COGNITIVE ASPECTS OF DIRECT-MANIPULATION INTERFACES

Jörg Beringer

Technische Hochschule Darmstadt
Institut für Psychologie
Hochschulstraße 1
6100 Darmstadt, West Germany

Abstract

A central purpose of human computer interfaces is to support the development of adequate conceptual and syntactic knowledge about the system and to stimulate and to support the learning of efficient sequences of actions. Among several other user interfaces direct manipulation interfaces have been of growing interest. Especially the permanent visualization of objects and the use of a pointing device are characteristic features of this kind of interface. Do direct manipulation interfaces have an advantage over traditional dialogue techniques with respect to learnability and usability? Quoting several products as an example, the advantages and limits of this kind of interface will be discussed.

1. INTRODUCTION

The design of user interfaces in terms of ease of use and ease of learning is of growing interest. The progress in computing and graphic capabalities of computer devices, and the widespread employment of such devices in our working life calls for sophisticated knowledge about human-computer interaction. Efficiency of usage certainly depends on the successful integration of technical and human aspects. This paper attempts to summarize aspects of the so-called direct-manipulation interface, which are relevant under the ergonomic point of view.

Besides various types of interface-techniques like menu-selection, form-filling, command-language, and natural-language, the direct-manipulation technique has become an independent class of user-interface of growing popularity. Many products, as for example the DEC-Vaxmate and the VAX-work-stations, implement direct-manipulation interfaces not only for applications (e.g. Interleave), but also for the operating system (MS-Windows) and for the management of multiple parallel processes (UIS-Window Manager, X-Windows).

Although this type of interface is often claimed to be easy to use and easy to learn, there are only a few systematic descriptions and empirical studies trying to characterize and to evaluate the direct-manipulation interface.

According to *Shneiderman* (1983) the most essential features of direct-manipulation are:

- Continuous representation of the objects in interest.
- Physical actions or labeled button presses instead of complex syntax.
- Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

These features maximize the directness of interaction and the degree of direct engagement with the work-object (*Hutchins, Hollan & Norman*, 1986). The interface simulates a world (real world metaphor) rather than being an I/O-device (conversational metaphor).

However, the question is which of these features add to the ease of use and which of them are unique for direct-manipulation. After a concise overview of the typical interaction techniques within direct-manipulation interfaces, the implications of the direct-manipulation technique in terms of cognitive aspects will be discussed.

2. INTERACTION TECHNIQUES OF DIRECT-MANIPULATION INTERFACES

The following interaction principles are usually implemented in direct-manipulation interfaces:

- Continuous Visualization
- Simple locating and selection techniques
- Object-oriented interaction
- Function-objects
- Generic commands and operations
- Metaphor
- Window technique

2.1 Continuous Visualization

An essential feature of direct-manipulation is the permanent visualization of relevant objects (*Shneiderman*, 1983). One of the most typical examples is the visualization of files, directories, and drives in form of icons (e.g., GEM and Interleave). Another example is the UIS-Window Manager which permanently visualizes processes or applications in separate windows. All these interfaces have in common that system internal states are parallelly represented on the external user interface. The degree of visualization differs considerably. Properties of objects, available functions, and object-function relations are only partly or not at all visualized.

2.2 Simple Location and Selection Techniques

In contrast to customary command language interfaces, the user of a direct-manipulation interface communicates to a great extent by pointing and dragging. On the screen an object can be selected and activated by pointing at it with a analogous input device. Switching between windows can be done by simply moving a mouse cursor to the next window.

The pointing device works asynchronously with the ongoing process which allows a state-independent usage of the pointing device while on the system side the interpretation of the asynchronous input can be done in a context dependent way. The efficiency of the pointing device, however, depends completely on which objects are displayed on the screen and which functions are mapped to the movement and buttons of the pointing device.

Due to constraints in space and resolution, many objects and functions can only be selected or activated via menus. The small number of buttons (one, two, or three) and the limited possibilities of movement-function mappings (see also section 2.4 and 2.5) forces the user to specify many operations by selecting menu-options instead of using the pointing device directly. The MS-DOS Executive of MS-Windows, for example, integrates only the function of activation and of changing a directory in its pointing device. All remaining operations on files and directories must be activated via pull-down menus. This limited usage of the direct pointing technique is partly due to the fact that MS-Windows does not allow to move files across the screen.

