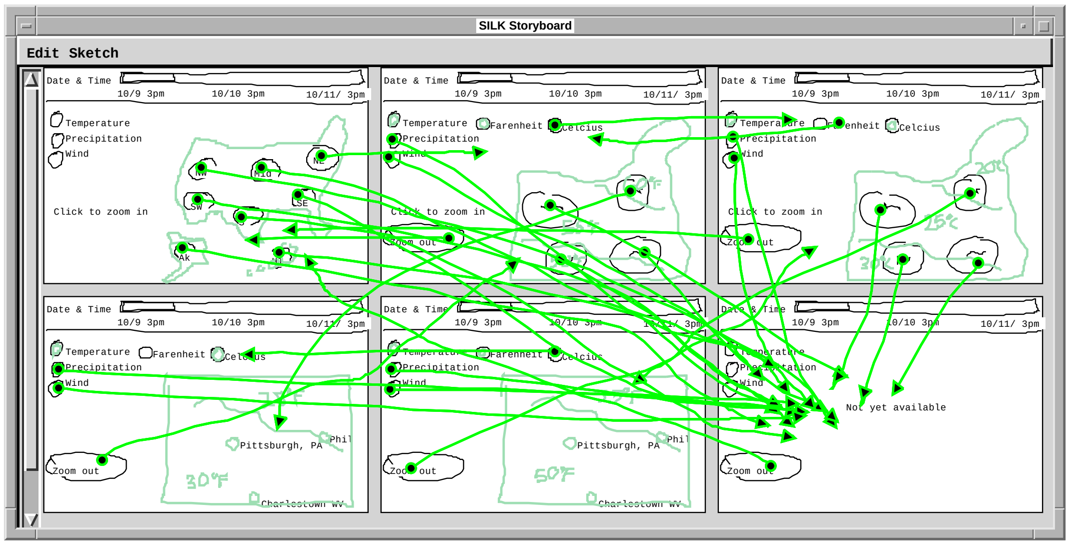


Figure C-9. First design of participant 15.



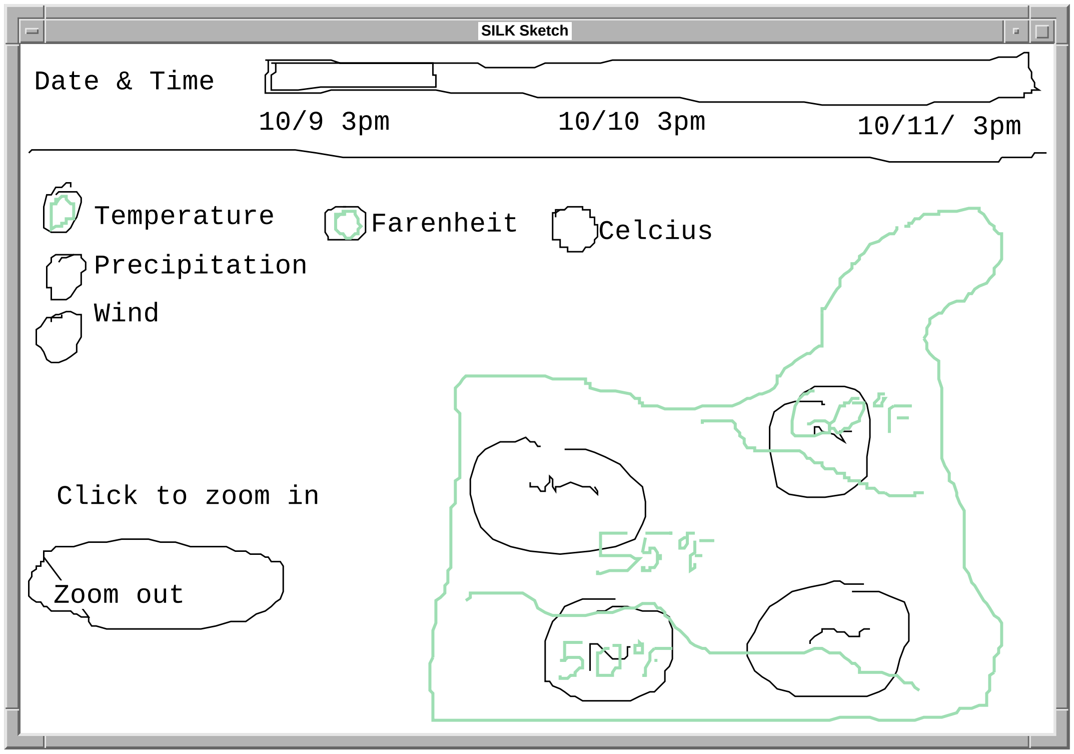


Figure C-10. Second design of participant 15.

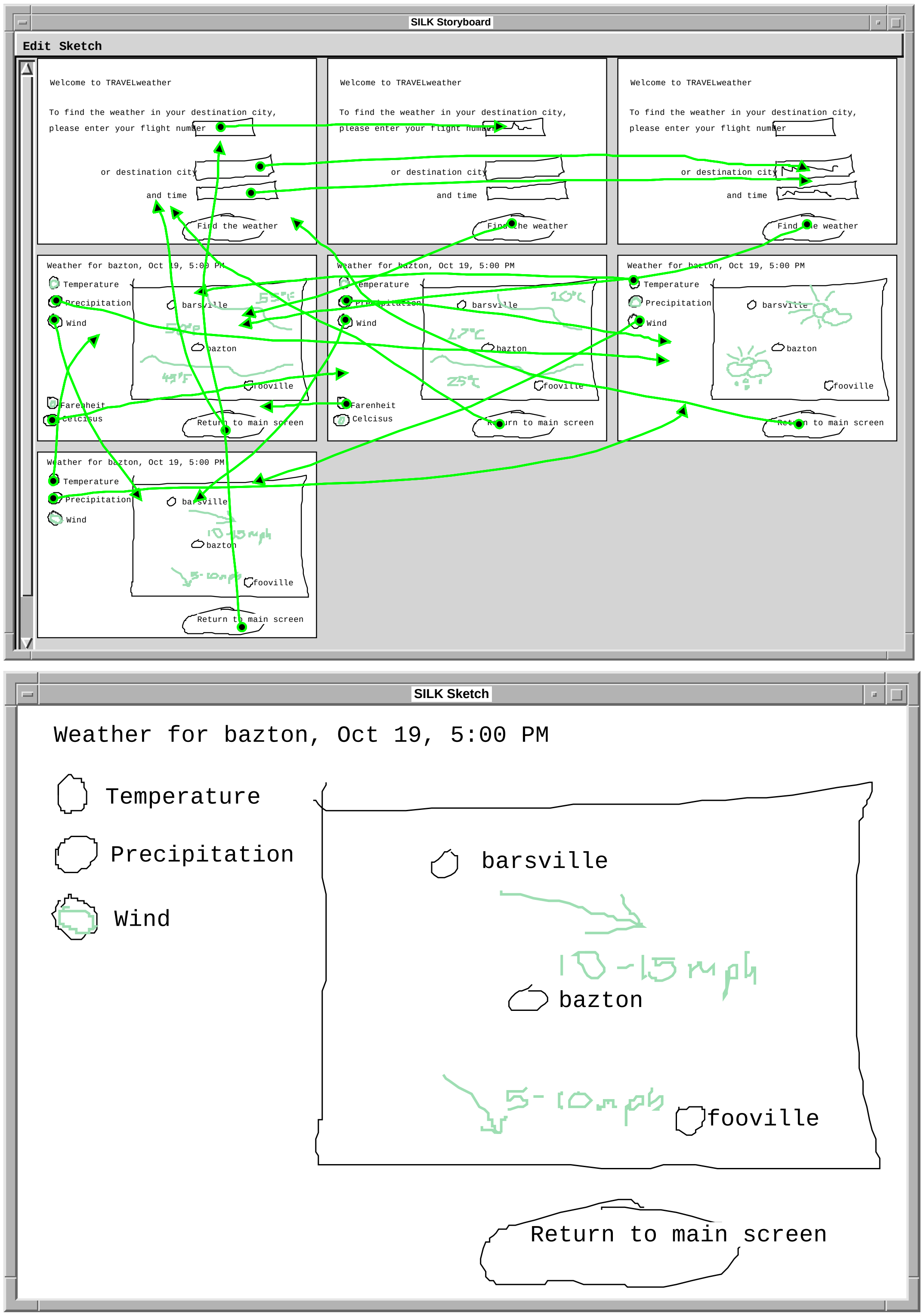


Figure C-11. Third design of participant 15.

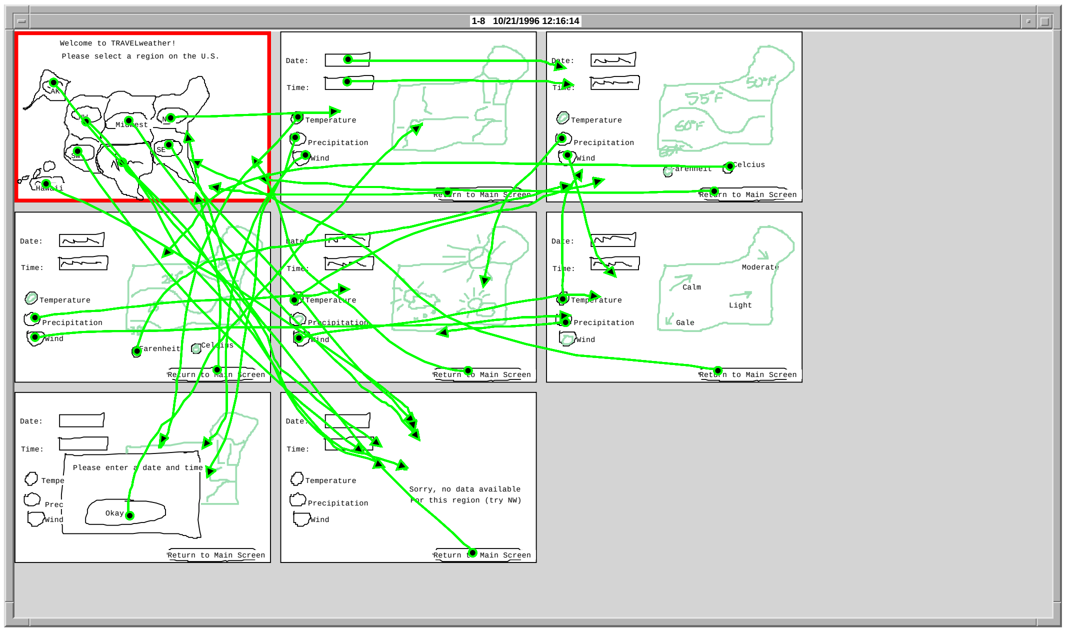


Figure C-12. Fourth design of participant 15. This design, a modification of participant 15's first design, was made during the engineering discussion. The only difference is that the state of Texas in the first screen (upper left) can be selected by clicking anywhere in the state rather than on a button contained in the state.

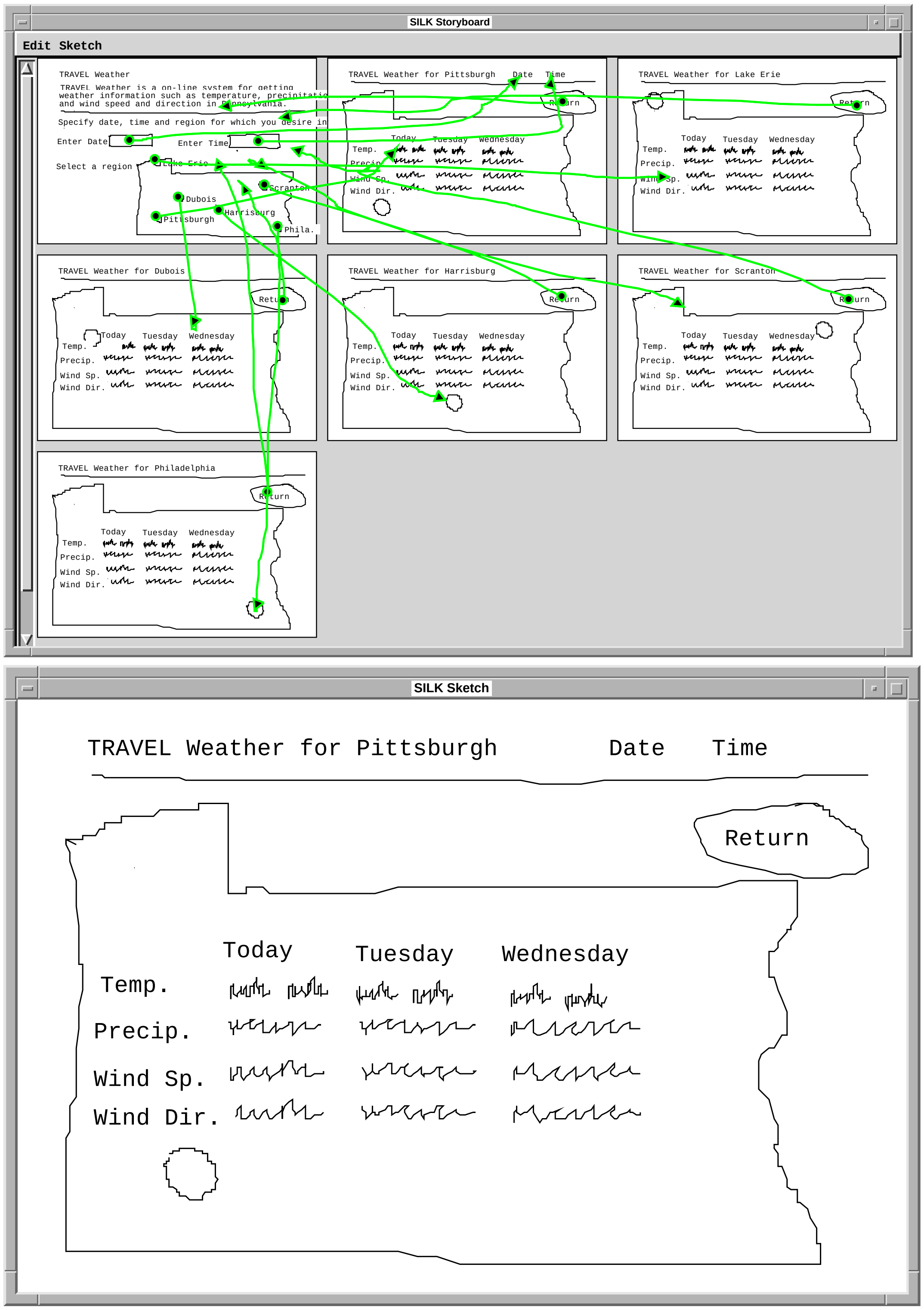
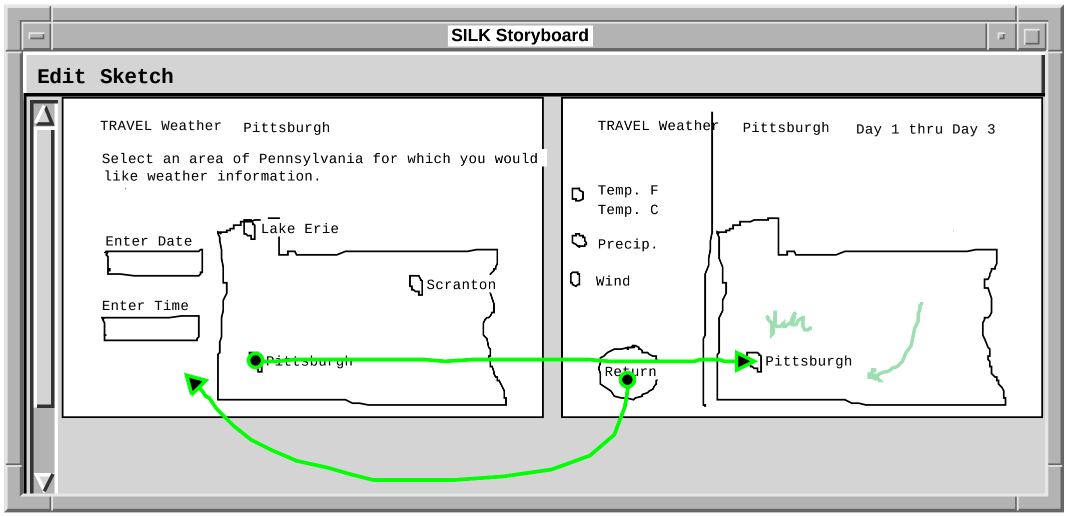

