Minimizing Modes for Smart Selection in Sketching/Drawing Interfaces

Eric Saund and Edward Lank

1 Introduction

User interface modes are ubiquitous in both mouse-keyboard and pen-based user interfaces for creating graphical material through sketching and drawing. Whether choosing the straight-line or oval tool in Photoshop or PowerPoint, or tapping a toolbar prior to lassoing a word in order to select it in OneNote, users know that, before they can perform the content-relevant action they want, they need to tell the computer the intent of what they are about to do by setting a mode. This chapter reviews our research exploring whether prior setting of modes is always necessary, and whether the future of user interface designs may promise more fluid and direct ways of creating and then selecting and editing words and pictures on a screen.

The purpose of modes is to allow actions performed with a single input device to mean more than one thing. Physical paper permits two fundamental operations, creation of marks, and erasure. For these, the user employs two basic tools, each physically suited to its purpose: a marking tool (pencil, pen, typewriter keys and ribbon) and an erasure tool (eraser, white-out). Computers are more powerful than this. They permit not only creation and deletion, but all manner of modification such as moving, resizing, duplicating, changing colors, changing line quality, changing fonts, controlling depth order, etc.

To effect modification of content, computer authoring and editing tools provide two dominant modes: a creation mode, and a selection mode. Modification of content is performed by first entering selection mode, then selecting graphical content on the screen, and finally performing operations manipulating the selected content.

Is it possible to design user interfaces that improve the fluidity and precision of the Selection step? Our research indicates that the answer can be yes. The key

Eric Saund
Palo Alto Research Center, e-mail: saund@parc.com

Edward Lank
University of Waterloo, e-mail: lank@cs.uwaterloo.ca

is to more fully exploit available information about user actions in the context of canvas content, to infer the user's intent. If the user's click, tap, or stroke gesture makes sense only in terms of one particular mode, then the program should allow the user to perform that operation without first explicitly setting the mode, and then post-facto interpret the action in terms of the correct mode.

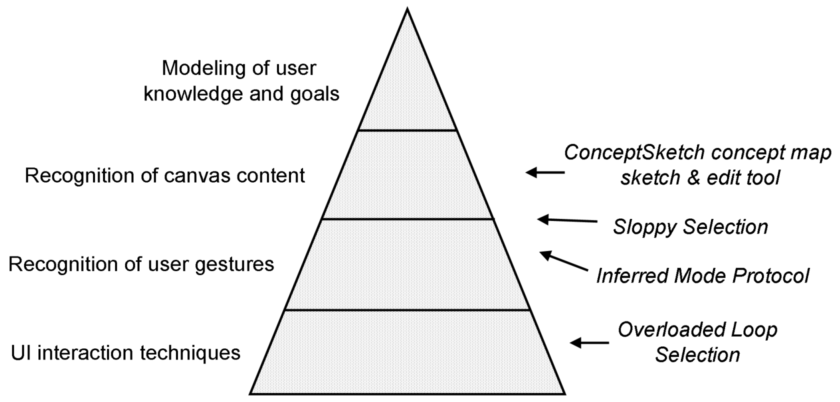

Modeling of user knowledge and goals
Recognition of canvas content
Recognition of user gestures
UI interaction techniques
ConceptSketch concept map sketch & edit tool
Sloppy Selection
Inferred Mode Protocol
Overloaded Loop Selection

Fig. 1 Increasingly sophisticated methods for inferring user intent build on one another. This chapter offers examples highlighting techniques at several levels.

This may require sophisticated analysis of user's gestures, the visual and semantic structure of canvas content, and even user desires and goals, as shown in Figure 1. Such a project entails risk, for if the program guesses wrong the user interaction can go seriously awry. But when done carefully, the principle can be extended to not only inferring mode but other aspects of user intent, to create new levels of intelligent user interfaces.

This chapter focuses on mode minimization in interfaces via smarter selection techniques. We address three challenges associated with selection:

- How best to incorporate multiple selection techniques into a sketch interface.
- The drawback of requiring mode switching between content creation and selection.
- The challenge of selecting and interacting with salient groups of content.

We address these challenges through the creation of novel interaction techniques contained within a series of experimental graphical creation and editing programs that we have built. The ScanScribe document image editing program eliminates the mode tool palette in a mouse/keyboard image editor by overloading mouse functions for multiple selection methods. The InkScribe draw/edit program for pen computers eliminates prior Draw/Select mode selection through an Inferred Mode interface protocol. A technique we call Sloppy Selection illustrates intelligent object selection by analysis of gesture dynamics coupled with visual segmentation of canvas content. And the ConceptSketch program for creation and editing of Node-Link diagrams shows how recognition of diagrammatic structure supports intelligent object selection by cycling Click/Tap operations. These points in the gesture/canvas-content analysis pyramid are discussed in the subsequent sections of this chapter.

2 The Cost of Modes

To motivate minimizing modes in interfaces, it is useful to examine the cost of having a large set of modes. The salient research question is whether some benefit, either in efficiency or accuracy, exists for reducing the set of modes in an interface. To examine this question, we describe our recent work in the cost of mode switching. We first examine the temporal cost of large mode sets, and then explore the effect a large set of modes has on mode switching errors within sketch interfaces.

2.1 The Temporal Cost of Modes

Many researchers have studied variations in interaction techniques for stylus input systems that seek to fluidly allow both command and input [2, 6, 9, 18]. This research can be broadly separated into research that seeks to improve the accessibility of software modes versus research that seeks alternatives to modes. While our work primarily falls into the latter category, i.e. in reducing the need for modes within interfaces, improving the accessibility of modes in interfaces is an alternative for improving the fluidity of sketch or graphical applications that contain multiple modes.

One open question is whether or not there exists an "optimal" mode switching technique, and if so, what the performance of that mode switching technique might be. To partially address this question, Li et al. [9] studied five different existing mode switching techniques. These include typical mode switching techniques that have been extensively used, i.e. use of the eraser end of a dual ended stylus, use of the barrel button on an electronic stylus, a press and hold technique similar to the Apple Newton, and use of the non-preferred hand. They also examined a pressure based technique based on work by Ramos et al. on pressure widgets [13]. In this list of mode switching techniques, we note the absence of software widgets to control modes, a result of general recognition of the fact that improvements are needed over software-based modes [10]. Based on experimental data, Li et al. concluded that, of the five techniques, non-preferred hand performed best based upon the metrics of speed (fastest), error rate (second lowest), and user preference (most preferred).

Given the apparent benefit of non-preferred hand mode switching, we explored in detail the specific temporal costs associated with non-preferred hand mode switching [7, 15, 16]. In this work, we looked at the time taken to initiate modes with the non-preferred hand, and the total time taken to perform a simple drawing task, given the need to switch modes. We found that, as the number of modes increased, the total time taken to perform the drawing task increased, and that this increase was a result of an increase in the time required to initiate modes with the non-preferred hand. We discovered [16], using an interface with between two and eight modes, that the cost of manipulating modes in an interface could be modeled using the Hick-Hyman Law [4, 5]. This law predicts a linear relationship between response time and the information entropy, H, associated with n different responses, i.e.

$$RT = a + bH \tag{1}$$

where the information entropy, as defined by Shannon, is:

$$H = \sum_{i=1}^{n} p_i \log_2 \left(\frac{1}{p_i} \right) \tag{2}$$

where n is the number of alternatives (in our study, the number of modes) and p_i is the probability of the ith alternative.

Figure 2, reproduced from [16], depicts the linear relationship between information entropy, H, and time to select a mode, as described in the previous paragraph. To generate this data, we performed an experiment where we presented subjects with a simple line bisecting task, and asked the subjects to draw a line of a specific color, indicated by a mode. We measured the time taken to activate the mode with the non-preferred hand, the time between mode activation and the pen tip touching the surface of the display, and the time taken to perform the drawing task on a tablet computer. Analysis of variance shows that there is a significant main effect of the number of modes on total time ($F_{3,5} = 12.593, p < .001$) for the task. Analysis of variance for the time to activate modes, i.e. the time to press the appropriate button with the non-preferred hand, shows a significant effect of condition ($F_{3,5} = 22.826, p < .001$). However, the time interval between mode switch and pen down and the time to perform the drawing task did not vary significantly with number of modes ($F_{3,5} = 1.460, p = 0.269$ and $F_{3,5} = 2.360, p = 0.101$ respectively).

This work on the cost of mode switching provides evidence that, regardless of the efficiency of any mode switching technique, as you add modes to an interface the cost, measured as the time, to select any individual mode within the interface increases. By reducing the number of modes within an interface, we increase the efficiency of the interface.

2.2 Mode Errors: The Mode Problem

In addition to temporal efficiency, the accuracy with which users can manipulate an interface is an important consideration. It seems logical that larger numbers of modes in interfaces increases the likelihood of mode errors. The web site Usability First [20] defines as mode error as:

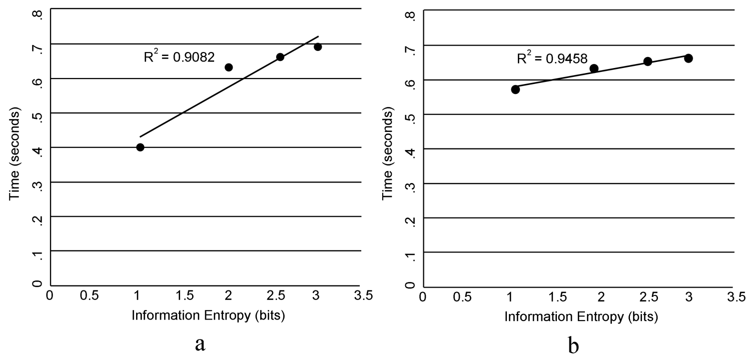


Fig. 2 By studying interfaces with 2, 4, 6, and 8 modes, we show a linear relationship between information entropy, H, and the time taken to select a mode. a. the case where all modes are equally probable. b. varies the probabilities for different modes in the interface.

