Distributed Architectures for Pen-Based Input and Diagram Recognition

Wayne Citrin
Department of Electrical and Computer Engineering
Campus Box 425
University of Colorado
Boulder, CO, USA 80309-0425
Tel: +1 303 492-1688
E-mail: citrin@cs.colorado.edu

Mark D. Gross
College of Architecture and Planning
Campus Box 314
University of Colorado
Boulder, Colorado, USA 80309-0314
Tel: +1 303 492-6916
E-mail: mdg@cs.colorado.edu

ABSTRACT

We present a system supporting pen-based input and diagram recognition that employs a personal digital assistant (PDA) as an intelligent input device for the system. Functionality is distributed between the PDA and the main computer, with the PDA performing low-level shape recognition and editing functions, and the back-end computer performing high-level recognition functions, including recognition of spatial relations between picture elements. This organization provides a number of advantages over conventional pen-based systems employing simple digitizing tablets. It provides the opportunity to use hardware specially designed for shape recognition and editing in a general diagram recognition system, it allows for improved performance through parallel processing, and it allows diagram entry to be performed remotely through use of the PDA front end in the field, with recognized shapes subsequently downloaded to the main diagram recognizer. We discuss the overall organization of the system, as well as the individual pieces and the communication between them, and describe two ongoing projects employing this architecture.

Keywords

Pen-based interfaces, diagram recognition, graphical editors

INTRODUCTION

As graphical interaction with computer systems becomes more frequent, improved entry methods for graphics must be developed. User tests make it clear that conventional mouse-and-palette editors are unsatisfactory for this purpose [3]; to create or edit a diagram using such an editor, additional user actions must be performed to switch between drawing and text modes, and to indicate the place on the screen where the graphics must go. In addition, frequent shuttling between the palette and the drawing surface slows down diagram entry and discourages users from adopting graphical systems. In order to address this problem, investigators have developed editors using pen-based interfaces. Experiments [1, 3] have shown that such interfaces speed the editing process through elimination or reduction of use of the tool palette, and through the ability to enter information in place, through gestures. Pen-based editors have the additional advantage of potentially being more portable, since the user does not need to carry or have table space for a keyboard or a mouse; pen-based devices may be made small and may be employed while the user is standing.

In order to support pen-based entry of diagrams, graphical editors must offer some sort of diagram recognition. At the very lowest level, editors must group sequences of points traced by the pen into individual strokes. At the next higher level, those strokes may be grouped into shapes. For example, a sequence of four strokes may be grouped into a square. At the next higher level, those shapes may be grouped into compound objects based on a hierarchical spatial grammar. In order to simplify the organization, these tasks are kept in separate modules.

A number of organizational features are desirable when designing pen-based graphical editors with diagram recognition systems. First, although the spatial grammars describing valid diagrams may vary by domain, and therefore modules performing the top-level implementation task may need to be reimplemented in each specialized graphical editor, the set of primitive shapes and strokes to be recognized is not likely to vary much from one editor to another. The set of editing gestures is also unlikely to vary much from one application to another. It would therefore be desirable to be able to reuse such low-level recognition routines and not have to reimplement them in each editor. It would be even more desirable to implement these routines in specialized hardware, since stroke recognition constitutes the inner loop of any pen-based editor.

Second, in order to support the impression of transparency in the pen-based interface, it is desirable for diagram entry and low level stroke and shape recognition to continue simultaneously with higher-level recognition functions. If the system does not recognize pen input while high-level recognition is being performed, or if the system does recognize input but does not rectify the digital ink into strokes and shapes, the delays will violate the illusion of transparency, and lead to lower-quality drawing and to user dissatisfaction. Therefore, it would be desirable for low-level recognition functions, including ink capture and shape and stroke recognition, to proceed in parallel with higher-level functions.

Third, because of the potential mobility of pen-based interfaces, it would be desirable for users to perform diagram entry in the field, wherever they can most usefully perform sketching and data collection tasks, rather than sketching on paper and later transcribing the drawings into the computer when the user returns to the office. Ideally, the full power of diagram recognition should be available in the field. As a reasonable compromise, it should be possible to capture diagrams and perform some low-level recognition in the field, store the diagrams, and later upload them to a host computer to perform the remaining diagram recognition.