SILK Storyboard
Edit Sketch
TRAVEL Weather
TRAVEL Weather is a on-line system for getting weather information such as temperature, precipitatio and wind speed and direction in Pennsylvania.
Specify date, time and region for which you desire in
Enter Date
Enter Time
Select a region
Lake Erie
Scranton
Dubois
Harrisburg
Pittsburgh
Phila.
TRAVEL Weather for Pittsburgh Date Time
TRAVEL Weather for Lake Erie
TRAVEL Weather for Dubois
TRAVEL Weather for Harrisburg
TRAVEL Weather for Scranton
TRAVEL Weather for Philadelphia
Return
Today Tuesday Wednesday
Temp.
Precip.
Wind Sp.
Wind Dir.
SILK Sketch
TRAVEL Weather for Pittsburgh Date Time
Return
Today Tuesday Wednesday
Temp.
Precip.
Wind Sp.
Wind Dir.

Figure C-13. First design of participant 17.



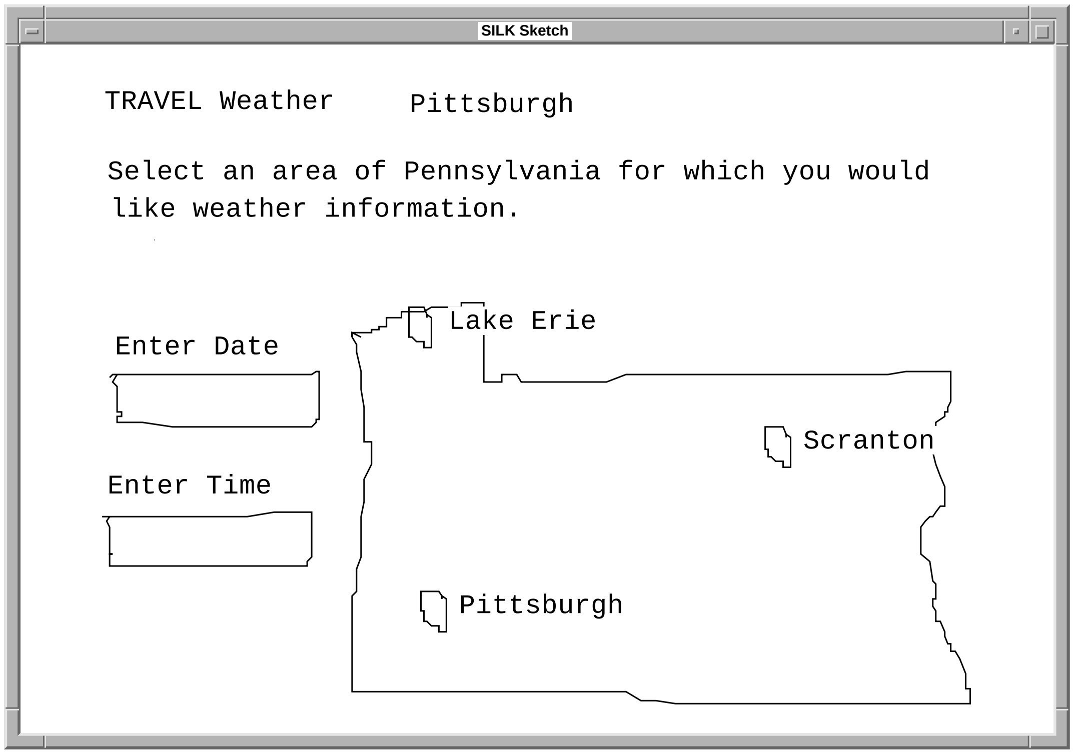


Figure C-14. Second design of participant 17.

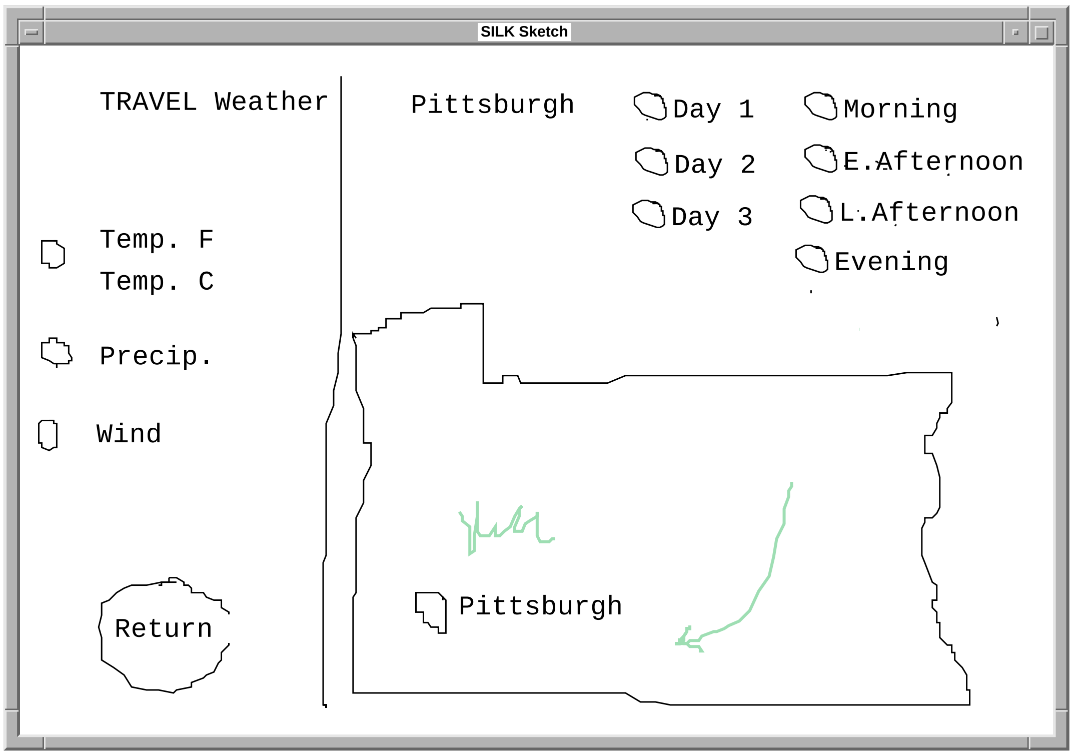


Figure C-15. Third design of participant 17. This design, a modification of participant 17’s second design, was made during the engineering discussion. The designer added two panels of buttons in the upper right corner.

C.5 Post-Evaluation Questionnaire Results

This section contains all of the participant's responses to the questionnaire administered after the usability test (see Chapter 6, "Evaluation"). For fixed response questions, the number of users who selected a particular response are reported. In addition, the mean and median are also reported for numerical questions. For discussion questions, all of the answers are shown. All answers are proceeded by the participant's identification number.

Designer Post-Evaluation Questionnaire

1. Assuming that the SILK system was significantly faster and less buggy than the prototype, do you think it would be a better tool for interface design than existing tools (i.e., paper and prototyping tools such as HyperCard, Director, and Visual BASIC)?

» (7) "It could be an excellent 1st draft tool."

» (9) "Better than Director – the linking with drawings rather than Lingo is excellent. Also, you can see right away what is going on."

» (11) "No."

» (13) "No, but it would be good for novice interface designers."

» (15) "Yes and no."

» (17) "May be useful for quickly structuring information."

Why or why not?

» (7) "It is as quick as paper sketching and provides a basis for interaction."

» (9) "I still would have liked using pencil and paper for fiddling with the problem – it was still faster by hand. But I could learn using the tool – I made some frames, and saw missing pieces as I was working, which is the same way it works using paper. That's good!"

» (11) "It takes too much time and it's very frustrating."

» (13) "Unable to implement rollovers, visual/perceptual cues, transitions from one screen to another."

» (15) "The ability to prototype screen-based interactions is GREAT! However, it seems difficult to do anything non-standard (e.g., radio buttons that change the screen)."

» (17) "But restricts typical visual design decisions frequently combined with prototyping (like type and color)."

Usability Problems

Did you experience problems with any of the following:

2. Understanding how to carry out the tasks (check one):

{7, 9, 13} -> 3 no problems {11, 15, 17} -> 3 minor problems 0 major problems

Please explain:

» (7) no comments

» (9) “For basic tasks, the tutorial was good enough to understand. I did have some questions as I went along about features – and got stuck a few times which may have been bug related – e.g., link from button didn’t work when text was too big.”

» (11) “It was just a matter of readjusting my thinking – learning something new.”

» (13) no comments

» (15) “I didn’t realize that uninterpreted regions could be links between screens.”

» (17) “Relationship of copy and paste on control window ambiguous. Instruction seems too long (text) and also prevents recovery from error problems.”

3. Knowing what to do next to complete your task (check one):

{9, 13} -> 2 no problems {7, 11, 15, 17} -> 4 minor problems 0 major problems

Please explain:

» (7) “The UI of the tool itself.”

» (9) “The tasks were fairly simple. The only problem might be re-arranging – I wasn’t sure how to do this without being able to drag.”

» (11) “Same as above.”

» (13) “Tools are few and simple – it’s easy to remember all possible combinations of things.”

» (15) “Linking in scrollbars to make them do something.”

» (17) “Annotate mode. And widget choices were not well known enough for me to rely on them.”

4. How well did the gesture recognition work?

» (7) “Well except that direction was too important.”

» (9) “X’s could be better. Caret worked very well.”

» (11) “Just OK.”

» (13) “Works well – and I am left handed. Preferred using backspace for delete, however.”

» (15) “Pretty well. It’s annoying that it quits if you go off the screen. Using the mouse with my left hand was probably responsible for a lot of problems.”

» (17) “I decided not to use it.”

5. What are the best aspects of SILK for the user?

» (7) “Ability of SILK to understand widget types.”

» (9) “The storyboard was nice – being able to draw arrows to indicate links was very fast. I liked the way it learned gestures, and the way it remains sketchy. I liked the ease with which you could test the interaction – it’s a very tight loop. Also, the storyboarding/sketching cycles were much better than with Director. It would be even better with drag & drop.”

» (11) “The idea of frame to frame.”

» (13) “Works like pencil and paper; is simple, shows logic of navigation.”

» (15) “The ability to see and edit the storyboard is great. It’d be nice if I could automatically straighten arrows.”

» (17) “View of information the users need can be explored quickly!”

6. What are the worst aspects of SILK for the user?

» (7) “It won’t understand non-standard widgets.”

» (9) “I got a little confused with mouse buttons and not being able to drag an object except by its handles. This might get better with time.”

» (11) “(1) Speed. (2) Lack of choices. (3) No actual shapes – spent a lot of time trying to get the program to recognize commands. (4) lack of manual control. (5) odd type input. (6) odd shapes.”

» (13) “Display is pixelated, hard to layer drawing objects, and type abilities are primitive. I would implement size, bold, italics."

» (15) "Fiddling with the recognizer. Towards the end, I just wanted to grab widgets from a palette. Clicking on the wrong mouse button when gesturing is annoying."

» (17) "Visual restrictions in design phase make it difficult to integrate static choices with specification."

7. What were the most common mistakes the system made?

» (7) "Drawing objects without figuring out what they are (buttons, text, etc.)."

» (9) "Not recognizing X's. Sometimes, such as with scrollbar, it didn't come up with any recommended widget and I couldn't force it to be any of them."

» (11) "Recognizing some shapes."

» (13) "I think some of the storyboard got lost when I deleted a screen."

» (15) "Every time I tried to do an ungroup gesture, it interpreted it as delete. Lots of my text scribbles were interpreted as lines."

» (17) "Primitive recognition."

8. How was the overall performance of the system? (circle one)

very poor 0 1 2 3 4 5 6 7 8 9 10 very good

» (7) "6.5", (9) "7", (11) "3", (13) "8", (15) "7", (17) "6.5"

mean = 6.3
median = 6.8
stdev = 1.7

9. What does a system like SILK offer you that Director or HyperCard does not?

» (7) “Rough drafting and simple storyboard interaction.”

# (9) “I liked being able to draw the connections. It was much faster than using Lingo in Director. I’m not a HyperCard user. However, I’m not sure I would use Director for that level of sketching... I would more likely use paper, and then Director would be used for a more visually rich/accurate interactive prototype. But that sort of functionality would be nice for Director even in those instances.”

» (11) “The idea of working from frame to frame.”

» (13) “To me, it’s like pencil and paper plus navigation logic, so it offers simplicity.”

» (15) “Storyboarding interface is really nice. I’d like to see it better integrated with sketching.”

» (17) “A storyboard view that is not only sequential (like movies). Ability to be fast and sloppy.”

Please write any other recommendations and comments you might have on SILK here and on the back:

» (7) “The widget detection and sketching allows for fast low level interaction. This tool is great for giving the idea of a progression through a program w/o getting into the details of the visual design. Often with tools like Director the high level of visual detail misleads people to thinking more about the visual refinement rather than the interaction.

The UI and interaction of the tool (SILK) itself needs some work. The tool is difficult to use at times and its functionality is not apparent much of the time.”

» (9) “Sometimes the links got messy (too many) and were hard to follow, but I don’t know how to fix that.

Being able to copy from the storyboard would be great; also drag & drop between sketch and storyboard and within the storyboard.

Pen input would also be easier for me to draw shapes with (nothing to do with the program, I guess).

I sometimes wanted different colors in ‘decorate’ mode – also different sizes of text. It was hard to get things looking proportional – such as how big the map is compared to buttons and text, etc.

I would have liked an easier way to share objects – sometimes I wanted to change all titles, but didn’t want to change each frame separately. Being able to copy directly from storyboard would help a little. However, I wouldn’t want to complicate the interface by having object names and attributes, etc. So, I’m not sure about this comment.

SILK seems most useful as something in between paper sketching/ideas and fuller-functioned prototypes, but I am not sure exactly how it would fit into the design process without trying it in that scenario.”

