Regular Paper

One Stroke Operations: A New Pen-based User Interface that can Integrate or Separate Operand Specification, Menu Opening and Selection, and Action Execution, in One or More Strokes

S. NAVANEETHA KRISHNAN† and SHINJI MORIYA†

This paper proposes a new pen-based user interface based on the concept of integrating and separating the operation stages in *one or more strokes*. That is, the user can perform operand specification, menu opening and selection, and action execution, in *one stroke* (without needing to lift the stylus pen from the tablet surface at any time during the operation) or in *two or more strokes* (lifting the pen-tip after completing any operation stage). The above concept enables *one stroke operations*, which we expect require fewer pen movements and lesser time. As the starting step, we designed and implemented these operations on a prototype pen-based ink-editor. *One stroke operations* were enabled due to two reasons — (i) Existence of the three elements, namely, *ink-states* which denote various units of ink-data for performing operations on, pen and pie menu and linear menu, and, *data-cum-tools* that possess properties of both data and tools, and, (ii) "Integration and separation of the operation stages" in *one or more strokes*. The concepts described in (i) and (ii) above can be extended to other user interfaces that use direct pointing devices (for example, mouse or hand as in touchscreen). Some examples are, mouse-based user interfaces, other pen-based user interfaces (such as pen-based text-editors or pen-based graphics-editors). We performed a preliminary experiment to compare *one stroke* operations with *two/three stroke* operations, and compared pie menu with linear menu.

1. Introduction

Pen-computers such as "Newton" and "ZAURUS" that use handwritten input are gaining popularity. And more research is being done to make them easy to use. Such research can focus on hardware or on software. This paper focusses on the software, specifically, the user interface of pen-computers.

This paper aims to enable *one stroke operations*. In this paper, *one stroke* denotes the movement of the pen, starting at the instant the pen-tip (that is, the tip of stylus pen) touches the tablet surface, and ending at the instant the pen-tip is lifted from the tablet surface. One *operation* includes *all* the operation stages — the stage of specifying the operand (the data to operate on), the stage of specifying the operator (the "tool" with which to operate, such as "Move") and the stage of executing the action (for example, moving the data). And *one stroke operations* are those operations in which users can complete all the above three operation stages in *one or more strokes*. Though the term *one or more strokes operations* may be more appropriate for these operations, however to shorten the term, we use the term *one stroke operations*.

As a specific example of performing operations in *one stroke*, let us consider "moving" the data from one location of the pen-computer screen to another. The user puts the pen-tip down on the tablet surface and specifies the operand by rubber-banding this operand. While the pen-tip is still in contact with the tablet-surface, the menu is displayed near the location of pen-tip and the user selects from this menu by moving the pen-tip in the direction of the menu-item. While the pen-tip is still in contact with the tablet surface, he/she executes the "move" action by continuing to drag the pen-tip to the final "move" location, and the moment he/she lifts the pen-tip, the data is redisplayed at its new location.

One stroke operations are essential in tasks such as discussions or teaching where users want to operate (for example, delete or move) their handwritten sketches or tables, while they are talking with others (for example, the discussion participants or the students). *One stroke operations* are also essential in situations where

† Department of Information and Communication Engineering, Tokyo Denki University

users are writing and operating on the narrow screens of pen-computers, where, frequent pen-down (putting the pen-tip down on the tablet-surface) and pen-up (lifting the pen-tip from the tablet surface) could result in mistakenly placing the pen-tip on an undesirable part of the screen, for example, the operand adjacent to the desired operand. In addition, *one stroke operations* can also be used in the important research area that studies user behavior in various tasks or situations, where users integrate the operation stages in *one stroke* or perform operation stages in *multiple strokes*.

Some of the current pen-based user interfaces let users perform operations using menu, some use handwritten pen gestures, and some use both.

In menu-based user interfaces[4),7)~12)], the operation stages (operand specification, menu selection, and action execution) are performed using multiple pen-downs and pen-ups. This requires large pen movements (especially in big-sized pen-computers as in Ref. 12)), and to avoid mistakes the user must carefully point at the operand or the menu. As a result, time would be consumed, the user would get tired and would be distracted from his/her work.

In gesture-based user interfaces[1)~3),5)~7),12)], the operand and the operator can be specified in *one stroke*. As a specific example, though Ref. 3) enables operations (only move, copy and delete) in *one stroke* using handwritten gestures, Ref. 3) integrates operand specification and action execution, that is, menu opening and menu selection are not integrated. Now, as gestures increase in number, users must practice them. Also, users must write gestures carefully to avoid misrecognition, thus slowing down the user. In addition, developers must construct the recognition algorithm.

This paper resolves the above-mentioned problems of menu-based pen user interfaces by integrating and separating the operation stages in *one or more strokes*. And by using the radial (or pie) menu instead of gestures, we resolved the above-mentioned problems of gesture-based pen user interfaces. This is because, pie menu enables menu selection when displayed, and gesture-like pen movements when not displayed (explained in more detail in Section 2.3.2).

For performing operations on pen-input systems, this research is the first to integrate and separate operand specification, menu opening and selection, and action execution, in *one or more strokes*. This research is also the first to use pie menu for operating on data while creating this data with pen-input in the same application.

As the starting step, we implemented *one stroke operations* on ink-data. Here ink-data refers to the on-line handwritten text, sketches, etc. that are input by writing on the tablet surface with stylus pen, and exists as a sequence of coordinate points denoting the pen-tip movements over the tablet surface. In this paper, unless otherwise specified, "data" refers to ink-data. Now, we targeted ink-data for two reasons. First, it would be convenient if users can perform operations on ink-data while creating this data side-by-side. Second, in many tasks it is practical to keep the handwritten text, figures, etc. in handwritten form instead of recognizing and converting into text-data, graphics-data, etc. For example, in discussions, keeping the sketches, etc. in handwritten form avoids misrecognition problems, and does not disrupt the flow of discussions.

Section 2 illustrates the concept of *one stroke operations*, and also describes the reasons that enabled these operations. Section 3 describes the design of this paper's user interface. Section 4 discusses how this paper's proposed concepts can be used in other user interfaces. Section 5 describes the experiment performed to evaluate *one stroke operations*. Finally Section 6 summarizes this paper.

2. Concepts of the Proposed User Interface

Section 2.1 describes the prototype ink-editor, Section 2.2 illustrates *one stroke operations*, Section 2.3 explains the first reason that enabled these operations, and Section 2.4 explains the second reason that enabled these operations.

2.1 The Prototype Ink-editor

The "ink-editor" refers to the pen-based editor for inputting ink-data and operating (deleting, moving, etc.) this data. **Figure 1** shows a hard-copy of the screen of the prototype ink-editor. This screen has three regions — (i) The wide region on the left for writing and operating on ink-data, (ii) "Global Menu" for performing operations (such as "File Load") that affect all the ink-data in the document being created and/or edited, (iii) "Micro Screen"[21)] for scrolling this document.

Figure 1 shows actually inputted ink-data in

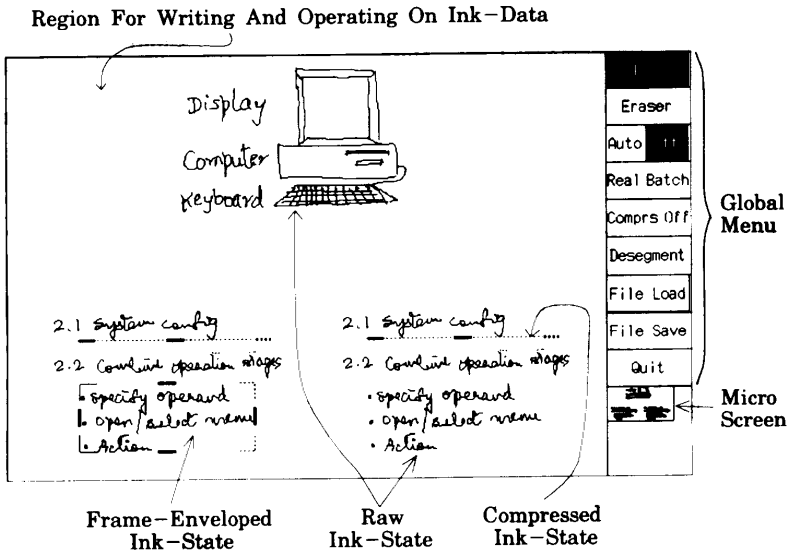

Fig. 1 Hard-copy of the screen of the prototype ink-editor. "Global Menu" is for operations that affect all the ink-data in the document. "Micro Screen"[21] scrolls the screen. The labels "Raw Ink-state", etc. below the screen denote various units of ink-data for operating on.

the form of various *ink-states* (explained later in Section 2.3.1). In "Global Menu" the number of menu items that remain visible at all times is few. This was done to reduce menu size and allow more space for writing and operating on ink-data. The other menu items for operating on ink-data are contained inside the pie menu (explained in Section 2.3.2) or the rectangularly shaped "linear menu" (explained in Section 2.2).

The prototype ink-editor runs on the MS-DOS based personal computer PC-9801 (NEC Corporation), to which a tablet-cum-display HD-640 and stylus pen SP-200A (WACOM Corporation) are connected.

