Direct Manipulation for Comprehensible, Predictable and Controllable User Interfaces

Ben Shneiderman
Human Computer Interaction Laboratory
Department of Computer Science
Institute for Systems Research
University of Maryland, College Park, MD 20742

Abstract: Direct manipulation user interfaces have proven their worth over two decades, but they are still in their youth. Dramatic opportunities exist to develop direct manipulation programming to create end-user programming tools, dynamic queries to perform information search in large databases, and information visualization to support network database browsing. Direct manipulation depends on visual representation of the objects and actions of interest, physical actions or pointing instead of complex syntax, and rapid incremental reversible operations whose effect on the object of interest is immediately visible. This strategy can lead to user interfaces that are comprehensible, predictable and controllable. Direct manipulation interfaces are seen as more likely candidates to influence advanced user interfaces than adaptive, autonomous, intelligent agents. User control and responsibility are highly desirable.

Note: This paper is adapted, with permission of the publisher, from the forthcoming book: *Designing the User Interface: Strategies for Effective Human-Computer Interaction (3rd Edition)*, Addison Wesley, Reading, MA (1997).

Introduction

Certain interactive systems generate a glowing enthusiasm among users that is in marked contrast with the more common reaction of grudging acceptance or outright hostility. The enthusiastic users' report the following positive feelings:

- Mastery of the interface
- Competence in performing tasks
- Ease in learning the system originally and in assimilating advanced features
- Confidence in the capacity to retain mastery over time
- Enjoyment in using the system
- Eagerness to show the system off to novices

• Desire to explore more powerful aspects of the system

These feelings are not universal, but this amalgam is meant to convey an image of the truly pleased user. The central ideas in these systems is visibility of the objects and actions of interest; rapid, reversible, incremental actions; and replacement of complex command-language syntax by direct manipulation of the object of interest—hence, the term *direct manipulation* (Shneiderman, 1983; Hutchins et al., 1986; Ziegler and Fähnrich, 1988; Frohlich, 1993).

The attraction of direct manipulation is apparent in the enthusiasm of the users. The designers of early direct manipulation systems had an innovative inspiration and an intuitive grasp of what users would want. Each example has features that could be criticized, but it seems more productive to construct an integrated portrait of direct manipulation :

1. Continuous representation of the objects and actions of interest
2. Physical actions or presses of labeled buttons instead of complex syntax
3. Rapid incremental reversible operations whose effect on the object of interest is immediately visible

Using these three principles, it is possible to design systems that have these beneficial attributes:

- Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
- Experts can work rapidly to carry out a wide range of tasks, even defining new functions and features.
- Knowledgeable intermittent users can retain operational concepts.
- Error messages are rarely needed.
- Users can immediately see if their actions are furthering their goals, and, if the actions are counterproductive, they can simply change the direction of their activity.
- Users experience less anxiety because the system is comprehensible and because actions can be reversed so easily.
- Users gain confidence and mastery because they are the initiators of action, they feel in control, and the system responses are predictable.

The success of direct manipulation stems from the visibility of the objects of interest so that interface actions are close to the high-level task domain. There is little need for the mental decomposition of tasks into multiple interface commands with a complex syntactic form. On the contrary, each action produces a comprehensible result in the task domain that is visible in the interface immediately. The closeness of the task domain to the interface domain reduces

operator problem-solving load and stress. This basic principle is related to stimulus–response compatibility discussions in the human-factors literature. The task concepts dominate the users' concerns, and the distraction of dealing with tedious interface concepts is reduced.

Direct-Manipulation Programming

Performing tasks by direct manipulation is not the only goal. It should be possible to do programming by direct manipulation as well, for at least some problems. Robot programming is sometimes done by moving the robot arm through a sequence of steps that are later replayed, possibly at higher speed. This example seems to be a good candidate for generalization. How about moving a drill press or a surgical tool through a complex series of motions that are then repeated exactly? In fact, these direct-manipulation programming ideas are implemented in modest ways with automobile radios that users preset by tuning to their desired station and then pressing and holding a button. Later when the button is pressed, the radio tunes to the preset frequency. Some professional television-camera supports allow the operator to program a sequence of pans or zooms and then to replay it smoothly when required.