» (11) “I don’t think this works well as a design tool – what does it offer that is better than what is out there already? Why not draw actual boxes – instead of having to spend time drawing shapes that are not easily/quickly recognized by the computer?”

» (13) “Implement size change for type plus bold and italic.
Allow irregular shapes to be made into buttons.
Allow for movable sprites/buttons (I know this is not easy!)
Allow storyboards to be ported to/edited in Director/Hypercard.”

» (15) “When you’re in run mode, you can still delete things from the storyboard. This is bad.

It might be cool if there wasn’t a separate sketch window. If you could zoom in and out of the storyboard, hide and show arrows, and move screens around, you wouldn’t need a sketch window. I found moving screens between the sketch and storyboarding windows to be very confusing.

Again, the value of sketching widgets seemed less and less to me as I had to group and ungroup, change guesses, and so on. I think a more direct interface to modifying widgets than HyperCard/VB would be really nice, though.

Add a color palette! 2 colors (black and green) is not enough for some prototypes, like a weather map.”

» (17) no comments

Engineer Post-Evaluation Questionnaire

1. Assuming that the SILK system was significantly faster and less buggy than the prototype, do you think it would be a better tool for interface design than existing tools (i.e., paper and prototyping tools such as HyperCard, Director, and Visual BASIC)?

» (8) "Yes, in some situations."

» (10) "Yes."

» (12) "Hard to say."

» (14) "N/A."

» (16) "Don't know."

» (18) participant did not answer this question

Why or why not?

» (8) "The editing, saving, and manipulation facilities beat pencil and paper, but they are slower to use. It seems quicker to use than VB or HyperCard, although the finished prototype is also less polished."

» (10) "It's more obviously a sketch, and sets the "drafting" tone more appropriately."

» (12) "I have no idea about the development of the user interface."

» (14) "I haven't used comparable systems, and therefore don't have points of comparison."

» (16) no comments

» (18) no comments

Usability Problems

Did you notice the designer having problems with any of the following:

2. Understanding how to carry out the tasks (check one):

{8, 12, 14} -> 3 no problems {10, 16, 18} -> 3 minor problems__ major problems

Please explain:

» (8) "But I had some difficulty figuring out how to understand the tasks."

» (10) “In trying to bring up a sketch for detailed looking she tried to use a sketching gesture!”

» (12) no comments

» (14) no comments

» (16) “Wasn’t aware of SILK’s capabilities (e.g., clickable image map).”

» (18) “Sometimes took a second attempt to make the right thing hap-pen. Unsure whether ‘copy’ had actually been made.”

3. Knowing what to do next to complete their task (check one):

{8, 10, 12, 14} -> 4 no problems_ {18} -> 1 minor problems___ major problems

» (16) participant did not answer this question

Please explain:

» (8) “Again, the designer had a better handle on what was going on than I did.”

» (10) no comments

» (12) no comments

» (14) no comments

» (16) no comments

» (18) “Still unsure of what to do with date/time.”

4. What are the best aspects of SILK for the user?

» (8) “Flexibility to sketch a wider variety of user interfaces than you can make with traditional widget toolkits.”

» (10) “Seemed easy to use and quick to put together a prototype in.”

» (12) “Easy to manipulate. Straightforward display.”

» (14) “Fast, easy modeling of user interface screens. Good modeling of relations between various phases in the user dialog.”

» (16) no comments

» (18) “Quick to make changes and then see how the interface would look.”

5. What are the worst aspects of SILK for the user?

» (8) “Manipulating the mouse to draw is slow. Its difficult to specify subtle UI manipulation issues (i.e., double-click vs. single-click).”

» (10) “Not sure, but it seemed slightly laborious for her to place text on multiple copies of the ‘same’ screen.”

» (12) “No hints on how to proceed.”

» (14) “From the interaction with the designer, it appeared there was not good way of modeling the system’s reaction to user input errors, also, no provisions for modeling certain types of UI controls (scroll boxes, etc.).”

» (16) “Seemed to go through quite a lot of interaction just to change a small thing.

Geared towards discrete modeling. Not easy to see the effects of sliding the scrollbar or for the designer to change them.”

» (18) “Occasional pauses were annoying.”

6. What were the most common mistakes the system made?

» (8) “Crashing. Not properly dealing with links that were modified and changed.”

» (10) “Didn’t notice any.”

» (12) “No major mistakes, just a bit slow.”

» (14) “No mistakes in this experiment.”

» (16) no comments

» (18) “Didn’t notice any.”

Please write any other recommendations and comments you might have on SILK here and on the back:

» (8) “Drawing with a mouse is difficult. It really needs a pen and tablet.

Although it provides much more free-form design capabilities than a drag & drop interface designer like Visual Basic or Builder Xcessory, it’s sometimes more of a pain to draw things like buttons than it is to drag & drop them from a palette. A hybrid design might be useful.

The storyboard and links are great.”

» (10) “At one point she said ‘I don’t really see it flashing [when changing from one snapshot of a screen to another] but rather the things that change changing by themselves.” – It might be good to support this.”

» (12) no comments

» (14) no comments

» (16) “Built-in graphics/icons would be helpful.”

» (18) “It seemed useful for quickly going through a simple interface design.”

APPENDIX D
SILK Tutorial and Reference

D.1 Introduction

This document is a tutorial for using the SILK user interface design system. SILK is a system in which a designer uses a mouse or electronic pad and pen to sketch user interface widgets, add behaviors via storyboarding, and test the interfaces out. This document will take the reader through building an example application that illustrates the features of SILK.

D.2 Sketching vs. Gestures

There are three main ways of interacting with SILK. It is important to get these clear before starting. You are able to draw widgets and other UI elements by holding down the left button on the mouse and drawing. You can use gestures to perform editing commands by holding down the middle button on the mouse and drawing. Finally, all editing commands have an equivalent that can be selected from the **`SILK Controls`** window.

D.3 Basic Components

The primary method of interaction with SILK is by sketching. SILK tries to recognize the basic shapes that the designer sketches with a single stroke of the pen (i.e., not lifting up the pen or mouse button during the stroke) and then presents the results of its guesses with the **`Primitive Type`** buttons at the bottom of the **`SILK Controls`** window (see Figure D-1).

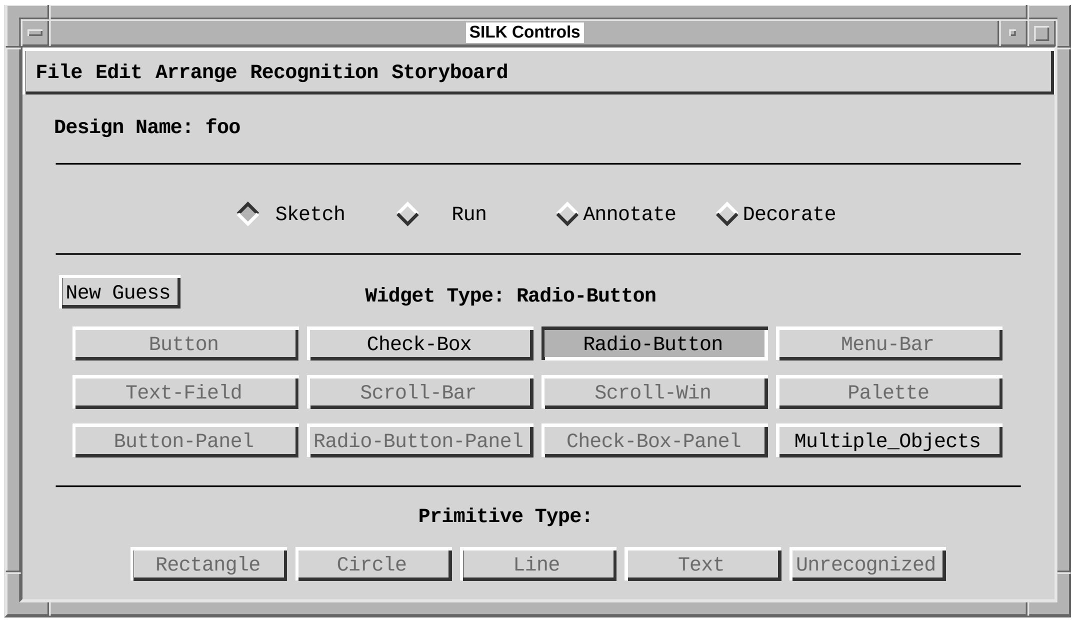


Figure D-1. SILK Controls window.

The basic components that SILK recognizes include rectangles, circles, lines, and text. If a basic component does not fit into one of those categories, it is labeled as `Unrecognized`. The basic components are illustrated in Figure D-2.

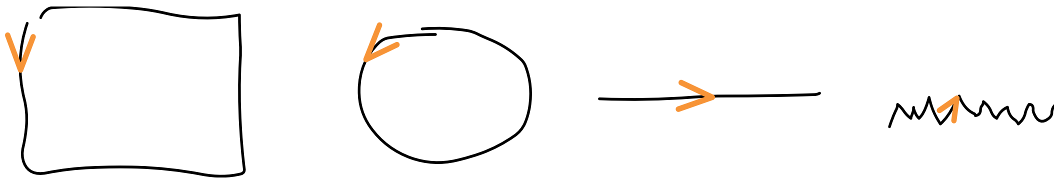


Figure D-2. Basic components recognized by SILK: rectangle, circle, line, and text.

The basic components can be sketched by moving the mouse while holding down the **`left button`**. Each shape must be drawn with a single stroke, that is, you **cannot** create a rectangle by drawing four lines. The suggested stroke directions are given by the arrows in Figure D-2.

When SILK makes a mistake, you can correct the system by clicking on the desired **`Primitive Type`** button. It is important to do this because over time the system will get better at recognizing the way you draw the basic components.

Exercise: *Try drawing each of these shapes in the* ***`SILK Sketch`*** *window and see how the system reports its guess of the type in the* ***`SILK Controls`*** *window.*

D.4 Editing Commands and Gestures

SILK changes the color of the last basic component drawn to purple so that you know which object is selected. You can resize and move objects by dragging with the **`left button`** or **`middle button`** on the selection handles (see Figure D-3). The black handles are used for resizing the object and the white handles are used to move the object. Clicking on an object with the **`left`** or **`middle button`** will also select the object. You must click on the outline of the object – not in the center.

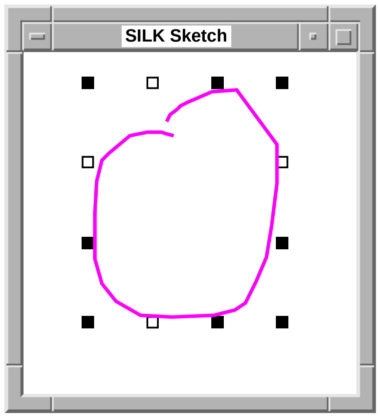


Figure D-3. A selected object with selection handles.

Exercise: *Draw a circle in the **`SILK Sketch`** window and then use the selection handles to resize it. Then move the circle. Now select another object that you drew previously and do the same.*

To delete an object, first select it and then hit the **`Back Space`** key. You can also use **`Cut`**, **`Copy`**, **`Paste`**, **`Clear`**, and **`Undo`** from the **`Edit`** menu.

SILK also allows the use of gestures for specifying many of the editing operations (see Figure D-4). Gestures are created by holding down the **`middle button`** and making the proper pen stroke.

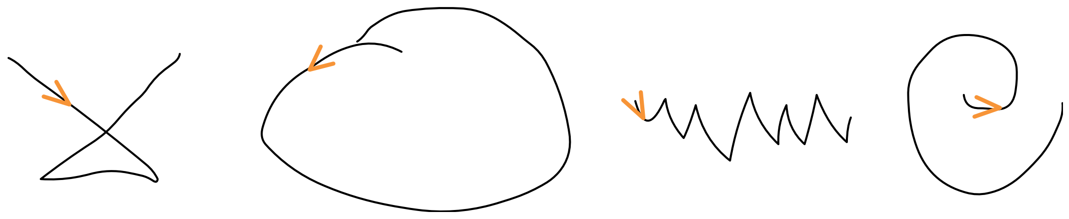


Figure D-4. Editing gestures: clear, group, ungroup, new guess.

Exercise: *Try grouping a few of the objects you have drawn and then move or resize the entire group. Now try deleting the group with the clear gesture. Do not try the "new guess" gesture at this time as it is only useful for reguessing the types of entire widgets rather than basic components. Note: Holding down the* ***`shift`*** *key while clicking can be used for selecting multiple objects at once.*

Exercise: *An easy way to clear the entire sketch is to select* ***`Clear Sketch`*** *from the* ***`Edit`*** *menu. Try that now.*

Note: The group and ungroup gestures are not too useful without an electronic pen, so you may wish to just use the equivalent menu items for these gestures while working with SILK.

D.5 Combining Basic Components Into Widgets

The basic components are usually combined together in order to sketch user interface widgets (e.g., scrollbars, buttons, and windows). For example, specifying a button is as easy as sketching a piece of text inside of a rectangle or oval (see Figure D-5)

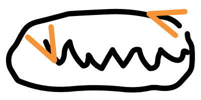


Figure D-5. Sketched button is made up of text inside of rectangle / oval.

Exercise: *Sketch a button.*

SILK tries to infer that the two separate basic components (i.e., the text and the oval) combine to make-up a button. It informs the user of this by displaying **`Button`** in the **`Widget Type`** field of the **`SILK Controls`** window (see Figure D-1).

Sometimes SILK may incorrectly infer what the widget type is. For example, radio buttons and check boxes look quite similar (see Figure D-6). The only difference

Figure D-6. Check box and radio button widgets.

between the two widgets is that a check box has a small rectangle to the left of some text, whereas a radio button has a small circle. If SILK makes the wrong inference, the designer can try the next best guess by clicking on the **`New Guess`** button in the **`SILK Controls`** window or by drawing the **`New Guess`** gesture with the middle mouse button held down. Alternatively, the correct inference can be selected by clicking on one of the buttons under the **`Widget Type`** field in the **`SILK Controls`** window.

Note: clicking on the **`New Guess`** button when there are no further inferences will cause SILK to ungroup the objects in question. Selecting one or more of them and then clicking on **`New Guess`** again will cause the original inference to be chosen again.

Hint: When objects do not group together the way you would expect, try moving them closer together and clicking on the **`New Guess`** button.

Exercise: *Sketch a radio button and then see which inference SILK comes up with. Click on the* **`New Guess`** *button to see the next best inference.*

Another widget that SILK can recognize is a text field. This is specified by sketching some text with a rectangle to the right of it (see Figure D-7).

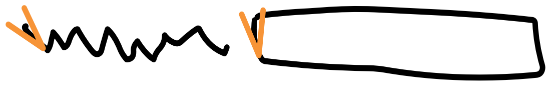


Figure D-7. Text field widget.

Exercise: *Sketch a text field.*

D.6 Practice Design Problem

You will design an interface to a service that will allow potential home buyers to view houses on the market using a home computer. The service will allow the user to specify various search criteria such as price and location, after which the system will show a series of photos for each of the available houses and allow the user to set up an appointment to visit the house.