2.2 Examples of Integrating and/or Separating Operand Specification, Menu Opening and Selection, and Action Execution, in *One* or *More Strokes*

One stroke operations are illustrated in **Fig. 2**. This figure simulates the situation of jotting down the main points of this paper. In this figure, the labels "Stage 1" to "Stage 4" represent the four stages of a typical operation that uses menu. From top to bottom, both Figs. 2a and 2b show the four stages of "Compress" operation. The ink-data in Stage 1 of Fig. 2a is the hard-copy of the bottom-right portion of Fig. 1, and the ink-data in Stage 1 of Fig. 2b is the hard-copy of the bottom-left portion of Fig. 1. The mock-up of the stylus pen was not actually input, but was pasted later on.

In Stage 1 of Fig. 2a, the user specifies the operand by constructing a "rubber-band" between the points labeled ① and ② (the method of constructing the rubber-band and how the end of operand specification is denoted using rubber-band, is explained in Section 3).

Immediately after operand specification, in Stage 2, the pie menu opens (that is, is displayed) at location ②, with its center exactly coinciding with the bottom-right corner of the rubber-band.

Without lifting the pen-tip from the tablet surface, in Stage 3, the user drags the pen-tip to the "Comp" (abbreviation of "Compress") menu item (the borders of "Comp" menu item are shown as thickened lines) and lifts the pen-tip from the tablet-surface.

The moment the pen-tip is lifted, the action (that is, "compression" of ink-data) is executed in Stage 4, and "compressed" ink-data is formed.

Thus, in *one stroke*, the user specifies the operand, opens the menu, selects the menu and executes the action.

Focussing on Stage 3 of Fig. 2a, if instead of selecting "Comp" menu item, the user selects "Seg" menu item (abbreviation of "Segment"), then, already specified ink-data (which we call as *frame-enveloped ink-state*, explained in detail in Section 2.3.1) is created, shown in Stage 1 of Fig. 2b. The moment this already specified ink-

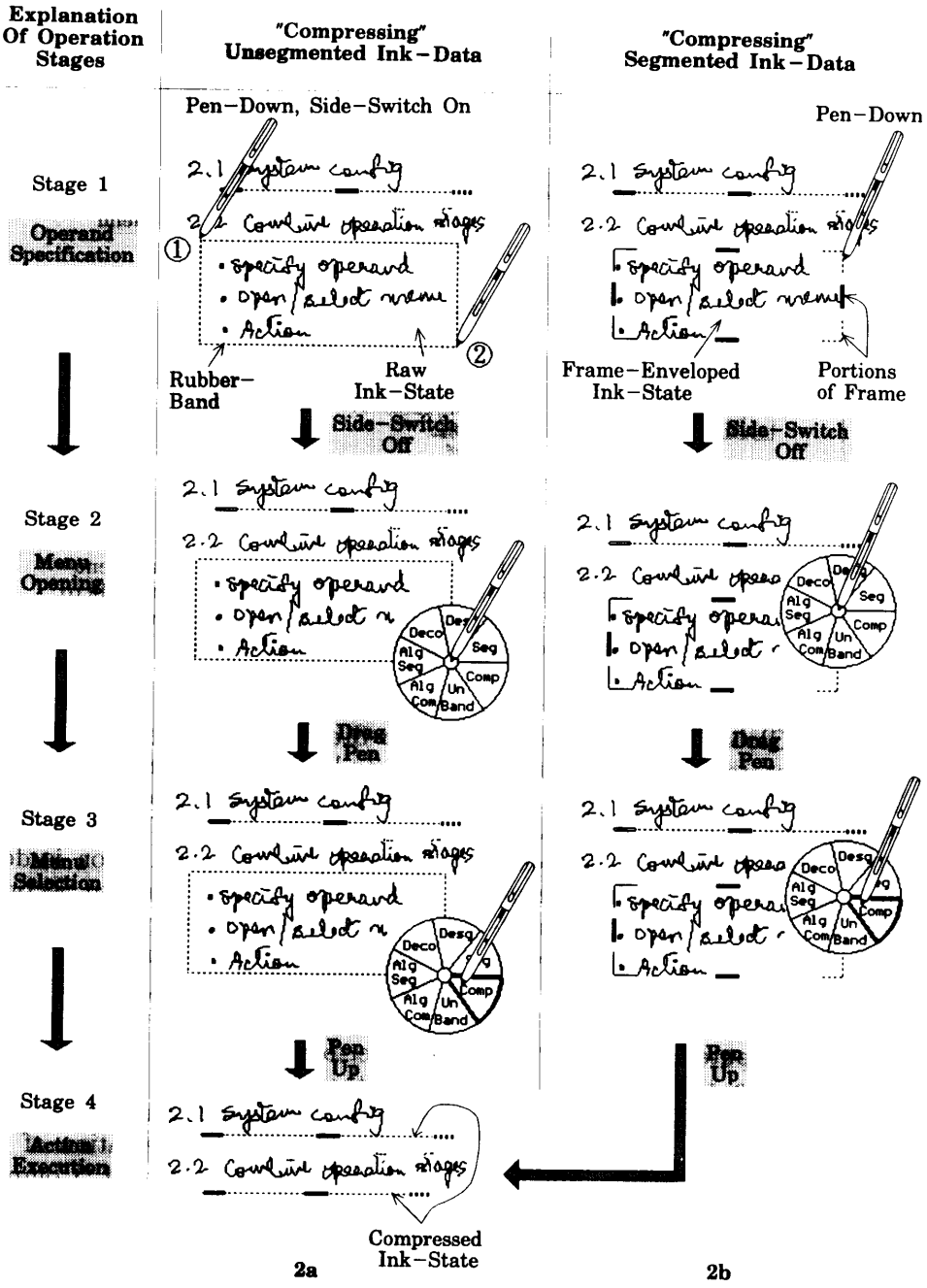

Fig. 2 Illustrates *one stroke operations* when a user "compresses" ink-data. The ink-data shown in Stage 1 to Stage 4 of 2a and 2b is the hard-copy of various portions of the screen of Fig. 1. The mock-up of the stylus pen was not actually written, but was pasted later on.

data is created, the *frame* is created along with it. The *frame*, displayed by the symbols " ⌐" "¬ " " └" and "┘ ", denotes that the ink-data contained within these symbols can be specified as operand simply by touching this *frame* with the pen-tip.

Now, focussing on Stage 1 of Fig. 2b, to "compress" already specified ink-data, the user simply puts the pen-tip down on the tablet surface at the location of the *frame*.

The instant the pen-tip touches the *frame*, in Stage 2, the pie menu opens exactly at the location of pen-down.

Without lifting the pen-tip from the tablet surface, in Stage 3 the user "drags" the pen-tip to "Comp" menu item and lifts the pen-tip

from the tablet surface.

The moment the pen-tip is lifted, the action is executed and "compressed" ink-data is formed.

As already explained in Section 1, the user can also perform operations in this user interface in multiple *strokes*, that is, the user can lift the pen-tip after completing any operation stage. With respect to Fig. 2a, in the first pen-down the user rubber-bands the operand and lifts the pen-tip. The rubber-band remains visible. The user puts the pen-tip down on the already displayed rubber-band which opens the menu, he/she selects the menu-item, and lifts the pen-tip, thus executing the action. Similarly, the "compress" operation of Fig. 2b can also be performed in *multiple strokes*.

The explanations till now described *one stroke operations* with respect to "one operand" type operations (such as "Compress" or "Segment"), in which the user specifies the operand and selects the operator. Below, we describe *one stroke operations* with respect to "two operand" type operations, that is, operations in which the user needs to specify the original operand as well as the destination for this operand. Two examples of "two operand" type operations are the "Move" and "Copy" operations.

Now, let us focus on **Fig. 3**, which shows "move" operation performed in *one stroke* by rubber-banding. This figure shows Stage 3 and Stage 4 of "move" operation, since Stage 1 to Stage 3 are similar to Fig. 2a. Specifically, in Stage 1 the user rubber-bands the ink-data from the right side of the tablet to the left (for reasons described later in Section 3.1.2), on doing which the pie menu containing the "Move" menu item opens in Stage 2. We can notice that the pie menu of Fig. 2 and Fig. 3 are different. Actually, this user interface has multiple pie menus (described later in Section 3.1.2).

Without lifting up the pen-tip, in Stage 3 the user drags the pen-tip to the "Move" menu item and continues to drag till the pen-tip goes outside the outer circle of pie menu. At this instant, the "Move" menu item gets selected, and Stage 4a of Fig. 3 is reached, that is, the rubber-band denoting the boundaries of the ink-data automatically moves towards the left with its bottom-left corner coinciding with the tip of stylus pen. This was done to eliminate the need to again move the pen-tip back to the operand.