Programming of physical devices by direct manipulation seems quite natural, but an adequate visual representation of information may make direct-manipulation programming possible in other domains. Several word processors allow users to create macros by simply performing a sequence of commands that is stored for later use. WordPerfect enables the creation of macros that are sequences of text, special function keys such as TAB, and other WordPerfect commands . EMACS allows its rich set of functions, including regular expression searching to be recorded into macros. Macros can invoke each other, leading to complex programming possibilities. These and other systems allow users to create programs with nonvarying action sequences using direct manipulation, but strategies for including loops and conditionals vary. EMACS allows macros to be encased in a loop with simple repeat factors. By resorting to textual programming languages, EMACS and WordPerfect allow users to attach more general control structures.

Spreadsheet packages, such as LOTUS 1-2-3 and Excel, have rich programming languages and allow users to create portions of programs by carrying out standard spreadsheet operations. The result of the operations is stored in another part of the spreadsheet and can be edited, printed, and stored in a textual form.

Macro facilities in graphic user interfaces are more challenging to design than are macro facilities in traditional command interfaces. The MACRO command of Direct Manipulation Disk Operating System (DMDOS) (Iseki and Shneiderman,

1986) was an early attempt to support a limited form of programming for file movement, copying, and directory commands.

Smith (1977) inspired work in this area with his Pygmalion system that allowed arithmetic programs to be specified visually with icons. A number of early research projects have attempted to create direct manipulation programming systems (Rubin et al., 1985). Maulsby and Witten (1989) developed a system that could induce or infer a program from examples, questioning the users to resolve ambiguities. In constrained domains inferences become predictable and useful, but if the inference is occasionally wrong users will quickly distrust it.

Myers (1992) coined the phrase *demonstrational programming* to characterize these efforts in which users can create macros by simply doing their tasks and letting the system construct the proper generalization automatically. Cypher (1991) built and ran a usability test with seven subjects for his EAGER system that monitored user actions within HyperCard. When EAGER recognized two similar sequences, a small smiling cat appeared on the screen to offer the users help in carrying out further iterations. Cypher's success with two specific tasks is encouraging, but it has proven to be difficult to generalize this approach.

It would be helpful if the computer could reliably recognize repeated patterns and automatically create a useful macro, while the user is engaged in performing a repetitive interface task. Then, with the user's confirmation, the computer could take over and automatically carry out the remainder of the task. This hope for automatic programming is appealing, but a more effective approach may be to give users the visual tools to specify and record their intentions. Rule-based programming with graphical conditions and actions offers a fresh alternative that may be appealing to children and adults (Smith et al., 1994). The screen is portrayed as a set of tiles and users specify graphical re-write rules by showing before and after tile examples. Another innovative environment conceived of initially for children is ToonTalk (Kahn, 1996) which offers animated cartoon characters who carry out actions in buildings using a variety of fanciful tools.

To create a reliable tool that works in many situations without unpredictable automatic programming, designers must meet the *five challenges of programming in the user interface* (PITUI) (Potter, 1993):

1. Sufficient computational generality (conditionals, iteration)
2. Access to the appropriate data structures (file structures for directories, structural representations of graphical objects) and operators (selectors, booleans, specialized operators of applications)
3. Ease in programming (by specification, by example, or by demonstration, with modularity, argument passing, etc.) and editing programs
4. Simplicity in invocation and assignment of arguments (direct manipulation, simple library strategies with meaningful names or icons, in-context execution,

and availability of result)
5. Low risk (high probability of bug-free programs, halt and resume facilities to permit partial executions, undo operations to enable repair of unanticipated damage)

The goal of PITUI is to allow users easily and reliably to repeat automatically the actions that they can perform manually in the user interface. Rather than depending on unpredictable inferencing, users will be able to indicate their intentions explicitly by manipulating objects and actions. The design of direct-manipulation systems will undoubtedly be influenced by the need to support PITUI. This influence will be a positive step that will also facilitate history keeping, undo, and online help.

The *cognitive dimensions* framework may help in analyzing design issues of visual programming environments such as those needed for PITUI (Green and Petre, 1996). The framework provides a vocabulary to facilitate discussion of high-level design issues, for example, "viscosity" is used to describe the difficulty of making changes in a program and progressive evaluation describes the capacity for execution of partial programs. Some of the other dimensions are consistency, diffuseness, hidden dependencies, premature commitment, and visibility.