This document will lead you through building a possible interface to this system. You should follow the steps in the document and try to build the interface in order to become familiar with SILK. Sometimes the system will make a mistake. That is, the system will not do what you expected it to. Please inform us when this happens so that we can take note of it.

The first step is to sketch the opening search screen that allows the user to specify a price range and city in which to search for available houses. The screen should have a title, two text fields for the price range, a text field for the location, and two buttons (one for starting the search and one for clearing the form). An example of such an interface is shown in Figure D-8.

Exercise: *Duplicate the interface shown in Figure D-8.*

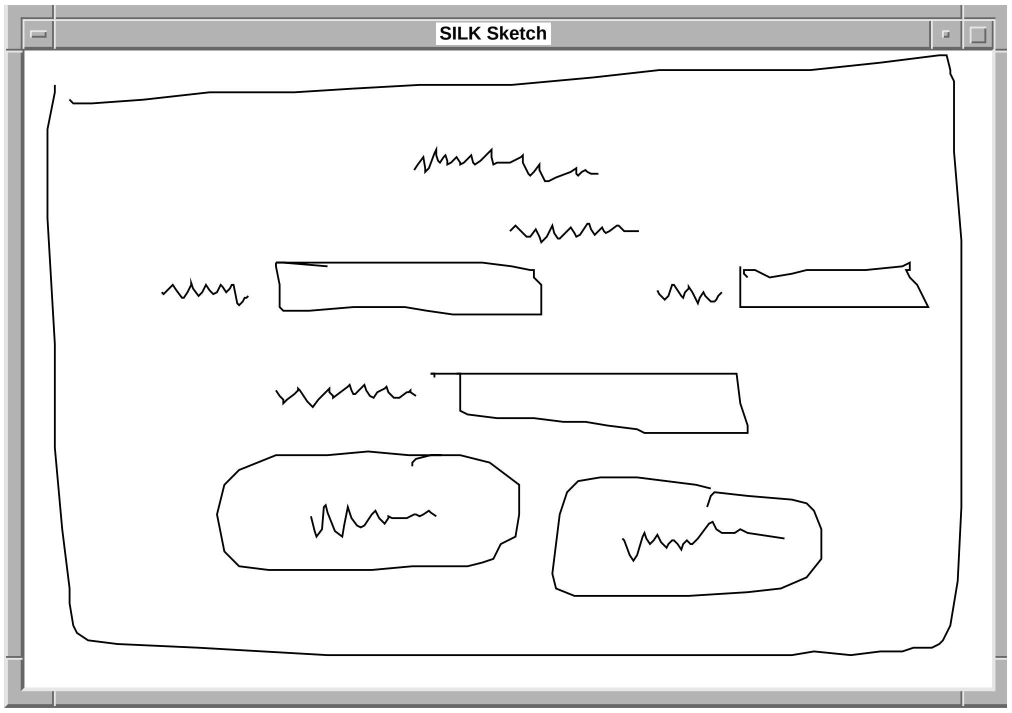


Figure D-8. Sketched search form.

D.7 Replacing Text

Now we will replace the “squiggly” lines representing text with typed text. This is done by drawing a small **`Caret`** gesture while the **`middle button`** is held down on top of the text (see Figure D-9). Pressing the **`Return`** key saves the text. Pressing **`Control-g`** cancels the text.

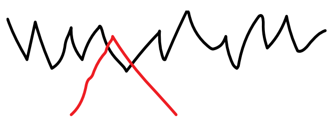


Figure D-9. A **`Caret`** gesture on top of some text to replace.

You can also use the same **`Caret`** gesture to edit existing typed text. Just draw the **`Caret`** gesture on top of the text to be edited and you will be given a cursor in the text.

Exercise: *Replace the "squiggly" text with real text as we have done in Figure D-10. Try editing the text of one of the items after you have already replaced the squiggly text.*

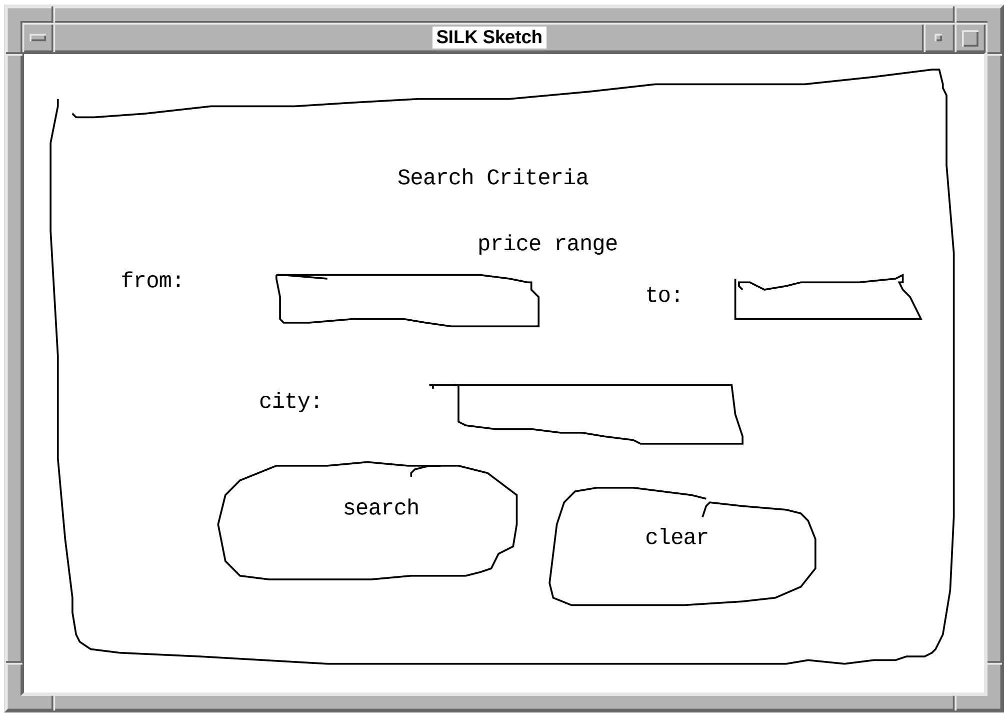


Figure D-10. The search form with real text for labels.

Now is a good time to save your work. You should do that after each major piece has been sketched so that it will be easy to recover if the system crashes. To save the sketch select **`Save Design...`** from the **`File`** menu of the **`SILK Controls`** window. Type in a filename in the dialog box and click on **`Save`**. Please use filenames of the form: *tutorial1, tutorial2,* etc. It would also be a good time to print your work. You can do this by selecting **`Print Design...`** from the **`File`** menu.

D.8 Storyboarding

Now you can copy the *screen* illustrated in the sketch to the **`SILK Storyboard`** window. You do this by selecting **`Copy Screen to Storyboard`** from the **`Storyboard`** menu of the **`SILK Controls`** window. You will leave it there for now and come back to it later.

Now you will type in some sample values in the text fields of the search form. In order to do this just use the **`Caret`** gesture on top of the text field boxes as you did earlier for replacing “squiggly” lines.

Exercise: *Fill in the text fields as in Figure D-11.*

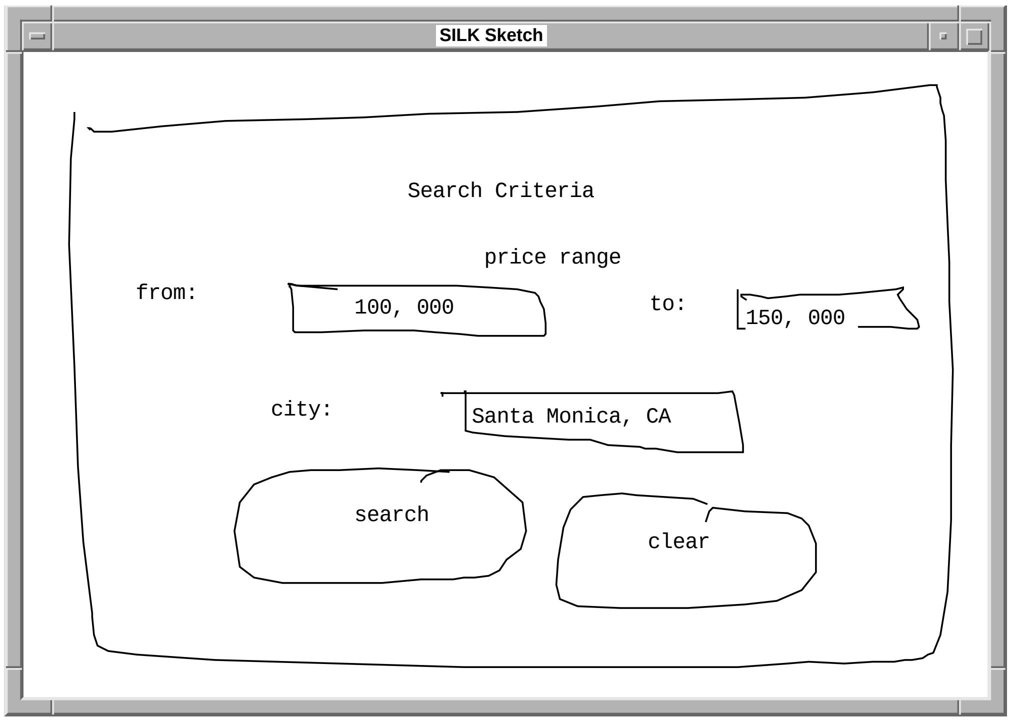


Figure D-11. The search form with values for the text fields.

Now you can copy this new version of the sketch to the **`SILK Storyboard`** window. You do this by selecting **`Copy Screen to Storyboard.`** You should now have two versions of the search criteria form in the **`SILK Storyboard`** window.

SILK storyboards allow you to flip between screens on button clicks. This is done by drawing arrows with the **`left`** mouse button in the **`SILK Storyboard`** window. We would like to go from the blank Search Criteria screen to the filled-in one when the user clicks on any of the text fields (we will pretend that they typed in the text). To do this you draw arrows in the **`SILK Storyboard`** window using the left button from the text fields in the blank form to the filled-in form as in Figure D-12. We would also like to go from the filled-in Search Criteria screen to the blank one when the user clicks on the Clear button. To do this you draw an arrow from the Clear button in the filled-in form to the blank form.

Drawing arrows means just drawing a line between the two points you wish to connect. SILK inserts the arrow anchor and pointer for you.

Exercise: *Draw the arrows as shown in Figure D-12.*

Now is a good time to save and print your work.

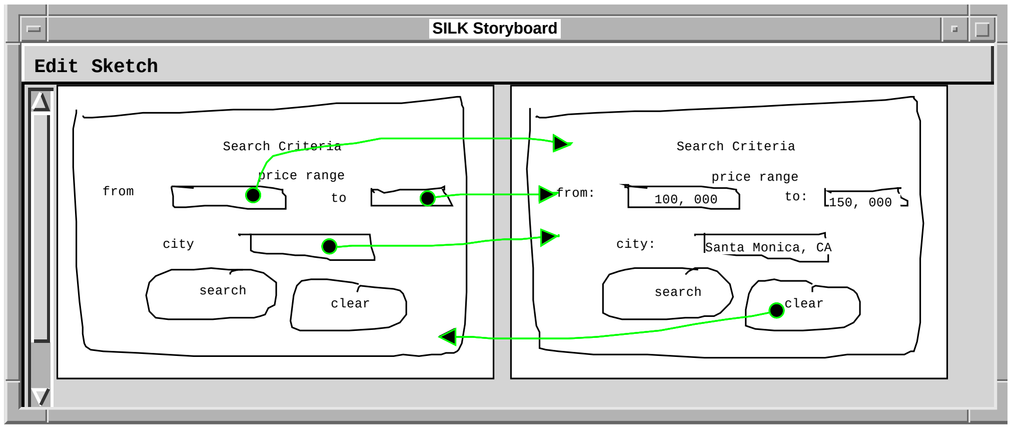


Figure D-12. Storyboard for filling-in and clearing the Search Criteria form.

D.9 Run Mode

SILK's *Run* mode will let you try this interface out. First you need to get the blank Search Criteria form back to the sketch window, as that is the screen to interact with first. Select that screen by clicking on it with the **`middle button`**. Then select **`Copy Screen to Sketch`** from the **`Sketch`** menu in the **`SILK Storyboard`** window.

To switch into *Run* mode, click on the **`Run`** radio button in the **`SILK Controls`** window. Note that the cursor changes to indicate the mode you are in. You can then test the interface by clicking on the text fields and the Clear button in the **`SILK Sketch`** window. The screen should make the desired transition.

Exercise: *Copy the proper screen to the **`SILK Sketch`** window and switch to run mode. Test the interface by clicking on the text fields and the Clear button.*

You can now design the screen that will appear when the user clicks on the Search button. The screen will consist of a scrolling window that contains two potential houses. The houses are sketched along with some information on the house and buttons from each house for making an appointment. In addition, there is a button to bring up the Search Criteria form in order to perform a new search. The rough sketch is illustrated in Figure D-13.

Exercise: *Make sure to switch back to sketch mode and then duplicate the rough sketch from Figure D-13.*

Now is a good time to save and print your work.

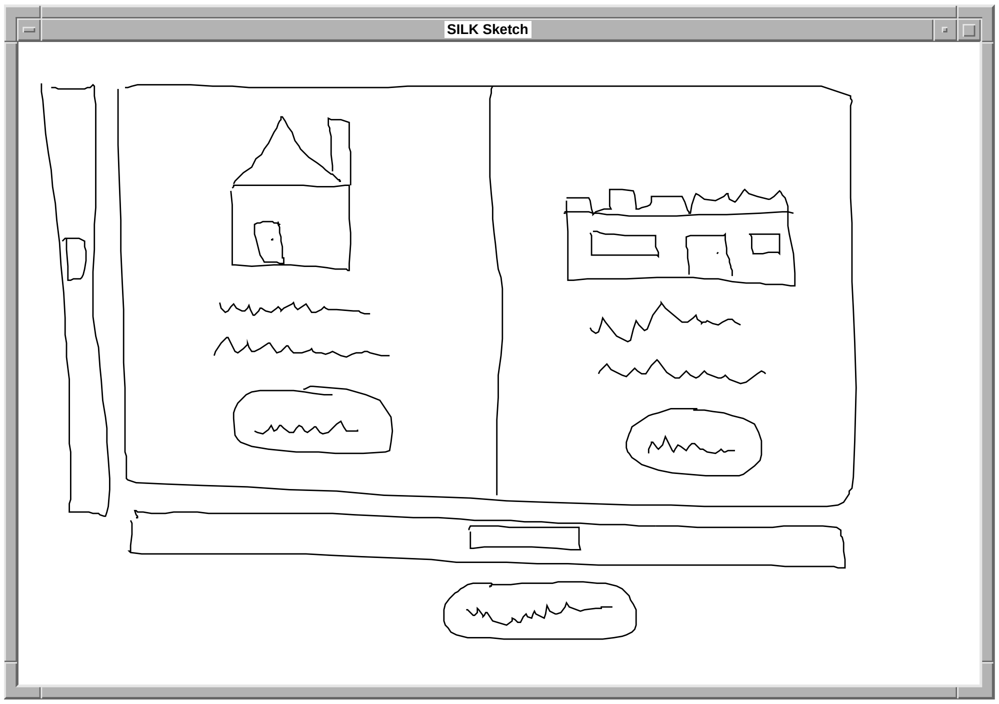


Figure D-13. Rough sketch of houses found from the search.