We present a prototype diagram recognition system that supports these three useful characteristics of a pen-based graphical editor. The editor employs a personal digital assistant (PDA) as the front end of the system, and a conventional computer as the back end. The PDA, which is light and inexpensive, may be used in the field for diagram capture. The PDA contains built-in primitive recognition routines, and may be programmed to integrate those routines with editing, diagram storage and retrieval, and communications. The PDA performs low-level recognition, recognizes shapes, and transmits them to the host computer, where higher-level diagram recognition is performed. The results of this high-level recognition are then transmitted back to the PDA, where they are displayed.

In addition, the PDA may be used remotely in the field, where sketches are entered and low-level recognition performed. The resulting sketches are then stored in the PDA and uploaded to the host computer at a later time.

This distributed recognition strategy is presented in the context of the prototype, but the system architecture is applicable to any diagram recognition application requiring high performance or mobility or both. To illustrate the usefulness of the architecture, we describe two ongoing projects, one which uses PDA-based sketching to support telecommunications workers in the field, and the other a pen-based editor for a visual programming system.

Sections of the paper below describe the system in general, then discuss in detail the design of the PDA-based front end and the diagram recognition system running on the host computer back end. We also describe the methods by which the two components of the system communicate. We discuss the above mentioned ongoing diagram recognition projects, and conclude with directions for future work.

GENERAL SYSTEM ORGANIZATION

The task of pen-based diagram recognition can be divided hierarchically into several subtasks (figure 1). At the lowest level, the pen or stylus moving across a digitizer yields a stream of points. These are typically x-y coordinate pairs, but they may be pairs of x and y increments, or direction / velocity vectors. At the second level segments of the point stream are combined into strokes, and at the third level strokes may be grouped into basic shapes such as polygons, polylines, and circles, which can often be characterized by a few key points and properties (the corner points of a square, for example). At the highest level, those shapes are grouped into complex, semantically rich, diagrams according to a spatial grammar. The subtasks are to differing degrees domain-invariant. Typically the gathering of points will not vary from diagram domain to diagram domain. In most cases, stroke recognition and basic shape recognition are also domain invariant. Where a domain recognizes different or additional basic shapes, shape recognition can be deferred to a later stage, and the additional shapes can be recognized as combinations of previously recognized strokes and basic shapes. It is typically only at the high-level diagram recognition level that recognition criteria vary between domains.

Since the first three levels (point capture, stroke recognition, and shape recognition) typically do not vary much, if at all, it is desirable if they need not be reimplemented for each graphical application. One strategy is to place the routines that perform these tasks into standard libraries [13], so that they can be incorporated into various applications.

Since lower-level diagram recognition functions constitute the inner loop of any diagram recognition system, efficient recognition requires that these low-level functions be performed efficiently. One way to do this is to design specialized hardware to handle these functions. The fact that these low-level functions are domain-invariant makes such hardware more economical than it might otherwise be.

Handling of handwriting recognition, particularly in the more general context of diagram recognition, is a special case. On the one hand, handwriting recognition is highly complex; handwriting consists of a large variety of strokes whose meaning depends on complex spatial relations. On the other hand, in a pen-based system handwriting occurs frequently enough as an entry method that it should be efficiently handled in any pen-based system. It has therefore been incorporated into the hardware of PDAs such as the Apple Newton MessagePad. Since handwriting and shapes may be entered in the same diagram, the Newton attempts to apply handwriting and shape recognition to the same set of strokes; the recognizer that comes up with the most confident match is the one that is chosen.

With these considerations in mind, it appears reasonable to construct diagram recognition systems so that the low-level functions are implemented in hardware, and the high-level, domain-specific, functions are implemented in retargetable software.

Our distributed digram recognition system consists of an Apple Newton MessagePad PDA as the front end, and an Apple Macintosh computer as the back end. Currently the two units are connected by a serial cable, but the Newton has additional communications capabilities, including infrared communications and AppleTalk networking. The device can also accept modems and wireless messaging devices. We plan to investigate some of these communications methods in the future to increase the bandwidth between the front and back ends.

