Fundamental States of Interaction for Pen, Touch, and Other Novel Interaction Devices

Ken Hinckley
Microsoft Research, One Microsoft Way, Redmond, WA 98052
kenh@microsoft.com

ABSTRACT
Traditional mouse-based desktop interfaces support a number of fundamental capabilities that frequently become problem areas for novel interaction devices with new capabilities. The same interaction and device design problems come up with these devices over and over again, with researchers and designers repeating the same mistakes. This note describes these fundamental capabilities, how they are supported on mouse-based interfaces, and identifies fundamental problems in extending them to novel interaction devices. My hope is that this can help designers identify basic mismatches between input device and interaction technique in the early stages of a design, as well as to consider design solutions (or to be more precise, various design compromises) that have been proposed.

Keywords
pens, hands, tables, mobile devices, touch, proximity, pressure, tracking, dragging, three-state model

INTRODUCTION
What do PDA's, Tablet PC's, touchscreens, interactive tables, and bare-hand sensing devices all have in common?

Despite their rich and exciting interaction possibilities, none of them can do everything a mouse can do.

This simple fact repeatedly leads designers to three fundamental choices in the design of their devices and interaction techniques. The options are simple, yet often equally unpalatable:

1. Design all interactions and applications such that they are based upon only dragging actions, with no intermediate cursor feedback. This can be very limiting, and prevents use of unmodified "legacy" applications that were not written to understand the special limitations of a device.
2. Add new capabilities to the input device to allow it to directly incorporate additional state transitions. These may include adding buttons or pressure-sensing capabilities to a device. However, these may not necessarily make the interaction that much simpler, especially for novices. For example, barrel buttons on a pen place constraints on how the user can hold the pen, and can be difficult to press without disturbing the position of the pen tip. Buttons on a trackball or touchpad often are awkward to hold down while moving the cursor, and may even require the user to employ a second hand just to hold the button.
3. Introduce special mechanisms that allow simulation of mouse click, right click, dragging, cursor feedback, and hover capabilities. These introduce indirect mechanisms that users must learn, which often gets away from the direct and simple interactions that make direct input devices and touch devices appealing in the first place.

Pick your poison: no matter how clever you are, you will have to make one (or more!) of the above compromises.

A TALE OF TWO STATES
Input devices taken in general can support three possible states: out-of-range, tracking, and dragging [1]. Practitioners refer to these three states as states 0, 1 and 2, respectively, as a simple shorthand (Fig X). However, many input devices, including traditional mice, can only support two of the three possible states. Conveniently, these are not the same two states that are sensed by most touch-based input devices. Even more confusing, as we shall discuss later, is that traditional WIMP (windows-icons-menus-pointer) interactions use these two basic mouse device states, plus the buttons on the mouse, to support five interaction states.

State	Description
0	*Out Of Range:* the device is not in its physical tracking range.
1	*Tracking:* moving the device causes the tracking symbol to move.
2	*Dragging:* allows one to move objects in the interface.

Fig. 1. Summary of the three-state model of input devices.

All of this leads to unending tension between designs for each type of device, and confusion about what is and what is not possible with a device.

Unpublished manuscript 2004

This work-in-progress underscores some of the difficult issues in supporting standard graphical user interface operations on pen and touch-operated devices.

It is currently more of a reminder of the general problems likely to be encountered, and is not as strong in pointing out solutions. For many combinations of input device and desired functionality it is important to be aware that there may not be any good or easy solution, however.

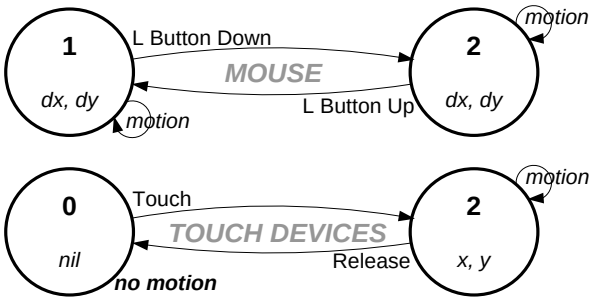

Fig. 2. States sensed by a mouse (**top**) versus states sensed by basic touch devices (**bottom**). The fundamental problem is that many touch devices only support one state of interaction that can sense the position (motion) of an input.

The 3-state model describes the mouse as a 2-state device, supporting state 1, the cursor tracking state, as well as state 2, the dragging state. State 1 allows the user to receive feedback of what position on the screen the mouse button will act upon, while State 2 allows the user to drag an object on the screen by pressing and holding the mouse button. Note that traditional graphical user interfaces assume the mouse can track its position in both the tracking and dragging states, represented in Fig. X by the (dx,dy) in each state, indicating relative motion tracking capability.