"A type of slip where a user performs an action appropriate to one situation in another situation, common in software with multiple modes. Examples include drawing software, where a user tries to use one drawing tool as if it were another (e.g. brushing with the Fill tool), or text editors with both a command mode and an insert mode where a user accidentally types commands and ends up inserting text."

Two of the most common mode errors include use of the CAPS-lock and Insert keys on keyboards, both of which alter the effect of keyboard input.

Systems normally mitigate against mode errors by providing some indicator for modes. However, Sellen et al. [19] studied the use of visual feedback and kinesthetic feedback to indicate modes. Visual feedback was provided by changing the shape of the cursor, and kinesthetic feedback by use of a footpedal. In their first study, they used a non-locking piano footpedal, and users were forced to maintain modes. In this experiment, they found that kinesthetic feedback was more effective at preventing mode errors than was visual feedback. They followed this study with a second study that contrasted a locking and non-locking footpedal, and found fewer errors with the non-locking footpedal. Based on Sellen's work, Jef Raskin, in his book *The Humane Interface*, advocates a mode switching technique he terms "quasimodes". With quasimodes, as with Sellen et al.'s nonlocking footpedal, a user holds down a key to indicate modes.

The non-preferred hand mode switching technique used by Li et al. [9] and by us in our work on the temporal cost of modes [16] is a quasimode, based on Raskin's definition. In Li et al.'s work in two-mode interfaces, non-preferred hand mode switching resulted in an error rate of approximately 1.1%, slightly worse than the using the eraser end of the electronic stylus. One question unanswered by Li et al. is whether a relationship exists between the number of modes and the frequency of mode errors. It seems likely that increasing the number of modes increases the frequency of mode errors: Users are forced to choose one from a larger number of alternatives, giving rise to a higher probability of selecting the incorrect mode from among the set of available modes. However, whether the increase in mode errors as number of modes increases is a logarithmic, linear, or other function of number of modes provides an understanding of the expected cost, in accuracy, of adding additional modes to an interface, and the corresponding benefit associated with reducing the mode set within an interface. To address this question, we examined mode errors as a function of number of modes in a sketch interface in our work modeling the cost of mode switching [16]. We observed error rates of between 3.3% in the two-mode condition and 7.5% in the eight mode condition [16]. Figure 3 depicts the mode error rate against number of modes in the interface. In this graph, we see a linear correlation ($R^2 = 0.94$) between number of modes and frequency of mode errors in our experimental task.

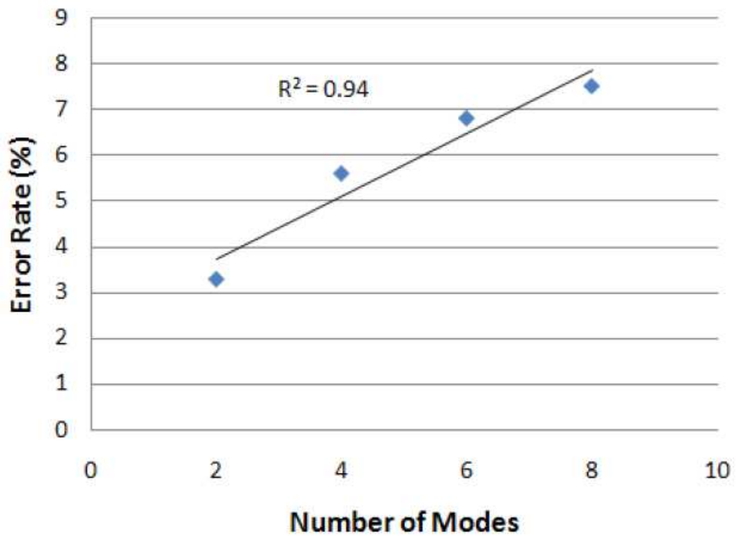


Fig. 3 Error rate as a function of number of modes in an interface.

Given our results on the relative efficiency and accuracy of interfaces as a function of the number of modes within the interface, we claim that reducing the number of modes is a worthwhile goal. In the following sections, we examine user interface techniques and recognition techniques that we have developed to accomplish this.

3 Overloaded Loop Selection: UI Design to Infer Selection Mode

Many graphical editing programs support multiple means for selecting image material through the use of tool palettes. For example, Photoshop offers both a rectangle selection tool and a lasso tool, among others. Selection of one of these tools puts the interface into a distinct Selection mode. The rectangle is faster for selecting isolated objects, but the lasso is capable of “threading the needle” and selecting objects among clutter, and generally of creating oddly shaped selection regions.

We propose that the most straightforward means for amplifying the selection options available to users without requiring attention to a tool palette is to mix them together in a single Select Mode, and infer the user’s intent from the gesture they actually produce. We invoke this idea in a technique called Overloaded Loop Selection. The user is free to drag a selection gesture that may take form as either a rectangle or a lasso. Both are displayed simultaneously. If the user proceeds to draw a nearly-closed loop, the rectangle disappears and the lasso region is chosen. But if the user releases the mouse while the rectangle is displayed, the rectangle selection region is used. See Figure 4.

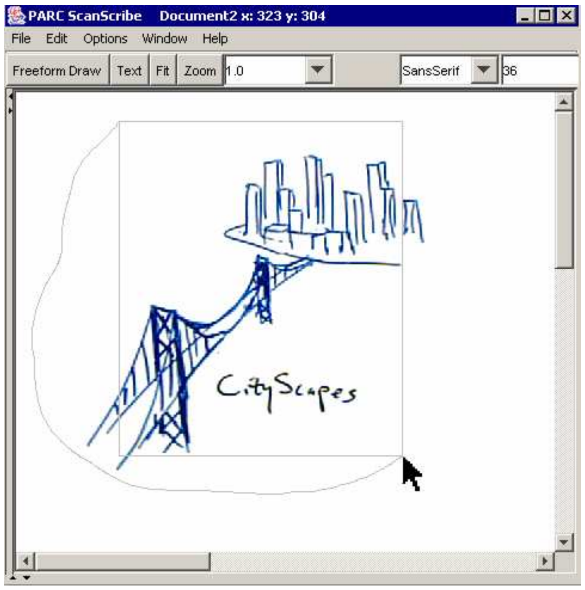

PARC ScanScribe Document2 x: 323 y: 304
File Edit Options Window Help
Freeform Draw
Text
Fit
Zoom
1.0
SansSerif
36
CityScapes

Fig. 4 Overloaded loop selection initiated by dragging the mouse with the left button held. Both a selection rectangle and lasso path are active. Closing the path causes the rectangle to disappear, leaving lasso selection. If the button is released while the rectangle is visible, rectangle selection is used instead.

Overloaded Loop Selection is employed by the ScanScribe document image editor program first introduced at UIST 2003 [17]. ScanScribe takes this idea two steps further. First, in addition to overloading rectangle and lasso selection, Selection Mode supports Cycle Click Selection, which extends the capability to select by clicking the mouse on an object. This is described in Section 6. Second, ScanScribe supports Polygon selection, by which users are able to select image material by placing and adjusting the vertices of an enclosing polygon. Polygon selection is invoked as a mode, but conveniently so by double-clicking the mouse over a background region, without the need for a separate toolbar.

Overloaded Loop Selection is an example of UI design minimizing prior selection of modes through analysis of the user action alone, without regard to the underlying canvas content. Other examples exist as interface techniques that analyze user action to determine effect in sketch interfaces. Hinckley et al. [6] proposed using a post-gesture delimiter technique, called a “pig-tail”, for determining gesture interpretation, and they compared the post-gesture delimiter to using a handle, a timeout, or a button to alter a gesture’s “mode”. Grossman et al. [3] proposed “hover widgets”, where the tracking state of a Tablet PC is used to access modes, and they compared it to using a software button to switch interface modes. Finally, Ramos and Balakrishnan [12] describe a “pressure mark” technique, where the different pressure associated with a mark maps to different interpretations. However, in each of these cases, the need exists to select from among the possible alternative interpretations, either during or after the action. As noted in Section 2, there is a cost associated with selecting amongst alternatives. By minimizing modes within an interface, we reduce the cost of selecting any mode within the interface.

While the UI design of the ScanScribe document image editor is modeled after and builds on PowerPoint, ScanScribe is designed primarily to be an editing tool for mouse/keyboard platforms and does not offer many options for entering new material. Freeform entry of sketch strokes is possible, but only by explicitly entering a separate Freeform Draw mode. The pen/stylus platform, on the other hand, demands more seamless interplay of drawing/sketching entry and select/command manipulation of canvas content.

4 The Inferred Mode Protocol for Stylus Drawing and Selection with a Pen

The prototypical application for pen/stylus computing platforms is the Electronic Whiteboard, which generally supports freeform drawing and handwriting, then selection of digital ink for cut, copy, move, resize, color change, etc. Unconstrained electronic notetaking applications fall within this definition. One of the first electronic whiteboard programs to gain significant contemplation was the Tivoli [11] program for the Xerox Liveboard.

