Redirecting Direct Manipulation or What Happens When the Goal is in Front of You but the Interface Says to Turn Left?

Wai-Tat Fu & Wayne D. Gray
Human Factors & Applied Cognition
George Mason University
Fairfax, VA 22030 USA
+1 703 993 1357
[wfu/gray]@gmu.edu

ABSTRACT
The feeling of directness arises when the interface permits the user to manipulate an interface object in a way analogous to manipulating the real object. However, we argue here that the essence of direct manipulation is not *directness* per se, but *manipulation* of *task* relevant *objects* in a *task* relevant *manner*. The research reported studies users of HyperCard after 20 hours of practice. We found that when users deviated from taught strategies that 25% of the time they invented new strategies that attempted a more direct manipulation of the task object than that permitted by the design of the interface.

Keywords
direct manipulation, difference-reduction, hill-climbing, means-ends analysis

INTRODUCTION
Two decades after they were introduced, most users would agree that direct manipulation interfaces are easy to learn and to use, and few developers would consider designing anything else. As the name claims, direct manipulation allows the user to directly manipulate objects rather than issuing commands to tell the computer to manipulate objects. In the first decade, essays on "directness" abounded (e.g., see Hutchins, Hollan & Norman 1985, Shneiderman 1982). However, none of these essays resulted in a recipe for making new direct manipulation interfaces. Indeed, after two decades of use, it is clear that some direct manipulation interfaces work better than others, but we have few, if any, engineering principles to guide our designs.

In a direct manipulation interface, users act as if the representations of objects are the objects themselves. The feeling of directness arises when the interface permits the user to directly manipulate an interface object in a way analogous to manipulating the real object. However, we argue here that the essence of direct manipulation is not *directness* per se, but *manipulation* of *task* relevant *objects* in a *task* relevant *manner*. This distinction becomes important when the interface imposes a structure on the task that is not inherent in the task itself. In these cases, rather than directly manipulating *interface objects*, users may attempt to directly manipulate *task objects*.

In this brief report, we present two situations in which taking the interface object as analogous to the real object suggested a simple, difference-reduction (i.e., hill-climbing) strategy for accomplishing the task. Unfortunately, in both of these situations, the optimal solution – that which was engineered by the designer – required specialized interface objects that were not an inherent part of the task. Both solutions required direct manipulation; however, the former suggested direct manipulation of a task object, whereas the later required direct manipulation of an interface object.

This paper is an advanced report on methods our users discovered that enabled them to successfully accomplish their assigned task. From the perspective of the task, the methods invented were more direct than the simpler methods favored by the designer. When the designer's more elegant solution required them to turn left, users discovered the less elegant, straight-ahead strategy.

METHOD
Participants and Training
Students were recorded as they completed their third, three card HyperCard stack. The first stack had been created by following step-by-step procedures in their textbook (Beekman, 1991). Hence, all students had been exposed to the designer's methods for building a three card stack. At the time of the study, diaries kept by the students indicated that they had worked on HyperCard for an average of 20 hr each (in class, homework, reading, doing). At this time, we have analyzed protocols from four students who successfully completed the stack.

The Task and Procedures
Complete specifications, including sample screen prints of the completed cards, were provided for creating a HyperCard stack. Variations of the positions of the buttons, text style and font, and the user-created graphics were acceptable. Students worked at their own pace.

RESULTS

Action protocols were transcribed. Whereas 98 designer methods were required to build the stack, the users averaged 145 methods (ranging from 106 to 210). The user methods included an average of 91 of the designer methods (from 80 to 97) with an average of 54 methods that deviated from the designers methods (ranging from 13 to 116).

Across users, we identified 69 unique methods that deviated from the 98 designers methods. These methods were classified into one of four categories. Three of these categories represented various mistakes or inefficiencies that concluded with the successful completion of the designers' method. The fourth category, *difference-reduction* (also known as hill-climbing), resulted in a more direct, but more inefficient, method of accomplishing the task. Difference-reduction represented 25% (17/69) of the deviate methods.