Touch-activated devices such as touchscreens, touchpads, and PDA screens (operated by stylus or finger) are also two state devices, but as Buxton points out, these devices do not sense the same two states as the mouse. A PDA with a stylus, for example, can sense the stylus when it is in contact with the screen; this is the equivalent of the mouse dragging state (state 2). The PDA can also sense when the stylus is removed from the screen, but once the stylus breaks contact, this enters state 0 (out-of-range), where no continuous property can be sensed at all (represented by *nil*, Fig. Y).

Thus, although PDA's support two separate interaction states, the lack of any motion sensing capability in state 0 means that there is going to be serious difficulty in supporting all of the same interactions that are possible with a mouse.

True Three-State Devices

Some touch-based input devices address this problem by providing additional sensing capabilities that support all three states of the 3-state model.

Proximity Sensing

One option is to support a proximity sensing capability. Many tablets, for example, can sense a special stylus when it is within about 2 centimeters of the screen surface. When the stylus is near the screen, the device enters state 1, providing intermediate cursor feedback of the screen location that the pen will act upon. Touching the pen to the screen surface enters state 2, which allows the user to click on objects or drag objects on the screen.

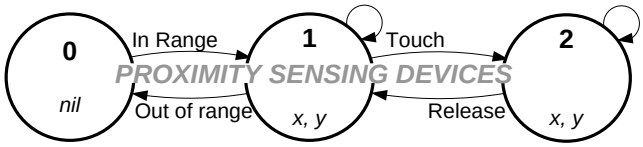

Fig. 3. A device that can track a pen when it is close to, but not touching the screen, and which can also distinguish this from when the pen is touching the screen, can use proximity to support state 1 (cursor tracking).

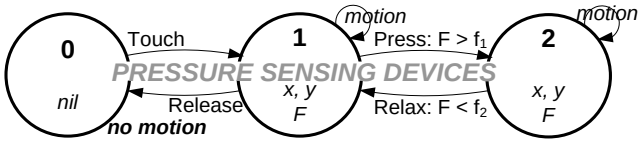

Fig. 4. A device that can sense pressure (or contact area) can use contact force to separate the tracking from the dragging states. Dual force thresholds should be used such that a high force is required to enter state 2, but a low force is required to revert back to state 1.

Most such devices offer an additional capability absent from traditional mice. Proximity sensing pens (tablets) can signal an input event when the stylus enters or leaves the range of the screen (i.e. an OutOfRange event for the state 1-0 transition, and an InRange event for a state 0-1 transition). These signals are often ignored, but can have interesting clever uses in interface design [1-3].

Pressure Sensing

A second option is to provide a pressure sensing capability.

In practice, many tablet computers support both range sensing and pressure sensing. Typically, in this case, proximity to the screen is used to perform state 0-1 transitions, and contact with the screen performs state 1-2 transitions. This leaves the interaction designer free to use pressure sensing as a continuously controllable parameter (e.g. some drawing programs change the shape of the user's pen stroke depending on the pen pressure). Users have a poor absolute sense of the pressure they are exerting, but can effectively control the exerted pressure with appropriate visual feedback encapsulated in "pressure widgets" [4].

The 5 Hidden States of Conventional WIMP Interfaces

Unfortunately, despite the techniques presented in the previous section, supporting both state 1 and state 2 with a pen or finger-based interaction is still not enough to support all the capabilities offered by the humble mouse. Modern graphical interfaces actually make use of at least 5 states. Here, we describe these states and introduce a specialized version of the 3-state model to describe them.

The five states of conventional graphical interfaces are as follows: Tracking (1), Hover (1_H), Left Click (2_L), Dragging (2_D), and Right Click (2_R). To complete this list, one must also consider double click. In principle this could be represented as a sixth state, but we prefer to think of it as a series of state transitions ($1\text{-}2_L\text{-}1\text{-}2_L\text{-}1$) in the model presented in Fig X.

State	Description
1	*Tracking:* moving the device causes the tracking symbol to move.
1_H	*Hover:* holding the mouse stationary over an object reveals Tooltips
2_L	*Left Click:* pressing the left mouse button. To complete the normal "Click" event, the user releases the mouse button (without moving).
2_D	*Dragging:* pressing the left mouse button and moving the mouse. Typically this drags the selected object, or activates rubber-band selection (if no object is selected).
2_R	*Right Click:* pressing the right mouse button activates context menus. The mouse can then be moved to make a selection.
-	*Double Click:* Double click is a series of (1-2_L-1-2_L-1) state transitions, with timing constraints, no mouse motion between clicks, and no mouse motion while in state 2_L.

