Foreword for

Your Wish is My Command, Edited by Henry Lieberman

Ben Shneiderman, Draft May 14, 2000

Ben Shneiderman	http://www.cs.umd.edu/~ben
Dept of Computer Science	301-405-2680
University of Maryland	301-405-6707 fax
College Park, MD 20742	http://www.cs.umd.edu/hcil

Setting an alarm clock is probably the most common form of programming. Users set a time and then put the clock in alarm mode. Older 12-hour mechanical clocks usually had a special alarm hand that could be moved to the time for the alarm to ring, and then the users turned the alarm switch on. A nice form of *direct manipulation programming* – easy to learn and use.

Direct manipulation is a term I coined in 1981 to describe the visual world of action in many successful graphical user interfaces such as video games, air traffic control and What-you-see-is-what-you-get word processors. The principles were to:

- represent the objects and actions visually,
- replace typing with pointing and dragging,
- provide rapid, incremental and reversible actions, and
- offer immediate and continuous feedback to users.

These principles can lead to interfaces that help novices and experts, prevent or at least reduce errors, and encourage exploration because reversibility is supported. Designers continue to refine and extend direct manipulation, but critics complain that direct manipulation only works for limited tasks. They often ignore the possibility of direct manipulation programming, which was part of the original conception [1, 2].

To explore the possibilities, we built a direct manipulation programming tool in 1984-85 that enables users to create macros for MS DOS. This tool, Direct Manipulation DOS (DMDOS) [3], enabled users to record and view their actions, and then store and replay macros. We were motivated by successful macro facilities for unix, word processors and spreadsheets. These early keyboard-oriented systems led us to joke that "those who ignore history are destined to retype it." We were also inspired by innovative programming-by-demonstration in David Canfield Smith's Pygmalion [4], graphical macro facilities in Dan Halbert's SmallStar [5] and Alan MacDonald's early call for Visual Programming [6]. These pioneers and other innovators believed in the goal of empowering users to create useful programs, extend existing interfaces, and build small just-in-time programs that automated daily tasks.

This important volume of papers carries forward the agenda of making direct manipulation programming (or programming-by-example, programming-by-demonstration, end-user-programming, programming-in-the-user-interface, etc.) a reality.

While there have been successes in the intervening years, such as programmable machine tools, visual programming languages, and a variety of macro building programs, widespread adoption is still elusive. Henry Lieberman deserves credit for his long devotion to this topic and for collecting the diverse strategies and application domains in this volume. He and the contributors to this volume remind us all of the breadth of opportunities and depth of ambition.

The allure of direct manipulation programming is its capacity to empower users, while minimizing learning of programming concepts. Researchers continue to seek simple cognitive models for programming that are in harmony with the cognitive model of the existing user interface. Just as the programmable mechanical alarm clock is tied to the familiar model of clock hands, researchers have wanted to build on the visual nature of graphical user interfaces.

This fundamental human-computer interaction challenge has inspired a generation of designers, who have come up with innovative strategies for supporting iteration, conditionals, parameter passing, modular design, pattern matching, and data representation. This treasure chest of strategies is likely to pay off in multiple solutions for direct manipulation programming and related problems. A successful strategy would not only be easy to learn, but also support rapid composition of common programs. Then it would also be easy to invoke, with comprehensible feedback about successful and unsuccessful executions.

One strategy represented in this book is to develop software that recognizes familiar patterns of action and infers a useful program. There may be some opportunities along this path but I prefer the second path of special tools for users to create a program, just as they move the special hand of an alarm clock to set the wake-up time.

A third path, also well represented in this book, is visual programming languages in which the users set out to write a program, but visually instead of textually. Visual programming languages may have a simple basis such as dragging items from a relational table to a screen-based form to create a report program. More elaborate visual programming languages have graphic symbols to represent objects, actions, conditionals, loops, and pattern matching.

A fourth path might be to add history capture environments for every interface. Unix command line interfaces had a history log that allowed users to conveniently review and reuse commands. World-Wide Web browsers support history keeping of page visits with relatively easy review and reuse. Microsoft Word captures a history of actions to support undo operations, but users cannot review the history or save it. Adobe Photoshop 5.0 added a nice history feature for graphic designers, demonstrating that even in complex environments rich history support is possible.

Our current efforts with Simulation Processes in a Learning Environment have emphasized history keeping, enabling users to review their work, replay it, annotate it, and send it to peers or mentors for advice [7]. An immediate payoff was that faculty

could run the simulation in exemplary or inappropriate ways and store the histories for students to use as a training aid.

The story of this field and this book is that there is magic and power in creating programs by direct manipulation activities, as opposed to writing code. The potential for users to take control of technology, customize their experiences, and creatively extend their software tools is compelling.

Eighteenth century scientists, like Ben Franklin, experimented with electricity and found its properties quite amazing. Franklin, Faraday, Maxwell, and others, laid the foundation for Thomas Edison's diverse applications, such as refinements of telegraphy, generators, and electric lighting. This book brings reports from many Franklins, Faradays, and Maxwells who are laying the foundation for the Thomas Edisons, still to come. It is difficult to tell which idea will trigger broad dissemination or whose insight will spark a new industry. However, the excitement is electric.

(Acknowledgement: Thanks to Richard Potter for his comments on my drafts)

1. Shneiderman, B., The future of interactive systems and the emergence of direct manipulation, *Behaviour and Information Technology 1*, 3, 1982, 237-256

2. Shneiderman, B., Direct manipulation: A step beyond programming languages, *IEEE Computer 16*, 8, August 1983, 57-69).

3. Iseki, O. and Shneiderman, B., Applying direct manipulation concepts: Direct Manipulation Disk Operating System (DMDOS), *ACM SIGSOFT Software Engineering Notes 11*, 2, (April 1986), 22-26.

4. Smith, D. C., *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*. Basel, Stuttgart, Birkhauser Verlag. 1977.

5. Halbert, Daniel, Programming by Example, Ph. D. dissertation, Department of Electrical Engineering and Computer Systems, University of California, Berkeley, CA, Available as Xerox Report OSD-T8402, Palo Alto, CA, (1984).

6. MacDonald, Alan, Visual programming, *Datamation 28*, 11 (October 1982), 132-140.

7. Plaisant, C., Rose, A., Rubloff, G., Salter, R., Shneiderman, B., The design of history mechanisms and their use in collaborative educational simulations, *Proc. Computer Supported Collaborative Learning Conference* (December 1999), 348-359.