2.3 Object-oriented interaction

The typical order of command specification is the selection of an object followed by the specification of the desired operation. To delete a file, the user first selects the file and then chooses the delete option out of the pull-down menu (MS-Windows) or drags the file-icon to the waste paper basket icon (GEM). This order of command specification is just the opposite to command language interfaces where the operation usually leads the ob-

ject specification (e.g., delete *.*). The difference between prefix and postfix notation in terms of ease of use seems to be not significant (*Beringer*, 1986; *Cherry*, 1986), but the inconsistent implementation of command order within one application adds to the complexity of the interface (*Barnard, Hammond, Morton, Long, Clark*, 1981).

2.4 Function-object

Due to constraints in possible mappings of functions onto the mouse buttons and types of mouse movements some direct-manipulation interfaces introduce additional function-objects (*Rosenberg & Moran*, 1985). A function-object, for example the waste paper basket in GEM, converts the function 'delete' into an object, so that moving an icon to the function-object means deleting the file presented by the icon. The regular meaning of moving an icon, which might be copying, is modified to a delete operation. The introduction of function-objects results in a larger set of operations which are implemented within an uniform interaction technique.

2.5 Generic commands and operations

The functionality of a system can be represented by listing the objects which can be manipulated and the operations which are provided by the system to manipulate these objects. For example, a word processor offers the possibility to delete, insert, cut and paste words, sentences, and paragraphs. According to *Rosenberg & Moran* (1985) the provided functions of a system can be classified according to their degree of generality in combining them with objects. A command is called generic if it can be combined with all objects. The introduction of generic commands minimizes the syntactic knowledge required to use the system.

Besides the degree of generality, one has also to consider how much the meaning of an operation changes depending on which object it is combined with. *Rosenberg & Moran* call this the degree of object-centeredness.

Direct-manipulation interfaces try to implement as many generic commands as possible for supporting the simplicity of the pointing and selection technique. Copying, deleting, activating, and renaming are typical candidates for generic commands.

The context independent implementation of a pointing and selection operation can also be seen as a generic command but the generic part is restricted to the physical operational level. Although the same input operation can be applied to many objects represented on the screen, the activated function can be completely different depending on the object. Thus, direct-manipulation interfaces have a one to many mapping between input operations and activated system functions.

2.6 Metaphor

Direct-manipulation interfaces usually make extensive use of metaphors. Objects and functions of a system such as files, directories, tools and so on are visualized as icons which resemble well-kown objects of our working-life. Naming conventions make also heavy use of metaphors. A user of a direct-manipulation system does not copy one file from one directory to the other, but moves a sheet of paper from one folder to another folder.

Metaphors are not restricted to icons and naming conventions. The pointing and selection technique itself provides the possibility to simulate operations in a spatial analogous way, i.e., arranging documents on a desktop or continuously changing the color pallette by dragging a virtual button on a panel.

2.7 Window Technique

Although windowing is not a necessary feature of direct-manipulation interfaces, many aspects of direct-manipulation are inherent to windowing.

Windowing supports the visualization of complex parallel processes (see section 2.1). Abstract concepts like processes, host-computers or active applications are represented as objects which can be directly manipulated (UIS-Window Manager). Second, windows provide a well-defined spatial-dependent interpretation of the asynchronous pointing device, thus supporting the universal usage of generic input operations (see section 2.5). Window management operations are mainly spatial in nature. Therefore spatial input operations, as changing size, moving, and zooming, can be implemented in an optimal analogous way, thus adding to the feeling of direct-manipulation (see section 1).

3. USER AND USER ACTIVITY

The evaluation of direct-manipulation interfaces with respect to cognitive ergonomics requires at least a coarse modelling of user behavior and user knowledge.

3.1 User action

Based on various models of user behavior (*Moran* 1981; *Norman* 1986) the human-computer interaction process can be described as a hierarchical cycle of action-planning and action-evaluation (see figure 1). While the planning process transforms abstract intentions into concrete physical operations (execution), the evaluation process yields the interpretation of physical feedback of a system with respect to the user's intention.