From Stage 4a, the user drags the pen-tip until reaching the final desired position (Stage 4b)

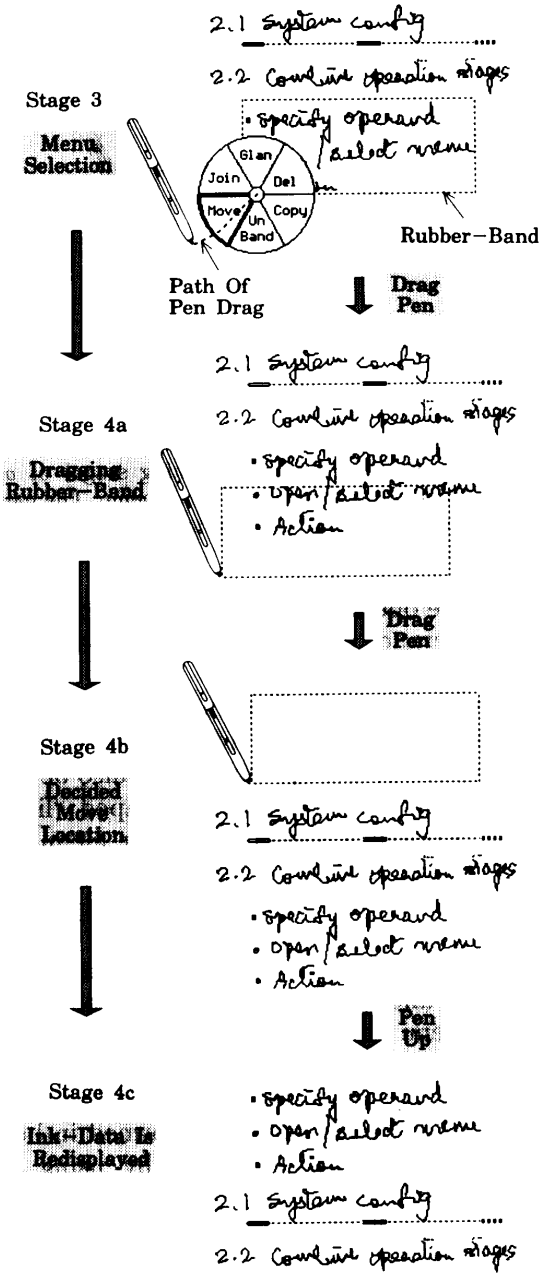

Fig. 3 *One stroke operations* for "Move". Stage 3 and Stage 4 (a to c) of "Move" are shown. In Stage 1, the user rubber-bands ink-data from the top-right to bottom-left corner of rubber-band; the pie menu or linear menu opens in Stage 2.

and then lifts the pen-tip. On doing this, in Stage 4c, the ink-data is redisplayed at its new position.

Thus, "two operand" type operations can also be performed in *one stroke*. The "move" operation of Fig. 3 can also be performed in *multiple strokes*, by lifting the pen-tip when it is over the "Move" menu item, then positioning the pen-

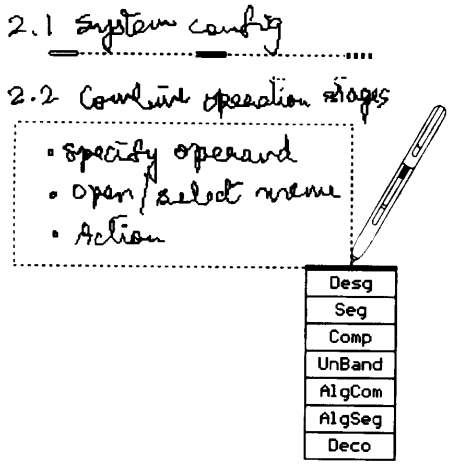

Fig. 4 Shows Stage 2 (menu opening stage) of *one stroke operations* using linear menu. The menu items in this menu are the same as the pie menu of Figs. 2a and 2b.

tip over the operand, putting the pen-tip down on the operand, and dragging the operand to the final destination.

Figures 2a, 2b and 3 illustrated *one stroke operations* using pie menu. Actually, in this user interface, *one stroke operations* can also be performed using the rectangularly shaped "linear menu". **Figure 4** shows the linear menu displayed in Stage 2 (that is, menu opening stage), after the user has finished operand specification. In exactly the same manner as Fig. 2a, the user drags the pen-tip towards the desired menu-item and lifts the pen-tip. As we can notice, the menu items in the linear menu of Fig. 4 are the same as the pie menu of Figs. 2a and 2b.

Thus, in this user interface, operations with one or two operands can be performed in both *one stroke* or *multiple strokes*, by using pie menu or linear menu.

Regarding pen-input systems, our research is the first to put forward the concept of integration and separation of operand specification, menu opening and selection, and action execution in *one or more strokes.*

The next two sections explain the reasons that enabled *one stroke operations.* Section 2.3 explains the three elements, and Section 2.4 the integration and separation of the operation stages.

2.3 The Three Elements: The First Reason that Enabled *One Stroke Operations*

This section describes the first of the two reasons that enabled *one stroke operations.* Sections 2.3.1, 2.3.2 and 2.3.3 respectively describe the three elements, namely, *ink-states*, pen and pie menu and linear menu, and *data-cum-tools.*

2.3.1 *Ink-states*

Here, we explain the first of the three elements that enabled *one stroke operations. Ink-states* denote different types of "operational units" of ink-data, where an "operational unit" of ink-data is that portion of ink-data on which the user performs operations as if this portion constitutes one block or unit of ink-data. The *ink-states* have been shown in Figs. 1 and 2.

Till now, in the user interfaces of most pen-based ink-editors, either ink-data was present (written) or absent (not yet written or deleted). Thus a concept of *ink-state* did not seem essential. However, to explain newer operational units of ink-data such as the one enveloped by the *frame* (Fig. 2b), and to explain the transformations between such operational units, the concept of *ink-state* became necessary.

The *ink-states* can be divided into *pure ink-states* and *compound ink-states.* The *pure ink-states* are shown in **Fig. 5** (explained a little later). The *compound ink-states* are formed by rubber-banding a mixture of *pure ink-states*, for example, rubber-banding the compressed ink-data and non-compressed ink-data of Fig. 2. Though basically the operations on both the *pure* and *compound ink-states* are similar, however, various combinations of the *pure ink-states* exist and different operations on these compound *ink-states* yield various ink-states. Since discussing all these combinations and results is outside the scope of this paper, this paper focusses only on *pure ink-states.*

Let us focus on the state chart of Fig. 5, where each circle along with the shaded label represents one *pure ink-state.* The contents inside each circle denote the operand on which operations (such as "Compress" or "Segment") are performed to transform them to another *ink-state.* For example, the ink-data inside the circle of *raw ink-state* is the top portion of Fig. 1. The arrows in Fig. 5 denote the *ink-state* transformations, and "Man. or Auto." means that the *ink-states* can be transformed manually (as in Figs. 2a and 2b) or automatically (described in Section 3.1.3).

Now, let us traverse Fig. 5 from the left to the right. The *raw ink-state* in Fig. 5 denotes the ink-data formed after the user finishes writing one stroke **and** no *ink-state* transformation has taken place.

Now moving right in Fig. 5, when "Segment"

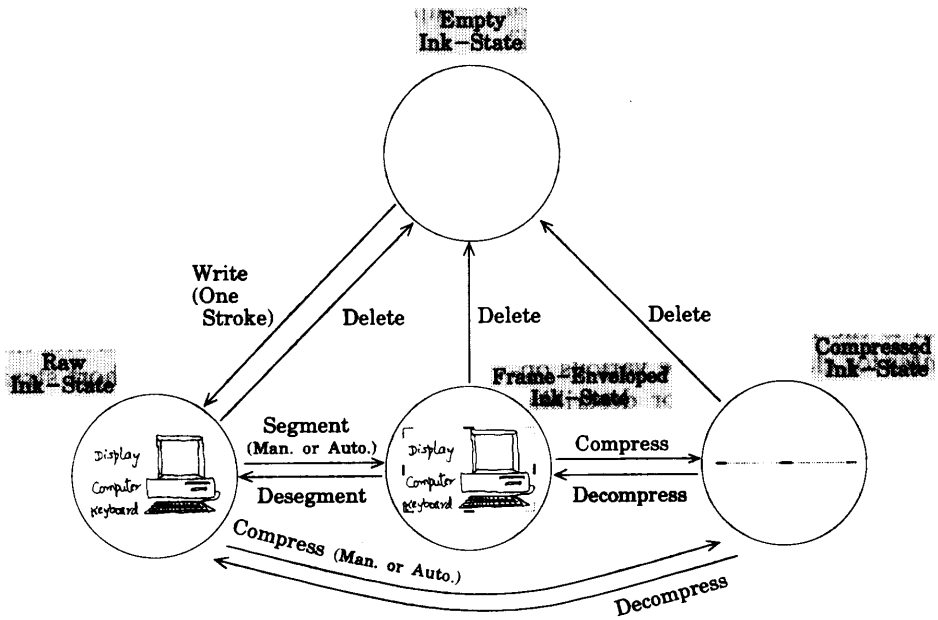

Fig. 5 State chart where each circle (along with the shaded label) denotes one *pure ink-state*. The arrows and the associated labels (such as "Compress" or "Segment") represent *ink-state* transformation operations.

operation is performed on *raw ink-state*, *frame-enveloped ink-state* is created. This *ink-state* is shown in the bottom-left of Fig. 1 and in Stages 1 to 3 of Fig. 2b. The "Segment" operation envelopes portions of ink-data by a *frame* (denoting spatial bounds), hence the name *frame-enveloped ink-state*. All the ink-data inside the *frame* becomes one operational unit of *frame-enveloped ink-state*. These spatial bounds are preserved along with the ink-data. We created this ink-state so that the user can specify all the ink-data inside an operational unit of this *ink-state*, simply by touching the *frame* with the pen-tip (Stage 1 of Fig. 2b).

Moving further right in Fig. 5, when "Compress" operation is performed on *frame-enveloped ink-state* (Fig. 2b) or on *raw ink-state* (Fig. 2a), *compressed ink-state* is formed, and is displayed as a one-pixel thick line, giving the impression that ink-data is compressed or flattened. This one-pixel line functions as an operational unit of ink-data, and also acts as the *frame* which envelopes compressed ink-data, enabling operand specification by simply touching the *frame* with the pen-tip.