Fig. 5. Fig D: Summary of the five states of conventional WIMP interfaces.

Note that state 0 is not part of the picture for conventional graphical interfaces, as conventional mice cannot sense state 0 (i.e., system software does not receive an event when the user picks up or puts down the mouse, nor when the user touches or lets go of the mouse). Such enhancements to the traditional mouse have been considered by researchers but have not yet found widespread use [3, 5].

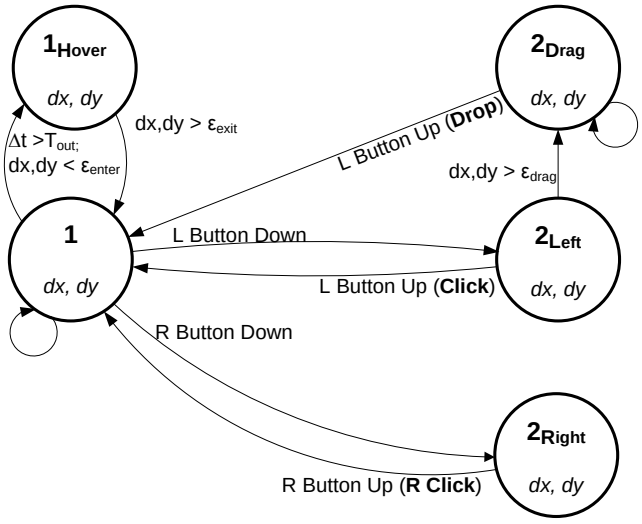

Fig. 6. Fig E: **Hidden states of interaction with the mouse.** The mouse is a "simple" device but there are actually 5 states needed by modern GUI's. These states support cursor tracking (state 1), clicking (state 2_L), dragging (state 2_D), right-clicking (State 2_R), and hovering (state 1_H). Double-clicking also must be considered, but we do not model this as a separate state (it is a 1-2_L-1-2_L-1 series of state transitions, with timing constraints and little or no cursor motion).

The Full Extent of Our Dilemma

We can now observe the full extent of the dilemma facing interaction designers working with basic touch devices (Fig. A, bottom) that cannot sense proximity or pressure. All current PDA's (e.g. Pocket PC's and Palm Pilots) represent examples of such devices. Such devices have just a single state, state 2, that can sense (x,y) position. In addition we have a pair of events, Touch and Release, that signal transitions to state 0. If one is to support all of the traditional mouse interactions that may be expected by an application with a conventional graphical interface, one must synthesize five states that rely on or make use of pointer motion (dx, dy) from this single position-sensing state and pair of events.

This is certainly difficult and a good solution may be impossible. We are not aware of any elegant solution that provides all five of these states for basic touch devices. It is possible to support all five states with more advanced three-state devices with proximity-sensing or pressure-sensing capabilities. However the resulting techniques do not have quite the same feel as mouse-based interfaces and suffer some significant deficiencies.

Of course, one can argue that pen-based interaction techniques should not attempt to imitate the techniques developed for the mouse and traditional graphical interfaces. Rather, interaction designers should focus on techniques, such as gesture recognition [6-9], marking [10-12], tracking menus [13], or pressure widgets [4], to name a few, which are uniquely well suited to pen or touch devices.

While we agree with this view in principle, in practice we anticipate that the inability of pen, touch, and other novel pointing techniques to adequately support traditional mouse-based interaction states will be a stumbling block for quite some time to come. Many applications are designed for mouse-based devices and ported to mobile devices. Many users are familiar with mouse-based interfaces and can have difficulty learning equivalent mechanisms for pen interfaces, or new learning new skills needed to succeed with pens.

Even casting all legacy issues aside, we believe that the ability to rapidly and/or unambiguously initiate one of several possible actions (or modes) with a pointing device represents a fundamental building block of graphical interaction techniques. The difficulty for the user to rapidly and/or unambiguously indicate, and for the system to recognize, multiple distinct interaction modes represents a continuous theme in the long history of research on pen, touch, 3D interaction and virtual environments, and other direct input techniques. The tricks that researchers and interaction designers have adopted to allow separation and rapid, unambiguous interpretation of inputs are often application-specific, may only work in specific contexts, or may become problematic or impossible to extend or maintain as new commands and features are added to a pen-

based interface. As such we believe these represent fundamental problems to the field.