Next, fill in the details on the rough sketch to get something like Figure D-14.

Exercise: *Fill in the details on your rough sketch like in Figure D-14 and copy the sketch to the* ***`SILK Storyboard`*** *window. We will come back to it later.*

Now is a good time to save and print your work.

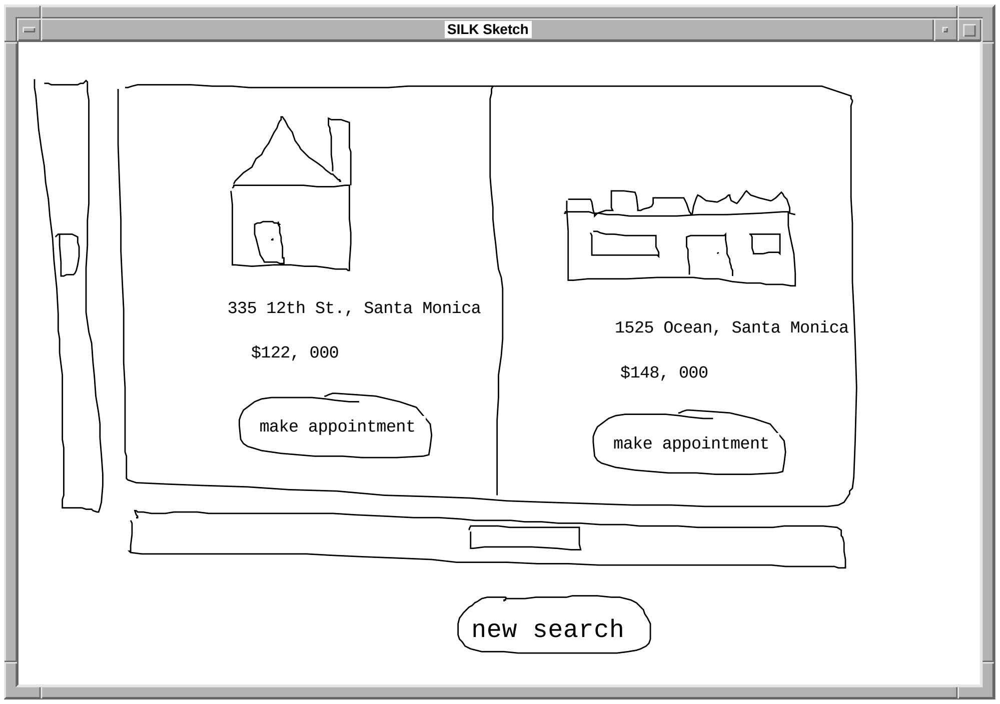


Figure D-14. Our houses screen after adding more details.

Finally, you need to design a window for scheduling an appointment for houses that the user likes. The example shown in Figure D-15 could be used for testing the application.

Exercise: *Sketch a rough version of the appointment screen and then fill in the details. When you are finished, copy it to the* ***`SILK Storyboard`*** *window.*

Now is a good time to save and print your work.

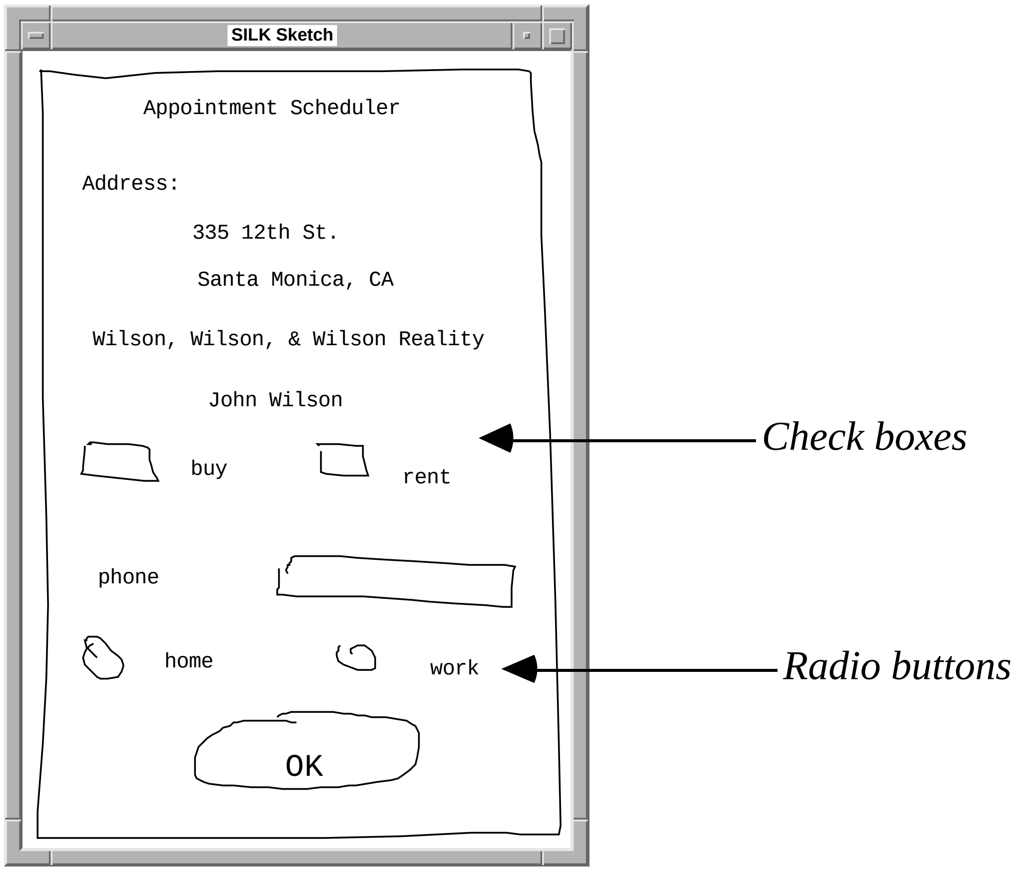


Figure D-15. Appointment scheduling screen.

Now you are ready to add the final arrows to finish our application. You need an arrow from the Search button of the filled-in Search Criteria form to the houses screen. You also need an arrow from the Make Appointment button of the houses screen to the appointment scheduling screen. This is illustrated in Figure D-16. Remember, you draw the arrows with the **`left`** mouse button.

Exercise: *Draw the needed arrows as shown in Figure D-16.*

Now is a good time to save and print your work.

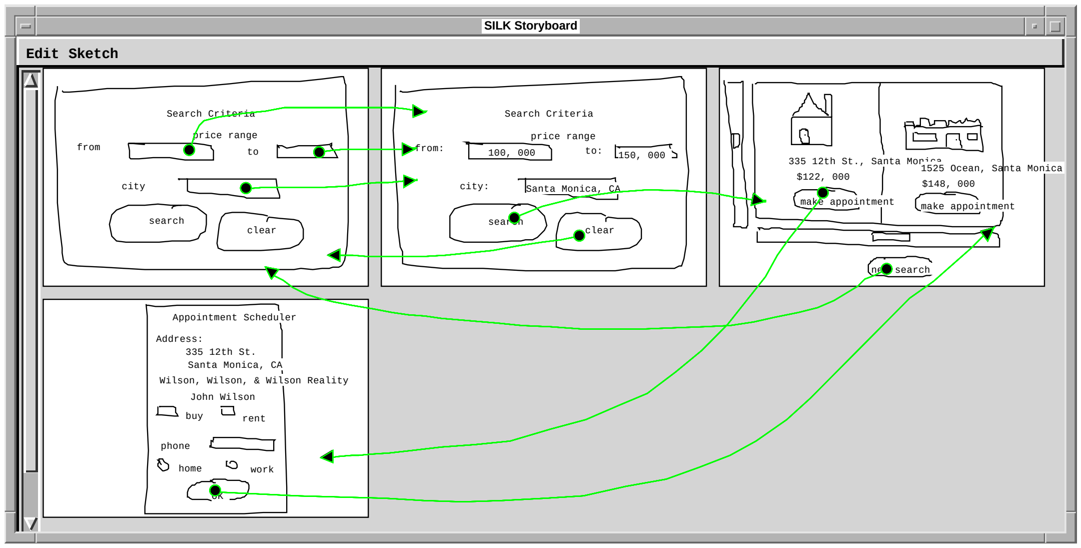


Figure D-16. Complete storyboard for our house search application.

Exercise: *Now, the entire application can be tested by copying the blank search screen to the* ***`SILK Sketch`*** *window, switching to run mode, and trying it out!*

You should now read the attached **SILK Quick Reference Manual** to make sure you understand the features available to you.

D.10 SILK Quick Reference Manual

This manual gives a complete listing of the different commands and operations supported by SILK.

D.10.1 SILK Modes

SILK supports the following modes of operation that can be selected from the radio buttons in the **`SILK Controls`** window:

- *Sketch* - all sketched objects are interpreted as possible widgets or other basic primitives.
- *Run* - allow the user to manipulate the interface widgets.
- *Annotate* - all strokes and typed text are annotations for documentation (these marks are hidden when in any other mode).
- *Decorate* - strokes and text are treated as uninterpreted graphics.

Note: Strokes can be cut & pasted between the different layers associated with each mode.

D.10.2 Mouse buttons

middle or left button click	select object
left button draw	sketch object
left button or middle button on selection handles	move or resize
middle button draw	editing gesture

Note: Holding down the **`shift`** key while clicking can be used for selecting multiple objects at once.

D.10.3 Supported Basic Components

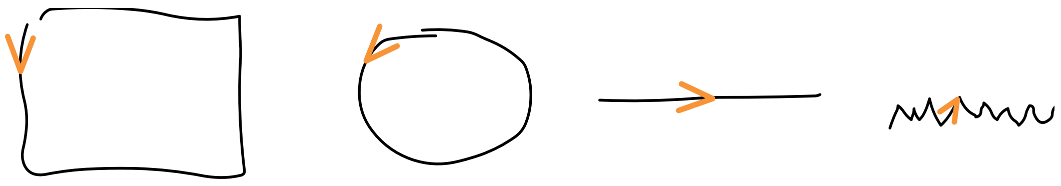


Figure D-17. Rectangle, circle, line, and text basic components.

D.10.4 Supported Editing Gestures

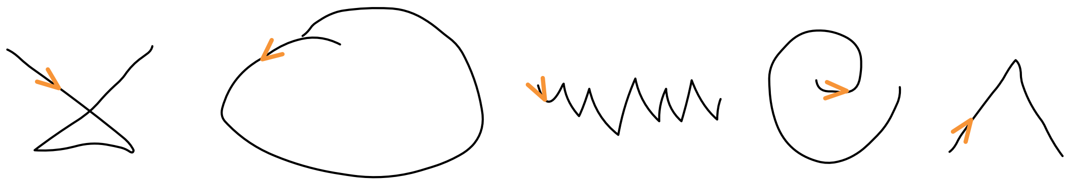


Figure D-18. Clear, group, ungroup, new guess, and text editing gestures.

D.10.5 Adding Typed Text

Typed text can be either typed directly on the sketch or can be used to replace the "squiggly" lines representing text. Both can be accomplished by drawing the **`Caret`** gesture (see Figure D-19)using the middle mouse button at the desired position of the new text. After typing the text, press the **`Return`** key to save the text. To cancel the text, press **`Control-g`**.

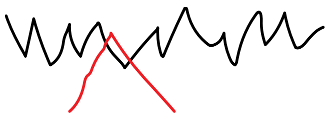


Figure D-19. Caret gesture for inserting typed text.

D.10.6 Supported Widgets

Note: SILK is sensitive to the position and sizes of the primitive components that make up the following widgets, so it is wise to sketch them in a similar manner to the examples. The arrows are drawn as a guide on where and in what direction to start a basic component. The objects do not need to be drawn in that manner, since correcting those primitive inferences will cause the system to learn, but SILK will do better if the directions and starting positions illustrated below are used.

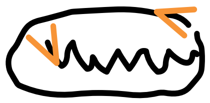


Figure D-20. A button.

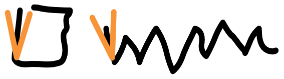


Figure D-21. A check box.

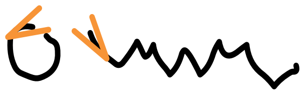


Figure D-22. A radio button.

SILK also recognizes vertical and horizontal sequences of buttons, check boxes, and radio buttons as panels.

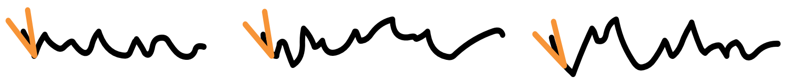


Figure D-23. A menu bar. (Note: must be drawn at upper left corner of the **`SILK Sketch`** window.)

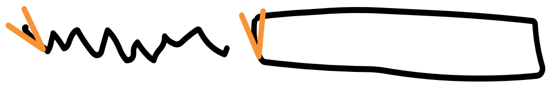


Figure D-24. A text field.

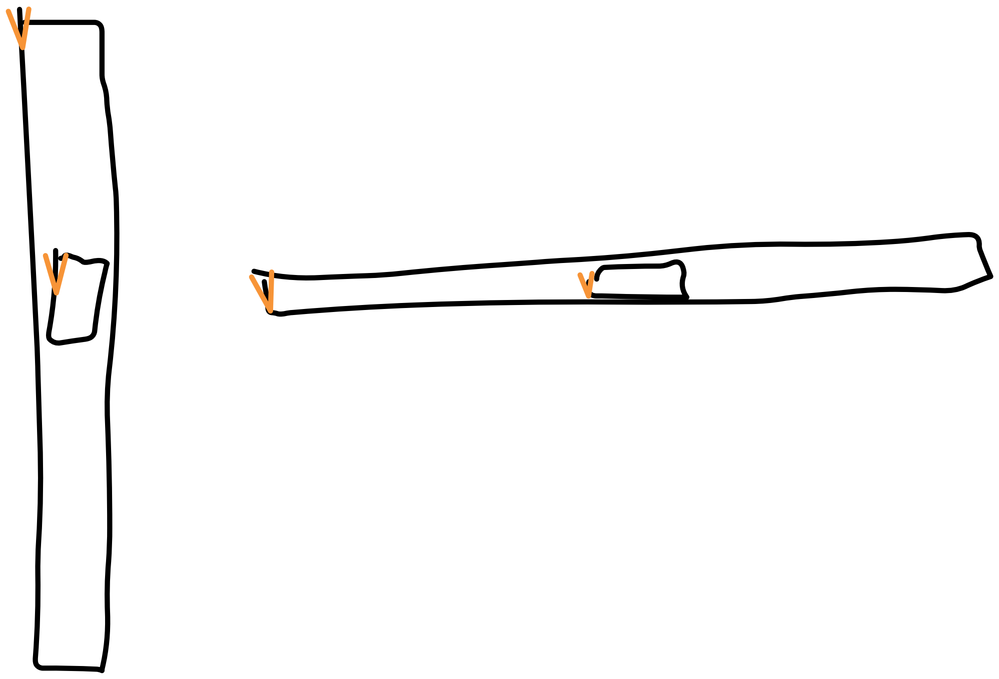


Figure D-25. Vertical and horizontal scroll bars.