Direct manipulation programming is an alternative to the agent scenarios. Agent promoters believe that the computer can automatically ascertain the users' intentions or take action based on a vague statements of goals. This author is skeptical that user intentions are so easily determined or that vague statements are usually effective. However, if users can specify what they want with comprehensible actions selected from a visual display, then they can more often and more rapidly accomplish their goals while preserving their sense of control and accomplishment.

Adaptive Agents and User Models versus Control Panels

Some designers promote the notion of adaptive and/or anthropomorphic agents that would carry out the users' intents and anticipate needs (Maes, 1994, 1995). Their scenarios often show a responsive, butler-like human being to represent the agent (a bow-tied, helpful young man in Apple Computer’s 1987 video on the *Knowledge Navigator*), or refer to the agent on a first-name basis (such as Sue or Bill in Hewlett-Packard’s 1990 video on future computing). Microsoft's unsuccessful BOB program used cartoon characters to create onscreen partners. Others have described “knowbots,” agents that traverse the World Wide Web in search of interesting information or a low price on a trip to Hawaii.

Many people are attracted to the idea of a powerful functionary carrying out

their tasks and watching out for their needs. The wish to create an autonomous agent that knows people’s likes and dislikes, makes proper inferences, responds to novel situations, and performs competently with little guidance is strong for some designers. They believe that human–human interaction is a good model for human–computer interaction and seek to create computerized partners, assistants, or agents. They promote their designs as intelligent and adaptive, and often, they pursue anthropomorphic representations of the computer to the point of having artificial faces talking to users. Anthropomorphic representations of computers, have been unsuccessful in bank terminals, computer assisted instruction, talking cars, or postal service stations, but some designers believe that they can find a way to attract users.

A variant of the agent scenario, which does not include an anthropomorphic realization, is that the computer employs a “user model” to guide an adaptive system. The system keeps track of user performance and adapts its behavior to suit the users' needs. For example, several proposals suggest that, as users make menu selections more rapidly, indicating proficiency, advanced menu items or a command-line interface appears. Automatic adaptations have been proposed for response time, length of messages, density of feedback, content of menus, order of menu items, type of feedback (graphic or tabular), and content of help screens. Advocates point to video games that increase the speed or number of dangers as users progress though stages of the game. However, games are quite different from most work situations, where users have external goals and motivations to accomplish their tasks. There is much discussion of user models, but little empirical evidence of their efficacy.

There are some opportunities for adaptive user models to tailor system responses, but even occasional unexpected behavior has serious negative side effects that discourages use. If adaptive systems make surprising changes, users must pause to see what has happened. Then, users may become anxious because they may not be able to predict the next change, interpret what has happened, or restore the system to the previous state. Suggestions that users could be consulted before a change is made are helpful, but such intrusions may still disrupt problem-solving processes and annoy users.

The agent metaphor is based on the design philosophy that assumes users would be attracted to “autonomous, adaptive, intelligent” systems. Designers believe that they are creating something lifelike and smart, however users may feel anxious and unable to control these systems. Success stories for advocates of adaptive systems include a few training and help systems that have been extensively studied and carefully refined to give users appropriate feedback for the errors that they make. Generalizing from these systems has proven to be more difficult than advocates hoped.

The philosophical contrast is with "user-control, responsibility, and

accomplishment" Designers who emphasize a direct manipulation style believe that users have a strong desire to be in control and to gain mastery over the system. Then users can accept responsibility for their actions and derive feelings of accomplishment (Lanier, 1995; Shneiderman, 1995). Historical evidence suggests that users seek comprehensible and predictable systems and shy away from complex unpredictable behavior, such as the pilots who disengage automatic piloting devices or VCR users who don't believe that they can properly program it to record a future show.

Comprehensible and predictable user interfaces should mask the underlying computational complexity, in the same way that turning on an automobile ignition is comprehensible to the user but invokes complex algorithms in the engine-control computer. These algorithms may adapt to varying engine temperatures or air pressures, but the action at the user-interface level remains unchanged.

A critical issue for designers is the clear placement of responsibility for failures. Agent advocates usually avoid discussing responsibility. Their designs rarely allow for monitoring the agent's performance, and feedback to users about the current user model is often given little attention. However, most human operators recognize and accept their responsible for the operation of the computer, and therefore designers of financial, medical, or military ensure that detailed feedback is provided.