Figure 2 shows the distribution of functionality between the two hardware units. The front end (a program called SmartPad) accepts pen-based input and echos it back to the user as digital ink. It also captures that ink and stores it away. When a stroke is completed, the system attempts to recognize a shape (possibly combining it with other shapes or strokes that have been entered). The shape is then transmitted in encoded form to the host computer functioning as the back end. The back end (a program called the Electronic Cocktail Napkin) records the shape and attempts to identify compound objects by grouping the new shape with shapes that have already been reported to the back end. The back end can also communicate with databases and other applications to provide information for the front end client. When higher-level groupings are recognized, the grouping is reported back to the front end, which may display it. While the back end is performing recognition and other tasks, further graphical entry and low-level recognition may be performed by the front end.

Figure 3 gives a simple scenario showing the operation of the system. The user draws a circular shape on the Newton (figure 3a). The system recognizes the shape as a circle (figure 3b) and transmits the circle to the back end. The user then draws a line underneath the circle (figure 3c), which is recognized as a straight line (3d). The back end has been configured to recognize this as the configuration `sunset' and that information is transmitted back to the front end and displayed (3e).

SMARTPAD PDA-BASED FRONT END

We chose the Apple Newton MessagePad as the hardware-based front end for the diagram recognition system for a number of reasons. First, it is one of the few pieces of diagram-recognition hardware available. It is inexpensive, it is easily programmable, it is designed to communicate with other computers, and it has its own expandable internal memory. The Newton provides a set of reusable low-level recognition functions, as well as integrated handwriting recognition. Although recognition in hardware should in theory provide efficiency improvements, the Newton is a fairly slow machine; however, future models should eliminate that problem. (We discuss this issue in the conclusions section.) Finally, because of the Newton's internal memory, the Newton provides mobility and off-line diagram entry that would not ordinarily be available in a host-based system with a tethered display tablet as the input device.

We have developed a program called SmartPad that runs on the Newton as a basic front end. SmartPad supports stroke and shape recognition, and captures and saves digital ink, as well as the recognized shapes. It supports basic gesture-based editing, including erasing, copying, moving, and changing the shapes of elements. Finally, it supports text entry, recognition, and editing, either through handwriting or through a soft keyboard. All of these facilities make use of hardware-based recognition, editing, and storage services provided by the Newton.

SmartPad can be configured to transmit shapes to the back end over the serial line as soon as they are recognized, or to save them while the PDA is operating remotely and then download them all at once when it is reconnected. SmartPad also organizes diagrams by combining them into pages, which the user can page through and edit at a later time.

The SmartPad front end also allows the Newton to function as an output device by accepting information from the back end and displaying text, shapes, or other graphical information. Because it has both input and output functions, two Newtons running SmartPad can be connected together and will function as a cooperative work environment, in which any drawing done on one PDA is displayed on the other. We describe this cooperative drawing application later in the paper.

SmartPad also receives information from the back end, and displays that information on the Newton's screen. Asynchronously, and concurrently with its other functions, SmartPad listens on the Newton's serial port for input from the back end. This information may include shape descriptors, text descriptors, and editing operations. For example, the high-level recognizers in the back end may identify configurations of elements in the drawing. The back end may also be connected to databases, simulations, or other applications and may deliver information from these applications back to the SmartPad front end. The back end program may also transmit editing commands, either from editing operations performed by a user on the Macintosh (host) side or by another PDA client. When a command is read from the port and decoded, SmartPad performs the appropriate operation, which may include updating the graphics and/or the soup. The communications protocol used between the front and back ends is explained in a later section.

ELECTRONIC COCKTAIL NAPKIN BACK END

The program running on the host computer back end is the Electronic Cocktail Napkin program, a trainable diagram recognition system. The system takes strokes and shapes uploaded by the PDA, combines them into aggregates based on their spatial relations. Since SmartPad also transmits the raw digital ink to the back end as a point stream, the Cocktail Napkin has the option of re-recognizing the input shapes based on other information that may not be available to the front end, including domain information and information based on the shape's context.