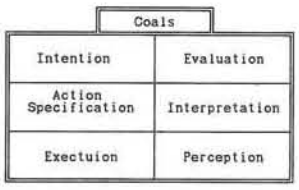

Figure 1:
User activity as modelled by Norman (1986).

In the course of action planning one can distinguish between the semantic level, which contains intentions and subgoals, the action specification level which maps intentions onto the system dependent dialogue language, and finally the actual execution of the input.

Following the execution the user must interpret the state of the system to decide if he has achieved his goal. Before evaluating the outcome of the action, the physical feedback must be transformed into the user's conceptual level.

Besides illustrating the different levels of abstraction of action planning and evaluation, *Norman's* model of user activity stresses also the fact that the transformation across these levels represents an additional effort for the user interacting with the system (*Hutchins, Hollan & Norman* (1986).

3.2 User knowledge

There is a variety of attempts to model the user's mental representation of a system and of the interaction with it (ACT*-Theory by *Anderson*, 1983; GOMS-Model by *Card, Moran & Newell*, 1983; Cognitive Complexity Theory by *Kieras & Polson* 1985; Task-Action Grammar by *Payne*, 1985). These different theoretical approaches will not be discussed further. Instead the user's knowledge will be illustrated corresponding to the levels of user activity as illustrated in figure 1, i.e., it is assumed that the mental representation of the system consists of knowledge about the conceptual level and knowledge about the syntactic level of the system (see figure 2).

The user must know about the overall functionality of the system, e.g., the fact that he can make goodlooking plots with a graphic programm or that he can 'talk' to a friend via a phone utility. Further, he needs to know in more detail what functions and objects the system offers for fulfilling a task. Then he must be able to specify his intentions in terms of the system's interaction language.

Additionaly, many papers (for an overview see *Bösser*, in press) stress the distinction between task-knowledge and tool-knowledge. While task-knowledge is the user's system-independent knowledge about the task, e.g., how to make a business report, tool-knowledge represents system-specific knowledge, e.g., how to use Interleave to write a report. These two sources of knowledge provide the necessary concepts and procedures for planning and performing the interaction.

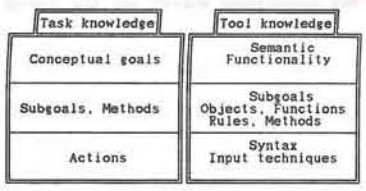

Figure 2: Components of the user's knowledge.

3.3 Syntactic level

Within the action cycle the syntactic level depends on the dialogue language of a system which can be more or less complex. Syntactic knowledge must be seen as the key or the barrier to a system.

Karat, Boyes, Weisberger & Schafer (1986) observed that in spite of a considerable conceptual similarity between two systems, users had difficulties when switching to the other system only because of small syntactic differences. Conceptual knowledge can apparently not be applied without syntactic knowldege.

There are also studies which show that the learning of adequate concepts is strongly supported by active use of the system (*Carroll & Mack*, 1985; *Hiltz & Kerr, 1986; Kamouri, Kamouri & Smith*, 1986). Thus, syntactic knowledge is a prerequisite for effective conceptual learning.

3.4 Conceptual level

Although the knowledge of the syntax of a system is very important and can be a barrier for beginners or casual users, the building of adequate concepts about the system is the key to an efficient way of using the system. Whether and in which way a user decides to use a system depends on his knowledge about the functionality of the system. In terms of figure 2 the user scans in the conceptual level of his tool knowledge for functions which can help him to achieve the goal that is represented in the conceptual level of his task knowledge.

Furthermore, the overall efficiency of action planning and evaluation increases if the user has appropriate tool-specific concepts. Concepts do not only represent declarative knowledge but they are also parts of procedural knowledge. *Payne & Green's* set-grammar (1983) stresses the fact that concepts are an important part of their rules. In terms of production systems (*Kieras & Polson*, 1985) conceptual knowledge determines both the left- and right-hand side of a production.

Thus, the development of efficient methods, procedures, and rules depends on the syntax of the system as well as on the semantic concepts and the mapping between both. If the user had no idea of the meaning of insert and overwrite mode in the context of a word processing system, he would never be able to build a production of the form: if the goal is to add a character and there is no space for it then go into insert mode. He could not recognize the state of insert and overwrite, either. It is therefore evident that user interfaces should strongly stimulate and support the learning of adequate concepts.