An alternative to agents and user models may be to expand the control-panel metaphor. Current control panels are used to set physical parameters, such as the speed of cursor blinking, rate of mouse tracking, or loudness of a speaker, and to establish personal preferences such as time, date formats, placement and format of menus, or color schemes. Some software packages allow users to set parameters such as the speed in games or the usage level as in HyperCard (from browsing to editing buttons to writing scripts and creating graphics). Users start at level 1, and can then choose when to progress to higher levels. Often, users are content remaining experts at level 1 of a complex system, rather than dealing with the uncertainties of higher levels. More elaborate control panels exist in style sheets of word processors, specification boxes of query facilities, and scheduling software that carries out processes at regular intervals or when triggered by other processes.

Computer control panels, like cruise-control in automobiles and remote controllers for televisions, are designed to convey the sense of control that users seem to expect. Increasingly, complex processes are specified by direct-manipulation programming or by graphical specifications of scheduled procedures, style sheets, and templates.

Dynamic Queries and Information Visualization

The remarkable human visual perception seems underutilized by today's graphical user interfaces. Seeing 40-60 icons on the screen seems like a modest ambition when the megapixel displays can easily show 4000-6000 glyphs and allow rapid user controlled animation on task-relevant animations (Shneiderman, 1994, 1996). Our work on the HomeFinder (Ahlberg, Williamson and Shneiderman, 1992) and the FilmFinder (Ahlberg and Shneiderman, 1994) demonstrated that users could find information faster than with natural language queries and that user comprehension and satisfaction was high for these interfaces. Double-boxed sliders enabled users to select ranges, for example the desired range of bedrooms in a house or length of a movie, and buttons allowed selection of binary attributes or from sets with small cardinality (Drama, Action, Mystery, Musical, etc.) (see Figure 1).

We applied the same sliding control for the National Library of Medicine's 50-gigabyte image database of the Visible Human (http://www.nlm.nih.gov) (north et al., 1996). Sliders over the body rapidly (less than 100 ms) reveal the slice images that are available for downloading (see Figure 2)

Medical patient histories are another difficult domain for which a user-controlled overview with dynamic queries to support zooming and filtering is proving to be effective (see Figure 3) (Plaisant et al., 1996).

Conclusion

Direct manipulation and its descendants are thriving. Visual overviews accompanied by user interfaces that permit zooming, filtering, extraction, viewing relations, history keeping, and details-on-demand can provide users with appealing and powerful environments to accomplish their tasks. I believe that most users want comprehensible, predictable and controllable interfaces that give them the feeling of accomplishment and responsibility.

References

Ahlberg, Christopher, Williamson, Christopher, and Shneiderman, Ben, Dynamic queries for information exploration: An implementation and evaluation, Proc. ACM CHI'92: Human Factors in Computing Systems, (1992), 619-626.

Ahlberg, Christopher and Shneiderman, Ben, Visual Information Seeking: Tight coupling of dynamic query filters with starfield displays , Proc. of ACM CHI94 Conference, (April 1994), 313-317 + color plates. Reprinted in Baecker, R. M., Grudin, J., Buxton, W. A. S., and Greenberg, S.

(Editors), Readings in Human-Computer Interaction: Toward the Year 2000, Second Edition, Morgan Kaufmann Publishers, Inc., San Francisco, CA (1995), 450-456.

Ahlberg, Christopher and Shneiderman, Ben, AlphaSlider: A compact and rapid selector, Proc. of ACM CHI94 Conference, (April 1994), 365-371.

Doan, K., Plaisant, C., and Shneiderman, B., Query previews for networked information services, Proc. Advances in Digital Libraries Conference, IEEE Computer Society (May 1996), 120-129.

Cypher, Allen, EAGER: Programming repetitive tasks by example, *Proc. CHI '91 Conference—Human Factors in Computing Systems*, ACM, New York (1991), 33–39.

Frohlich, David M., The history and future of direct manipulation, *Behaviour & Information Technology 12*, 6 (1993), 315-329.

Green, T. R. G. and Petre, M., Usability analysis of visual programming environments: A 'cognitive dimensions' framework, *Journal of Visual Languages and Computing 7*(1996), 131-174.

Hutchins, Edwin L., Hollan, James D., and Norman, Don A., Direct manipulation interfaces. In Norman, Don A. and Draper, Stephen W. (Editors), *User Centered System Design: New Perspectives on Human–Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1986), 87–124.