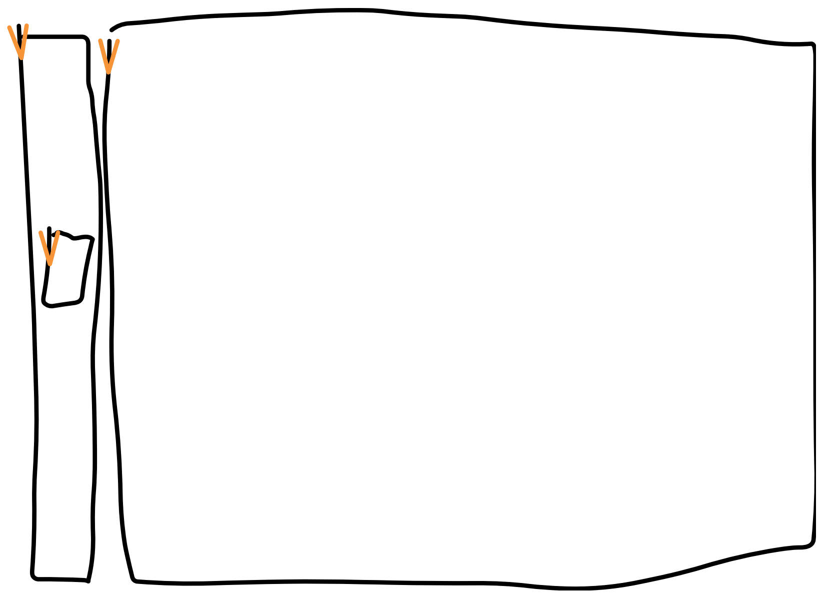


Figure D-26. A scrolling window. (Note: can have any combination of scroll bars at left and below).

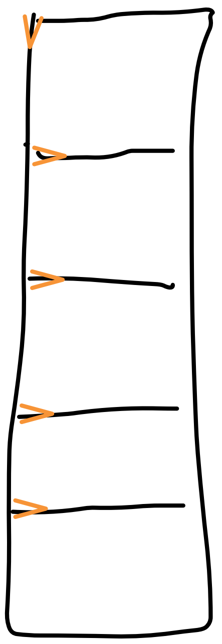


Figure D-27. A palette. (Note: arbitrary graphics can be drawn in the boxes.)

D.10.7 Storyboarding

We have chosen a simple model for our storyboarding language. Our visual language has two types of objects, *screens* and *arrows*. Each screen is a sketch of an interface in a particular state. Arrows connect objects contained in one screen with a second screen. The arrow indicates that when the object in the first screen is manipulated (our current model is limited to mouse clicks), the system should display the second screen in the **`Sketch`** window instead of the first. The example shown in Figure D-28 specifies that the dialog box should appear when the left button is pressed.

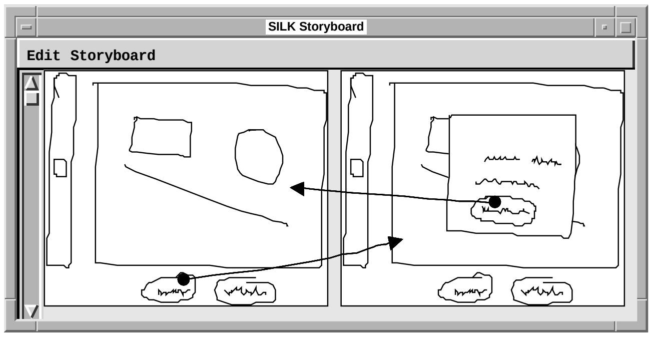


Figure D-28. Storyboard that causes the dialog box to appear and disappear.

The designer constructs storyboards by sketching screens in the SILK **`Sketch`** window. Screens are then copied to the storyboard window using either **`Copy Screen to Storyboard`** or **`Paste Screen to Selected Storyboard Screen`** from the **`Storyboard`** menu in the SILK **`Controls`** window. At this point, the original screen can be modified in the sketch window to show how its state might change. After this, the new screen is also copied to the storyboard window. Now, the designer can start drawing arrows in the storyboard window that indicate screen sequencing or more screens can be produced.

The arrows can be drawn from any widget, graphical object (*e.g.*, decorations), or the background to another screen. Thus, the designer can cause transitions to occur when the user clicks on any of these items. The arrows show an anchor point on the object they were drawn from and an arrowhead on the screen they are drawn to.

Arrows and screens can be deleted from the **`Storyboard`** window. Screens can also be cut, copied, and pasted.

D.10.8 Giving Hints to the Inference Engine

There are a few ways to help SILK's inference engine be more successful at guessing the proper widgets.

- always correct the system by clicking on the proper **`Primitive Type`** button when SILK makes a mistake at that level.
- use multiple-selections (e.g., select multiple pieces of a sketched widget) to inform SILK of related items and to ignore internal rules for "nearness" and "containment". For example, by selecting a circle and a text object that are far from each other, we can still get the system to infer it as a radio button.

D.10.9 Menu Items

This section describes all of the commands available from the menus in the **`SILK Controls`** window and the **`SILK Storyboard`** window.

D.10.9.1 Control Window Menu Items

File menu:

New Design - delete current design from both the **`Sketch`** and **`Storyboard`** windows

Open Design... - load a saved design from disk

Save Design... - save a design to disk

Print Design... - print design to printer named by shell environment variable $PRINTER

Quit - exit SILK

Edit menu:

Undo - undo last cut, copy, paste, or clear

Cut - delete selected object and move to the copy buffer

Copy - copy selected object to copy buffer

Paste - paste object from copy buffer

Clear - delete selected object

Clear Sketch - delete all objects in **`Sketch`** window

Select All - select all objects in **`Sketch`** window

Scale Selection... - scale selected object(s)

Arrange menu:

Group - group selected objects

Ungroup - ungroup selected group object

Make Opaque - makes the given object or group opaque

Recognition menu:

Transform... - transform widgets in **`Sketch`** window to a Motif interface and place them in the **`Finished`** window

New Guess - reinfer widgets related to selected object

Storyboard menu:

Copy Screen to Storyboard - copy screen in **`Sketch`** window to **`Storyboard`** window

Paste Screen to Selected Storyboard Screen - copy screen in **`Sketch`** window and paste it over the selected screen in the **`Storyboard`** window

D.10.9.2 Storyboard Window Menu Items

Edit menu:

Undo - unimplemented

Cut - delete selected screen and move to the copy buffer

Copy - copy selected screen to copy buffer

Paste - paste screen from copy buffer

Clear - delete selected screen or arrow

Clear Storyboard - delete all screens and arrows in **`Storyboard`** window

Search Annotations... - brings up a dialog box that allows a text string search of the annotations on all screens in the storyboard.

Sketch menu:

Copy Screen to Sketch - copy selected screen from **`Storyboard`** window to **`Sketch`** window.

APPENDIX E

SILK Rules

This appendix contains the rules used by SILK for recognizing widgets from among the primitive components and previously recognized widgets sketched by the designer. The description of the recognition algorithm is given in Chapter 3, "Widget Recognition". The actual rules used by SILK are LISP functions that may include `if` statements and iterators (e.g., `dolist`) over lists of objects that satisfy nearness, containment, or sequence relationships. Thus, there is no need to test the rest of the rule if no such relationship exists. These rules have been rewritten here for clarity to contain only `and`'s, `or`'s, and calls to simple functions that test for the existence of a property (e.g., `contains-p`). In addition, only the "test" parts of the rules are shown. The "then" parts simply return a certainty value and a function to add interactive behavior to the parts of the widget identified by the "test." These "then" parts are all of the same format as the one shown in Figure 3-20 on page 45.

```
(and (contains-p container containee)
     (or (rectangle-p container)
         (circle-p container))
     (text-p containee)
     (button-size-p container))
```

Figure E-1. Test for button rule which results in certainty value of 1.

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

Figure E-2. Test for vertical scrollbar rule which results in certainty value of 1.

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

Figure E-3. Test for horizontal scrollbar rule which results in certainty value of 1.

```
(and (scroll-bar-p comp1)
     (vertical-p comp1)
     (rectangle-p comp2)
     (horizontal-left-p comp1 comp2)
     (near-p comp1 comp2)
     (close-height-p comp1 comp2))
```

Figure E-4. Test for new scrolling window rule with scrollbar on left which results in certainty value of 1.

```
(and (scroll-bar-p comp1)
     (horizontal-p comp1)
     (rectangle-p comp2)
     (vertical-above-p comp1 comp2)
     (near-p comp1 comp2)
     (close-width-p comp1 comp2))
```

Figure E-5. Test for new scrolling window rule with scrollbar below which results in certainty value of 1.

```
(and (scroll-bar-p comp1)
     (vertical-p comp1)
     (scroll-win-p comp2)
     (null (has-v-scroll-bar-p comp2))
     (horizontal-left-p comp1 comp2)
     (near-p comp1 comp2)
     (close-height-p comp1 comp2))
```

Figure E-6. Test for scrolling window rule with new scrollbar on left which results in certainty value of 1.

```
(and (scroll-bar-p comp1)
     (horizontal-p comp1)
     (scroll-win-p comp2)
     (null (has-h-scroll-bar-p comp2))
     (vertical-above-p comp1 comp2)
     (near-p comp1 comp2)
     (close-width-p comp1 comp2))
```

Figure E-7. Test for scrolling window rule with new scrollbar on bottom which results in certainty value of 1.

```
(and (rectangle-p comp1)
     (text-p comp2)
     (near-p comp1 comp2)
     (aligned-p comp1 comp2 :bottom)
     (horizontal-left-p comp1 comp2)
     (check-box-width-p comp1))
```

Figure E-8. Test for check box rule which results in certainty value of 1.

```
(and (circle-p comp1)
     (text-p comp2)
     (near-p comp1 comp2)
     (aligned-p comp1 comp2 :bottom)
     (horizontal-left-p comp1 comp2)
     (check-box-width-p comp1))
```

Figure E-9. Alternative test for check box rule which results in certainty value of 0.8.

```
(and (circle-p comp1)
     (text-p comp2)
     (near-p comp1 comp2)
     (aligned-p comp1 comp2 :bottom)
     (horizontal-left-p comp1 comp2)
     (check-box-width-p comp1))
```

Figure E-10. Test for radio button rule which results in certainty value of 1.

```
(and (rectangle-p comp1)
     (text-p comp2)
     (near-p comp1 comp2)
     (aligned-p comp1 comp2 :bottom)
     (horizontal-left-p comp1 comp2)
     (check-box-width-p comp1))
```

Figure E-11. Alternative test for radio button rule which results in certainty value of 0.8.

```
(and (or (button-p component)
         (button-panel-p component))
     (in-sequence-p component sequence))
```

Figure E-12. Test for button panel rule which results in certainty value of 1.

```
(and (or (check-box-p component)
         (check-box-panel-p component))
     (in-sequence-p component sequence))
```

Figure E-13. Test for check box panel rule which results in certainty value of 1.

```
(and (or (radio-button-p component)
         (radio-button-panel-p component))
     (in-sequence-p component sequence))
```

Figure E-14. Test for radio button panel rule which results in certainty value of 1.

```
(or (and (menu-bar-p component)
         (in-sequence-p component sequence :horizontal))
    (and (text-p component)
         (in-sequence-p component sequence :horizontal))
         (upper-left-p (first sequence))))
```

Figure E-15. Test for menu bar rule which results in certainty value of 1.

```
(and (contains-p container containee)
     (or (rectangle-p container)
         (palette-p container))
     (or (and (line-p containee)
              (close-size-p containee container
                                     :width))
         (not (line-p containee))))
```

Figure E-16. Test for palette rule which results in certainty value of 1.

```
(and (text-p comp1)
     (rectangle-p comp2)
     (near-p comp1 comp2)
     (aligned-p comp1 comp2 :bottom)
     (horizontal-left-p comp1 comp2))
```

Figure E-17. Test for text field rule which results in certainty value of 1.

References

[Apple 1993] Apple. *HyperCard*. Apple Computer Inc., Cupertino, CA. 1993.

[Apte 1993a] Ajay Apte and Takayuki Dan Kimura. “A Comparison Study of the Pen and the Mouse in Editing Graphic Diagrams”. *Proceedings of 1993 IEEE Symposium on Visual Languages*, Bergen, Norway, IEEE Computer Society Press. Aug. 24–27, 1993. pp. 352-357.

[Apte 1993b] Ajay Apte, Van Vo and Takayuki Dan Kimura. “Recognizing Multi-stroke Geometric Shapes: An Experimental Evaluation”. *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '93, Atlanta, GA, November 3–5, 1993. pp. 121-128.

[Bederson 1994] Benjamin B. Bederson and James D. Hollan. “Pad++: A Zooming Graphical Interface for Exploring Alternative Inteface Physics”. *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '94, Marina del Rey, CA, November 2–4, 1994. pp. 17-26.

[Black 1990] Alison Black. “Visible Planning on Paper and on Screen: The Impact of Working Medium on Decision-making by Novice Graphic Designers,” *Behaviour & Information Technology*. **9**(4): pp. 283-296, 1990.

[Borland 1996] Borland. *Delphi*. Borland International Inc., 100 Borland Way, Scotts Valley, CA 95066, (408) 431-1000. 1996.

[Boyarski 1994] Daniel Boyarski and Richard Buchanan. “Computers and Communication Design: Exploring the Rhetoric of HCI,” *Interactions*. **1**(2): pp. 24-35, April, 1994.

[Branco 1994] V. Branco, A. Costa and F.N. Ferreira. “Sketching 3D Models with 2D Interaction Devices,” *Computer Graphics Forum*. Proceedings of Eurographics '94. **13**(3): pp. 489-502, Sept. 12–16, 1994.

[Brocklehurst 1991] E. R. Brocklehurst. “The NPL Electronic Paper Project,” *International Journal of Man-Machine Studies.* **34**(1): pp. 69-95, 1991.

[Burgess 1993] John Burgess. “The Newton: PC Prophecy, or a Pratfall”. *The Washington Post*. Washington, DC, October 24, 1993. sec. H, p. 1.

[Buxton 1986] William Buxton. “There's More to Interaction Than Meets the Eye: Some Issues in Manual Input.” *User Centered Systems Design: New Perspectives on Human-Computer Interaction.* D. A. Norman and S. W. Draper, Eds. Hillsdale, N.J.: Lawrence Erlbaum Associates. pp. 319-337. 1986.

[Buxton 1983] W. Buxton, M.R. Lamb, D. Sherman and K.C. Smith. “Towards a Comprehensive User Interface Management System,” *Computer Graphics.* Proceedings of SIGGRAPH '83. **17**(3): pp. 35-42, July, 1983.