In Fig. 5, the operations "Desegment" and "Decompress" respectively revert the *frame-enveloped ink-state* and *compressed ink-state* back to their former *ink-states*.

Lastly, we come to *empty ink-state* of Fig. 5, which exists by default immediately after the user boots up the ink-editor. The state where a stroke has not yet been written is *empty ink-state*. In Fig. 5, the arrow "Write (one stroke)" denotes that the user is in the process of writing a stroke, that is, the pen-tip is in contact with the tablet surface. The instant the user lifts the pen-tip, that is, signals the end of this stroke, a new stroke (of *raw ink-state*) is created and the transformation from *empty ink-state* to *raw ink-state* is completed. Similarly, the arrow from *raw ink-state* to *empty ink-state* labeled "Delete" means that the user is deleting a stroke. When the pen-tip is lifted up after deleting, *raw ink-state* is transformed into *empty ink-state*. Similarly, the other arrows can be explained.

From the viewpoint of enabling *one stroke operations*, the *ink-states* described thus far can be divided into "mandatory" *ink-states* and "optional" (though useful) *ink-states*. There are three "mandatory" *ink-states*— *empty ink-state*, *raw ink-state* and *frame-enveloped ink-state*, and one "optional" *ink-state*— *compressed ink-state*.

The *empty ink-state* and *raw ink-state* taken together are mandatory since they represent the most basic tasks of writing and erasing. The *frame-enveloped ink-state* is mandatory since it enables operand specification with a simple

pen-down on the *frame*. Regarding *compressed ink-state*, though this *ink-state* is "optional" from the viewpoint of enabling *one stroke operations*, it is nevertheless useful because, by creating operational units of *compressed ink-state*, users can efficiently use the narrow screen space of pen-computers.

The *raw ink-state* and *empty ink-state* are the only two *ink-states* that exist in the user interfaces of most ink-editors.

Our research is the first to propose the concept of *ink-states* for classifying the various operational units of ink-data and for describing the transformations among them.

Regarding the creation of *ink-states*, both the *frame-enveloped ink-state* and *compressed ink-state* can be created either manually (as in Figs. 2a and 2b) or automatically (described later in Section 3.1.3).

Regarding the *frame-enveloped ink-state*, though we first put forward the approach of segmenting ink-data[13),14)], the product "ZAURUS" enables operations in units of segmented ink-data by using gestures. However in "ZAURUS" the spatial bounds of segmented ink-data cannot be modified, and the individual strokes inside segmented ink-data cannot be edited[25)]. Whereas, in the *frame-enveloped ink-state* of this paper, first, the bounds can be modified. Second, the individual strokes inside the *frame-enveloped ink-state* can be edited without needing to first desegment the *frame-enveloped ink-state*, for example, tapping the strokes inside this *ink-state* (after selecting "Eraser") erases the strokes, and so on.

2.3.2 Pen, Pie Menu and Linear Menu

Here, we explain the second of the three elements that enabled *one stroke operations*. As shown in Figs. 2a and 2b, to enable *one stroke operations*, after the operand is selected, the operator choices should appear exactly at pen-tip location. Soon after operator selection, these operator choices should disappear from view, and let the user complete action execution in the same *one stroke*.

Two example candidates that could possibly satisfy the above requirements of operator choices are pop-up linear menu and pie menu. Both of these can be displayed at exactly the location of pen-tip, and the desired menu item can be selected.

The comparison of pie menu and linear menu with respect to performing *one stroke operations* and *two/three stroke* operations is taken up in Section 5.

From the viewpoint of memorizing the menu item locations, we expect that the pie menu would be relatively better than linear menu because the radial arrangement of pie menu items enables users in memorizing their locations, making possible menu selection by "marking"[18),19)] (that is, writing a straight mark in the direction of the menu item without displaying the pie menu). "Marking" is like writing gestures, with the merits that these gestures are faster to write (short straight marks), and easier to memorize since users learn marks by actually using pie menus. "Marking" speeds up overall operations as it cuts the display time of pie menu, and additionally does not obscure ink-data.

This paper is the first to use pie menu to perform operations on data created with pen-input in the same application. Now let us look at the other pie menu research.

The research of Ref. 17) empirically compares linear menu and pie menu, however no operations are performed on data using pie menu. The research of Ref. 18) investigates how many levels and items in hierarchial pie menu are practical with "marking", however no operations are performed on data using pie menu. Though the research of Ref. 19) uses pie menu to operate on data, however this data is not created with pen-input. The research of Ref. 3) uses pie menu to create (coded) graphics data (each pie menu item denotes a geometrical shape), however pie menu is not used to operate on this data. The research of Ref. 20) creates (coded) text characters (symbols, etc.) using pie menu, but no operations are performed on data using pie menu.

2.3.3 *Data-cum-tools*

Here, we explain the last of the three elements that enabled *one stroke operations*, namely, *data-cum-tools*. In this user interface, there are three *data-cum-tools*. The first *data-cum-tool* is the rubber-band along with the ink-data contained within it. The second *data-cum-tool* is the *frame* (of *frame-enveloped ink-state*) along with the ink-data contained within it. The third data-cum-tool is the *frame* (of *compressed ink-state*) along with the compressed ink-data contained within it. All these three *data-cum-tools* possess properties of both data and tools. These three have properties of data, since when the user manipulates the *frame* (Fig. 2b), rubber-band (Fig. 2a) or the *frame*

of *compressed ink-state*, the operand (ink-data) inside gets operated on. And all the three *data-cum-tools* have properties of tools since they contain embedded tools (pie menu or linear menu, and the operators inside pie menu or linear menu) which pop up immediately after operand specification.

We created *data-cum-tools* so that the user need not change modes between writing (and erasing) and performing operations. These modeless operations are realized because a pen-down on the *data-cum-tool* denotes the start of an operation, and a pen-down on any other area of the screen denotes writing, erasing, etc. depending on the mode in use at that time, for example, writing mode, erasing mode, etc. Thus, the strokes need not be recognized to distinguish between writing/erasing/etc. and start of operation, and users need not worry about misrecognitions during writing. The modeless feature of *data-cum-tools* greatly assists developers as it relieves them from having to develop an ink recognition algorithm.

The data-cum-tools of this user interface can be divided into "action-deferred" *data-cum-tool*, and, "on-the-fly" *data-cum-tool*. The *frame* of *frame-enveloped ink-state* and the *frame* of *compressed ink-state* are "action-deferred" *data-cum-tools*, since users can create them and defer action on them till later on. Whereas the rubber-band is the "on-the-fly" *data-cum-tool* since the rubber-band is formed and immediately operated on.

Though the researches in Refs. 22) and 23) describe the simultaneous specification of the operand and operation by using the "See-Through Tools"[22] and "Magic Lenses"[23], they do not talk about the concept of an object possessing properties of both data and tools. But, at the end of the presentation of Ref. 23) at CHI '94 conference (Boston, April 1994), the discussant talked about the concept of integrating data and tools (which is incidentally the concept behind our *data-cum-tools*). However, this paper is the first to actually implement this concept.

Now, the tools of Refs. 22) and 23) can be used with one hand or two hands. When used with one hand, the operations are performed in two distinct steps—in the first step the tools are positioned over the operand, and in the second step the operation is performed. When two hands are used, the tools are positioned over the operand with one hand, while the operation is performed with the other hand. That is, the tools and the data are originally separated.

In contrast, in the *data-cum-tools* of this paper, since the data and tools are integrated, one step operations are possible at all times.

2.4 Integration and Separation of the Operation Stages: The Second Reason that Enabled *One Stroke Operations*

Here, we explain the second of the two reasons that enabled *one stroke operations*. "Integration of operation stages" in *one stroke* means that, the user transfers from one operation stage to the next, while the pen-tip is in contact with the tablet surface.

When the operation stages are performed in separate pen-down (and pen-up) motions, as in most of the current pen-based user interfaces, then the operation stages are said to be "separated".

As explained in Section 2.2, in this user interface, the user is "free" to integrate all the operation stages in *one stroke*, or to separate the operation stages, that is, "free" in the sense that he/she can lift the pen-tip after completing any operation stage.

This research is the first to put forward the concept of integration and separation of the operation stages, and to actually implement this concept in a pen-based user interface.

Now, in Fig. 2a, we call the integration of Stage 1 and Stage 2 as "automatic integration" because, immediately after the operand is specified, the pie menu opened automatically without further prompting. Whereas Stage 2 and Stage 3 of Fig. 2a are "manually integrated", since the user must perform the drag action to select the menu.

Thus, from Sections 2.3 and 2.4, we can see that *one stroke operations* were enabled because of the existence of the three elements (*ink-states*, pen and pie menu and linear menu, and, *data-cum-tools*) and the "integration and separation of the operation stages" in *one or more strokes*.