Iseki, Osamu and Shneiderman, Ben, Applying direct manipulation concepts: Direct Manipulation Disk Operating System (DMDOS), *Software Engineering Notes 11*, 2, (March 1986), 22–26.

Lanier, Jaron, Agents of alienation, *ACM interactions 2*, 3 (1995), 66-72

Maes, Pattie, Agents tha reduce work and information overload, *Communications of the ACM 37*, 7 (July 1994), 31-40.

Maes, Pattie, Artifical life meets entertainment: Lifelike autonomous agents, *Communications of the ACM 38*, 11 (November 1995), 108-114.

Maulsby, David L. and Witten, Ian H., Inducing programs in a direct-manipulation environment, *Proc. CHI '89 Conference—Human Factors in Computing Systems*, ACM, New York (1989), 57–62.

Myers, Brad A., Demonstrational interfaces: A step beyond direct manipulation, *IEEE Computer 25*, 8 (August, 1992), 61-73.

North, C., Shneiderman, B., and Plaisant, C., User controlled overviews of an image library: A case study of the Visible Human, *Proc. 1st ACM International Conference on Digital Libraries* (March 1996), 74-82.

Plaisant, C., Rose, A., Milash, B., Widoff, S., and Shneiderman, B., LifeLines: Visualizing personal histories, *Proc. of ACM CHI96 Conference* (April 1996), 221-227, 518.

Potter, Richard, Just in Time Programming, Cypher, Allen (Editor), *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA (1993), 513-526.

Rubin, Robert V., Golin, Eric J., and Reiss, Steven P., Thinkpad: A graphics system for programming by demonstrations, *IEEE Software 2*, 2 (March 1985), 73–79.

Shneiderman, Ben, Direct manipulation: A step beyond programming languages, *IEEE Computer 16*, 8 (1983), 57–69.

Shneiderman, B., Beyond intelligent machines: Just Do It!, *IEEE Software 10*, 1 (January 1993), 100-103.

Shneiderman, B., Dynamic queries for visual information seeking, *IEEE Software 11*, 6 (1994), 70-77.

Shneiderman, Ben, Looking for the bright side of agents, *ACM Interactions 2*, 1 (January 1995), 13-15.

Shneiderman, B., The eyes have it: A task by data type taxonomy of information visualizations, *Proc. IEEE Symposium on Visual Languages '96*, IEEE, Los Alamitos, CA (September 1996), 336-343.

Shneiderman, Ben, Designing the User Interface: Strategies for Effective Human-Computer Interaction: Third Edition, Addison-Wesley Publishing Co., Reading, MA (1997).

Smith, David Canfield, *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*, Birkhauser Verlag, Basel, Switzerland (1977).

Ziegler, J. E. and Fähnrich, K.-P., Direct manipulation. In Helander, M. (Editor), *Handbook of Human–Computer Interaction*, Elsevier Science Publishers, Amsterdam, The Netherlands (1988), 123–133.

Direct manipulation for comprehensible, predictable and controllable user interfaces. Full Text: PDF Get this Article.Â Osamu Iseki , Ben Shneiderman, Applying direct manipulation concepts: direct manipulation dik operating system (DMDOS), ACM SIGSOFT Software Engineering Notes, v.11 n.2, p.22-26, April 1986 [doi>10.1145/382248.382815].Â Brad A. Myers, Demonstrational Interfaces: A Step Beyond Direct Manipulation, Computer, v.25 n.8, p.61-73, August 1992 [doi>10.1109/2.153286]. 15. Figure 1: The GestureQuery user interface. Users can select a database, drag tables from the tray on to the workspace, where they can perform a series of gestural queries directly on the table data. The multitouch database interface, as shown in Figure 1, is divided into three parts: the header, tray and workspace.Â [16] B. Shneiderman. Direct manipulation for comprehensible, predictable and controllable user interfaces. IUI, 1997. [17] M. M. Zloof. Direct manipulation user interfaces have proven their worth over two decades, but they are still in their youth. Dramatic opportunities exist to develop direct manipulation pro-gramming to create end-user programming tools, dynamic queries to perform information search in large databases, and information visualization to support network database browsing.Â This strategy can lead to user interfaces that are comprehensible, predictable and controllable. Direct manipulation interfaces are seen as more likely candidates to influence advanced user inter-faces than adaptive, autonomous, intelligent agents. User control and responsibility are highly desirable.