Touchscreen Interfaces for Visual Languages

Michael Hackett and Philip T. Cox
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada
e-mail: mhackett@cs.dal.ca, pcox@cs.dal.ca

***Abstract*—Through the construction of a simple mock visual programming language (VPL) editor, this study compares two different styles of touchscreen interface and demonstrates the natural fit between touch input and visual programming. The touchscreen removes a layer of indirection introduced by the mouse and allows for a more direct relationship—literally "hands on"—with the objects on the screen. The addition of "multi-touch" also opens up intriguing possibilities for two-handed, immersive interfaces, with the potential for greater efficiencies than possible with the mouse's single point of interaction.**

Keywords-Touchscreens; visual programming; bimanual; user interfaces; kinaesthetic feedback

I. Introduction

An often cited advantage of visual programming environments is the greater sense of *direct manipulation* that these environments provide, through the immediate feedback received while working with some graphical representation of the code. However, when the manipulation is performed with a mouse or a trackpad, as is typical in existing systems, an unnatural barrier is placed between the user and the software environment. Additionally, having to frequently switch hand positions and "operating modes" between mouse and keyboard slows users down and breaks their sense of flow, cutting into their productivity.

Recently, large touchscreen devices have become inexpensive and readily available, while new user interface designs are being developed that are tailored to touch input. Through these devices, users can interact more directly with their data, manipulating visual representations with the touch of a finger. And while the lack of a physical keyboard would seem to make these devices poorly suited for traditional text-based programming, the more direct control of on-screen objects that the touchscreen affords might just make them ideal for visual programming languages (VPLs).

This paper describes the design and implementation of a simple VPL code editor—codenamed "Flow"—for a touchscreen tablet device (an Apple iPad). The prototype was built in order to experiment with various user-interface options and to compare the usability and efficiency of the interfaces. The initial version has two different styles of interface, one based on the familiar drag-and-drop paradigm and the second using a novel bimanual (two-handed) approach. To minimize the influence of language syntax on overall usability, the VPL is closely modeled on Prograph [1], an existing well known dataflow programming language. At this stage, the prototype is only a facade of a true visual programming environment—editing is very limited and execution is not yet supported.

II. Background and Related Work

Although there is little prior work to draw upon with respect to visual programming on a touchscreen, the design of the Flow prototype was informed by more general touch-input and user-interface research. Of particular interest and inspiration was research into bimanual interfaces, kinaesthetic feedback, and touchscreen gesture design.

A. Two-Handed Input

Bimanual interfaces take advantage of the use of both of the user's hands together, usually in an asymmetric fashion. Yee [2] and Wu et al. [3] suggest using the non-dominant (NP) hand to establish and maintain modes while the dominant (P) hand does work within that context (as suggested in Guiard's Kinematic Chain model of bimanual action [4]).

This type of arrangement seems particularly well suited to today's "multi-touch" displays (which can distinguish multiple simultaneous touches), and the larger screens make it feasible to create a touch interface that supports the use of two hands at once. Hence, it was decided early on to experiment with at least one interface of this style.

B. Kinaesthetic Feedback

Sellen et al. [5] provide details on two experiments that compared kinaesthetic feedback (using a foot pedal) with visual feedback, and user-maintained feedback with system-maintained. They found that *actively* user-maintained kinaesthetic feedback significantly reduces mode errors (that is, instances in which a user performs an action appropriate for a mode that the application is not currently in), and allows for faster resumption of activity after an interruption. Wu et al. [3] use the term "kinaesthetically held modes" to describe this interface technique, while Raskin suggests the more succinct "quasimodes" [6, p. 55].

In contrast, “latching” modes, where a mode stays in effect until cancelled or changed, tend to lead to more mode errors [5][6], and users often struggle to figure out how to cancel the mode selection if they change their minds or realize they have made an error.

C. Touchscreen Gesture Design

Nielsen et al. [7] have defined a number of principles and guidelines for designing gestures with usability and ergonomics in mind. They also presented a procedure for building a “gesture vocabulary” (the set of gestures in an interface) through user studies, then refining and testing the resulting gesture set. They stress the importance of keeping ergonomics in mind and warn against basing gesture design on the recognition capabilities of the hardware, as this may result in gestures that are illogical and “stressing or impossible to perform [for] some people”.

Mauney et al. [8] performed a user study with participants from eight different countries and found that there was strong agreement across cultures for gestures with a physical or metaphorical correspondence to the objects being manipulated (“direct manipulation gestures”), but fairly low agreement on gestures that were symbolic in nature. This suggests that symbolic gestures should be avoided (a point also made by Frisch et al. [9]), except perhaps as expert-level shortcuts.