3. Design of the Proposed User Interface

As an application of the concepts of Section 2, here we explain the design of the proposed user interface. We aim to show the design principles behind the design of each operation stage. We describe the ink-state transformation operations in Section 3.1 and editing operations in

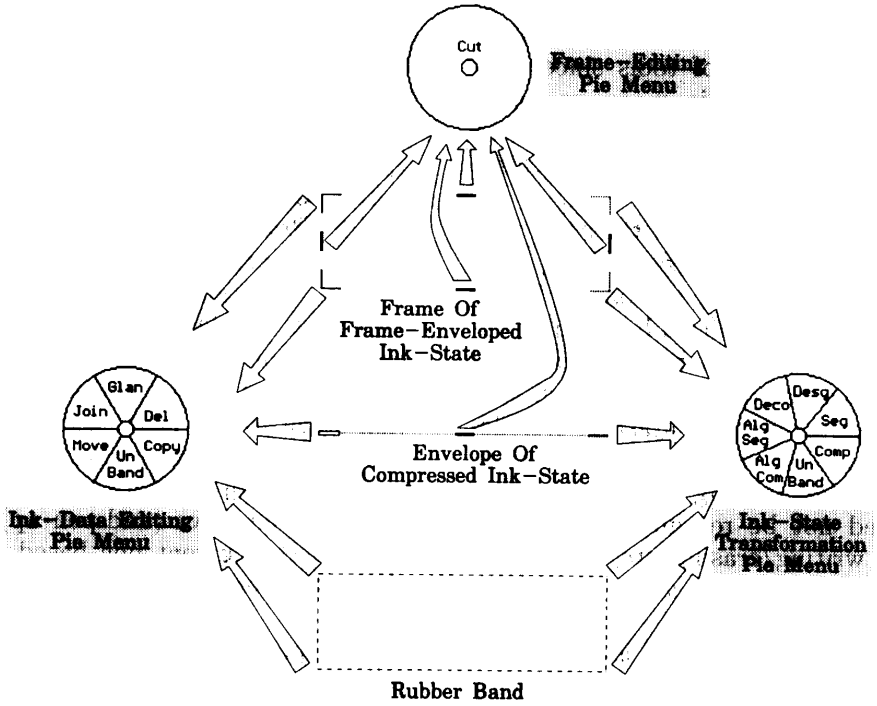

Fig. 6 Shows the three pie menus and the *data-cum-tools* of this paper. The shaded arrows indicate which pie menu is displayed when the tip of stylus pen comes in contact with the respective portion of the *data-cum-tool.*

Section 3.2. Though Section 2 described the concept of *one stroke operations* using both pie menu and linear menu, in this chapter, we describe the design of this user interface using only pie menu.

3.1 Design of the Ink-state Transformation Operations

Section 3.1.1 explains the design of the integration and separation of the operation stages. Next we describe the design of each operation stage in Sections 3.1.2 to 3.1.4, by starting with the design of pie menu opening.

3.1.1 Design of the Integration and Separation of the Operation Stages

Figures 2a and 2b showed one design method in which all the operation stages were integrated in *one stroke.* Other possible design methods are, for example, specify the operand in the *first stroke* and integrate the other operation stages in the *second stroke;* specify the operand and open the menu in the *first stroke* and integrate the other operation stages in the *second stroke;* and so on. It remains a future research issue for us to determine which design of the integration or separation of the operation stages leads to faster operations and fewer errors.

Though the research of Ref. 3) explains about deferred operations in which the user specifies the operand and the operation in two steps, however Ref. 3) does not mention about other separations of the operation stages.

3.1.2 Design of Pie Menu Opening

Let us focus on **Fig. 6** which shows the three pie menus of this user interface. The shaded arrows indicate which pie menu opens when the pen-tip comes in contact with which part of the *data-cum-tool.* This user interface has three pie menus instead of only one, so as to enable *one stroke operations* when using rubber-band, as explained below.

Immediately after specifying the operand with rubber-band, the pie menu should be displayed. If all the operators (corresponding to all the *ink-state* transformation and editing operations) are crowded into one pie menu, spotting the menu items will take time and errors will rise during menu selection. Possible ways of reducing the pie menu items are, using "hierarchial pie menus"[18], requiring users to make distinct pen movements to indicate which pie menu should be opened, dividing the pie menus based on functionality, etc. We adopted the ap-

proach of dividing the pie menus based on functionality.

As shown in Fig. 6, when the user rubber-bands from the left side of the tablet to the right, the *ink-state transformation pie menu* opens, using which the user selects the pie menu item and executes the action in the same *stroke*. And when he/she rubber-bands from the right to the left, the *ink-data editing pie menu* opens. Thus, *one stroke operations* are enabled with rubber-band.

Now let us consider "undo" and error recovery during pie menu opening. After the pie menu opens, the pie menu can be closed by lifting the pen-tip from the tablet-surface while the (x, y) coordinates of the pen-tip lie inside the inner circle of pie menu. In the event the user has already moved the pen-tip to one of the menu items, he/she can simply bring the pen-tip back into the inner circle and lift the pen-tip.

Now let us consider that the user opened the wrong pie menu by rubber-banding from quite the opposite side of rubber-band than he/she must have. For example, the user wanted to open the *ink-data editing pie menu*, however he/she wrongly rubber-banded from the left side of the tablet to the right and opened the *ink-state transformation pie menu*. Out of the various error-recovery design options possible, in this user interface, the following two methods exist.

(1) The user lifts the pen-tip, which closes the wrongly opened pie menu. Then, he/she simply "taps" with the pen-tip on the same corner of the rubber-band where he/she opened the wrong pie menu. This pen-tap acts as a toggle switch which opens the next pie menu. If this pie menu is not the desired pie menu, the user taps again, and so on, until he/she opens the desired pie menu. For example, if the wrong pie menu opened after rubber-banding is the *ink-state transformation pie menu*, the "tap" opens the *ink-data editing pie menu*. And another tap opens the *ink-state transformation pie menu*. This method of tapping at the same corner eliminates the need to move the pen-tip to the other rubber-band corners, and thus reduces pen movements.

(2) He/she lifts up the pen, which closes the opened pie menu. Then he/she moves the pen-tip to the horizontally opposite corner, and puts down the pen-tip, thus opening the desired pie menu.

To make the pie menu division visually distinguishable, the left, center and right portions of the *frame* of *frame-enveloped ink-state* and *compressed ink-state* are displayed as lines of various styles (solid, bold, etc.). However the top and bottom portions of the *frame* (of *frame-enveloped ink-state*) are identical, to maintain consistency between and left (or right) portions of the *frame* of *frame-enveloped ink-state* and *compressed ink-state*, and the rubber-band.

In order that the user be able to open the pie menu by pointing at the *data-cum-tool* in the first try, we did the following. When the pen-tip is directly above or very close to a portion of the *frame* of *frame-enveloped ink-state* or *compressed ink-state*, this *frame* portion temporarily widens (a dotted rectangle appears around the respective portion of the *frame* of *frame-enveloped ink-state* or *compressed ink-state*) to show current pen-tip location. Thus, even if the location of pen-down does not exactly coincide with the location of *frame*, but lies inside this widened *frame*, the *frame* is said to be selected.

3.1.3 Design of Operand Specification

We discuss manual operand specification in (1) and automatic operand specification in (2).

(1) Manual operand specification

Let us consider "rubber-banding". Stage 1 of Fig. 2a showed the rubber-band being constructed between the locations ① and ②. The starting point and ending point of rubber-band could be denoted in various ways, such as, keeping the pen-tip still for a pre-determined time at ① and ②, varying the pressure of the pen-tip on the tablet surface at ① and ②, etc. Focussing on Stage 1 of Fig. 2a, the start of rubber-band is denoted by pressing the side-switch of stylus pen at ①, and the end is denoted by releasing the side-switch at ②. In this user interface, the start and end of rubber-band are denoted in this way, in order to differentiate between the pen-down of writing ink-data and the pen-down of constructing a rubber-band. In this method, the user could mistakenly press or release the side-switch. Thus as an alternative we implemented another method of denoting the end of rubber-band. Specifically, on reaching the location ②, the user keeps the pen-tip still for about 0.3 seconds, and the end of rubber-band is specified.

Now, during rubber-banding, only those strokes that lie completely inside the rubber-band are included as the operand.

If the user mistakenly releases the side-switch at an undesirable part of ink-data, he/she can "undo" (that is, remove) the rubber-band by selecting "Un Band" (abbreviation of "Undo Rubber Band") from the pie menu that opens just after operand specification.

(2) Automatic operand specification

"Automatic" means that the spatial bounds are created around ink-data, without needing to rubber-band every time. The automatic mode is activated by tapping the "Auto Off" button of "Global Menu". "Automatic compression" means that the ink-data gets compressed in addition to being enveloped by spatial bounds.

For automatic operand specification, this user interface uses a slightly modified version of the segmentation algorithm of Refs. 13), 14), that segments handwritten text lines from the original ink-data. This algorithm looks for "break" in the pen movements while writing. "Break" represents a "big change" in the values of the pen-input parameters (such as the x and y coordinates of the pen-tip position, pressure at pen-tip position, etc.). And "big change" is measured relative to pre-determined threshold values, and depends on the type of "break". Some examples of "break" are, a "break in line while writing" is said to occur when the user ends writing on one line and is about to write on another line, "break while writing and drawing" occurs when the user just finishes writing a stroke of a character and is about to write the stroke of a sketch, etc. (For more details about this algorithm, kindly refer to Refs. 13), 14).) This paper detects only "break in line while writing".

Mistakenly segmented ink-data are corrected using *frame-editing pie menu* shown in Fig. 6, and is described later in Section 3.2.2.

Users can automatically create *frame-enveloped ink-state* or *compressed ink-state* while they are writing (that is, in real-time) or at an arbitrary time (that is, by batch processing). These two modes are toggled by tapping the "Real Batch" button of "Global Menu". The batch processing mode is useful when users want to eliminate mistaken segmentation of handwritten figures, tables, etc. by the segmentation algorithm. Specifically, after manually segmenting the figures, etc. he/she uses the segmentation algorithm to segment the remaining handwritten text.