4. ERGONOMIC ASPECTS OF DIRECT-MANIPULATION INTERFACES

4.1 Syntactic simplicity

Direct-manipulation interfaces implement only a few generic input operations, i.e., moving, pointing, dragging, pressing mouse buttons (see section 2.5). According to *Payne's* task-action grammar (1985) this type of syntax can be generated by only a small set of rules and terminal symbols. Therefore, the syntactic level of direct-manipulation interfaces should be easy to learn because it consists of less rules and terminal symbols (mouse activities) than for example a formal command language.

Furthermore, syntactic knowledge can be transferred quite easily to new applications which use the same input technique. In terms of the diagram presented in figure 2, the tool specific syntactic knowledge can be transferred to other tools if they are using the same pointing and selection technique.

However, two caveats about the simplicity of direct-manipulation interfaces must be mentioned. First, direct-manipulation interfaces almost always consist of a mixture of input techniques, i.e., pointing, menu selection, form filling, and even command language. For example, MS-Window's MS-DOS Executive has only a very small set of functions which can be activated by simple pointing operations. If the user wants to copy a file, he is confronted with the DOS-specific syntax for specifying the target directory in a dialogue box.

To give another example, the UIS-Window Manager offers simple pointing operations only in the domain of window management functions, i.e., moving, shrinking, switching windows. Other functions must be activated via pop-up menus. Although the menu options are selected with the pointing device, the direct-manipulation component of the interaction is reduced. When the user activates the help system , he is confronted with the keyboard-based VMS-Help Utility.

Second, the generic operations represent only the very basic part of the entire syntax of a system. For example, given the knowledge about the window-management operations and about how to select menu options within the UIS-Window Manager, the first step into a new application might be facilitated for a user. However, it still remains to learn how to tell the system to insert a footnote in a word processing system or how to rotate a figure in a CAD-system.

4.2 Self-explanatory and natural syntax

The pointing and selection technique is often described as a very natural technique which is supposed to be self-explanatory. Correspondingly, the on-line help for pointing techniques is extremely rare.

Fähnrich & Ziegler (1985) compared the performance of experts in direct-manipulation with users which were also computer experts, but unexperienced in direct-manipulation techniques. They found a significant advantage for subjects who were experienced with direct-manipulation.

Obviously the user needs specific syntactic knowledge about the direct-manipulation technique. Neither can syntactic task-knowledge be applied directly to action planning nor can syntactic tool-knowledge which was acquired in the context of other interface techniques. Even syntactic knowledge which was acquired using another direct-manipulation system can cause negative transfer if there are slight differences in syntax and presentation techniques (*Miller, Hill, Kendree, Masson, Blumenthal, Terveen & Zaback*, 1987).

4.3 Visualization

Visualization supports not only the interpretation and evaluation of action but also the building of tool-specific conceptual knowledge (see figure 2). Visualization is one of the most important design principles independently of the learning phase. It can potentially support the building of concepts in an optimal way. By visualization the user gets implicit help in learning the relevant key concepts of a system. Permanent visualization also implies immediate feedback which supports the learning of syntax and concepts by exploration.

The power of supporting the building of tool-specific concepts and functions, however, depends on the concrete design of the interface (*Wandmacher*, 1986). The object-centeredness of many direct-manipulation interfaces causes functions and object-properties to be less visualized. For example, available functions are often visualized in form of short and sometimes misleading menu labels. The visualization of the MS-DOS Executive is limited to the level of one directory plus icons which represent the available drives. The location of the active directory is only visualized in a very formalized way, i.e., the MS-DOS syntax.

Although the UIS-Window Manager permanently visualizes the activated processes and applications, it uses the same window design for each window, no matter whether it is a graphic output window, a VT220 terminal emulation window or any other kind of window. Thus, underlying conceptual and functional differences, e.g., input vs output window, local vs host process, type of emulation, are not made as transparent as it would be possible.

4.4 Metaphor

Direct-manipulation interfaces explicitly use metaphors for presenting and naming objects. The desktop metaphor with its folders, papers, waste-paper baskets is probably the most popular example. Metaphors are used to make the system appear familiar to the user and to help him in understanding the meaning of objects and functions.