III. Design Principles

While far from an exhaustive list, this section describes three key principles that guided the design of the prototype.

Discoverability: Norman and Nielsen [10] have criticized today’s commercial touch tablet interfaces, citing numerous examples of gestures that are not easily discoverable or guessable, and are often learned only by reading about them elsewhere. To support discoverability in Flow, a toolbar displays buttons for all available commands, obviating the need for a series of hidden (and often arbitrary) gestures that need to be learned. On the other hand, moving graphical objects by dragging them with a finger is intuitive and easily discoverable; it is not necessary to create a special button for this action.

Minimizing Mode Errors: Flow uses kinaesthetic feedback, in the form of screen contact, to regulate modes. A mode becomes active when a toolbar button is touched or a gesture is recognized, and remains active only as long as the user maintains that particular contact with the screen. Modes are disengaged automatically when all fingers are lifted from the screen, providing an easy and reliable way to return to the default application state.

Responsiveness: Although this is only an early prototype, it was believed that the interface would need to be smooth and responsive in order for any user feedback to be meaningful. To give the user a real sense of direct manipulation, objects would have to react without any hesitation or sluggishness. (That this intuition was correct was borne out in the comments from users who were all very impressed and pleased with the responsiveness of the application.)

IV. Prototype Application

For the initial prototype of Flow, two versions were created, each employing a different user interaction style. The visual layout for both is essentially the same, featuring a toolbar along one side of the display and a large area for displaying the code being edited. (See Figures 1–3.) The toolbar contains buttons for creating each of the language elements (initially just three types of operations; input and output nodes; and datalinks for connecting nodes) as well as various commands (only Delete at the moment). The toolbar can be placed along either the left or right edge of the screen (for right- or left-handed users, respectively), as selected through a user preference.

In one version, the toolbar buttons control the current operating mode of the editor—adding an operation, a node, or a datalink, or deleting any of the above. A button must be “held down” (the touch must be maintained on the button) to engage the corresponding mode, thereby providing continuous, user-maintained kinetic feedback. (Figure 1.) While the mode is active, a second touch performs some manipulation within the context of the mode, such as tapping or dragging in the main editor pane. Although this can all be done (somewhat awkwardly) with one hand, the intent is for this interface to be used with two hands, the NP hand selecting the mode and the P performing the manipulation. This will be referred to as the “quasimodal” version, using Raskin’s term.

In the second version, all of the buttons for creating new objects are operated by dragging a finger from the button onto the code pane, which creates an object for the user to drag into place. (Figure 2.) The delete command is, for the moment, still operated using the two-handed method, but that could be replaced or supplemented with a gesture (such as making a stroke through an object, or dragging it to a trash area along another edge of the screen). This interface, despite the mixing of styles, will be referred to as the “drag-and-drop” version.

The drag-and-drop style should be familiar to most users from desktop GUIs, but the quasimodal interface may require a bit of explanation. It would initially be unfamiliar to most, but it was hoped that, once its operation is explained, it would be easy to understand and offer a viable alternative to more customary designs.

In both versions, the editor operates in a default mode when no buttons are pressed. In this mode, objects can be moved simply by dragging them (Figure 3). In the quasimodal version, objects cannot be moved while any of the buttons are engaged, as that action might be confused with a gesture related to the active quasimode. Take, for

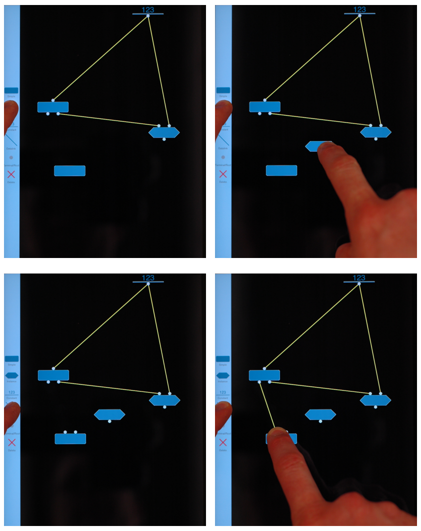

Figure 1. Quasimodal operation: Creating a new instance operation involves actively maintaining a touch on the Instance operation button (top-left photo) and tapping with another finger to place the operation icon in the code pane (top-right). To create a new datalink between nodes, the Datalink button is held (bottom-left) while another finger draws a line from the start node to the end node (bottom-right).