To automatically segment only a portion of the document, the appropriate ink-data is rubber-banded and "Alg Seg" (abbreviation of "Algorithm Segmentation") is selected from the *ink-state transformation pie menu*. To automatically compress only a portion of the document, "Alg Com" (abbreviation of "Algorithm Compression") is selected.

To revert (or undo) the *frame-enveloped ink-state* and *compressed ink-state* to their older *ink-states*, "Desg" (abbreviation of "Desegment") and "Deco" (abbreviation of "Decompress") are selected from the pie menu. To speedily desegment all the ink-data in the document, the "Desegment" button of "Global Menu" is tapped.

3.1.4 Design of Pie Menu Selection and Action Execution

During menu selection, if the user mistakenly drags the pen-tip to an undesired menu item, he/she can drag the pen-tip back to the correct menu item, then lift up the pen-tip to complete menu selection or he/she can drag the pen-tip outside the outer circle of pie menu. The method of dragging the pen-tip outside the outer circle of pie menu was provided to enable "move" ("copy", etc.) operations without needing to first lift up the pen-tip (already illustrated in Fig. 3).

Regarding pie menu selection and action execution, some of the issues to be tackled are, determining the optimal size (radius of inner and outer circle) of pie menu, optimal number of items inside pie menu, optimal placement of the menu items, etc.

3.2 Design of the Editing Operations

Section 3.1 explained the design of each design stage. Here we describe only the design issues related to the editing operations. Sections 3.2.1 and 3.2.2 respectively describe the design of the ink-data editing operations and the frame-editing operations.

3.2.1 Design of the Ink-data Editing Operations

The most basic ink-data editing operations are writing and erasing. Writing is done while the "Pen" stationery tool in "Global Menu" is highlighted. To erase ink-data, the user toggles between the pen/eraser tools, and taps on the ink-data strokes or drags the pen-tip over the ink-data strokes that are to be erased. The user can toggle between the pen/eraser tools in two ways — by pressing twice the side-switch of the pen, or by tapping the "Eraser" button of "Global Menu". The former toggling method was provided to reduce pen movements and dis-

traction.

The other ink-data editing operations are "move", "copy", "delete" and "glance" operations. All these operations can also be performed in *one stroke* or *multiple strokes*. Here we consider the "move" and "glance" operations as examples.

In the explanation of the move operation in Section 2.2, it was mentioned that the "move" operation can be performed in two methods—in *one stroke* or in *multiple strokes*. While both these "move" methods have their merits and demerits, performing "move" in *one stroke* is useful while the user is performing other tasks, for example, talking during discussions. In discussions, if the *multiple stroke* "move" operation is adopted, then the user is distracted from his/her main task of discussion, because he/she has to move his/her eyes and/or hand back to the operand to position the pen-tip, and then drag the operand.

An important thing to note is that, after the "move" (or "copy" or "delete") operation is completed, the other ink-data (not specified as operands of these operations) do not automatically move to occupy the space created as a result of "move" (or "copy" or "delete") operation. This is because the ink-data in this user interface does not exist in any pre-determined structure (such as the sequential structure in "ZAURUS"), because of which the user will not expect the unrelated ink-data to automatically move.

The "glance" operation is used to quickly browse *compressed ink-state*. The user puts the pen-tip down on the left edge of *compressed ink-state* and selects "Glan" (abbreviation of "Glance") in the *ink-data editing pie menu*. On doing so, about one-third the width of ink-data inside this *ink-state* comes into view. Only a portion of this ink-state is displayed, since the user can recollect the contents of this *ink-state* just by looking at this limited portion.

By creating *compressed ink-state* and browsing it with "glance", users can view both the local details as well as global context of the whole document. Though other research targeting the viewing of details and global context (such as Refs. 15), 16)) have their own merits and demerits, our approach of "compressing" ink-data and "glancing" is especially useful when writing on the narrow screens of pen-computers.

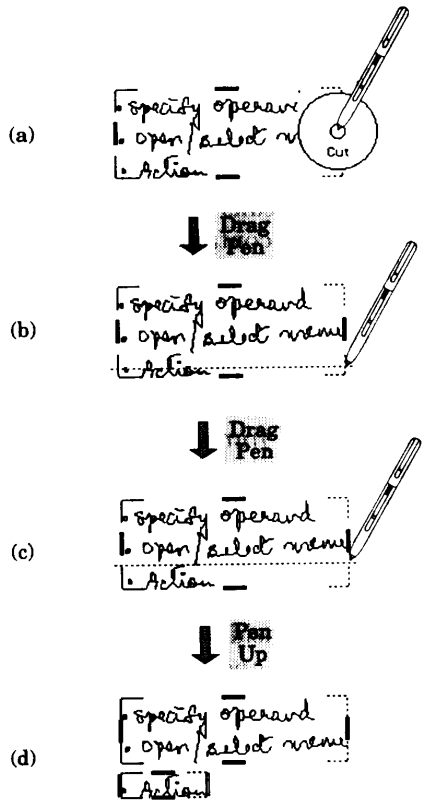

Fig. 7 "Cut" operation for breaking up one unit of the *frame-enveloped ink-state* into two, performed in *one stroke*. (a) to (d) respectively show pie menu opening, the instant just after menu selection, adjusting the "cut" position, and the ultimate formation of two units of *frame-enveloped ink-state*.

3.2.2 Design of the Frame-editing Operations

The frame-editing operations are "cut" and "join", and they modify the *frames* of *frame-enveloped ink-state*, for example, when the segmentation algorithm mistakenly segments or when users want to break up or join together the operational units of this *ink-state*. Here, the "cut" operation is explained using **Fig. 7**.

Figure 7 shows "cut" operation performed in one stroke for cutting the *frame-enveloped ink-state* into two. Fig. 7 (a) shows the instance the pie menu opens when the user puts the pen-tip down on the *data-cum-tool* at the center of the right-edge of *data-cum-tool*. He/she drags the pen-tip towards the outside of the pie menu (here, in the downward direction). The moment the pen-tip goes outside the pie menu, a horizontal dotted line appears, as shown in Fig. 7 (b). By continuing to drag the pen-tip in the up-down direction, he/she positions this dotted line at an appropriate position between

two handwritten lines of text (Fig. 7 (c)), and then lifts the pen-tip. On doing so, the original *ink-state* is divided into two as shown in Fig. 7 (d).

Figure 7 shows an example of "horizontal cut". In this user interface, there is also "vertical cut", which breaks up (for example) a handwritten text line (containing handwritten characters) into left and right portions. Regarding *compressed ink-state*, after the user opens the *frame-editing pie menu* and selects "Cut", this *ink-state* remains unchanged.

The "join" operation joins two or more operational units of *frame-enveloped ink-state*, or joins *raw ink-state* with *frame-enveloped ink-state*. To "join", the user rubber-bands the necessary ink-data and selects "Join" from the *ink-data editing pie menu*. Since the "join" operation is a type of editing operation, we placed the "Join" operator inside the *ink-data editing pie menu*. The "Join" operator has not been placed inside the *frame-editing pie menu* along with "cut" because, the "join" operation is not performed from the positions at which the *frame-editing pie menu* opens.

4. Extending the Concepts of this User Interface to Other User Interfaces

The concepts of this paper, namely, "integration and separation of the operation stages" in *one or more strokes*, *ink-states*, pen and pie menu and linear menu, and *data-cum-tools*, can be extended to other user interfaces that use direct pointing devices (such as mouse or hand/finger as in touchscreen). We describe this extension of our concepts, by taking up editing of text-data and graphics-data in (1) and (2).

(1) Using our concepts for editing text-data in other user interfaces

Let us suppose that a user wants to delete a paragraph (say, of a research paper) using a mouse-based Windows™ word processor. To specify this paragraph as the operand, the user would press the mouse button and drag the mouse from the start of the paragraph to its end, and release the mouse button. At this instant, a pie menu could be displayed, and the "Delete" pie menu item selected. To speed up operand specification, the user could create *frame-enveloped text-state* (such as individual lines or paragraphs) surrounded by a frame, so that such operands can be selected simply by clicking the mouse button when the mouse cursor is on the *frame*.

We can similarly explain *one stroke operations* for pen-based text-editors, the design issue being the detection of the start/end of the operand, for example, by toggling the side-switch of the pen.

And, to perform the operation of the above example on a touchscreen, the user would drag his/her finger or hand from the start to the end of the paragraph, then select the pie menu item. Here also the design issue is the detection of the start and end of operand specification, for example, keeping the finger at the same position for a pre-determined time, etc.

Now, the concept of *ink-states* (or more generally, *data-states*) can be extended to all the above three examples. In addition to *frame-enveloped text-state* explained above, *empty text-state* exists just after boot up, *raw text-state* is formed on inputting a character from the keyboard (for mouse-based user interfaces) or recognizing and converting a handwritten character into text-data (for pen-based or touchscreen-based text-editors).

(2) Using our concepts for editing graphics-data in other user interfaces

In a mouse-based graphics application, after rubber-banding the desired graphics-data, a pie menu could pop up and the menu item selected. To quicken operand specification, *frame-enveloped graphics-state* could be created and operated on by clicking the mouse button on the *frame*.

In similar manner, the above graphics editing operation can be performed in *one stroke* in pen-based graphics-editors and touchscreen-based graphics-editors.

And, in addition to *frame-enveloped graphics-state* explained above, we could have *empty graphics-state* and *raw graphics-state* in the mouse-based, pen-based and touchscreen-based graphics-editors.