Example Tricks of the Trade: Solutions, Problems, and Issues to Consider

Timeouts and/or dwell. Use timing cues, such as a tap-and-hold gesture, to allow the introduction of a separate mode (state) to the interaction, such as right-click simulation with pen interfaces. **Downside**: (1) the timeout adds extra time to every single activation of the associated functionality. (2) The timeout can be confounded with mental preparation time; the feature will be repeatedly activated by accident when the user is just thinking about what to do next.

Time delays are risky to use as implicit state transitions:

- Users may be thinking about their next action
- This introduces a delay at least as long as the timeout into every single user action involving this state transition. This can be tedious to the user, and irritating. Typically the user must wait significantly longer than the timeout threshold to be sure the action is not performed too soon.
- Appropriate delays may vary significantly between users. Novices may require long delays to prevent accidental mode switches, while more experienced users will quickly grow tired of delays of as little as 0.3 to 0.5 seconds.
- Time delays may be unituitive or mysterious. Typically there is no visual feedback associated with such delays, and the user may not realize what has triggered a differing behavior. Of course, such feedback may be added, such as the "circling" icon feedback provided by Pocket PC 2003 devices for the tap-and-hold action.

Extra state transitions: using a specific series of state transitions to segment gestures or indicate a separate state, e.g. using a "tap-and-a-half" gesture to allow dragging on a laptop touchpad [14]. Another example would be drawing a dot after a series of ink strokes to indicate that the "command" is complete. Downside: accidental contact with the screen or incidental movements may produce the same series of state transitions, triggering accidental and unwanted behavior.

Using a series of state transitions to synthesize extra states is possible, but this can be time consuming and may be error prone. For example, many laptops include touchpads that allow the user to "click" by tapping on the pad. This is a 0-1-0 state transition. However, if the user accidentally touches or brushes against the pad, this also triggers a 0-1-0 state transition. It can be difficult to reliably determine if this really represents a user's intent to "click" or not. More complex behaviors require increasingly complex tricks, such as a tap-and-a-half (state 0-1-0-1) gesture to initiate dragging. These become increasingly difficult for the user to perform, and increasingly error prone. They also lead to special cases (what happens when the user drags but hits the edge of the touchpad?) that introduce further complexity and modes in to the user interaction.

Out of band signals – e.g. resorting to an extra button on the device, barrel button on the pen, etc. These are simple and unambiguous signals. **Downside**: restricts how you hold the device(s), may prevent single-handed interaction, may not be self-revealing to the user, may be activated by accident anyway…

Highly modal interfaces – let the user explicitly select a mode of operation. Robust, simple. **Downside**: It takes time to select and to switch modes. User may enter information or perform actions in the wrong mode by accident. Mental load imposed to remember current mode. Mynatt & Igarashi [15] represents a good example of this design approach.

Speculative execution. This is a fancy way of saying "assume that the user is going to perform a default action, until proven otherwise." A good example of this is the "circling timer" speculative feedback provided by the PocketPC 2003 for right-click functionality via tap-and-hold. Bartlett provides another example in the context of using tilt to scroll [16]. **Downside**: Typically the designer can only "assume" one default action, so it is difficult to extend (it buys you one additional mode). It also requires "undoing" a user action that has already started, if the user does take the non-default interaction path. Thus it can only be applied to actions that are easily reversible, and there remains the potential to confuse the user or have misleading feedback for a short time.

Special case gestures for commands. For example, in the context where most pen input is treated as "ink," a special-case gesture, such as scribbling to erase, can be introduced. **Downside**: (1) Naturally occurring movements can be confused with the command (e.g. scribbling to shade an area in a picture is interpreted as a "scratch-out" gesture to delete ink strokes). (2) The approach scales poorly– it is extremely difficult to add more than a few commands in this way.

Clever tricks with input transfer functions, e.g. varying input device gain with velocity of movement [17], or touchscreen selection on lift-off with high resolution touchscreen pointing [18, 19]. Such techniques can be intuitive, even invisible, to the user if designed well. Downside: (1) May result in undesired or aberrant behavior in unforeseen circumstances or for some users. (2) may be too clever or require learning or modification of naturally occurring behavior; may make a simple device complicated to use. (3) may be very difficult to design well.

These are the main approaches. There are others that have been considered, mostly in specialized contexts. More to be fleshed out.

Some other random points to consider and remaining issues to be discussed. Under construction.