The adequateness of metaphors, hoewever, is often very limited. A sentence as "your screen is like your desktop" illustrates the obvious limits of such a comparison. As *Halasz & Moran* (1983) mentioned, metaphors sometimes seem to be rather harmful than helpful. Nevertheless, there is a surprising flexibility in excluding those elements of the original domain of the metaphor (i.e., the desktop), which do not correspond with the elements of the target domain (i.e., the system) as for example the fact that a table has four legs.

With respect to this flexibility *Carroll & Mack* (1985) argue that the real meaning of metaphors is not to induce a complete transfer of the original domain of the metaphor but to give beginners a first sense of the system. By testing the appropriateness of a metaphor and by detecting the differences between the source and the target domain of the metaphor, the user will actively deduce adequate concepts.

In terms of the diagram presented in figure 2 metaphors draw the user's attention to a specific part of his task- or tool-knowledge which he can then use as the source domain for analogical reasoning. The knowledge elements which are transferred may belong to any of the knowledge components. A metaphor like "the computer is like a typewriter" means that the predominate form of interaction is typing on a keyboard. Thus, this metaphor aims at the interaction level. In contrast, the metaphor "a directory is like a folder" clearly aims at the functionality of the directory, i.e., it can contain a set of items. Even multiple use of different metaphors can help to build the adequate concepts by modifying and combining the partial models (*Rumelhart & Norman*, 1981).

However, metaphors seem to be helpful in acquiring tool-specific knowledge only in the very beginning of learning. After an initial learning phase, metaphors become unnecessary and in fact obstructive. For a discussion of this aspect see *Wandmacher* (1987).

It becomes obvious that the permanent implementation of metaphors in user interfaces may decrease the long-term efficiency of usage, except if the system is consequently designed to fit the metaphor (*Benest, Morgan & Smithurst*, 1987), but this usually cannot be achieved .

4.5 Conceptual flexibility

The principle of visualizing all objects in interest inherently determines the conceptual model of the system. The conceptual model of the system must meet the user's expectations. It must be task adequate and flexible in task handling. In comparison to other types of interfaces todays direct-manipulation interfaces are not very flexible with respect to their conceptual model.

The MS-DOS Executive of MS-Windows is a striking example of this limited flexibility. The conceptual model of the operating system is restricted to the level of one directory containing a set of files. This might be satisfying for activating programs but as an interface to an entire operating system the visual presentation and the functionality of the MS-DOS Executive is too limited.

The level of perspective must be more flexible both horizontally and vertically. In order to find duplicates on a harddisk the user should have a file listing across directories, or for organizing the harddisk the user should see the entire directory structure of an harddisk. On other occasions, the detailed information about one single file may be relevant.

These simple examples illustrate that in order to meet the user's expectations and the task requirements, the conceptual flexibility of many direct-manipulation interfaces must be improved.

Another aspect is the functional flexibility. Usually the set of provided functions is not flexible at all. However, the interface should be adaptable to meet the individual needs of advanced users, e.g., programming new function-objects or hooking-in new menus or menu options.

A third and more pragmatic aspect of flexibility is the capability of individual tuning of an interface. Imagine a secretary whose task is writing reports by using a word processor and managing the print process as well as the weekly back-up of files. Instead of learning the entire operating system the secretary could get an individually tuned interface which provides only the functions and objects relevant to these tasks. If the direct-manipulation system does not offer the tuning, a non direct-manipulation system, e.g., a flexible menu interface or command language interface could be the better choice.

Although the development of direct-manipulation interfaces is by far not settled, pure direct-manipulation interfaces seem to be more adequate for limited task domains (*Fähnrich & Ziegler*, 1985). In more complex task domains interfaces will always have to resort to other interaction techniques.

With respect to the learning vs using issue one can state that the fixed build-in conceptual model of many direct-manipulation interfaces might be helpful for the beginner but advanced users need a more flexible and more powerful conceptual model.

5. SKILLED BEHAVIOR

While in the beginning the interaction with a new system is rather a problem solving process, experienced users can accomplish many tasks or subtasks by routine. The expert's tool-specific knowledge contains well-suited procedures for performing standard tasks, and the tool knowledge is highly integrated into the task knowledge.