The fundamental problem with pen electronic whiteboard programs is how to support drawing, selection, and commands on selected material through a single pen/stylus channel. The designers of Tivoli experimented with pen barrel buttons, tap-tap gestures, and post-lasso pigtail gestures, among other things, but eventually settled on explicit setting of Draw/Select mode through a side toolbar. Later, the Microsoft Journal program for the TabletPC settled on prior setting of Select mode through either tapping on a toolbar icon or else stationary holding of the pen for a predetermined length of time. All of these methods for mode setting fail to deliver seamless fluid user action. Barrel buttons are awkward to use. Toolbars require redirection of user focus away from the canvas. And stationary hover requires waiting for the hover threshold timeout and also leads to inadvertent entry of Select mode when the user may intending to draw but momentarily simply pausing to think with the pen down. The problem, we believe, is not *how* the user is supposed to set Draw versus Select mode, but that they have to do it at all.

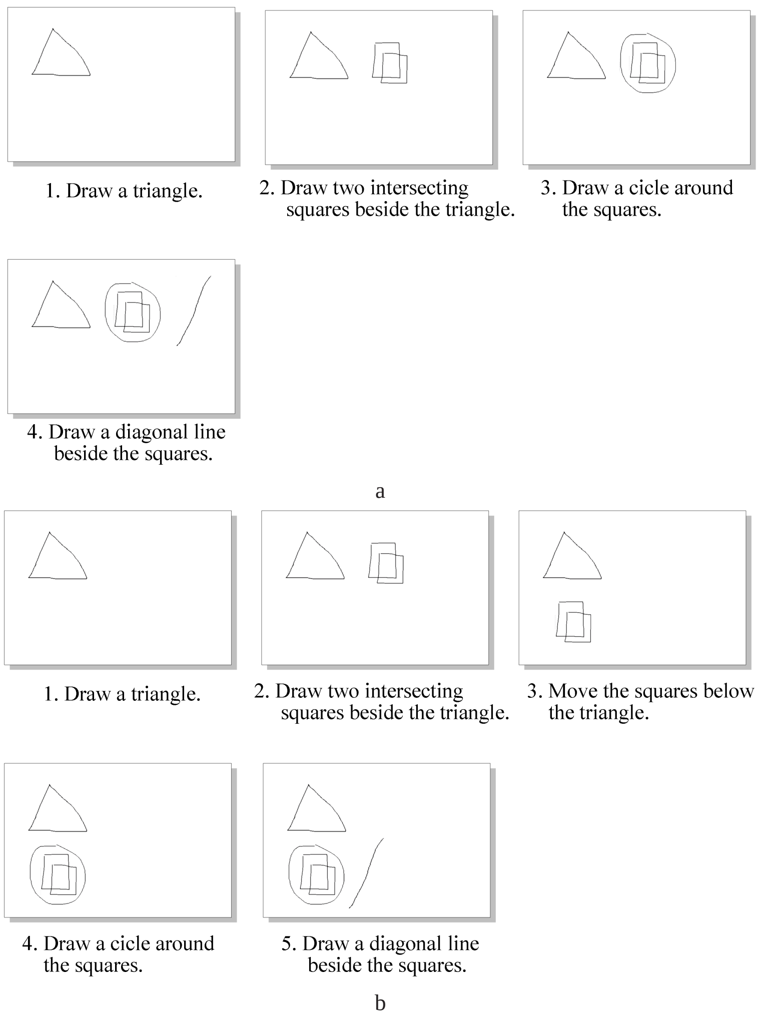

1. Draw a triangle.
2. Draw two intersecting squares beside the triangle.
3. Draw a cicle around the squares.
4. Draw a diagonal line beside the squares.
a
1. Draw a triangle.
2. Draw two intersecting squares beside the triangle.
3. Move the squares below the triangle.
4. Draw a cicle around the squares.
5. Draw a diagonal line beside the squares.
b

Fig. 5 Two simple tasks for a pen drawing/editing platform. a. Task I involves only draw a series of shapes. b. Task II involves drawing, then selecting and moving some of the drawn objects (as if the user changed their mind about where to place them), then subsequent additional drawing

4.1 The Mode Problem in Electronic Whiteboard Programs

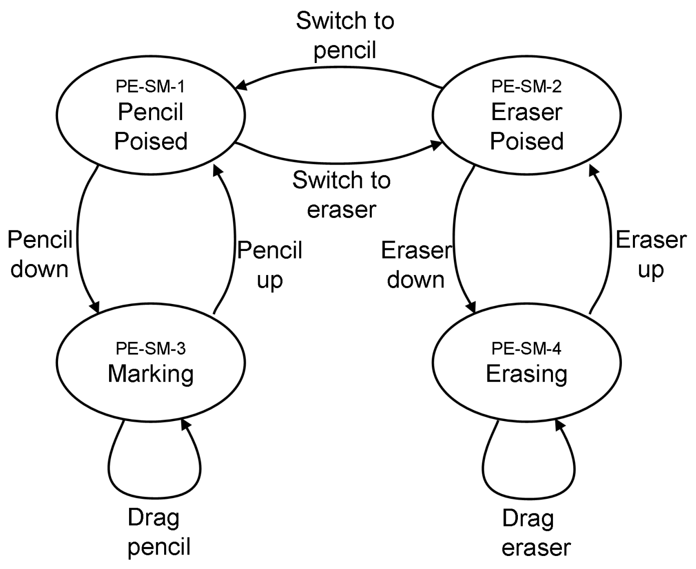
We illustrate the mode problem through two simple tasks which could be part of a larger document creation/editing session. The purpose of these tasks is not to achieve the final result as efficiently as possible, but rather to simulate the process a user might go through, including changing their mind in midstream and rearranging material they have already placed on the canvas. In Task I (Figure 5a) the user draws a triangle, some overlapping squares, and a diagonal line. In Task II (Figure 5b), they draw these same objects, but midway through, they decide to change the location of the overlapping squares. To do this, they would need to use the drawing tool's edit capabilities to select the squares and drag them to their new desired position on the canvas. This is where trouble lies. Under a conventional mode-based interface design, the user would enter a selection mode and draw a lasso around the squares to select them. Then, they would have to exit selection mode to return to drawing. If, in the creative moment, these extra UI steps are not completed correctly, the task is thrown off track.

4.2 Analytical Tool: The Interaction Flow Diagram

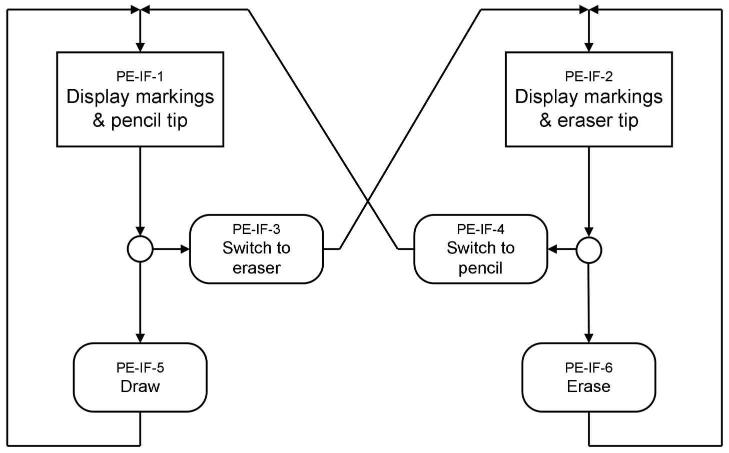
In order to gain insight into how and why the requirement for mode setting can become a serious problem for pen-based drawing and editing systems, we introduce an analytical tool for graphically tracing the steps of interaction between user actions and program interfaces. The *Interaction Flow Diagram* is a form of state diagram, but one that emphasizes the modal state of the program and the operations available to users within each mode. In a conventional user interface state machine diagram, nodes denote internal states of the program and arcs denote possible transitions between them. In the Interaction Flow diagram, nodes are differentiated into three primary types: (1) those that depict internal machine state and information available to the user through the machine's display (rectangles); (2) those that indicate intentional user actions (rounded rectangles); (3) those that indicate a choice or decision point for the user (circles). The Interaction Flow diagram is particularly useful in dissecting user interaction bugs and aspects of user interface design that enable them.

The difference is illustrated in Figure 6, which presents the state machine diagram and Interaction Flow Diagram representing the simple interaction afforded by paper, pencil, and eraser (or equivalently, whiteboard, marker, and eraser). There is no computer program here, the only action object in this diagram is the user's writing/drawing activities, which include two functions, creating marks, and erasing them.

The State Machine diagram represents the use of pencil, eraser and paper as transition among four states: Pencil Poised, Marking, Eraser Poised, and Erasing. The transition arcs reflect the logic of the system, for example the fact that before one can create a mark, one must first hold the pencil, then place its tip to the paper.



a



b

Fig. 6 a. State Machine and b. Interaction Flow Diagram for pencil and eraser. Rectangles represent a quiescent state of the interface. Rounded rectangles represent user actions. Circles represent user choices among available actions, given the presentation state.