The spatial relations recognized by the Electronic Cocktail Napkin may be specified through example, from a database, or by direct entry through a relation editor. When a group of shapes and the spatial relations among them are recognized by the Cocktail Napkin, it identifies the group, assigns the group a name, and downloads that information to the front end, where SmartPad displays it in association with the corresponding shapes. For example, if the Cocktail Napkin is given a rectangle and a triangle, with the triangle above the rectangle, as in figure 2, it may be configured, through training or otherwise, to recognize the two shapes as a `house'. Shapes may also be displayed, entered, and edited directly into the Cocktail Napkin, either with a mouse or with a conventional digitizing tablet, in which case they are downloaded for display onto the Newton.

The Electronic Cocktail Napkin is a pen based drawing environment for computer aided design [4]. Figure 4 shows the ECN's working screen, with simulated trace overlays (tabs, lower right of drawing board), a bulletin board for setting drawings aside while working (along the top). and a sketchbook for storing and retrieving previously made sketches (right). It has been used as front end for diagram based query for visual and other databases, for constraint based diagram editing [5], and for support of conceptual and creative design [6]. It has been linked to visual databases and case based design tools, interactive design simulations, and Netscape. The Electronic Cocktail Napkin is written in Macintosh Common Lisp, and in addition to the Newton PDA, users can draw on Wacom digitizing tablets or the mouse and trackball. In the latter cases, the Cocktail Napkin must perform its own stroke, shape, and handwriting recognition. When the Newton is used as the input device, it does not, but rather makes use of the higher-level messages transmitted by the Newton. It supports multiple users either sharing a single drawing device (it identifies different pens) or using two or more separate tablets. The program provides low level recognition of hand drawn glyphs and higher-level recognizers that identify spatial configurations; both levels of recognizer are easily programmed on the fly by end users. A simple constraint mechanism can be employed to enforce recognized relations as constraints on the drawing.

When the user draws on the Newton PDA, the Cocktail Napkin receives the recognized shapes, and higher level recognizers parse the drawing (that is, the set of shapes, or glyphs, received so far) for configurations of glyphs in certain spatial relations.

The Cocktail Napkin monitors the serial line and detects when the Newton begins to send data according to the protocol described below. The Napkin interprets the protocol and creates its own representation of the recognized diagram. Since the communication protocol also contains the raw stroke (that is, point stream) information, the Cocktail Napkin, if so configured, may employ that information to "second guess" the Newton, and make a new attempt at recognition. The system can be configured to suppress the stroke information, which leads to more efficient communication, but no longer allows the Cocktail Napkin the possibility of doing its own secondary low-level recognition. When the Cocktail Napkin recognizes a high-level construct, or re-recognizes a shape, it reports it to the Newton, which updates its own internal representations of the diagram and displays the updated diagram.

PDA/COCKTAIL NAPKIN PROTOCOL

The protocol that controls communications between the front and back ends was designed to be simple and flexible. It was also designed to be modeless, so that any message could be sent at almost any time, without devices at any end having to remember state information. Finally, it was designed so that the same messages could be sent and received by both the back and front end. This means that not only can a front and back end be connected, but two PDA front ends can be usefully connected together and communicate with each other. The protocol is naive, but this is a first attempt at such a design; we expect that future protocol designs will incorporate more internal state to improve efficiency, while still allowing mobile off-line drawing and deferred high-level diagram recognition.

The messages supported by the protocol describe editing operations performed by the front or back ends that must be reflected or otherwise noted by the device receiving the message. The page management messages (New Page, Erase Page, Delete Page, and Turn to Page) are used to organize diagrams, particularly when the sketching has been done remotely.

The shape creation message (New Shape) is sent whenever either device has created a new shape. The message packet contains an indicator indicating the type of shape it is, a bounding box indicating where the shape appears on the screen, a list of key points used to recreate the shape, and a list of ink points used to recreate the original ink in case further editing is necessary. If communication speed is essential, the ink points may be omitted from the message packet, at the cost of possible loss of further recognition opportunities. A new shape packet also contains a new unique shape identifier number which is used to identify the shape in subsequent editing operations. (This is the only place in the protocol where devices must remember state, but since the shape identifiers are stored with the other shape information, this is not a problem.)