How well a user can develop skills depends on the task adequateness or semantic directness, i.e., the system must conceptually correspond to the problem space of the task (*Hutchins, Holland & Norman*, 1986). Furthermore, consistency within and between the syntactic and the conceptual levels determines the building of action schemes (*Moran*, 1983; *Payne*, 1985; *Payne & Green, 1983*).

How well a user is supported by the direct-manipulation technique to utilize his skills will be discussed in the next two sections.

5.1 Open loop performance

Skilled action differs from problem solving in that the units of action which are planned in advance become larger. The expert can internally represent the systems main characteristics and thus can simulate and predict the system's responses (*Norman*, 1983). In contrast to the beginner, the advanced user does not require continuous feedback. Instead, he or she will check the system's state only at specific moments. A more or less long sequence of actions can be planned and executed without the evaluation phase. With advanced usage the interaction activity changes from closed-loop behavior (see figure 1) to open-loop behavior (*Green, Payne, Gilmore & Mephan*, 1985; *Rasmussen*, 1983).

The special interaction technique used in direct-manipulation interfaces constrains open-loop activity. The pointing and selecting technique forces the user to permanently monitor the display and to wait for the system's response. The user cannot enter a sequence in advance but is forced to synchronize his pointing with the current presentation of objects and menus.

Although an entire sequence of actions might be formulated on the conceptual level, the flow of action is interrupted at the interaction level. This side-effect of direct-manipulation interfaces becomes intuitively clear if one imagines the task of repetetively copying a file between directories by using a pointing device within MS-Windows. First, there is no way to short cut the input sequence, because the file has to be selected each time and the pop-up menu must be popped up each time. Second, during each repetition the user has to monitor the screen for locating the file icon and the menu option.

It becomes obvious that interfaces cannot rely on pointing techniques only if they want to be suitable for advanced users.

Optional input media as keyboard or speach and the flexible spontaneous building of macro functions are suggested.

5.2 Levels of feedback

With regard to its restrictive effects on open loop behavior the attractivity of the pointing technique and its frequent usage in interfaces which also offer keyboard input (e.g., MS-Windows) is surprising. For menu selection tasks *Karat, McDonald & Anderson* (1985) found that the mouse device has no advantage over keyboard input either in terms of attitude nor in terms of performance.

One reason for the preference of a pointing device can be the fact that generation of mouse input is in some way less effortful than the generation of keyboard commands (e.g., typing the first letter of a menu option).

Investigations of stimulus-response compatibility reveal that position of a stimulus is automatically encoded even if it is not important (*Craft & Simon*, 1970). From this it follows that after succesfuly scanning a screen or a menu for a specific item, the user implicitly knows the position without further cognitive processing. He can immediately start to locate the item. Typing in the first letter of the item's label probably requires additional processing because the label is not automatically represented in form of its letter constituents. Although the following locating sequence may take longer than the execution of the keyboard command, users apparently prefer the input device which is less effortful in terms of planning.

There also seems to be a systematic difference in the amount of interference between planing low level input and performing the task. *Rasmussen* (1983) postulates different types of information which differ in their semantic depth of encoding. The most simple type is the signal which only represents time or physical data. Signals serve as feedback for sensory-motor performance. Signs serve to activate well-learned procedures or action schemes. Symbols are abstract and are related to external events.

The different types of feedback also imply different amounts of interference. Positioning a mouse cursor on an item on the screen belongs to the lowest type of information. Positioning is an activity which does not involve any semantic reference to the ongoing task. Therefore, a positioning sequence can be performed without causing any semantic interference. If the interaction with the system is rather knowledge-based (i.e., problem solving), the user maybe prefers the direct pointing technique because it interfers less with the ongoing conceptual planning.

There is some evidence by a study which investigated performance and preferences with respect to different dialogue techniques (*Eberleh & Korffmacher & Streitz*, 1987). In speed conditions subjects prefered commands. Commands did interfere with secondary calculation tasks more than direct drawing did whereas direct drawing interferred more with secondary spatial tasks than commands did. Although these results are not significant, they are in line with the argumentation stated above.