The Interaction Flow diagram portrays the interaction in a manner more closely resembling the user's experience. State display nodes, represented by rectangular boxes, indicate information visually (or through other senses) available to the user. In particular, in the quiescent state between actions, the user can see the markings on the page, and they can sense whether the pencil or eraser is poised above the page. Circles indicate deliberative choices, such as between either making a mark or switching to the eraser. The Interaction Flow diagram thus re-configures selected arcs exiting from nodes in the formal State Machine diagram to make explicit certain decisions the user can make at the level of significant functional operations of the tool. Finally, actual user actions are shown as rounded boxes. Often Interaction Flow diagrams package up tedious details of the State Machine diagram. For example, the state transition subgraph of touching, dragging, and lifting the pencil are wrapped into the functional action (rounded box) labeled "draw".

The Interaction Flow diagram in Figure 6b reflects the simplicity of the interaction model for pencil and paper. The current draw/erase mode is always indicated by visual and/or tactile display. The choice to switch modes is always available. To execute a mode switch the user carries out the physical act that brings the desired tool end into position for use. Once in Draw or Erase mode, the system stays in that mode by default. The acts of continuously writing or continuously erasing are tight loops through states in Figure 6b. When writing fluidly the user may effectively ignore the choice to switch into erase mode. And significantly, for the purpose of managing their interaction with the pencil, the user has no requirement to attend to the information display (i.e. the markings on the surface and the pencil tip in view). Rather, they are free to write or draw "open loop," paying attention to the content of their writing instead of the user interface features of the tool.

With the greater functionality of computer programs for creating and editing graphical material comes greater complexity of the user interface. Perhaps the most successful of these is PowerPoint. A simplified User Interaction Flow Diagram for the PowerPoint-style interface is shown in Figure 7.

The fundamental operations here are creation of new text or graphic objects, selection of objects, and modifying selected objects. These are reflected in three state display nodes (rectangular boxes), in Figure 7. When nothing is selected (Node PPT-IF-2), the interface is in Select mode. From here the user has the option of performing a selection operation (PPT-IF-8) or else entering Create/Entry mode by choosing an object type to create by clicking a menu or toolbar icon (PPT-IF-5). Either of these choices results in an internal change of machine state, and also in an augmentation of the display, such as highlighting of selected material (PPT-IF-3), or change from an arrow to crosshair cursor (PPT-IF-1). Once something is selected (PPT-IF-3), the interface enters Command Mode, in which selected material is highlighted. From here the user has a choice to deselect it, modify it, select additional objects, or switch to a create mode.

The default Mode of PowerPoint is Select Mode. PowerPoint permits users to select graphical material by either of two means, by tapping on an object, or by dragging a rectangle which results in selection of all objects entirely enclosed.

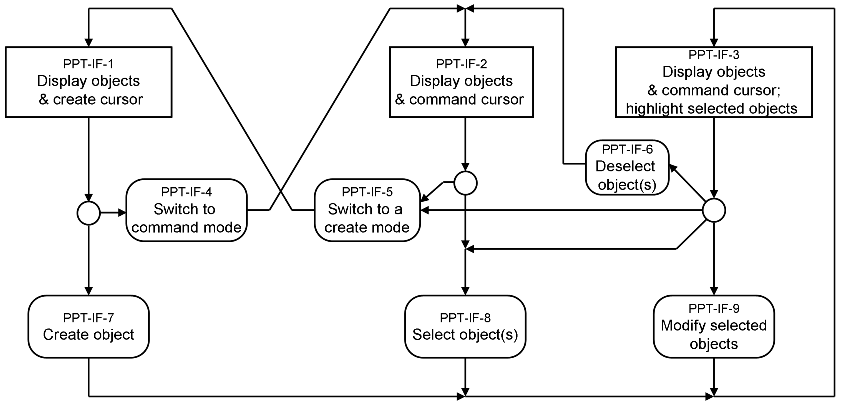


Fig. 7 Simplified Interaction Flow Diagram for the PowerPoint structured graphics editor.

4.3 Interaction Flow Analysis of Mode-Based Selection and Drawing

The Interaction Flow Diagram provides insight into exactly what can go wrong with prior selection of mode in an electronic whiteboard program. Let us consider in detail Task I and Task II of Figure 5 in terms of the Interaction Flow for a conventional mode-based Electronic Whiteboard program, such as Tivoli or Microsoft Journal. See Figure 8. This protocol bears strong resemblance to the mouse-based interaction protocol design of PowerPoint and other structured graphics editors. The main difference is that Create/Entry Mode (also known as Draw Mode for a pen/stylus program) and Command Mode are persistent. When in Draw Mode (the leftmost Display/User Action column of the diagram) the act of making repeated marks with the stylus is fluid and unconstrained, just as with a physical pen or pencil. From Draw Mode, the user may switch to Select Mode by an explicit action such as tapping a toolbar item or releasing the stylus barrel button. In Select Mode the user may select objects by tapping or lasso.

Although Draw and Select modes are independent nodes in the Interaction Protocol (CS-IF-1 and CS-IF-2), an Electronic Whiteboard program may or may not actually provide a visible indicator of the current mode. In tablets and electronic whiteboards whose hardware provides pen hover detection, alternative Draw and Select cursors can do this. In purely touch-based stylus systems any visual mode indicator must be placed peripherally if at all. In either case, users are famous for ignoring mode indications rendered via cursor shape.

The mode problem arises when users perform as if the system were in one mode when in fact it is in another. Our sample draw/edit tasks illustrate where the interaction protocol can lead users to make errors. Task I is not a problem. This involves simply adding strokes one after another, in draw mode, as shown in Figure 9.

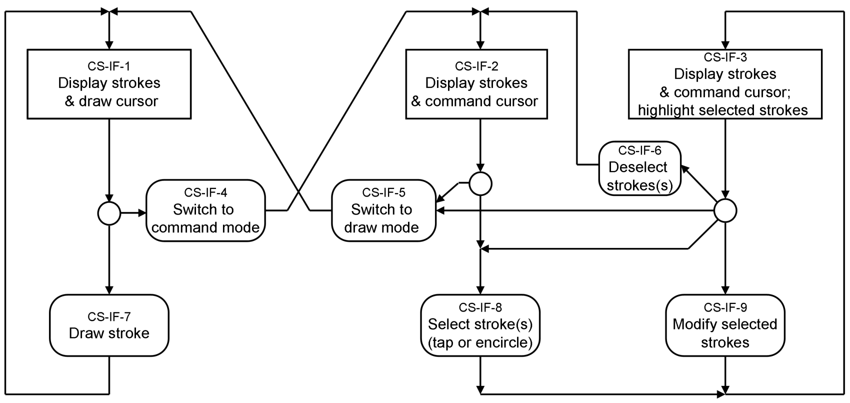


Fig. 8 Representative Interaction Flow Diagram for an Electronic Whiteboard program for Pen/Stylus platforms.

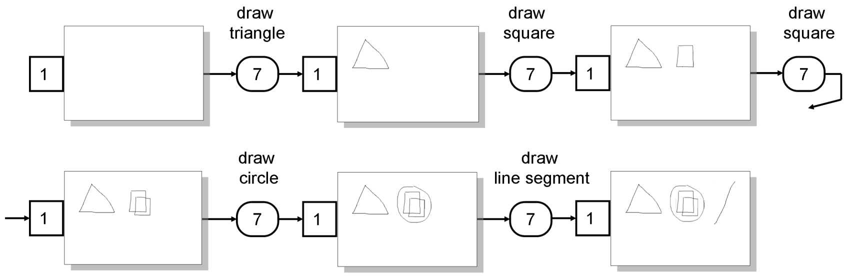


Fig. 9 Steps of the interaction flow for Task I under the Interaction Flow protocol of a conventional Electronic Whiteboard program. Numbers indicate nodes of the Interaction Flow Diagram of Figure 8.

The interaction flow for *correct* performance of Task II is shown in Figure 10. Note that in order to move the pair of squares the user must first switch to Select mode, then draw a lasso around the squares in order to select them, then drag the selected objects to another position, and finally switch back to Draw mode.

The common interaction bug in this protocol is failure to switch modes before executing the next pen gesture or stroke. Figure 11 shows the interaction flow that results from failing to enter Select mode, CS-IF-4. The user behaves as if they are proceeding from Node CS-IF-2, performing what is intended to be a selection gesture. But the program interprets this as a drawn stroke, and renders it as such. Seeing a drawn circle instead of a visual indication of strokes selected, the user is alerted to the problem. They must then execute a repair protocol of at least three additional actions, plus they must devote attention to the display to verify that they are back on track, before proceeding with the intended task.

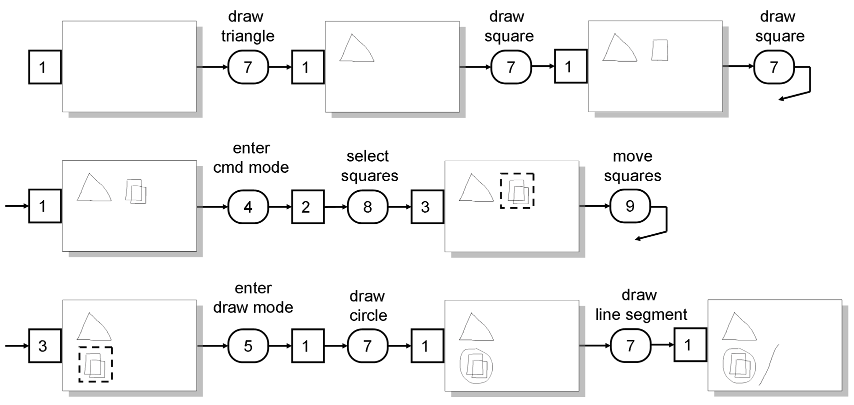


Fig. 10 Steps of the interaction flow for correct performance of Task II under the Interaction Flow protocol of a conventional Electronic Whiteboard program. Numbers indicate nodes of the Interaction Flow Diagram of Figure 8.