For example, a major subtask entailed typing and centering the title on each of the three cards. The designers' method included opening the font palette, selecting *style*, selecting *center*, closing the palette. Clicking on the middle of the card, typing. In contrast, two of the four students opened the font palette, selected style, and closed the palette (without selecting *center*). They then clicked on the card, typed the title, clicked on the lasso tool, circled the words of the title, and dragged the words, centering them by eye. The students' method enabled them to work directly on the task, first reducing one difference (no text – type it), then reducing a second difference (text not centered – center it). In contrast, the designers' method, although more efficient, was less direct as the object manipulated was not a task object, but an interface object.

Of the four users, only s01 flawlessly executed the designers' methods for centering text. Compared to s01, the two users – s02 and s03 – who used the difference-reduction strategy required 176% and 268% more time to type the titles on the three cards. The difference-reduction strategy required more error-prone actions. Both s02 and s03 spent 13.4% and 29.7%, respectively, of their time in error recovery. Although the strategy was time-consuming and error-prone, both s02 and s03 consistently used this method throughout the three cards.

A second example deserves a brief mention. Two of the cards required labeled buttons. The designer method for labeling a button entails opening the button palette, typing a button name in the field provided, and clicking on a checkbox labeled "show button name." An alternative difference-reduction method attempted by two of the users, entailed attempting to type directly on the button object – not in the button palette. Making this approach work requires typing the label on the card (not on the button – that is impossible in HyperCard), making the button transparent, and moving the button so that it is on top of the label. This method requires many more steps than the designer's method. However, it allows the user to directly manipulate task objects – text and buttons – rather than an interface object – the palette.

DISCUSSION

HyperCard is a flexible tool that allows for many ways of accomplishing the same task. The difference-reduction methods, invented by the users, were much more effortful than the designer's methods. Their apparent advantage is that they permit the user to directly manipulate a task object (e.g., the text being centered or the button being labeled) rather than a purely interface object (the font palette or the button palette). When the task object was in front of them, these users preferred to go straight rather than turning left as per the designer method. Unfortunately for our users, they could not beat the designer. In the cases we have studied, the shortest distance between two points was not a straight line, but a zig. The hill-climbing strategy brought the users to a local minimum and they ended up spending more effort than necessary to finish their tasks. Although the designers' methods were direct manipulation methods, they directly manipulated the interface but only indirectly manipulated the task.

These results yield a suggestion for redirecting direct manipulation. To the extent that the tool follows the structure of the task, then direct manipulation is fine. However, to the extent that the tool imposes a structure that the task must follow, then direct manipulation may misdirect users (Gray, 1998). Our proposed redirection emphasizes the distinction between the device space and the task space that has been noted by other (e.g., Payne, Squibb, & Howes, 1990). Direct manipulation works by providing an interface that is *transparent* to the user. A transparent interface should provide an environment in which the users do not feel that they are using the device, but are directly accomplishing their tasks.

ACKNOWLEDGMENTS

The work on this paper was supported by a grant from the National Science Foundation (IRI-9618833) to Wayne D. Gray.

REFERENCES

1. Beekman, G. (1991). *HyperCard 2 in a hurry*. Belmont, CA: Wadsworth Publishing Company.
2. Gray, W. D. (1998). *Cognitive reverse-engineering of a simple, rule-based task: Performance, errors, error detection and correction* (ARCH Lab Report 98-08/06).
3. Hutchins, E. L., Hollan, J. D. & Norman, D. A. (1985). Direct Manipulation Interfaces. *Human Computer Interaction 1, 311 – 338.*
4. Payne, S. J., Squibb, H. R., & Howes, A. (1990). The Nature of Device Models: The Yoked State Space Hypothesis and Some Experiments with Text Editors. *Human-Computer Interaction, 5*(4), 415-444.
5. Shneiderman, B. (1982). The future of interactive systems and the emergence of direct manipulation. *Behavior and Information Technology, 1*, 237 – 256.