The shape editing operations (Deleted Shape, Moved Shape, Selected Shape(s), and Edited Shape) refer to a previously assigned shape identifier. The operations send the minimum information necessary to convey the message. In the case of a deleted shape, this is simply the shape identifier, in the case of a moved shape, it is the new location of the upper left corner of the bounding box. In the case of a selection, it is the list of identifiers of shapes selected, and in the case of a general shape editing operation, it is a completely new shape descriptor.

Text messages allow text to be placed on the screen unassociated with any object (Unassociated Text message), or associated with a group of shapes (Associated Text), which is generally what happens after the back end performs high-level recognition and assigns an annotation (like 'sunset' in the earlier example) to a group of shapes. This association is a set of unique shape identifiers. When associated text is displayed, SmartPad determines where the text should be positioned. Currently it naively displays it at the upper left corner of the box bounding the associated shapes (as in figure 3). There is also a general Edited Text operation that is sent in the case of any text editing operations. (Text objects are generally simpler and shorter than shape objects, so it is less important to make their transmission more efficient.) Text objects, like shape objects, have unique identifiers.

The prototype protocol is not particularly robust. There is no way to detect, report, or recover from errors like deleting a non-existent shape, for example. We plan to add them to future implementations.

Synchronization and locking are important aspects of collaborative drawing systems, and we expect that several PDAs running SmartPad will often be configured in a single system, with or without an Electronic Cocktail Napkin back-end to mediate the communication. While we have not devoted much time to the consideration of these issues, the underlying communications services in the Newton provide some support for synchronization. When two Newtons are connected directly, users can simultaneously draw on both devices and the strokes each appear on the other device with no loss of data. No code was added to SmartPad to enable this coordination; the SmartPad program simply requests asynchronous serial communication services.

We have not considered the problem of two users simultaneously editing a single shape on different PDAs. Currently, when this happens, the edited shapes are exchanged between the two devices, and the resulting diagrams become unsynchronized. In future versions of the protocol, we plan to add the ability of one PDA to lock out other PDAs while a shape is being edited.

Although the current prototype employs a serial connection, the described protocol is suitable to other media, including infrared, AppleTalk networks, and wireless communication, assuming that the underlying connection is point-to-point, and the required connection has been set up. If this is not the case, it will not be difficult to modify the protocol to accommodate destination addresses in the message packets.

USE OF THE DISTRIBUTED ARCHITECTURE

We are currently developing two applications incorporating variants of the prototype distributed diagram recognition system discussed above.

The first system is designed to assist telecommunications service workers in the field, by providing a vehicle for obtaining visual information relevant to the task at hand and recording it for future users. Typically a service worker arrives at a site with a work order, having little prior knowledge of what is there. As shown in figure 6a, the worker may encounter a confusing tangle of wires, which must first be understood before work can begin. Therefore often the first thing a worker does when arriving on site is to make a diagram to understand the initial configuration, as for example in figure 6c. After understanding the work site, the worker plans the necessary rewiring , adding to the diagram to reflect the needed changes, and finally implements the plan by making changes to the wiring. However, currently once the task is complete, the diagrams made to understand and to plan changes are discarded because they are sketchy, and there is no convenient way to keep them. (Occasionally, the worker sensibly leaves the diagram on site, taped to the inside of a wiring box). Usually the next worker must repeat the process of understanding the configuration.

The host back end will download annotated photographs (figure 6b) and wiring diagrams (figure 6d) on demand to workers' PDA front ends from a central database, first over telephone lines, later over cellular communication links. Workers will be able to consult the photographs and diagrams in making repairs and will be able to record repairs by modifying and adding to the downloaded diagrams. Low-level tasks (recognition of newly drawn lines and editing of previously existing lines, for example) will be done on the PDA. The modified diagrams will later be uploaded to the host machine where high-level recognition will be performed to convert the new lines into wires and connections that will be added into the wiring diagram database. In addition, workers will be able to create digital photographs of their work using Apple's QuickTake digital cameras, transfer the photographs to the Newton, annotate them with pen input, and upload them to the host computer. It is hoped that, in addition to giving field service workers timely and accurate information, this system will encourage workers to report their wiring modifications and thereby keep the wiring databases current with minimal extra effort.