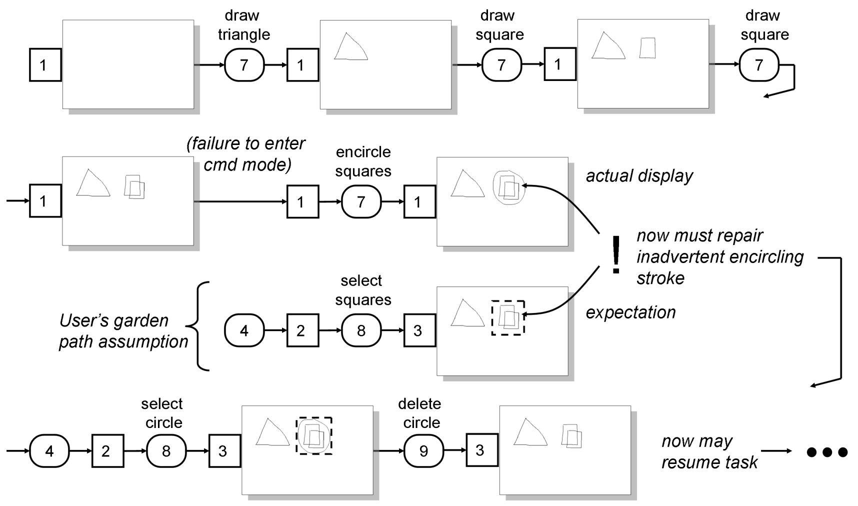


Fig. 11 Steps of the interaction flow for disrupted performance of Task II due to a common mode error, under the Interaction Flow protocol of a conventional Electronic Whiteboard program. Numbers indicate nodes of the Interaction Flow Diagram of Figure 8.

In a similar fashion, by failing to return to Draw mode after performing an edit operation, users are alerted to the problem and must interrupt their flow of interaction in order to recover and re-synchronize their mental model of the interaction with the machine state of the program.

4.4 Inferred Mode Protocol: Inferring Draw/Select Mode

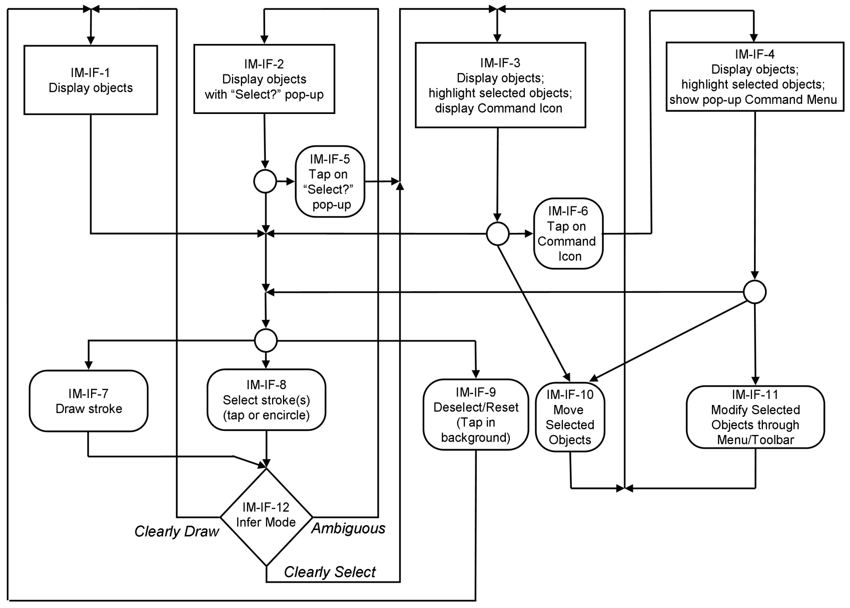
To address the Draw/Select Mode problem for pen/stylus interfaces, we introduced a technique called the Inferred Mode protocol [18], used in the InkScribe pen-based sketch tool. This protocol allows the user to perform either a draw/entry or lasso selection gesture without a priori specification of mode. The intent of the stroke is inferred from the stroke's shape and its relation to existing canvas content. If the stroke is not closed, or if it is closed but does not enclose any existing material, then it cannot be a lasso selection gesture so is interpreted as new ink. If however it is approximately closed and does enclose markings on the canvas (which can be any combination of digital ink and bitmap image), the gesture is ambiguous. In this case, the interface presents a pop-up menu labeled, "Select?", in a nearby but out-of-the-way location. The user may then elect either to tap the menu item to select the enclosed material, or else simply ignore it and keep writing or drawing.

The Inferred Mode Protocol also supports Cycle Tap Select, described in Section 6. In doing so, the protocol prohibits the user from drawing dots, or short tap strokes, on top of or very near to existing markings.

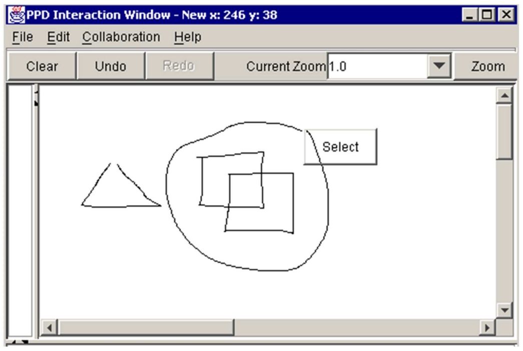
The Interaction Flow Diagram for the Inferred Mode Protocol is shown in Figure 12. Note that there are no user action nodes by which the user explicitly switches to a Draw or Command mode. Instead, the logic of mode switching is embedded in the inference of user intent based on the user's actions in context.

At quiescence the user can be faced with one of four visually distinguished situations: nothing is selected (IM-IF-1); nothing is selected but the pop-up menu item saying "Select?" is displayed (IM-IF-2); one or more strokes are selected (IM-IF-3); one or more strokes are selected and a command menu is visible (IM-IF-4). From these four possibilities the flow of control converges onto one unified set of choices that are always available regardless of the selection state. Namely, the user can at any time draw more material (IM-IF-7), they can at any time perform a selection gesture (IM-IF-8), and they can at any time reset the selection status to nothing selected by tapping in the background (IM-IF-9). The final options, to perform a gesture to move or modify selected material (IM-IF-10 and IM-IF-11), are operative only when something is actually selected.

The Inferred-Mode protocol introduces a new type of node to the Interaction Flow notation. This is the Intent Inference node, shown as a diamond (IM-IF-12), which represents a decision process that the system performs on the input gesture drawn at user action nodes IM-IF-7 or IM-IF-8. Note that IM-IF-7 or IM-IF-8 reflect only user intent, not any overtly distinguishable action or state. The purpose of this decision is to determine whether an input pen trajectory is clearly a drawn stroke,


IM-IF-1
Display objects
IM-IF-2
Display objects with "Select?" pop-up
IM-IF-3
Display objects; highlight selected objects; display Command Icon
IM-IF-4
Display objects; highlight selected objects; show pop-up Command Menu
IM-IF-5
Tap on "Select?" pop-up
IM-IF-6
Tap on Command Icon
IM-IF-7
Draw stroke
IM-IF-8
Select stroke(s) (tap or encircle)
IM-IF-9
Deselect/Reset (Tap in background)
IM-IF-10
Move Selected Objects
IM-IF-11
Modify Selected Objects through Menu/Toolbar
IM-IF-12
Infer Mode
Clearly Draw
Ambiguous
Clearly Select

a

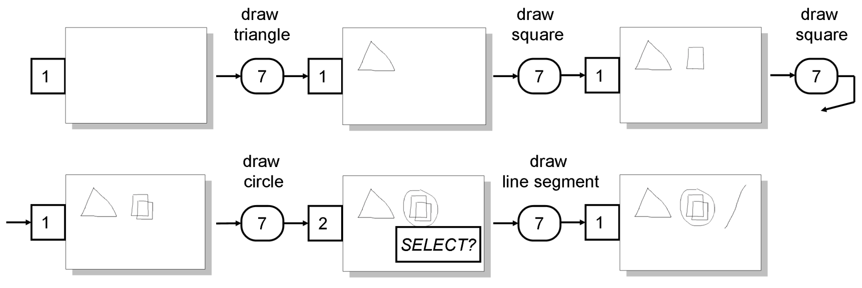


b

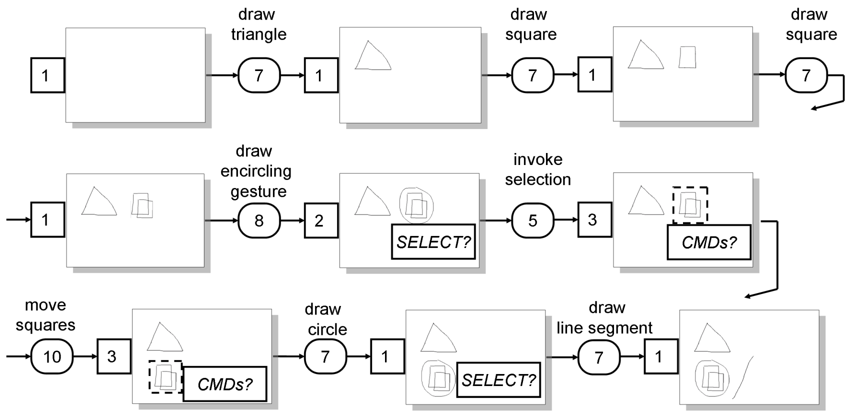
Fig. 12 Interaction Flow diagram for the Inferred Mode protocol. The diamond represents the program inferring the user's intended mode. If the intent is ambiguous, a pop-up mediator choice (b) is presented which the user may either tap to select encircled material, or ignore and continue writing or drawing.