5. Experiment

5.1 Aim of the Experiment

This section describes the preliminary experiment performed to compare the following two:

(1) Operations with *one stroke*, *two strokes*, and *three strokes*

(2) linear menu, and, pie menu

Here, "operations with *one stroke*" are those in which the user completes the whole operation (that is, from the start of operand specifi-

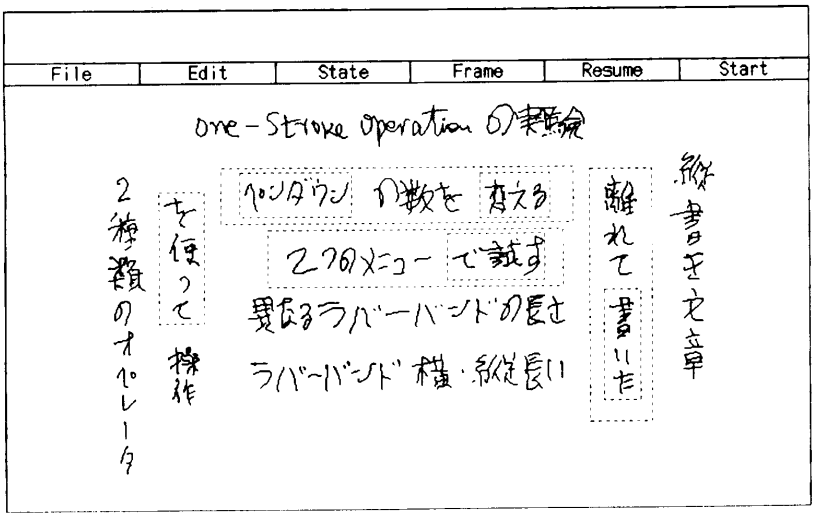

Fig. 8 Hard-copy of the screen of the experiment tool, showing the ink-data used. The eight dotted rectangles, used only for explaining, denote the eight operands t hat the subjects operated on.

cation to the end of execution) without lifting the pen-tip. "Operations with *two strokes*" are those operations in which the user lifts the pen-tip only once during the operation, that is, the user completes the operation in two pen-downs. For example, in the first pen-down, he/she puts the pen-tip down on the tablet-surface, rubber-bands the operand and lifts up the pen-tip. Then, in the second pen-down he/she puts the pen-tip down on the tablet-surface, opens the menu, selects the menu and executes the action. And in the "operations with *three strokes*", he/she completes the operation in three pen-downs. Specifically, in the first pen-down the user completes rubber-banding, and lifts up the pen-tip. In the second pen-down, he/she opens the menu, selects the menu-item, and lifts the pen-tip. In the third pen-down, he/she executes the action.

We compared operations with *one stroke* and operations with *two/three strokes*, in order to determine whether operations with *one stroke* are faster than and produce fewer errors as compared to operations with *two/three strokes*.

Now, referring to (2) above, we aimed to determine whether pie menu is faster than and produces less errors as compared to linear menu.

5.2 Experimental Method

The experimental method is explained using **Fig. 8** which is a hardcopy of the screen of the experiment tool. The dotted rectangles in Fig. 8 (denoting the operands to be operated on) are used only for explaining, and are not actually displayed during the experiment.

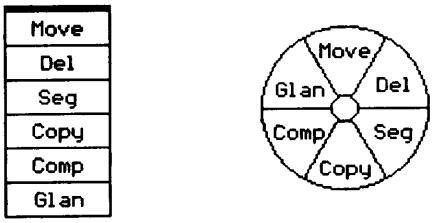

Fig. 9 The linear menu and pie menu actually used in the experiment. Each menu contains six menu items, including the "one-operand type" operators ("Del", "Seg", "Comp", "Glan") and "two-operand type" operators ("Move", "Copy").

Each subject performs a fixed number of operations, each operation performed according to the following four steps.

① Each operation is triggered by tapping the "Start" button (upper-right portion of the screen) with the pen. On doing which, one operand out of eight possible operands, is randomly displayed on the screen as bold strokes, that is, the ink-data contained in one of the eight dotted rectangles of Fig. 8 is displayed on the screen as bold strokes, and the operator (for example, "Delete") is shown on the top of the screen.

② The subject envelopes the specified operand by creating a rubber-band around this operand.

③ The subject opens the menu (linear menu or pie menu of **Fig. 9**) and selects the menu-item corresponding to the operator displayed on the top portion of the screen. This being a

preliminary experiment, we used only six menu-item linear menu and six-item pie menu.

④ The subject executes the action, for example, moving the operand from its initial position to the final destination. Here, the final destination is displayed as a bold rectangle on a random position on the screen. The random positions are explained a bit later.

There were four subjects, and each subject performed 480 operations in all. The number "480 operations" is obtained by multiplying the following four quantities:

(i) The **3** "operational methods", that is, operations with *one/two/three strokes.*
(ii) The **2** menu shapes, that is, linear menu and pie menu.
(iii) The **4** menu items in each of pie menu and linear menu.
(iv) The **24** operations with each of the four menu-items mentioned above.

Though multiplying the above four quantities yields a total of 576 operations, however, out of this, 96 operations can be deleted as explained below.

In this experiment, we used "one operand type" operations (in which users need to simply select from the menu, as already explained in Section 2.2) and "two operand type" operations (in which users need to specify both the original and final positions of the operands). Now, the operations with *three strokes* were not performed with "one operand type" operations, because the "one operand type" operations require maximum only *two strokes* to complete the whole operation. Therefore, 96 operations (that is, 24 times each of the four menu items) can be deleted from the total of 576 operations. Therefore, we obtain 480 operations in all.

These 480 operations were spread over six sessions. These six sessions were divided on the basis of operational method types (that is, *one/two/three stroke* operations) and menu-shapes (linear/pie menu). For example, in a particular session, the subject performed the specified 96 operations by using the method of operations with *one stroke* and linear menu. While in another session, the same subject performed the same 96 operations using the method of operations with *one stroke* and pie menu, and so on. The order of the six sessions was different for the four subjects. Now, the subjects were allowed to rest after completing each session, and they were allowed to practice for a few minutes before each session.

All the four subjects performed the 480 operations on the same operands and operators, however, the order of the operands and the operators was random, in order to make it difficult for the subjects to guess. Regarding the operands, two types of operands were selected — (i) horizontally written text, (ii) vertically written text.

Regarding the "two operand type" operations, the location of the destination of "move" and "copy" was generated at random, out of a possible 16 locations on the screen, that is, eight directions (up, down, left, right, and the 4 directions in between them) and two distances. These directions as well as distances are measured with respect to the bottom-right corner of the rubber-band, since the subjects rubber-banded from the top-left corner to bottom-right corner of the rubber-band. Here, two distances — "far" (near the screen edge in each direction) and "near" (roughly 1/3 of the "far" distance) were set up.

5.3 Experimental Results

Figure 10 shows the experimental results. The horizontal axis shows the six combinations of the operational method types and menu-shapes, and the vertical axis shows the average time taken to perform each operation. Here the time taken is measured from the start of rubber-banding to the end of execution. The start of rubber-banding is the instant the pen-tip touches the tablet surface. The end of execution is slightly different for "one operand type" and "two operand type" operations. For "one operand type" operations, the end of execution is the instant the pen-tip is lifted from the tablet surface just after selecting the menu item, or the instant the pen-tip goes out of the outer circle of pie menu. And the end of execution for "two operand type" operations is the instant the pen-tip is lifted from the tablet surface after dragging the operand to its destination.

As shown in Fig. 10, from the viewpoint of average time taken, operations with *one stroke* are relatively faster than both operations with *two strokes* and operations with *three strokes.* Also, operations using pie menu are relatively faster than using linear menu for all the three operational methods (operations with *one/two/three strokes*).

Statistical analyses of variance, however, did not yield significant differences ($F_{1,4} = 0.58$, $p < 0.05$) between the three operational meth-

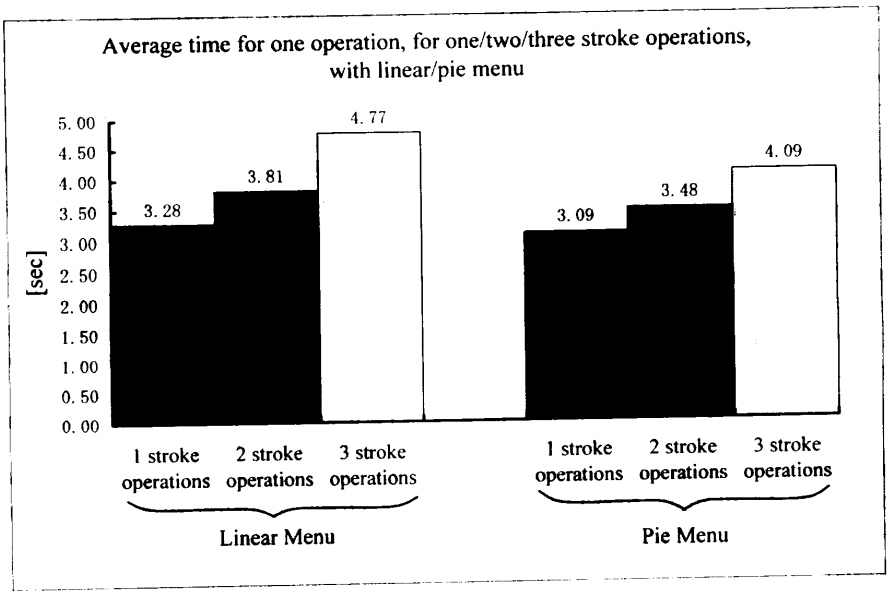

Fig. 10 Graph showing the average time taken to complete one operation, where each operation is performed with one/two/three strokes and two menu shapes (linear menu and pie menu).

ods (operations with *one/two/three strokes*) for both the linear menu and pie menu. Also, no statistically significant differences ($F_{1,6} = 0.65$, $p < 0.05$) were found between linear menu and pie menu for all the three operational methods (operations with *one/two/three strokes*).