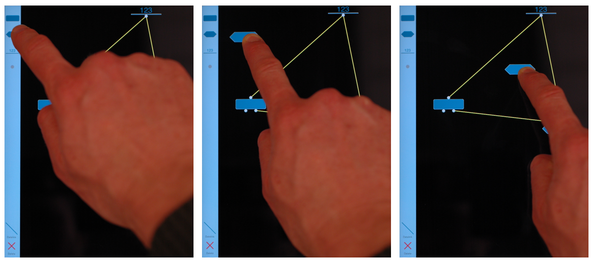

Figure 2. Drag-and-drop operation: New operations are dragged from the toolbar onto the editor pane and placed wherever desired. (Sequence shown left-to-right.)

example, creating a datalink between two nodes, which involves dragging one's finger across the screen from one node to the other (while the Datalink quasimode is engaged). Nodes are attached to the top and bottom edges of operation icons, so the point of contact would be on or near the edge of the node's operation icon. It would be hard to reliably distinguish this action from an attempt to move the operation, if it were not for the active quasimode providing context. Because a finger touch is not nearly as precise as a mouse or stylus, and some of the targets are necessarily small, the software must be forgiving and allow a good deal of leeway in hitting targets. By limiting the set of potential targets within a mode or quasimode, recognition accuracy can be much higher, which makes the interface seem more intelligent.

In the drag-and-drop interface, creating new code objects by dragging them from the toolbar invokes a kind of

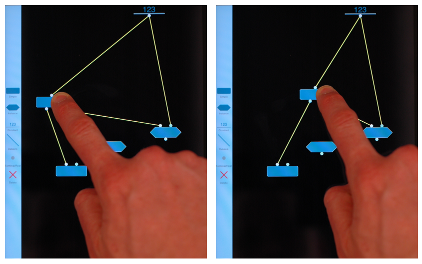

Figure 3. Moving objects: In the default editing mode (and at any time in the drag-and-drop interface), operation icons can be moved simply by dragging them.

quasimode as well, in as much as the creation mode is active only as long as the user maintains screen contact with the finger that started the drag gesture. However, unlike in the full quasimode interface, where engaging a toolbar button locks the entire interface into the corresponding quasimode, here the quasimode applies only to that one finger. Other fingers can continue to move existing objects in the editor, or begin additional creation actions. This feature is not actually inherent in the design; it was more an accident of some implementation choices, but it seems in the spirit of an immersive multi-touch interface to allow this to remain.

The target hardware for the prototype is a first-generation Apple iPad. The iPad has a 9.75″ (247 mm diagonal) screen with a capacitive touchscreen overlay, which can simultaneously detect independent touches from all ten fingers. The software was developed as a "native app", using Objective-C and Apple's iOS SDK [11]. Videos demonstrating the prototype are available at [12].

V. User Evaluation

A small sampling of users known to the authors were given the software to try, and were interviewed during and after using it. Some were expert Prograph users and so fully understood the model of the visual editor and the symbols used, while some others were non-programmers or novice Prograph users who, nonetheless, were able to give feedback on the responsiveness and naturalness of the interface options.

Overall, there was a strong consensus that the design was intuitive and very responsive. All but one user (and all of the expert users) felt that the sense of direct manipulation was greater than with a mouse, and that the responsiveness of the interface to touches was an important part of that.

The expert users all initially assumed that drag-and-drop would be the way to operate the toolbar buttons, while some novice users first tried to tap on the buttons to latch them on. They generally attributed this expectation to their previous computer experience. After an explanation was given (often as little as suggesting that one try using two hands), all but one quickly mastered the quasimodal operation, and the majority said that they preferred this version to using drag-and-drop. Most found it faster to create new objects with the two-handed interface than by dragging items from the toolbar. And the expert users found adding nodes by dragging them from the toolbar to be somewhat awkward, perhaps because there was no analogue to this in the desktop version. No one had a problem adding nodes by tapping using the quasimodal version.

Two users expressed a preference for the drag-and-drop interface, primarily because they preferred to use just one hand. This seemed to be largely because the form factor of the tablet invites one to sit back and hold it in one hand, rather than placing it on a table so that both hands are free. One user suggested moving the toolbar buttons up towards the top edge of the screen so that the unit could be gripped up higher (better balancing the tablet in the hand) while still leaving the thumb free to press buttons. It was also suggested that a landscape orientation for the editor (it is currently portrait orientation only) would accommodate holding the tablet with both hands while operating the interface with one's thumbs.