clearly a selection operation, or else ambiguous. The decision is made on the basis of certain rules which make use of the machine's prior state, plus the stroke's location, shape, and proximity to other strokes on the canvas. For example, a trajectory creating a closed path is interpreted in the following way:

- If the path encloses no other strokes then it is clearly a drawn stroke.
- If the path encloses at least one other stroke AND some other strokes are selected, then the path is interpreted as a selection gesture that adds the enclosed strokes to the set of selected strokes.
- If nothing is selected and the path encloses at least one other stroke, then the intent is ambiguous. The user could be intending to select the enclosed strokes, or they could simply want to draw a circle around them.



a



b

Fig. 13 Interaction flow for Task I (a) and Task II (b) under the Inferred Mode Protocol. Numbers indicate nodes in Figure 12. Note that the user's actions are identical until the point at which they either ignore or tap the pop-up "Select?" button at step IM-IF-3 (3 in the figure).

Critically, this gesture interpretation is made after the stroke, and the burden is lifted from the user to specify the correct Draw or Command mode prior to performing the motion. Only if the stroke is ambiguous is the user presented with the “Select?” mediator, at which time they have the choice of tapping the pen to select the enclosed material, or else ignoring it and proceeding to draw either additional digital ink strokes or else an entirely different enclosing gesture to select something else (IM-IF-6).

Figure 13 details the interaction flow for Sample Tasks I and II under the Inferred Mode interaction protocol. Tasks I and II are performed identically through the first four actions, where the user executes a circular pen trajectory enclosing the squares. At this point the program cannot know whether the user intends to draw a circle or select the squares it encloses. The system displays the popup “Select?” menu item. Here the two tasks diverge. Under Task I, the user ignores the menu item and continues drawing, completing the task with the entry of the final diagonal line. Under Task II, where the user intends to move the squares, they tap on the “Select?” button and the squares become highlighted as selected objects. The user then drags them to the target location, and, without explicitly switching modes, proceeds to complete the task by drawing a circle around the squares, then the final diagonal line.

The Inferred Mode Protocol for pen/stylus interfaces makes minimal use of structure analysis of canvas content, limited simply to determining whether a stroke is approximately closed and if so, whether it encloses existing markings. Further development of intelligent user interfaces involves more sophisticated analysis of the visible canvas in conjunction with the dynamics of the user’s stroke.

5 Sloppy Selection: Inferring Intended Content of an Ambiguous Selection

Let us assume that the digital ink and bitmap images on a canvas are not arbitrary, random strokes and images, but are meaningful, structured objects. Most instances in which a user intends to cut, copy, move, or otherwise modify material, they do so with respect to this structure. It makes sense to bias interpretation of the user’s actions in terms of the coherent objects and groupings present on the canvas. The most commonplace application of this principle applies to the characters, words, lines, and paragraphs comprising text. While always permitting exceptions, selection operations should tend toward selection of these units.

In accordance with this principle, we have suggested a user interface technique for lasso selection called *Sloppy Selection* [8]. Sloppy Selection observes that users’ lasso gestures may at times only approximately encircle the object(s) they intend to select. To the extent that the user perceives objects on the canvas as being organized into salient chunks, a quick, approximate selection gesture may be “good enough”. Conversely, we assume that if users intend to select arbitrary, non-salient regions of the canvas, they will do so slowly and deliberately. The Sloppy Selection technique thus analyzes the dynamics of the user's lasso gesture to ascertain whether and in what portions of the gesture the user is performing a rough, quick-and-dirty stroke, versus a careful, deliberate partitioning of selected versus excluded material.

In order to implement Sloppy Selection, we must employ a model of user gestures under casual and deliberate intent. We assume that casual, "sloppy" strokes are performed balistically, with a single motor planning event involving minimal mid-course correction. This type of motion is known to follow the minimum jerk principle, from the biological motor control literature. Figure 14a illustrates the speed profile of a fast, single-motion lasso gesture. Slowing occurs at locations of highest curvature according to a 2/3 power law. Conversely, careful, deliberate strokes occur at a much slower speed more closely obeying a "tunnel law" of motion [1], as seen in Figure 14b.

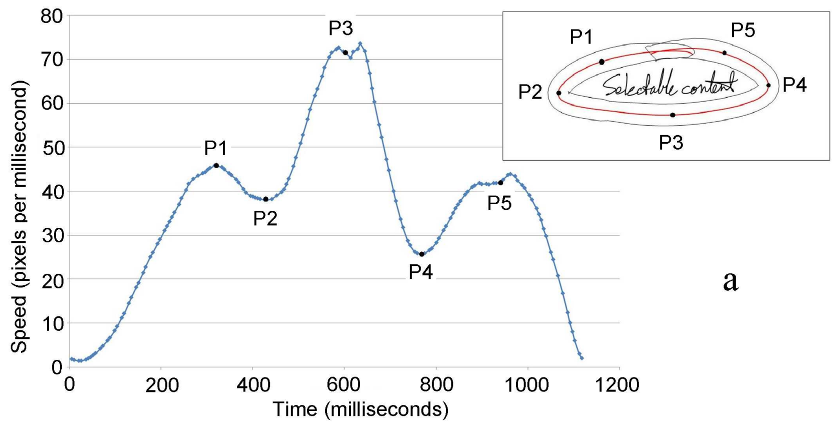
We exploit the difference between fast casual gestures and slow, deliberate gestures by inverting the local speed profile along a gesture to infer what we intpret as an effective selection tolerance width. Where a gesture's speed is less than would be predicted by a minimum jerk motion, we assume that the user is deliberately slowing down to more carefully adjust the gesture path, and therefore the effective tolerance narrows.

To combine the tolerance width with image structure analysis, we first construct candidate salient objects by performing visual segmentation and grouping on the existing canvas digital ink. Then, at the conclusion of a potential selection stroke we analyze the user's inferred selection tolerances. Where a lasso's selection tolerance permits, we divide included from excluded material according to the segmented units. But where the selection tolerance narrows, we split words or stokes literally along the lasso path, as shown in Figure 15.

6 Cycle Tap Selection: Exploiting Structure Recognition

The simplest and most direct method of selecting material with a mouse or pen is, respectively, mouse click (typically using the left mouse button) or pen tap. The problem is that this action is ambiguous with respect to the meaningful structure of canvas objects, because any given section of digital ink or fragment of bitmap image may belong to multiple coherent objects. The dominant PowerPoint UI design for graphics interfaces addresses this ambiguity through the use of groups. Primitive objects can be grouped hierarchically into groups that collectively form tree structures. See Figure 16b. Clicking on any primitive object automatically causes selection of the collection of primitive objects descending from the root node of any grouping tree the clicked object belongs to.

In PowerPoint-type UIs, groups are both a blessing and a curse. Once the user has grouped an object, in order to select that object and modify its location or properties, they must first un-group it. At this point, the group structure is lost and to get it back the user has to reconstruct it manually, which can become quite tedious. Thus,



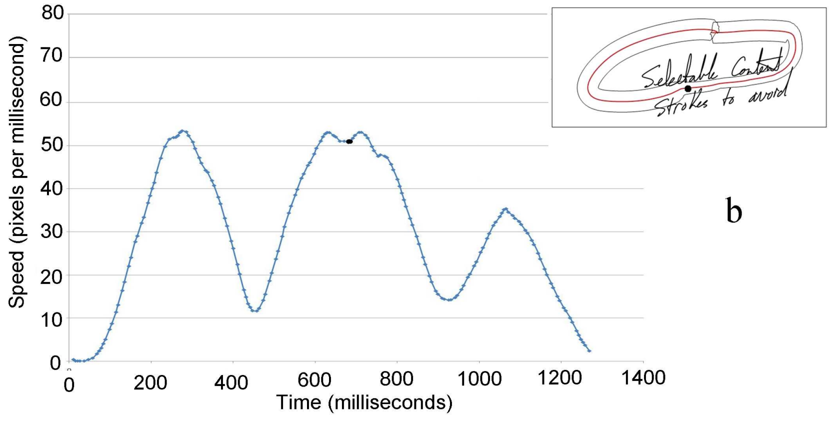


Fig. 14 a. Speed profile for a "sloppy" selection gesture. b. Speed profile for a "careful" selection gesture. Note the relatively slower speed for the straight section where the gesture is threading between two lines of text.

ambiguity and actionable membership in multiple groups as such are not actually supported.

We extend the notion of grouping primitive elements into meaningful larger structures in two stages, each of which carries design for intelligent UIs a step further. These steps are first, lattice groups, and second, automatic group formation through structure recognition.