Possible reasons for not observing significant differences between the three operational methods and between the two menu shapes could be that, the subjects were too few in number or the operations were few in number.

We need to further determine whether there are statistically significant differences between the three operational method types (operations in *one/two/three strokes*), that is, whether the integration/separation of operation stages produce statistically significant differences, and if so, then determine the reasons for these differences and determine which design of integration or separation of the operation stages is best or worst. Further, we need to determine whether there are statistically significant differences between the two menu shapes (linear menu and pie menu). In order to do all the above, we need to perform further experiments by increasing the number of subjects and/or operations.

6. Conclusion

This paper proposed *one stroke operations*, in which operand specification, menu opening and selection, and action execution, can be integrated and separated in *one or more strokes*. The following two reasons enabled *one stroke operations* — (i) Existence of the three elements, namely, *ink-states*, pen and pie menu and linear menu, and *data-cum-tools*, and, (ii) "Integration and separation of the operation stages" in *one or more strokes*.

We showed that the concepts described in (i) and (ii) above resolve the problems of multiple pen-down and pen-up in menu-based pen user interfaces, and the problems of gesture recognition in gesture-based pen user interfaces.

We performed a preliminary experiment to compare operations with *one stroke* and operations with *two/three strokes*, and to compare linear menu with pie menu. Comparison of the average time taken for each operation showed that the operations with *one stroke* were relatively faster than operations with *two strokes* and *three strokes*, and that pie menu was relatively faster than linear menu. However no statistically significant differences were found between the three operational methods (operations with *one/two/three strokes*), and between linear menu and pie menu.

Regarding the originality of this paper, for performing operations on pen-input systems, this paper is the first to put forward the concept of integration and separation of operand specification, menu opening and selection, and action execution, in *one or more strokes*. This paper is also the first to classify the operational units of ink-data into *ink-states*, and also the first to

use pie menu for performing operations on data while creating this data using pen-input in the same application. In addition, this paper is the first to actually realize *data-cum-tools*.

By developing a prototype ink-editor based on the proposed concepts, we showed that these concepts can be designed and implemented in a pen-based application. The operations in this ink-editor are performed in consistent manner, which we expect will reduce the learning time.

We showed that the concepts of *ink-states*, pen and pie menu and linear menu, *data-cum-tools*, and "integration and separation of the operation stages" in *one or more strokes*, can be extended to other user interfaces that use direct pointing devices (such as mouse, hand/finger, etc.) and process various data (text-data, graphics-data, etc.).

As future research work, we need to do the following:

- Perform further experiments by increasing the number of subjects, and compare *one/two/three stroke* operations, and compare linear menu with pie menu, and the menus used in conventional graphical user interfaces.
- Determine which design of the integration and separation of the operation stages is suitable in which task(s).
- Design, implement and evaluate *one stroke operations* in mouse-based and touch-screen based user interfaces.
- Explore how to further enhance *one stroke operations* so as to enable multiple "operations" (where each "operation" includes operand specification, menu opening and selection, and action execution) in *one stroke*.
- Determine in what situations, users want to create which type of *frame-enveloped ink-state*, for example, handwritten text lines, figures, etc.

Acknowledgments The authors would like to thank the reviewers for the constructive comments.

References

1) Hardock, G., Kurtenbach, G. and Buxton, W.: A Marking Based Interface for Collaborative Writing, *Proc. UIST '93*, pp.259–266 (1993).
2) Zhao, R.: Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors, *Proc. INTERCHI '93*, pp.95–100 (1993).
3) Kurtenbach, G. and Buxton, W.: Issues in Combining Marking and Direct Manipulation Techniques, *Proc. UIST '91*, pp.137–144 (1991).
4) Moriya, S., Morita, T., Inai, K. and Shimizu, S.: Stroke Editor, and Direct Pointing and Manipulation (in Japanese), *Trans. IPS Japan*, Vol.32, No.8, pp.1022–1029 (1991).
5) *PenPoint User Interface Design Reference*, Addison-Wesley, (1992).
6) Kamo, O., Sakurai, Y., Sakurai, Y., Shitanda, H. and Sugita, T.: Personal Pen Input Technology (in Japanese), *Human Interface News and Report (32nd Meeting on Human Interface)*, Vol.10, No.1, pp.59–64 (1995).
7) aha! InkWriter Handbook, aha! Software Corporation (1994).
8) Minneman, S.L. and Bly, S.A.: Managing a Trois: A Study of a Multi-user Drawing Tool in Distributed Design Work, *Proc. CHI '91*, pp.217–224 (1991).
9) Ishii, H., Kobayashi, M. and Grudin, J.: Integration of Inter-personal Space and Shared Workspace: ClearBoard Design and Experiments, *Proc. CSCW '92*, pp.33–42 (1992).
10) Wolf, C.G. and Rhyne, J.R.: Communication and Information Retrieval with a Pen-based Meeting Support Tool, *Proc. CSCW '92*, pp.322–329 (1992).
11) Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E., Pier, K., Tang, J. and Welch, B.: Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration, *Proc. CHI '92*, pp.599–607 (1992).
12) Pedersen, E.R, McCall, K., Moran, T.P. and Halasz, F.G.: Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings, *Proc. INTERCHI '93*, pp.391–398 (1993).
13) Moriya, S., Shimizu, S. and Krishnan, S.N.: Classification of Handwritten Stroke Data into Lines (in Japanese), *Trans. IEICE*, Vol.J73-D-II, No.7, pp.973–981 (1990).
14) Krishnan, S.N. and Moriya, S.: Segmentation of Handwritten Text and Editing-symbols from Ink-data, *Proc. HCI Intl. '93*, pp.1010–1015 (1993).
15) Sarkar, M. and Brown, M.H.: Graphical Fish-eye Views of Graphs, *Proc. CHI '92*, pp.83–91 (1992).
16) Sarkar, M., Snibbe, S.S., Tversky, O.J. and Reiss, S.P.: Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens, *Proc. UIST '93*, pp.81–91 (1993).
17) Callahan, J., Hopkins, D., Weiser, M. and Shneiderman, B.: An Empirical Comparison of Pie vs. Linear Menus, *Proc. CHI '88*, pp.95–100 (1988).

18) Kurtenbach, G. and Buxton, W.: The Limits of Expert Performance Using Hierarchic Marking Menus, *Proc. INTERCHI '93*, pp.482–487 (1993).
19) Kurtenbach, G. and Buxton, W.: User Learning and Performance with Marking Menus, *Proc. CHI '94*, pp.258–264 (1994).
20) Venolia, D. and Neiberg, F.: T-Cube: A Fast, Self-Disclosing Pen-Based Alphabet, *Proc. CHI '94*, pp.265–270 (1994).
21) Moriya, S. and Taninaka, H.: Concept of Minute Operation and Its Application to Pen-based Computers, *Proc. HCI Intl. '93*, pp.1034–1039 (1993).
22) Bier, E.A., Stone, M.C., Fishkin, K., Buxton, W. and Baudel, T.: A Taxonomy of See-through Tools, *Proc. CHI '94*, pp.358–364 (1994).
23) Stone, M.C., Fishkin, K. and Bier, E.A.: The Movable Filter as a User Interface Tool, *Proc. CHI '94*, pp.306–312 (1994).
24) Krishnan, S.N. and Moriya, S.: A New Pen-based User Interface that Combines Operand Specification, Menu Opening and Selection, and Operation Execution, *Human Interface News and Report (33rd Meeting on Human Interface)*, Vol.10, No.2, pp.143–152 (1995).
25) "ZAURUS" User Reference Manual, SHARP Corporation.

(Received May 29, 1995)
(Accepted September 12, 1996)

S. Navaneetha Krishnan was born in India in July 1965. He received the B.Sc. (Honours) degree in Physics from Delhi University in 1986. He joined Tokyo Denki University, Tokyo, Japan in 1988 and received his M.E. degree in Electrical Engineering in 1991. He was awarded the Ph.D. degree from Tokyo Denki University in 1996. His research interests include pen-based interactions, and pattern recognition and understanding. He is a member of IEICE Japan, IPSJ Japan and Society of Instrument and Control Engineers of Japan.

Shinji Moriya was born in December 1944. He received the Ph.D. degree from Tokyo Denki University in 1980. He was Visiting Associate Professor in the State University of New York at Buffalo in 1981 and in the University of Illinois at Urbana-Shampaign in 1982. He was Yunnan Computer Society Professor (China) in 1992 and Visiting Professor in the Yunnan Polytechnic University in China in 1994. He is currently professor of Tokyo Denki University, Tokyo, Japan. He was with the Editorial Review Board of Information Resources Management Journal, and the Special Editorial Board of Interacting with Computers Journal. His research interests include pen-based interactions, voice input user interfaces, and evaluation and modeling of human-computer interaction. He is a member of IEICE Japan, IPSJ Japan, Society of Instrument and Control Engineers of Japan, Ergonomics Society of Japan, Television Society of Japan, ACM and IEEE.