The second project using the distributed diagram recognition architecture is the design of a pen-based editor for the VIPR visual programming language [2]. VIPR programs are currently created using a specially designed editor that employs a conventional mouse-and-palette interface (figure 7). The graphical elements of VIPR are quite simple, consisting of rings, arrows, and text, and a pen-based entry method for VIPR programs is a natural extension. Rather than completely design and implement a new pen-based VIPR editor, we plan to use a PDA-based distributed architecture. The Newton-based front end will recognize the three basic graphical elements of VIPR programs and and allow such programs to be drawn on the Newton's screen. By recognizing component shapes and comparing the positions of these shapes with the positions of shapes already on the screen, the Newton will transmit editing commands to the back end VIPR editor over the serial link. Instead of receiving editing commands through mouse clicks, the editor will poll the serial port and receive the commands through that port. The editor will then modify the underlying semantic structure of the edited program accordingly. The advantage of such an approach, aside from the generic advantages of supporting remote diagram entry and simultaneous high- and low-level recognition, is that the only changes that will have to be made are (1) a new program from the Newton front end, most of which is either directly supported by the device, or can be modified from the existing SmartPad program, and (2) the back end editor is the current editor with the minor modification that incoming commands are received through the serial port rather than through mouse clicks and palette selections. Such a back end would probably be simpler than the original editor.

RELATED WORK

We do not know of other efforts to distribute low and high level recognition of hand drawn input across a PDA and a back end host computer. However, we discuss here three categories of related work: pen based drawing and recognition, computer supported collaborative work (CSCW) and distributed drawing environments, and other projects using PDA's.

Low-level (i.e. single stroke, single glyph) recognizers for hand drawn diagrams similar to the Newton's built in recognizer have been built, for example by Rubine[13], Apte and Kimura [1], and Zhao [14]. Any of these algorithms could be used for the low level PDA recognition, but the Newton's built in recognizer is adequate and it is convenient. We pass strokes the Newton cannot recognize to the Napkin's trainable recognizer, which can be customized interactively to recognize domain specific marks and gestures. Higher level recognition and parsing [7, 9], which in our distributed architecture takes place on the back end host computer, features in diagram editors for visual programming, but these parsers typically do not work on hand drawn input. Landay and Myers' SILK program [10] does parse hand drawn diagrams of (user interface designs).

Distributed drawing environments such as ClearBoard [8] and Commune [11] enable two or more users to share a virtual drawing surface, but these systems do not attempt recognition, nor do they distribute different leveles of functionality across connected hardware.

PDA's are often used in conjunction with other hardware devices, which leads to interesting multi-device user interface design problems. For example, Robertson et al. describe a PDA and TV-display interface for real estate agents [12]. In their system the user interacts only with the PDA, which controls the display of still images and video on the TV. Most PDA systems in commercial use do not use the PDA for entering diagrams, but for text and digital ink signatures (as in the system that United Parcel Service drivers use for logging deliveries), or with an even more limited button-menu interface.

CONCLUSIONS AND FUTURE WORK

The prototype pen-based diagram editor and recognition system described here provides most of the features previously identified as being desirable in such a system. The system supports remote and detached operation of the PDA-based front end, allowing truly mobile and portable pen-based data entry. A user can hold the system in his or her hand; no surface space is necessary for keyboard and mouse, and the user can use the system in the field, far from the host computer.

Second, the system supports continuous graphical data entry and recognition simultaneously with the high-level recognition performed by the back-end computer. This allows faster feedback, and supports the impression of transparency ideally provided by pen-based systems.

There are three drawbacks to our system as it currently stands, but we feel that these are a matter of a currently inadequate technology that will soon improve to acceptable levels:

The PDA's low-level recognition is too slow. Current shape and handwriting recognition on the Newton is indeed slow. However, results from the Newton's use of the 25 MHz ARM 610 microprocessor from Advanced RISC Machines, Ltd. Future versions of the Newton will use the 100 MHz StrongARM chip jointly developed by Digital Equipment Corporation and Advanced RISC Machines, in which case the recognition speed problem is likely to go away.

The PDA's screen is too small. The usable area of the Newton MessagePad 100's screen is 3"¥4-1/8" (7.62cm¥10.48cm). This is clearly inadequate for creation of many useful diagrams. Future Newton-based devices are under development with 8"¥11" (20.32cm¥27.94cm) screens, which should be adequate for most problems. In the meantime, we intend to include a scrolling facility in future PDA-based front-end systems.