To permit primitive strokes and bitmap objects to belong to more than one group simultaneously, we reformulate group structure from a hierarchical tree to a lattice. In a lattice, a child node may have more than one parent, and thus may participate in more than one group. This idea is taken to an extreme in the ScanScribe document image editor and the InkScribe digital ink sketch creation and editing tool. In these

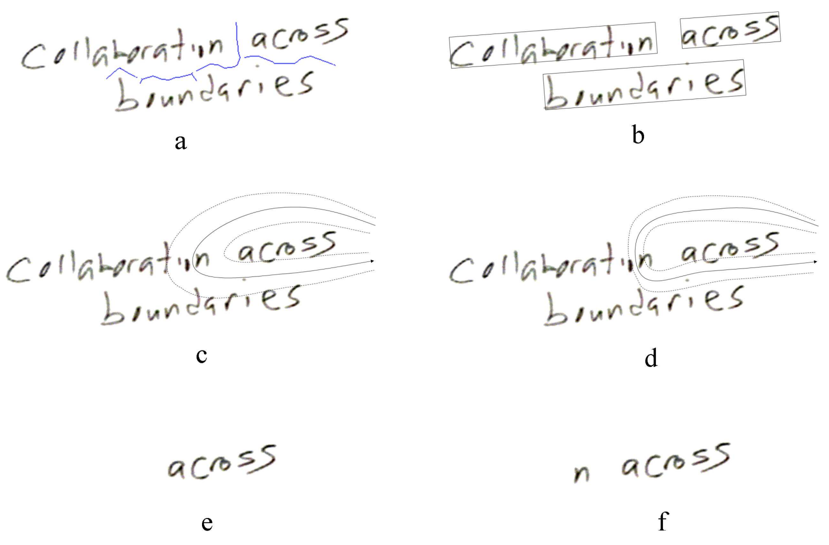


Fig. 15 Steps in the sloppy selection gesture interpretation technique. a., b. Detection of word objects. c., d, Inference of gesture carefulness vs. sloppiness. e., f. Decision whether to select based on word groups or precise gesture path.

programs, the lattice is flat, consisting of only primitives and a layer representing groups of primitives. Figure 16c illustrates that, for example, a lattice representation is sensible for maintaining the meaningful groupings of a tabular arrangement of cells. Any given cell simultaneously belongs to a row, a column, and the entire table.

The user interface design problem posed by lattice groupings is, how to give the user control over selection in terms of the multiple available options. A straightforward approach is called Cycle Click/Tap Select. Clicking (a mouse) or tapping (a pen) once on a stroke or bitmap object causes the primitive object itself to be selected. Tapping again selects one of the groups that object belongs to to become selected. Tapping repeatedly then cycles through the available groups. Our experience with ScanScribe and InkScribe suggest that the Cycle Click/Tap Selection technique is effective when the groups are all sensible and limited to about five in number. Each tap requires visual inspection of the selection (indicated for example by a highlight halo).

In basic ScanScribe and InkScribe, groups are formed in either of two ways. Any combination of primitives can be selected manually by clicking with the shift key (in ScanScribe for the mouse platform) or tapping individual objects (in InkScribe for the pen/stylus platform). Then, an explicit menu item permits explicit creation of a group. Or, groups may be formed automatically when the user manually selects a

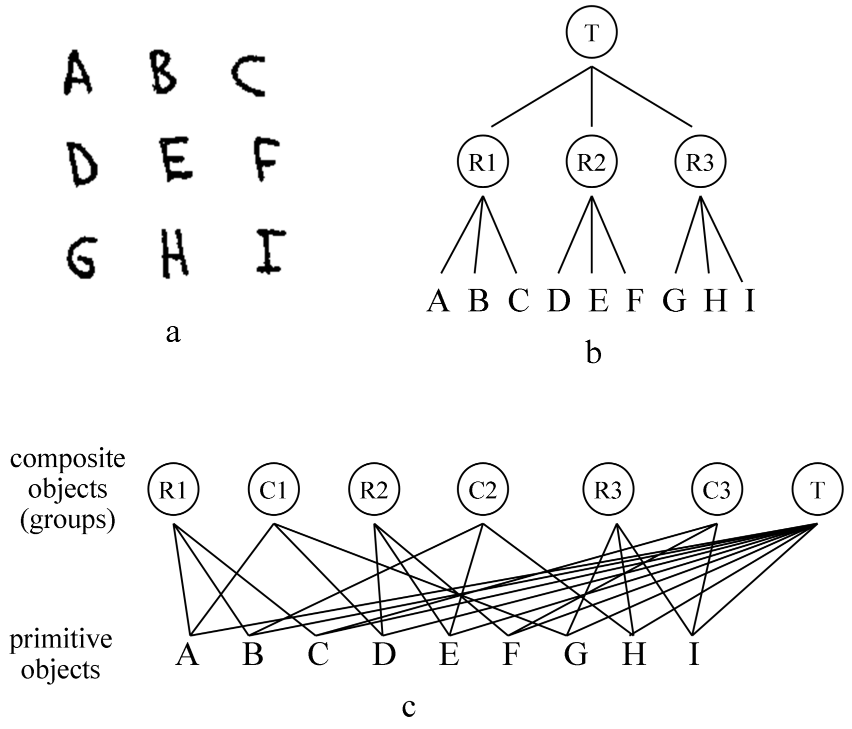


Fig. 16 a. Items arranged in tabular layout. b Hierarchical groupings as rows then table. c. Lattice structure permitting elements to belong to both row and column groups as well as the entire table.

collection of primitives, and then performs any operation such as moving, copying, changing color, etc.

This approach to multiple, overlapping group structure forms the basis for a second, more advanced form of meaningful group-based selection of by direct Click/Tap. That is for groups to be formed automatically through automatic structure recognition.

Automatic structure recognition is exemplified in a program we have developed for creating and editing node-link diagrams, called ConceptSketch. Node-link diagrams are the basis for a popular graphical notation, called variously Concept Maps, or Mind Maps, for brainstorming and organizing information through labeled nodes representing cognitive concepts, and (optionally labeled) links depicting relations among concepts. The popularity of concept maps is evidenced by a multitude of free and commercial programs available for creating and editing these diagrams. At this writing, however, none of the available programs offers a truly fluid user interface permitting users to simply draw a concept map in freeform fashion and then select nodes, links, and labels as meaningful objects to rearrange, form and delete new nodes and links. and label or annotate. By design, ConceptSketch is an extension of a basic Electronic Whiteboard that knows about node-link diagrams and automatically recognizes the constructs of this notation automatically as the user creates it.

The key to a powerful concept mapping program is automatic recognition algorithms that can identify and organize the elements of a concept map, including text representing node labels, graphical node indicators, links, arrows, link labels, and arbitrary annotative text and graphics. The recognition strategies we have developed lie beyond the scope of this article. Here, of interest are the user interface techniques for accessing the meaningful diagrammatic objects once they have been recognized.

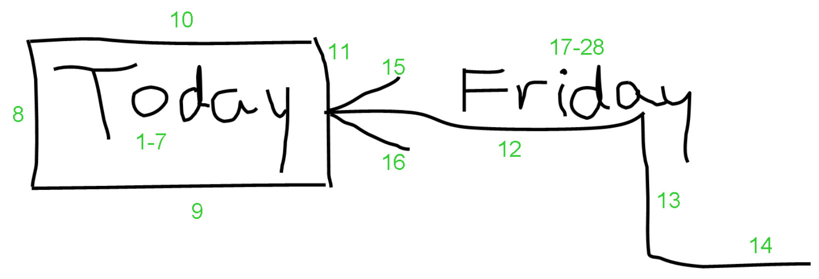
The Cycle Click/Tap select technique serves this purpose. See Figures 17 and 18. In prototypical use, we presume that the user's overall goal is to evolve a rough and malliable sketch into a formalized diagram. The meaningful objects here are: (1) the graphical object representing the concept nodes; (2) the textual labels of these nodes; (3) entire nodes consisting of both node graphics and their textual labels; (4) the curvilinear lines linking concepts; (5) arrows or other terminating graphics of link graphics; (6) textual labels associated with graphical links; (7) entire links consisting of the link lines, their terminator graphics, and their labels; (8) ancillary text; (9) ancillary graphics. To support Cycle Click/Tap select, recognition algorithms need to build structured representations of the canvas that reflect these groupings of primitive digital ink and text objects. Any given primitive may belong to more than one group. Figure 18 illustrates that in the ConceptSketch program, the user may select different levels of structure by repeated tapping. Tapping once on the side of a rectangle forming the enclosing graphic of a node causes that node to be selected, including its text label; tapping again at the same place cycles to selection of just the rectangle; tapping again cycles to selection of just the side of the rectangle.

For creation and editing of Concept Maps using a pen/stylus in ConceptSketch, the Cycle Tap Select protocol is embedded within the Inferred Mode protocol of Figure 12 in particular, within the Tap selection Node IM-IF-8.