VI. Conclusion and Future Work

The feedback obtained so far indicates that users find the environment appealing to use and that the sense of direct manipulation, a key feature of visual programming, is enhanced by the touchscreen interface. Having multiple, simultaneous touch points instead of single cursor opens up many possibilities for immersive and more efficient programming interfaces. While it is a bit early to begin judging coding efficiency, the experienced Prograph users were very excited and are looking forward to putting that question to the test with a future, more fully realized version.

However, it appears that users do need at least a small amount of instruction to figure out how to operate the quasimodal interface, given its unfamiliarity. Because mobile applications rarely come with any external documentation, some sort of in-program assistance should be added. Progressive enhancement techniques could also be used to gradually introduce novel interface features. Mobile games may offer a useful model to follow, as it is common for games to have specialized rules and control systems that must be learned before or during play. Testing so far does suggest, however, that once users learn the "trick", many do prefer the quasimodal interface, so the small amount of effort put into learning the unfamiliar system seems worthwhile.

Obviously, much more rigorous user testing is required, and we intend to conduct such testing as the prototype matures. Future work will largely be focused on experimenting with designs for navigating higher-level language constructs, such as methods, classes, and libraries. There are a number of general approaches to consider, such as zooming, overview+detail, focus+context (all described in [13]), and/or a more traditional hierarchical system. And within those approaches, there are several ways to map them to a small screen and to take advantage of touch input. Again, particular attention will be paid to the option of bimanual input, with the hope that this can spur further investigation into the effectiveness of this mode of interaction.

Acknowledgment

The authors would like to thank the members of the Dalhousie University Visual Languages and Design Group for their valuable input during the development of the Flow prototype, and their former Pictorius colleagues who took the time to provide expert-user feedback on the prototype.

References

[1] P. T. Cox, F. R. Giles, and T. Pietrzykowski, "Prograph: a step towards liberating programming from textual conditioning," in *Proc. 1989 IEEE Workshop on Visual Languages*, 1989, pp. 150–156.

[2] K. Yee, "Two-handed interaction on a tablet display," in *CHI '04 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2004, pp. 1493–1496.

[3] M. Wu, C. Shen, K. Ryall, C. Forlines, and R. Balakrishnan, "Gesture registration, relaxation, and reuse for multi-point direct-touch surfaces," in *Proc. 1st IEEE Int. Workshop on Horizontal Interactive Human-Computer Systems (TableTop 2006)*, 2006, pp. 185–192.

[4] Y. Guiard, "Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model," *Journal of Motor Behavior*, vol. 19, pp. 486–517, 1987.

[5] A. J. Sellen, G. P. Kurtenbach, and W. A. S. Buxton, "The prevention of mode errors through sensory feedback," *Human-Computer Interaction*, vol. 7, pp. 141–164, Jun. 1992.

[6] J. Raskin, *The Humane Interface: New directions for designing interactive systems*. Reading, MA, USA: Addison-Wesley, 2000.

[7] M. Nielsen, M. Störring, T. B. Moeslund, and E. Granum, "A procedure for developing intuitive and ergonomic gesture interfaces for HCI," in *Gesture-Based Communication in Human-Computer Interaction*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 2915, pp. 105–106.

[8] D. Mauney, J. Howarth, A. Wirtanen, and M. Capra, "Cultural similarities and differences in user-defined gestures for touchscreen user interfaces," in *Proc. 28th Int. Conf. Human Factors in Computing Systems (extended abstracts) (CHI EA '10)*. New York, NY, USA: ACM, 2010, pp. 4015–4020.

[9] M. Frisch, J. Heydekorn, and R. Dachselt, "Diagram editing on interactive displays using multi-touch and pen gestures," in *Diagrammatic Representation and Inference*, ser. Lecture Notes in Computer Science, A. Goel, M. Jamnik, and N. Narayanan, Eds. Springer Berlin / Heidelberg, 2010, vol. 6170, pp. 182–196.

[10] D. A. Norman and J. Nielsen, "Gestural interfaces: a step backward in usability," *interactions*, vol. 17, pp. 46–49, Sep. 2010.

[11] Apple Computer. (2011) iOS Reference Library. [Online]. Available: http://developer.apple.com/library/ios/navigation/

[12] M. Hackett, "Multitouch dataflow editing (video)," 2011. [Online]. Available: http://web.cs.dal.ca/~pcox/visual/Multitouch/

[13] A. Cockburn, A. Karlson, and B. B. Bederson, "A review of overview+detail, zooming, and focus+context interfaces," *ACM Computing Surveys (CSUR)*, vol. 41, p. 2:1–2:31, Jan. 2009.