Communication between the PDA and the back end is too slow. Our experience shows that the 9600 baud connection over the serial line between the front and back ends is too slow for truly transparent shape recognition. As mentioned earlier, we plan to investigate faster, higher bandwidth communications methods, including infrared, AppleTalk, and wireless. In addition, future versions of the Newton operating system will support a full TCP/IP stack, allowing the use of high-bandwidth network communication between the front and back ends. We intend to exploit that technology when it becomes available.

As mentioned before, the protocol for communications between the front and back ends was designed so that the front and back ends would both send and receive the same sets of messages. This means that it should be possible to connect two SmartPad front ends together and have them communicate. The result is a cooperative drawing system in which strokes, shapes, and text drawn on one Newton's screen will appear on the other Newton's screen. Such systems provide an inexpensive way to support collaborative design. We have tested such a configuration and will soon expand the system so that the Newtons communicate over an AppleTalk network, with greater physical separation between the users. We have also tested cooperative drawing configurations with more than two Newtons, and with a high-level recognizer back end mediating between the collaborators. In such a system, the back end accepts input from both PDA front ends, add semantic interpretation to the diagrams by performing high-level recognition, and reports the results to the PDAs. It should also be possible to use the back end as a database server from which graphical and textual operation could be downloaded, either on demand or as a result of high-level recognition and integrated into the data being manipulated on the PDAs.

REFERENCES

[1] Apte, A. and T. D. Kimura, "A Comparison Study of the Pen and the Mouse in Editing Graphic Diagrams," in Proc. IEEE 1993 Symposium on Visual Languages. 1993. Bergen, Norway, 352-357.

[2] Citrin, W., M. Doherty, and B. Zorn, "Formal Semantics of Control in a Completely Visual Programming Language," in IEEE Symposium on Visual Languages (VL '94). 1994. St. Louis, 208-215.

[3] Citrin, W. V., "Requirements for Graphical Front Ends for Visual Languages," in Proc. IEEE 1993 Workshop on Visual Languages. 1993. Bergen, Norway, 142-151.

[4] Gross, M. D., "The Electronic Cocktail Napkin - computer support for working with diagrams." Design Studies, 1996. 17(1): to appear.

[5] Gross, M. D., "Stretch-A-Sketch: A Dynamic Diagrammer," in IEEE-CS Symposium on Visual Languages. 1994. St. Louis, 232-238.

[6] Gross, M. D. and E. Y.-L. Do, "Shape-Based Reminding in Creative Design," in Computer-Aided Architectural Design Futures (CAAD Futures '95). 1995. Singapore, 1-11.

[7] Helm, R., K. Marriott, and M. Odersky, "Building Visual Language Parsers," in ACM Conference on Human Factors in Computing Systems (CHI '91). 1991. New Orleans, 105-112.

[8] Ishii, H. and M. Kobayashi, "ClearBoard: A Seamless Medium for Shared Drawing and Conversation with Eye Contact," in ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '92). 1992. Monterey, CA, 525-532.

[9] Lakin, F., "Visual Grammars for Visual Languages," in National Conference on Artificial Intelligence (AAAI '87). 1987. Seattle, 683-688.

[10] Landay, J. A. and B. A. Myers, "Interactive Sketching for the Early Stages of Interface Design," in ACM Conference on Human Factors in Computing Systems (CHI '95). 1995. Denver, 43-50.

[11] Minneman, S. L. and S. A. Bly, "Managing à trois: a study of a multi-user drawing tool in distributed design work," in Human Factors in Computing Systems (CHI '91). 1991. New Orleans, 217-224.

[12] Robertson, S., C. Wharton, C. Ashworth, et al., "Dual Device User Interface Design: PDAs and Interactive Television," in Human Factors in Computing Systems (CHI '96). 1996. Vancouver, to appear.

[13] Rubine, D., "Specifying Gestures by Example." Computer Graphics, 1991. 25(4): 329-337.

[14] Zhao, R., "Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors," in Proc. INTERCHI '93. April 1993. Amsterdam, 95-100.