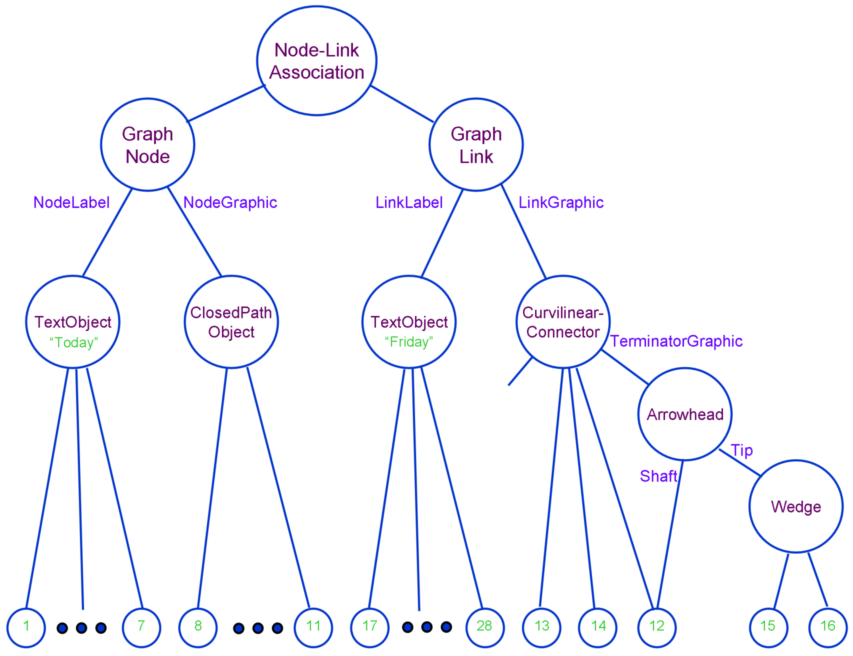
This principle of course applies to all types of graphical structure, across all domains for which effective recognition algorithms can be devised, including circuit diagrams, mathematical notation, physical simulations, engineering and architectural drawings, chemical diagrams, UML diagrams, etc.

7 Conclusion

The goal of creating computer tools that anticipate and understand user actions in terms of their purpose and intent is an ambitious one that will not be realized for quite some time. We can however realize some of the lower levels of the pyramid of sophistication that will be required. Our emphasis in this chapter has been on minimizing the requirement that the user pre-specify modes in order to communicate to the program how their subsequent action should be interpreted. We have shown how at the most basic UI level, Overloaded Loop Selection enables multiple methods for selection without the use of a toolbar. We have introduced conservative forms of



a



b

Fig. 17 a. Example sketch. Strokes are labeled in order of input by a pen. b. Hierarchical graph representing the objects and relations of the sketch in terms of the elements of a Node-Link diagram.

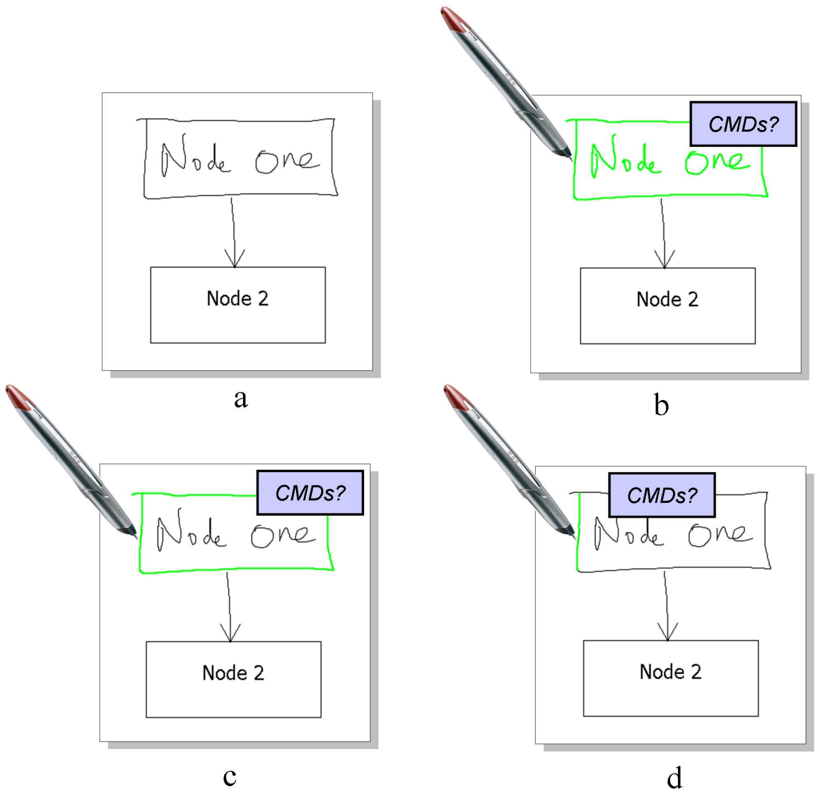


Fig. 18 Cycle tap select of salient objects in a node-link diagram. a. Diagram partially formalized. b. At the first pen tap on the enclosing graphic, the entire node is selected (the enclosing graphic plus text label). c. At the next pen tap the enclosing graphic alone is selected. d. At the next pen tap just the side of the graphic rectangle is selelected.

recognition of a user's gestural intent by considering gestures' paths and dynamics in context of canvas content; these are the Inferred Mode protocol and the Sloppy Selection technique. And we have shown how recognition of canvas content enables easy selection of meaningful objects through the simple tap/click command, under the Cycle Tap Selection protocol.

We believe that many more techniques will fill in these levels of the pyramid. Indeed, we have taken note of several very interesting contributions by our co-workers in this field. And we look forward to future developments in cognitive modeling of user tasks and goals that will lead to truly intelligent user interfaces.

8 References

References

1. Accot, J. and Zhai, S.; 1997; "Beyond Fitts' law: models for trajectory-based HCI tasks," *Proc ACM CHI* pp. 295-302.
2. Accot, J. and Zhai, S; 2002; "More than dotting the i's — foundations for crossing-based interfaces," Proc. CHI '02 (SIGCHI conference on Human factors in computing systems), pp. 73-80.
3. Grossman, T., Hinckley, K., Baudisch, P., Agrawala, M., Balakrishnan, R.; 2006; "Hover widgets: using the tracking state to extend the capabilities of pen-operated devices", *Proc. ACM CHI*, 861-870.
4. Hick, W.; 1952; "On the rate of gain of information", Quarterly Journal of Experimental Psychology, V. 4, pp. 11-36.
5. Hyman, R.; 1953; "Stimulus information as a determinant of reaction time," Journal of Experimental Psychology, V. 45, pp. 188-196.
6. Hinckley, K., Baudisch, P., Ramos, G., and Guimbretiere, F.; 2005; "Design and analysis of delimiters for selection-action pen gesture phrases in scriboli," Proc. CHI '05 (SIGCHI conference on Human factors in computing systems), pp. 451-460.
7. Lank, E., Ruiz, J., and Cowan, W.; 2006; "Concurrent bimanual stylus interaction: a study of non-preferred hand mode manipulation," GI '06 (Proceedings of Graphics Interface 2006), pp. 17-24, Quebec.
8. Lank, E., and Saund, E.; 2005; "Sloppy Selection: Providing an Accurate Interpretation of Imprecise Selection Gestures," Computers and Graphics, Special Issue on Pen Computing V. 29, No. 4, August 2005, pp. 183-192.
9. Li, Y., Hinckley, K., Guan, Z., and Landay, J.; 2005; "Experimental analysis of mode switching techniques in pen-based user interfaces," Proc. CHI '05 (SIGCHI conference on Human factors in computing systems), pp. 461-470.
10. Norman, D.; 1982; "Steps toward a cognitive engineering: Design rules based on analyses of human error," Proceedings of the 1982 conference on Human factors in computing systems, pp. 378-382, Gaithersburg, Maryland.
11. Pedersen, E., McCall, K., Moran, T, and Halasz, F.; 1993; "Tivoli: An electronic whiteboard for informal workgroup meetings," *Proc ACM CHI*, 391-398.
12. Ramos, G., and Balakrishnan, R.; 2007; "Pressure Marks", *Proc. ACM CHI*, 1375-1384.
13. Ramos, G., Boulos, M., and Balakrishnan, R.; 2004; "Pressure widgets,", Proc. CHI '04 (SIGCHI conference on Human factors in computing systems), pp. 487-494.
14. Raskin, J., *The Humane Interface*; 2000; Addison Wesley.
15. Ruiz, J. and Lank, E.; 2007; "A study on the scalability of non-preferred hand mode manipulation," ICMI '07: Proceedings of the 9th international conference on Multimodal interfaces, pp. 170-177, Nagoya, Aichi, Japan.
16. , Ruiz, J., Bunt, A., and Lank, E.; 2008; "A model of non-preferred hand mode switching," GI '08 (Proceedings of graphics interface 2008), pp. 49-56, Windsor, Ontario, Canada.
17. Saund, E., Fleet, D., Larner, D., and Mahoney, J.; 2003; "Perceptually-Supported Image Editing of Text and Graphics," Proc. UIST '03 (ACM Symposium on User Interface Software and Technology), pp. 183-192.
18. Saund, E., and Lank, E; 2003; "Stylus Input and Editing Without Prior Selection of Mode," Proc. UIST '03 (ACM Symposium on User Interface Software and Technology), pp. 213-216.
19. Sellen, A., Kurtenbach, G., and Buxton, W.; 1992; "The Prevention of Mode Errors through Sensory Feedback," Human-Computer Interaction, 7:2, p. 141-164.
20. http://www.usabilityfirst.com, June, 2008.