Instability of Touch/Pen Devices

It is hard to hold a stylus stationary on a screen

- When holding screen in other hand
- During tap or double tap
- When moving around with device
- Location of the cursor is disturbed when removing the pen/finger from the screen.
- With proximity tracking devices, the location of the cursor while holding the pen above the screen may change by the time user makes contact with the screen. For example, try dragging a standard window resize control on the Tablet PC. You hover to get the pointer just right, but by the time you press down and apply enough pressure the pointer has moved and you miss the resize handle.

Direct Input Devices are Special

On direct input devices, the hand or stylus acts as its own physical cursor, and occludes a portion of the screen. A virtual cursor separate from the hand or stylus can be introduced for more precise pointing, but arguably this interferes with the direct nature of the input. A problem can be making every pixel of the screen available for pointing. A constant offset for a virtual cursor relative to the finger must change depending on which edge of the screen the user may be closest to. Of course, parallax error is another common problem.

CONCLUSION

xxx

REFERENCES

1. Buxton, W. *A three-state model of graphical input*. in *Proc. INTERACT'90*. 1990: Amsterdam: Elsevier Science.
2. Hinckley, K., M. Czerwinski and M. Sinclair. *Interaction and Modeling Techniques for Desktop Two-Handed Input*. in *Proceedings of the ACM UIST'98 Symposium on User Interface Software and Technology*. 1998. San Francisco, Calif.: ACM, New York.
3. Hinckley, K. and M. Sinclair. *Touch-Sensing Input Devices*. in *ACM CHI'99 Conf. on Human Factors in Computing Systems*. 1999.
4. Ramos, G., M. Boulos and R. Balakrishnan. *Pressure Widgets*. in *CHI 2004*. 2004.
5. Rekimoto, J., T. Ishizawa, C. Schwesig and H. Oba. *PreSense: Interaction Techniques for Finger Sensing Input Devices*. in *UIST 2003*. 2003.
6. Buxton, W., E. Fiume, R. Hill, A. Lee and C. Woo. *Continuous hand-gesture driven input*. in *Proceedings of Graphics Interface '83*. 1983.
7. Mynatt, E.D., T. Igarashi, W.K. Edwards and A. LaMarca. *Flatland: New Dimensions in Office Whiteboards*. in *ACM SIGCHI Conference on Human Factors in Computing Systems*. 1999. Pittsburgh, PA.
8. Moran, T., P. Chiu and W. van Melle. *Pen-Based Interaction Techniques for Organizing Material on an Electronic Whiteboard*. in *Proc. ACM UIST'97 Symp. on User Interface Software & Technology*. 1997.
9. Kramer, A. *Translucent Patches--Dissolving Windows*. in *Proc. ACM UIST'94 Symp. on User Interface Software & Technology*. 1994.
10. Buxton, W., *Touch, Gesture, and Marking*, in *Readings in Human-Computer Interaction: Toward the Year 2000*, R. Baecker, et al., Editors. 1995, Morgan Kaufmann Publishers. p. 469-482.
11. Kurtenbach, G. and W. Buxton. *Issues in Combining Marking and Direct Manipulation Techniques*. in *Proc. UIST'91*. 1991.
12. Kurtenbach, G. and W. Buxton. *The Limits of Expert Performance Using Hierarchic Marking Menus*. in *Proc. INTERCHI'93*. 1993.
13. Fitzmaurice, G., A. Khan, R. Pieke, B. Buxton and G. Kurtenbach. *Tracking Menus*. in *UIST 2003*. 2003.
14. MacKenzie, I.S. and A. Oniszczak. *A Comparison of Three Selection Techniques for Touchpads*. in *Proc. ACM CHI'98 Conf. on Human Factors in Computing Systems*. 1998.
15. Nelson, L., S. Bly and T. Sokoler. *Quiet Calls: Talking Silently on Mobile Phones*. in *CHI 2001*. 2001.
16. Bartlett, J.F., *Rock 'n' Scroll Is Here to Stay*. IEEE Computer Graphics and Applications, 2000(May/June 2000): p. 40-45.
17. Hinckley, K., E. Cutrell, S. Bathiche and T. Muss. *Quantitative Analysis of Scrolling Techniques*. in *currently submitted to CHI 2002*. 2001.
18. Sears, A. and B. Shneiderman, *High Precision Touchscreens: Design Strategies and Comparisons with a Mouse.* International Journal of Man-Machine Studies, 1991. **34**(4): p. 593-613.
19. Sears, A., C. Plaisant and B. Shneiderman, *A New Era for High Precision Touchscreens*, in *Advances in Human-Computer Interaction*, Hartson and Hix, Editors. 1992, Ablex Publishers. p. 1-33.