[Bylinksy 1993] Gene Bylinksy. “Russian Help for Apple's Newton,” *Fortune.* **128**(1): p. 12, July 12, 1993.

[Callahan 1988] Jack Callahan, Don Hopkins, Mark Weiser and Ben Shneiderman. “An Empirical Comparison of Pie vs. Linear Menus”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '88 Conference, 1988. pp. 95-100.

[Cardelli 1987] Luca Cardelli. *Building User Interfaces by Direct Manipulation.* Report #22, Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. 1987.

[Cardelli 1988] Luca Cardelli. “Building User Interfaces by Direct Manipulation”. *Proceedings of ACM SIGGRAPH Symposium on User Interface Software and Technology*, UIST '88, Banff, Canada, October, 1988. pp. 152-166.

[Carr 1991] Robert Carr and Danny Shafer. The Power of PenPoint. Addison-Wesley. 1991.

[Chang 1987] Shi-Kuo Chang. “Visual Languages: A Tutorial and Survey,” *IEEE Software.* **4**(1): pp. 29-39, January, 1987.

[Chang 1986] Shi-Kuo Chang, Tadao Ichikawa and Panos A. Ligomenides, Eds. Visual Languages. New York: Plenum Press. 1986.

[Charniak 1991] Eugene Charniak. “Bayesian Networks without Tears,” *AI Magazine.* **12**(4): pp. 50-63, Winter, 1991.

[Citrin 1996] Wayne Citrin and Mark D. Gross. “Distributed Architectures for Pen-based Input and Diagram Recognition”. *Proceedings of ACM Workshop on Advanced Visual Interfaces*, AVI '96, Gubbio, Italy, May, 1996.

[Citrin 1995] W. Citrin and McWhirter J. “Diagram Entry Mechanisms in Graphical Environments”. *Proceedings of Human Factors in Computing Systems (Conference Companion)*, ACM CHI '95, Denver, CO, May 7–11, 1995. pp. 294-295.

[Citrin 1993] Wayne V. Citrin. “Requirements for Graphical Front Ends for Visual Languages”. *Proceedings of 1993 IEEE Symposium on Visual Languages*, Bergen, Norway, IEEE Computer Society Press. Aug. 24–27, 1993. pp. 142-150.

[Cypher 1993] Allen Cypher. Watch What I Do: Programming by Demonstration. Cambridge, MA: MIT Press. 1993.

[Cypher 1995] Allen Cypher and David Canfield Smith. “KidSim: End User Programming of Simulations”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '95, Denver, CO, May 7–11, 1995. pp. 27-34.

[Davis 1964] M.R. Davis and T.O. Ellis. “The Rand Tablet: A Man-Machine Graphical Communication Device”. *Proceedings of Fall Joint Computer Conference Proceedings (26),* AFIPS, 1964. pp. 325-331.

[Design 1993] Fractal Design. *Sketcher.* Fractal Design Corp., P.O. Box 2380, Aptos, CA 95001, (408) 688-8800. 1993.

[Design 1996] Fractal Design. *Painter 4*. Fractal Design Corp., P.O. Box 2380, Aptos, CA 95001, (408) 688-8800. 1996.

[Draper 1986] Donald A. Norman and Stephen W. Draper, Ed. User Centered System Design: New Perspectives on Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates. 1986.

[Elrod 1992] Scott Elrod *et. al.* “Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '92, Monterey, CA, May 3–7, 1992. pp. 599-607.

[Fehr-Snyder 1995] Kerry Fehr-Snyder. “What Ever Happened to Bob, Lisa & Newton? Computer Landscape is Littered With Products that Commercially Just Couldn't Cut the Mustard”. *Arizona Republic*. Phoenix, AZ, September 11, 1995. sec. E, p. 1.

[Ferguson 1993] Paula Ferguson and David Brennan. Motif Reference Manual, Volume 6B. O'Reilly & Associates. 1993.

[Fish 1990] J. Fish and S. Scrivener. “Amplifying the Mind’s Eye: Sketching and Visual Cognition,” *Leonardo.* **23**(1): pp. 117-126, 1990.

[Fisher 1992] Gene L. Fisher, Dale E. Busse and David A. Wolber. “Adding Rule-Based Reasoning to a Demonstrational Interface Builder”. *Proceedings of ACM Symposium on User Interface Software and Technology,* UIST '92, Monterey, CA, November 15–18, 1992. pp. 89-97.

[Fitzgerald 1994] Michael Fitzgerald. “This Handwriting is Recognizable,” *Computerworld.* **28**(40): p. 44, October 3, 1994.

[Fodor 1975] Jerry A. Fodor. The Language of Thought. Cambridge, MA: Harvard University Press. 1975.

[Frankish 1995] Clive Frankish, Richard Hull and Pam Morgan. “Recognition Accuracy and User Acceptance of Pen Interfaces”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '95, Denver, CO, May 7–11, 1995. pp. 503-510.

[Fujisaki 1990] T. Fujisaki, T.E. Chefalas, J. Kim and C.C. Tappert. “Online Recognizer for Runon Handprinted Characters”. *Proceedings of 10th International Conference on Pattern Recognition (1)*, Atlantic City, NJ, IEEE Computer Society Press. June 16–21, 1990. pp. 450-454.

[Furnas 1986] George W. Furnas. “Generalized Fisheye Views”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '86, Boston, MA, 1986. pp. 16-23.

[Genau 1995] Andreas Genau and Axel Kramer. “Translucent History”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '95, Denver, CO, May 7-11, 1995. pp. 250-251.

[Gleicher 1994] Michael Gleicher and Andrew Witkin. “Drawing With Constraints,” *The Visual Computer.* **11**(1): pp. 39-51, 1994.

[GO 1992] GO. PenPoint User Interface Design Reference. Reading, MA: Addison-Wesley. 1992. ISBN 0-201-60858-8.

[Goel 1995] Vinod Goel. Sketches of Thought. Cambridge, MA: The MIT Press. 1995.

[Goldberg 1993] David Goldberg and Cate Richardson. “Touch-typing With a Stylus”. *Proceedings of Human Factors in Computing Systems*, ACM INTERCHI '93, Amsterdam, The Netherlands, April 24–29, 1993. pp. 80-87.

[Golin 1990] Eric J. Golin. *A Method for the Specification and Parsing of Visual Languages*. Ph.D. dissertation, Department of Computer Science, Brown University, Providence, R.I. 1990.

[Gomoll 1990] Kathleen Gomoll. “Some Techniques for Observing Users.” *The Art of Human-Computer Interface Design.* B. Laurel, Ed. Reading, MA: Addison-Wesley. pp. 85-90. 1990.

[Gould 1985] J.D. Gould and C. Lewis. “Designing for Usability: Key Principles and What Designers Think,” *Communcations of the ACM.* **28**(3): pp. 300-311, March, 1985.

[Gross 1994] Mark D. Gross. “Recognizing and Interpreting Diagrams in Design”. *Proceedings of the ACM Conference on Advanced Visual Interfaces '94*, Bari, Italy, June, 1994. pp. 89-94.

[Gross 1996] Mark D. Gross and Ellen Yi-Luen Do. “Ambiguous Intentions: A Paper-like Interface for Creative Design”. *Proceedings of ACM Symposium on User Interface Software and Technology,* UIST '96, Seattle, WA, November 6–8, 1996. pp. 183-192.

[Halbert 1984] Daniel C. Halbert. *Programming by Example*. Computer Science Division, EECS Department, University of California, Berkeley, CA. 1984.

[Hamilton 1992] Booz Allen Hamilton. *NeXTStep vs. Other Development Environments; Comparative Study.* Report available from NeXT Computer, Inc.

[Harel 1987] David Harel. “Statecharts: A Visual Formalism for Complex Systems,” *Science of Computer Programming.* **8**(3): pp. 231-274, 1987.

[Helm 1991] Richard Helm, Kim Marriott and Martin Odersky. “Building Visual Language Parsers”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '91, April 27–May 2, 1991. pp. 105-112.

[Henderson Jr 1986] D. Austin Henderson Jr. “The Trillium User Interface Design Environment”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '86, Boston, MA, April, 1986. pp. 221-227.

[Herbsleb 1993] James D. Herbsleb and Eiji Kuwana. “Preserving Knowledge in Design Projects: What Designers Need to Know”. *Proceedings of Human Factors in Computing Systems*, INTERCHI '93, Amsterdam, The Netherlands, April 24-29, 1993. pp. 7-14.

[Hosaka 1977] Mamoru Hosaka and Fumihiko Kimura. “An Interactive Geometrical Design System with Handwriting Input”. *Proceedings of Information Processing*, IFIP '77, Toronto, Canada, North-Holland Publishing Co. Aug. 8–12, 1977. pp. 167-172.

[Hussain 1972] A.B. Shahidul Hussain, Godfried T. Toussaint and Robert W. Donaldson. “Results Obtained Using a Simple Character Recognition Procedure on Munson's Handprinted Data,” *IEEE Transactions on Computers.* **C-21**(2): pp. 201-205, 1972.

[Hutheesing 1996] Nikhil Hutheesing. “The Mother of Development,” *Forbes*. **157**(2): pp. 88-89, Jan. 22, 1996.

[Insight 1995] Insight. *Squiggle*. Insight Development Corp., 2420 Camino Ramon, Suite 205, San Ramon, CA 94583, (510) 244-2000. 1995. Purchased from Premisys Corp.

[Instruments 1989] National Instruments. *LabVIEW.* National Instruments Corp., 6504 Bridge Point Pkwy, Austin, Texas, 78730. 1989.

[IST 1996] IST. *X-Designer.* Imperial Software Technology, 120 Hawthorne Ave., Suite 101, Palo Alto, CA 94301. (415) 688-0200. 1996.

[Jacob 1986] Robert J. K. Jacob. “A Specification Language for Direct-manipulation User Interfaces,” *ACM Transactions on Graphics.* **5**(4): pp. 283-317, October, 1986.

[Karsenty 1992] Solange Karsenty, James A. Landay and Chris Weikart. “Inferring Graphical Constraints with Rockit”. *Proceedings of People and Computers VII*, British Computer Society HCI '92, York, England, September, 1992. pp. 137-153.

[Kolli 1993] Raghu Kolli and Jim Hennessey. “Deriving the Functional Requirements for a Concept Sketching Device: A Case Study”. *Proceedings of Human Computer Interaction, Vienna Conference*, VCHCI '93, Vienna, Austria, Springer-Verlag. September 20–22, 1993. pp. 184-195.

[Kramer 1994] Axel Kramer. “Translucent Patches – Dissolving Windows”. *Proceedings of ACM Symposium on User Interface Software and Technology*, UIST '94, Marina del Rey, CA, November 2–4, 1994. pp. 121-130.

[Kurlander 1993] David Kurlander. *Graphical Editing by Example*. Ph.D. dissertation, Department of Computer Science, Columbia University, New York. 1993.

[Kurtenbach 1991a] Gordon Kurtenbach and Bill Buxton. “GEdit: A Test Bed for Editing by Contiguous Gestures,” *ACM SIGCHI Bulletin.* **23**(2): pp. 22-26, 1991.

[Kurtenbach 1991b] Gordon Kurtenbach and William Buxton. “Issues in Combining Marking and Direct Manipulation Techniques”. *Proceedings of ACM SIGGRAPH and SIGCHI Symposium on User Interface Software and Technology*, UIST '91, Hilton Head, SC, November 11–13, 1991. pp. 137-144.

[Lakin 1986] Fred Lakin. “Spatial Parsing for Visual Languages.” *Visual Languages.* S.-K. Chang, T. Ichikawa and P. A. Ligomenides, Eds. New York, NY: Plenum Press. pp. 35-85. 1986.

[Lakin 1989] Fred Lakin, John Wambaugh, Larry Leifer, Dave Cannon and Cecilia Sivard. “The Electronic Design Notebook: Performing Medium and Processing Medium,” *The Visual Computer.* **5**(4): pp. 214-226, 1989.

[LaLomia 1994] Mary J. LaLomia. “User Acceptance of Handwritten Recognition Accuracy”. *Proceedings of Human Factors in Computing Systems (Conference Companion)*, ACM CHI '94, Boston, MA, April 24–28, 1994. p. 107.

[Landay 1991] James A. Landay. “Tools Review: Serius – A Visual Programming Environment,” *Journal of Visual Languages and Computing.* **2**(3): pp. 297-303, 1991.

[Landay 1996a] James A. Landay. “SILK: Sketching Interfaces Like Krazy”. *Proceedings of Human Factors in Computing Systems (Conference Companion)*, ACM CHI '96, Vancouver, Canada, April 13–18, 1996. pp. 398-399. Formal video program.

[Landay 1993] James A. Landay and Brad A. Myers. “Extending an Existing User Interface Toolkit to Support Gesture Recognition”. *Proceedings of Human Factors in Computing Systems (Adjunct Proceedings)*, ACM INTERCHI '93, Amsterdam, The Netherlands, April 24–29, 1993. pp. 91-92.

[Landay 1995a] James A. Landay and Brad A. Myers. “Interactive Sketching for the Early Stages of User Interface Design”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '95, Denver, CO, May 7–11, 1995. pp. 43-50.

[Landay 1995b] James A. Landay and Brad A. Myers. *Just Draw It! Programming by Sketching Storyboards*. Report #CMU-CS-95-199, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA. 1995.

[Landay 1996b] James A. Landay and Brad A. Myers. “Sketching Storyboards to Illustrate Interface Behavior”. *Proceedings of Human Factors in Computing Systems (Conference Companion)*, ACM CHI '96, Vancouver, Canada, April 13–18, 1996. pp. 193-194.

[Lansdown 1995] John Lansdown and Simon Schofield. “Expressive Rendering: A Review of Nonphotorealistic Techniques,” *IEEE Computer Graphics and Applications.* **15**(3): pp. 29-37, May, 1995.

[Larkin 1987] Jill Larkin and Herbert Simon. “Why a Diagram is (sometimes) Worth 10,000 Words,” *Cognitive Science.* **11.** 1987.

[Lee 1994] Yvonne L. Lee. “PDA Users Can Express Themselves With Graffiti,” *InfoWorld.* **16**(40): p. 30, October 3, 1994.

[Lifeboat 1995] Lifeboat. *Dan Bricklin's Demo*. Lifeboat Publishing, Inc., 1163 Shrewsbury Ave, Shrewsbury, NJ 07702, 800-336-1166, 908-389-9227. 1995.

[Lu 1996] Cary Lu. “Newton 2.0 and Three Contenders Try to Renew the PDA Promise,” *Macworld.* pp. 102-107, July, 1996.

[Macromedia 1994] Macromedia. *Director.* Macromedia Inc., San Francisco, CA. 1994.

[Martin 1990] Gale Martin, James Pittman, Kent Wittenburg, Richard Cohen and Tom Parish. “Sign Here, Please (Interactive Tablets),” *BYTE.* **15**(7): pp. 243-251, July, 1990.

[Meyer 1996] Jonathan Meyer. “EtchaPad – Disposable Sketch Based Interfaces”. *Proceedings of Human Factors in Computing Systems (Conference Companion)*, ACM CHI '96, Vancouver, Canada, April 13–18, 1996. pp. 195-198.