6. DISCUSSION

The direct-manipulation technique will always be only a part of an entire interface. If the interface exceeds a moderate functionality, the implementation of other dialogue techniques seems to be unavoidable. As we have seen, so-called direct-manipulation interfaces as MS-Windows have implemented only very few direct-manipulation components. The main part of the interaction is done by using menus and dialogue-boxes. The fact that menu options can be selected by a pointing device is not sufficient to become a direct-manipulation interface. If the real-world metaphor is not completely realized in the sense of *Hutchins, Hollans & Norman* (1986), it seems more accurate to speak of a direct-manipulation technique and not of a direct-manipulation interface. The main characteristic of direct-manipulation techniques is the visual presentation of object-like items which can be directly manipulated and which provide a task adequate mapping of functions to spatially analogous input operations.

When evaluating direct-manipulation interfaces it becomes obvious that this type of interface is not inherently easy to use or easy to learn. Looking at todays products, one can speculate that a direct-manipulation interface offers a relatively fast access to beginners, but the asymptotic level of long-term performance will be worse in comparison to well designed command interfaces because of the dominance of the pointing technique, the fixed implemented metaphors and the limited conceptual and functional flexibility.

Many interface characteristics which support learning such as visualization, consistency, generic commands, task-adequate conceptual structure and visualization can be implemented in non direct-manipulative interfaces as well. Valid investigations about short-term and long-term usability of interfaces using direct-manipulation techniques are not available. These studies compared interfaces using direct-manipulation or command techniques which were designed in a completely different way with respect to many characteristics listed in this paper.

Karat, Fowler, & Gravelle (1987) compared a direct-manipulation prototype with the MS-DOS command language but the prototype for example visualized the entire directory structure. It is no surprise that the prototype revealed better performance, but it is surprising that they related this advantage to direct-manipulation although other things were not equal. Results from our own laboratory show that a simple menu oriented interface can give better results than MS-DOS and GEM when beginners learn to master a specified set of operating system functions (*Arnold, Hoffman & Prottung, 1986*).

Literature

Anderson, J.R. (1982). Acquisition of cognitive skill. Psychological Review, 89, 369-406.

Arnold, R., Hoffmann, B., & Prottung, S. (1986). Performanzvergleich bei ungeübten Benutzern unter einer kommandosprachlichen, einer menüorientierten und einer Benutzeroberfläche der direkten Manipulation. Darmstadt: Fachbereich Informatik und Institut für Psychologie der THD (Unveröffentlichte Studienarbeit).

Barnard, P.J., Hammond, N.V., Morton, J., Long, J.B., and Clark, I.A. (1981). Consistency and compatibility in human-computer dialogue. International Journal of Man-Machine Studies, 15, 87-134.

Benest, I.D., Morgan, G., & Smithurst, M.D. (1987). A Humanized interface to an electronic library. Human-Computer Interaction - Interact'87, H.J. Bullinger & B.Shackel (Eds.), North-Holland.

Beringer, J., (1986). Transfer einer Menüstruktur. Darmstadt: Institut für Psychologie der THD (Unveröffentlichtes Manuskript).

Bösser, T. (in press). Learning in Man-Computer Interaction. A critical review of the literature. To appear in Springer Lecture Notes.

Card, S.K., Moran, T.P., & Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Carroll, J.M., & Mack, R.L. (1985). Metaphor, computing system, and active learning. International Journal of Man-Machine Studies, 22, 39-57.

Cherry, J.M. (1986). An experimental evaluation of prefix and postfix notation in command language syntax. International Journal of Man-Machine Studies, 24, 365-374.

Craft, J.L. & Simon, J.R. (1970). Processing symbolic information from a visual display: Interference from an irrelevant directional cue. Journal of Experimental Psychology, 83, 415-420.

Eberleh, E., Korffmacher, W., & Streitz, N.A. (1987). Denken oder Handeln: Zur Wirkung von Dialogkomplexität und Handlungsspielraum auf die mentale Belastung. In W.Schönpflug & M.Wittstock (eds.), Software-Ergonomie '87, B.G. Teubner: Stuttgart.

Fähnrich, K.-P., Ziegler, J. (1985). Workstations using direct manipulation as interaction mode - aspects of design, application and evaluation. In B. Shackel (ed.), IFIP conference on Human-Computer Interaction - Interact'84, North-Holland-Amsterdam, 693-698.

Green, T.R.G., Payne, S.J., Gilmore, D.J., & Mephan, M. (1985). Predicting Expert Slips. In B. Shackel (ed.), IFIP conference on Human-Computer Interaction - Interact'84, North-Holland-Amsterdam, 519-525.

Halasz, F. & Moran, K. (1983). Analogy considered harmful. Proceedings of Human Factors in Computing Systems, March 15-17, 1982, Gaithersburg, Maryland (pp. 383-386). Washington, DC: ACM.

Hiltz, S.R. & Kerr, E.B. (1986). Learning modes and subsequent use of computer-mediated communication systems. CHI'86 Conference Proceedings: Human factors in computing systems, Boston, 149-155.

Hutchins, E.L., Hollan, J.D., & Norman, D.A. (1986). Direct manipulation interfaces. In D.A. Norman and S.W. Draper (Eds.), User centered system design. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Kamouri, A.L., Kamouri, J., & Smith, K.H. (1986). Training by exploration: facilitating the transfer of procedural knowledge through analogical reasoning. International Journal of Man-Machine Studies, 24, 171-192.

Karat, J. Boyes, L., Weisberger, S., & Schafer, C. (1986). Transfer between word processing systems. CHI'86 Conference Proceedings: Human factors in computing systems, Boston, 67-71.

Karat, J., Fowler, R., & Gravelle, M. (1987). Evaluating user interface complexity. Human-Computer Interaction - Interact'87, H.J. Bullinger & B.Shackel (Eds.), North-Holland.

Karat, J., McDonald, J.E., & Anderson, M. (1985), A comparison of selection techniques: touch panel, mouse and keyboard. Human-Computer Interaction - Interact'84, B. Shackel (Ed.), North-Holland.

Kieras, S. & Polson, P.G. (1985). An approach to the formal analysis of user complexity. International Journal of Man-Machine Studies, 22, 365-394.

Miller, J.R., Hill, W.C., McKendree, J., Masson, M.E.J., Blumenthal, B., Terven, L., & Zaback, J. (1987). The role of the system image in intelligent user assistance. Human-Computer Interaction - Interact'87, H.J. Bullinger & B.Shackel (Eds.), North-Holland.

Miller, J.R., Hill, W.C., McKendree, J., Masson, M.E.J., Blumenthal, B., Terveen, L., & Zaback, J. (1987), The role of the system image in intelligent user assistance. Human-Computer Interaction - Interact'87, H.J. Bullinger & B.Shackel (Eds.), North-Holland.

Moran, Th.P. (1981). The command language grammar: a representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies, 15, 3-50.

Moran, Th.P. (1983). Getting into a system: External-internal task mapping analysis. CHI'83 Conference Proceedings: Human factors in computing systems, Boston, 45-49.

Norman, D.A. (1983). Some observation on mental models. In D. Gentner & A.L. Stevens (Eds.), Mental Models, Hillsdale, New Jersey: Erlbaum.

Norman, D.A. (1986). Cognitive engeneering. In D.A. Norman and S.W. Draper (Eds.), User centered system design. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Payne, S.J. & Green, T.R.G. (1983). The user's perception of the interaction language: A two-level model. CHI'83 Conference Proceedings: Human factors in computing systems, Boston, 202-206.

Payne, S.J. (1985). Task-action grammars. In B. Shackel (ed.), IFIP conference on Human-Computer Interaction - Interact'84, North-Holland-Amsterdam, 527-532.

Rasmussen, J. (1983). Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. IEEE Transactions on Systems, Man and Cybernetics, 13(3), 257-266.

Rosenberg, J.K. & Moran, T.P. (1985). Generic commands. In B. Shackel (ed.), IFIP conference on Human-Computer Interaction - Interact'84, North-Holland-Amsterdam, 245-249.

Rumelhart, D.E. & Norman, D.A. (1981). Analogical processes in learning. In J.R. Anderson (Ed.), Cognitive skills and their acquisition. Hillsdale, New Jersey: Erlbaum.

Schneiderman, B. (1983). Direct Manipulation: A step beyond programming. IEEE Computer, 16 (8), 57-69.

Wandmacher, J., (1986) Software-Ergonomie I. Darmstadt: Institut für Psychologie (Unveröffentlichtes Manuskript zur Vorlesung).

Wandmacher, J. (1987). Schema- und handlungstheoretische Grundlagen der Mensch-Computer-Interaktion. Forschungsbericht, Darmstadt: Institut für Psychologie.