[Microsoft 1993] Microsoft. *Visual Basic*. Microsoft Corp., Redmond, WA. 1993.

[Modugno 1995] Francesmary Modugno. *Extending End-User Programming in a Visual Shell with Programming by Demonstration and Graphical Language Techniques*. Ph.D. dissertation, Report #CMU-CS-95-130, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. March, 1995.

[Modugno 1993] Francesmary Modugno and Brad A. Myers. “Graphical Representation and Feedback in a PBD System.” *Watch What I Do: Programming by Demonstration.* A. Cypher, Ed. Cambridge, MA: MIT Press. pp. 415-422. 1993.

[Moran 1995] Thomas P. Moran, Patrick Chiu, William van Melle and Gordon Kurtenbach. “Implicit Structures for Pen-Based Systems Within a Freeform Interaction Paradigm”. *Proceedings of Human Factors in Computing Systems,* CHI '95, Denver, CO, May 7–11, 1995. pp. 487-494.

[Myers 1988] Brad A. Myers. Creating User Interfaces by Demonstration. Boston: Academic Press. 1988.

[Myers 1990a] Brad A. Myers. “A New Model for Handling Input,” *ACM Transactions on Information Systems.* **8**(3): pp. 289-320, July, 1990.

[Myers 1990b] Brad A. Myers. “Taxonomies of Visual Programming and Program Visualization,” *Journal of Visual Languages and Computing.* **1**(1): pp. 97-123, March, 1990.

[Myers 1991] Brad A. Myers. “Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs”. *Proceedings of ACM SIGGRAPH and SIGCHI Symposium on User Interface Software and Technology,* Hilton Head, SC, November, 1991. pp. 211-220. Proceedings UIST'91.

[Myers 1992] Brad A. Myers. “Demonstrational Interfaces: A Step Beyond Direct Manipulation,” *IEEE Computer.* **25**(8): pp. 61-73, August, 1992.

[Myers 1995] Brad A. Myers. “User Interface Software Tools,” *ACM Transactions on Computer Human Interaction.* **2**(1): pp. 64-103, March, 1995.

[Myers 1986] Brad A. Myers and William Buxton. “Creating Highly Interactive and Graphical User Interfaces by Demonstration,” *Computer Graphics.* ACM SIGGRAPH '86 Conference Proceedings. **20**(4): pp. 249-258, August, 1986.

[Myers 1990c] Brad A. Myers *et. al.* “Garnet: Comprehensive Support for Graphical, Highly-interactive User Interfaces,” *IEEE Computer.* **23**(11): pp. 71-85, November, 1990.

[Myers 1993] Brad A. Myers, Richard G. McDaniel and David S. Kosbie. “Marquise: Creating Complete User Interfaces by Demonstration”. *Proceedings of Human Factors in Computing Systems,* ACM INTERCHI '93, Amsterdam, The Netherlands, April 24–29, 1993. pp. 293-300.

[Myers 1989] Brad A. Myers, Brad Vander Zanden and Roger B. Dannenberg. “Creating Graphical Interactive Application Objects by Demonstration”. *Proceedings of ACM SIGGRAPH Symposium on User Interface Software and Technology,* UIST '89, Williamsburgh, VA, November, 1989. pp. 95-104.

[Negroponte 1973] Nicholas Negroponte. "Recent Advances in Sketch Recognition". *Proceedings of 1973 National Computer Conference and Exposition (42)*, AFIPS '73, New York, AFIPS Press. June 4–8, 1973. pp. 663-675.

[Negroponte 1971] Nicholas Negroponte and James Taggart. "HUNCH – An Experiment in Sketch Recognition." *Computer Graphics*. W. Giloi, Ed. Berlif. 1971.

[NeXT 1991] NeXT. *NeXTStep and the NeXT Interface Builder*. NeXT Inc., 900 Chesapeake Drive, Redwood City, CA 94063. 1991.

[Nielson 1993] Jakob Nielson. Usability Engineering. Boston, MA: Academic Press. 1993.

[Ousterhout 1991] John K. Ousterhout. "An X11 Toolkit Based on the Tcl Language". *Proceedings of Winter 1991 USENIX Conference*, Dallas, TX, Jan 21–25, 1991. pp. 105-115.

[Pavlidis 1985] Theo Pavlidis and Christopher J. Van Wyk. "An Automatic Beautifier for Drawings and Illustrations," *Computer Graphics*. ACM SIGGRAPH '85 Conference Proceedings. **19**(3): pp. 225-234, July, 1985.

[Pearl 1988] Judea Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. San Mateo, CA: Morgan Kaufmann. 1988.

[Pedersen 1993] Elin Ronby Pedersen, Kim McCall, Thomas P. Moran and Frank G. Halasz. "Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings". *Proceedings of Human Factors in Computing Systems*, ACM INTERCHI '93, Amsterdam, The Netherlands, April 24–29, 1993. pp. 391-398.

[Perlin 1985] Ken Perlin. "An Image Synthesizer," *Computer Graphics*. SIGGRAPH '85 Proceedings. **19**(3): pp. 287-296, July, 1985.

[Pictorius 1995] Pictorius. *Prograph*. Pictorius Inc., 2000 Barrington St., 4th Floor, Halifax, Nova Scotia, Canada B3J 3K1, (902) 492-2880. 1995.

[Polya 1973] G. Polya. How to Solve It. Princeton, New Jersey: Princeton University Press. 1973.

[Pugh 1993] David Pugh. *Using Interactive Sketch Interpretation to Design Solid Objects*. Ph.D. dissertation, Report #CMU-CS-93-147, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. April, 1993.

[Ravden 1989] Susannah J. Ravden and Graham I. Johnson. Evaluating Usability of Human-Computer Interfaces: A Practical Method. Chichester, England: Ellis Horwood Limited. 1989.

[Repenning 1993] Alex Repenning. *AgentSheets: A Tool for Building Domain-oriented Dynamic, Visual Environments*. Ph.D. dissertation, Department of Computer Science, University of Colorado at Boulder, Boulder, CO. 1993.

[Repenning 1992] Alex Repenning and Tamara Sumner. "Agentsheets: A Tool for Building Visual Programming Environments". *Proceedings of Human Factors in Computing Systems (Posters and Short Talks),* ACM CHI '92, Monterey, CA, May 3–7, 1992. p. 30.

[Rettig 1994] Marc Rettig. "Prototyping for Tiny Fingers," *Communications of the ACM.* **37**(4): pp. 21-27, April, 1994.

[Rhyne 1987] Jim Rhyne. "Dialogue Management for Gestural Interfaces," *ACM Computer Graphics.* **21**(2): pp. 137-142, 1987.

[Rhyne 1991] Jim Rhyne, Doris Chow and Michael Sacks. "Enhancing the X-Window System: Adding a Paperlike Interface and Handwriting Recognition," *Dr. Dobb's Journal.* **16**(12): pp. 30-38, December, 1991.

[Rhyne 1993] James R. Rhyne and Catherine G. Wolf. "Recognition-Based User Interfaces." *Advances in Human–Computer Interaction.* H. R. Hartson and D. Hix, Eds. Norwood, NJ: Ablex Publishing. **4.** pp. 191-250. 1993.

[Rieman 1996] John Rieman. Personal Communication.

[Rubin 1990] Robert V. Rubin, James Walker II and Eric J. Golin. "Early Experience with the Visual Programmer's WorkBench," *IEEE Transactions on Software Engineering.* **16**(10): pp. 1107-1121, 1990.

[Rubine 1991a] Dean Rubine. Personal Communication.

[Rubine 1991b] Dean Rubine. *The Automatic Recognition of Gestures*. Ph.D. dissertation, Report #CMU-CS-91-202, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. December, 1991.

[Rubine 1991c] Dean Rubine. "Specifying Gestures by Example," *Computer Graphics.* ACM SIGGRAPH '91 Conference Proceedings. **25**(3): pp. 329-337, July, 1991.

[Salisbury 1994] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel and David H. Salesin. "Interactive Pen-and-Ink Illustration," *Computer Graphics.* ACM SIGGRAPH '94 Conference Proceedings. pp. 101-108, July 24–29, 1994.

[Saund 1994] Eric Saund and Thomas P. Moran. "A Perceptually-Supported Sketch Editor". *Proceedings of ACM Symposium on User Interface Software and Technology,* UIST '94, Marina del Rey, CA, November 2–4, 1994. pp. 175-184.

[Schumann 1996] Jutta Schumann, Thomas Strothotte, Andreas Raab and Stefan Laser. "Assessing the Effect of Non-photorealistic Rendered Images in CAD". *Proceedings of Human Factors in Computing Systems*, ACM CHI '96, Vancouver, Canada, April 13–18, 1996. pp. 35-41.

[Seiter 1996] Charles Seiter. "MessagePad 120 with Newton OS 2.0," *Macworld.* **13** p. 57, March, 1996.

[Senior 1992] A.W. Senior. *Off-line Handwriting Recognition: A Review and Experiments*. Report #CUED/F-INFENG/TR 105, Cambridge University Engineering Department, Cambridge, England. 1992.

[Shu 1988] Nan C. Shu. Visual Programming. New York: Van Nostrand Reinhold Company. 1988.

[Singh 1990] Gurminder Singh, Chun Hong Kok and Teng Ye Ngan. "Druid: A System for Demonstrational Rapid User Interface Development". *Proceedings of ACM SIGGRAPH Symposium on User Interface Software and Technology,* UIST '90, Snowbird, UT, October, 1990. pp. 167-177.

[Software 1990] Visual Edge Software. *UIM/X*. Visual Edge Software Ltd., 3950 Cote Vertu, Montreal, Quebec H4R 1V4. Phone (514) 332-6430. 1990.

[Strothotte 1994] Thomas Strothotte, Bernhard Preim, Andreas Raab, Jutta Schumann and David R. Forsey. "How to Render Frames and Influence People". *Proceedings of Eurographics '94,* Oslo, Norway, September, 1994. pp. 455-466.

[Suen 1980] Ching Y. Suen, Marc Berthod and Shunji Mori. "Automatic Recognition of Handprinted Characters – The State of the Art," *Proceedings of the IEEE.* **68**(4): pp. 469-487, 1980.

[Sun 1991] Sun. *DevGuide: OpenWindows Developer's Guide*. Sun Microsystems, Inc., 2550 Garcia Ave., Mtn. View, Ca 94043-1100. 1991.

[Sutherland 1963] Ivan E. Sutherland. "SketchPad: A Man-Machine Graphical Communication System". *Proceedings of AFIPS Spring Joint Computer Conference (23),* 1963. pp. 329-346.

[Tappert 1990] Charles C. Tappert, Ching Y. Suen and Toru Wakahara. "The State of the Art in On-Line Handwriting Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence.* **12**(8): pp. 787-808, 1990.

[Thomas 1981] Frank Thomas. Disney Animation: The Illusion of Life / Frank Thomas and Ollie Johnston. New York: Abbeville Press. 1981.

[Thomson Software Products 1996] Thomson Software Products. *TeleUSE*. Aonix, 595 Market Street, 10th Floor, San Francisco, CA 94105. (415) 543-0900. 1996.

[Venolia 1994] Dan Venolia and Forrest Neiberg. "T-Cube: A Fast, Self-Disclosing Pen-Based Alphabet". *Proceedings of Human Factors in Computing Systems (Conference Companion),* ACM CHI '94, Boston, MA, April 24–28, 1994. p. 218.

[Verstijnen 1996] I.M. Verstijnen, R. Stuyver, J.M. Hennessey, C.C. van Leeuwen and R. Hamel. "Considerations for Electronic Idea-Creation Tools". *Proceedings of Human Factors in Computing Systems (Conference Companion),* ACM CHI '96, Vancouver, Canada, April 13–18, 1996. pp. 197-198.

[Wagner 1990] Annette Wagner. “Prototyping: A Day in the Life of an Interface Designer.” *The Art of Human-Computer Interface Design.* B. Laurel, Ed. Reading, MA: Addison-Wesley. pp. 79-84. 1990.

[Wang 1993] W. Wang and G.G. Grinstein. “A Survey of 3D Solid Reconstruction From 2D Projection Line Drawings,” *Computer Graphics Forum.* **12**(2): pp. 137-158, June, 1993.

[Weiser 1993] Mark Weiser. “Some Computer Science Issues in Ubiquitous Computing,” *Communications of the ACM.* **36**(7): pp. 74-84, 1993.

[Wellner 1989] Pierre D. Wellner. “Statemaster: A UIMS Based on Statecharts for Prototyping and Target Implementation”. *Proceedings of Human Factors in Computing Systems*, ACM CHI '89, Austin, TX, April 30 – May 4, 1989. pp. 177-182.

[Winkenbach 1994] Georges Winkenbach and David H. Salesin. “Computer-Generated Pen-and-Ink Illustration,” *Computer Graphics.* SIGGRAPH '94 Conference Proceedings. pp. 91-100, July 24–29, 1994.

[Wolber 1991] David Wolber and Gene Fisher. “A Demonstrational Technique for Developing Interfaces with Dynamically Created Objects”. *Proceedings of ACM SIGGRAPH Symposium on User Interface Software and Technology*, UIST '91, Hilton Head, SC, November 11–13, 1991. pp. 221-230.

[Wolf 1988] Catherine G. Wolf. “A Comparative Study of Gestural and Keyboard Interfaces.” *Proceedings of the Human Factors Society 32nd Annual Meeting.* pp. 273-277. 1988.

[Wolf 1989] Catherine G. Wolf, James R. Rhyne and Hamed A. Ellozy. “The Paper-Like Interface.” *Proceedings of the Third International Conference on Human-Computer Interaction.* pp. 494-501. 1989.

[Wong 1992] Yin Yin Wong. “Rough and Ready Prototypes: Lessons From Graphic Design”. *Proceedings of Human Factors in Computing Systems (Posters and Short Talks)*, ACM CHI '92, Monterey, CA, May 3–7, 1992. pp. 83-84.

[Wong 1993] Yin Yin Wong. “Layer Tool: Support for Progressive Design”. *Proceedings of Human Factors in Computing Systems (Adjunct Proceedings)*, ACM INTERCHI '93, Amsterdam, The Netherlands, April, 1993. pp. 127-128.

[Zeleznik 1996] Robert C. Zeleznik, Kenneth P. Herndon and John F. Hughes. “SKETCH: An Interface for Sketching 3D Scences,” *Computer Graphics.* SIGGRAPH '96 Conference Proceedings. pp. 163-170, August 4–6, 1996.

[Zhao 1992] Rui Zhao. “On-line Geometry Recognition Using C++, an Object-Oriented Approach”. *Proceedings of Tools Europe '92*, Dortmund, Germany, Prentice Hall. March 30, 1992. pp. 371-378.

[Zhao 1993a] Rui Zhao. “Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors”. *Proceedings of Human Factors in Computing Systems*, ACM INTERCHI '93, Amsterdam, The Netherlands, April 24–29, 1993. pp. 95-100.

[Zhao 1993b] Von Rui Zhao. *Handsketch-based Diagram Editing*. Ph.D. dissertation, University of Paderborn, Paderborn, Germany. 1993.