INTEGRATED INFORMATION MANIPULATION SYSTEMS (IMS) -- A COGNITIVE VIEW

Gerhard Fischer
Man-Machine Communication Research Group
Institut fuer Informatik
Universitaet Stuttgart
Stuttgart, W-Germany

Abstract

The personal computer of the future will offer its owner an **information manipulation system** (IMS). It will be a totally integrated system being able to manipulate arbitrary information structures, eg programs, prose, graphical objects and sound.

An IMS will be an important step towards achieving the goal that we can do all our work **on-line** -- placing in computer store all of our specifications, plans, designs, programs, documentation, reports, memos, bibliography and reference notes and doing all of our scratch work, planning, designing, debugging and most of our intercommunication via the consoles.

We outline the basic principles underlying the design of an IMS. We discuss the cognitive dimensions (specifically for text processing and programming systems) which should serve as the design criteria for systems whose goal is to reduce the cognitive burden and augment the capabilities of a human user.

Keywords

man-machine communication, problem solving, routine cognitive skill, text processing, programming, display-oriented interfaces, uniformity, integrated systems

1. Information manipulation systems (IMS)

1.1 Function and structure of an IMS

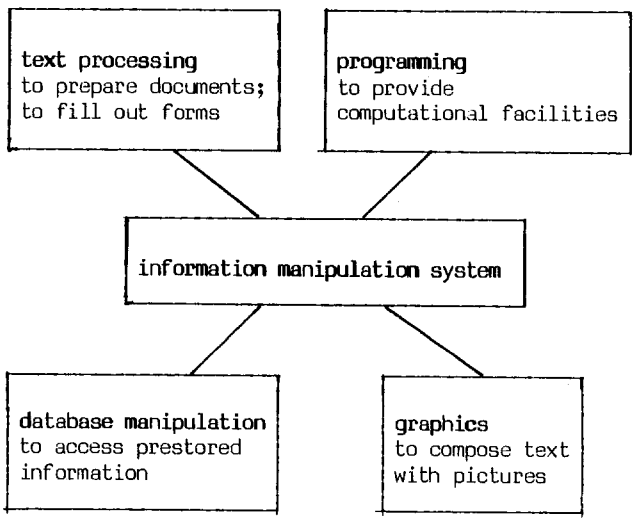
The rapidly increasing sophistication and cheap availability of computers make it likely that interactive man-machine systems will increasingly be exploited to deal with complex problems in many domains. IMSs should be prototypes for systems in which the computer and the human cooperate to solve problems and achieve tasks more quickly and more rapidly than either could do working alone.

The effective utilization of such combined man-machine systems will require that the **information-processing capabilities of the human component** be as well understood and designed as those of the computer.

The **basic hardware** to support an IMS consists of a personal computer dedicated to a single user which will have a high resolution, all points addressable display and a mouse as a pointing device. Individual machines will be connected in a network and they will possess computational power comparable to that of todays largest timesharing-machines. The LISP Machine (WEINREB & MOON, 1979) and the SMALLTALK Machine (as a first step toward the idea of a "DYNABOOK"; KAY 1977) are first examples of the technology we have in mind.

The structure of an IMS is illustrated in diagram 1.

Diagram 1: The structure of an IMS



Systems of this sort will be used for many applications: as office automation systems, as personal information systems (LAUBSCH, FISCHER and BOECKER 1979), as research tools etc.

We are convinced that real problems require an **IMS** and not only a programming language or a text processing system, like the following examples demonstrate:

1) to write a paper for a conference, we need
   - graphics (to include diagrams and pictures)
   - database (to retrieve the references)
   - programming (to sort the references, to include test runs, etc)

2) to support the development and modification of programs, we need an interactive program development system (FISCHER and LAUBSCH, 1980), including all the helpful features of the INTERLISP system (TEITELMAN 1978) like "Do what I mean (DWIM)", Programmer's assistant, UNDO and History facilities

This paper extends the work and the ideas expressed in FISCHER (1980).

1.2 Uniformity

One of the obstacles computer systems present to the user is the diversity of different languages and conventions which a user has to know to get a certain task done. To write an ordinary program in a conventional system the user has to know a large number of different languages, sublanguages and conventions, eg:

* the programming language itself (with conventions for specifying the control flow, external and internal data description etc)
* the operating system (job control language, linkage editor and loader)
* the debugging system (diagnostic system, symbolic assembler etc)
* the text processing system (editor and formatter)

The need for an **integrated** system is obvious to anybody who has tried to struggle through all the idiosyncracies of the different systems mentioned above.

An **IMS** offers uniformity in several dimensions to cope with this problem:

Linguistic uniformity: All tools (eg the programming system and superimposed modules as well as more specific creations of the user) are made from the same material and thus part of the same conceptual world. This has the sociological benefit that the system's implementor and users share the same culture. Each module in the system can be regarded as a "**glass-box**", ie it can be inspected by the user and the system can be explored all to the edges. This gives the user an amount of control over his environment which is not reachable in other systems.

Uniformity of interaction: This is based on a good interface, which provides a uniform structure for finding, viewing and invoking the different components of the system. The crucial aspect for this interface is the use of the display screen, which allows for many tasks the real-time, direct manipulation of iconic information structures which are displayed on the screen. Each change is instantly reflected in the document's image, which reduces the cognitive burden for the user. The screen should be regarded as an extension of the limited capacity of our short term memory (ie it provides a similar support like pencil and paper does for the multiplication of two large numbers).

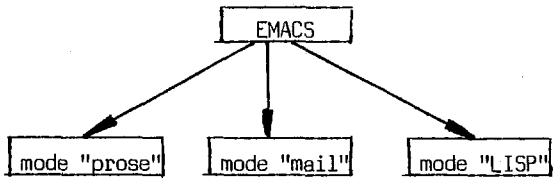
2. Experiences with existing systems

The author has had opportunities to work intensively with several advanced systems during the last few years. These systems form the background for the ideas expressed in this paper; they are the currently existing systems (known to the author) which come closest to our idea of an information manipulation system.

2.1 EMACS and MACLISP

EMACS (STALLMAN 1979) is a real-time display oriented editor, which can be **extended by the user.** This allows users to make extensions that fit the editor better to their own diverse applications, to experiment with alternative command languages and to share extensions which are generally useful. It runs on large timesharing machines (eg PDP-10) and large personal computers (eg LISP machine; WEINREB and MOON, 1979). It contains special subsystems ("modes"; see Diagram 2) to take advantage of the structures which occur in the systems to be edited. EMACS is a single key-stroke system, which puts a heavy demand on our recall memory. For these reasons, it is specifically suited for the expert user.

Diagram 2: Extensibility and Uniformity in EMACS
(extensibility means that arbitrary modes can be implemented and uniformity implies that the user does not need to learn a seperate editor for each system)



* contains commands for words, sentences and paragraphs;
* fill and justify commands
* transforms regions from upper to lower case

* contains commands for s-expressions
* operations for automatic indenting ("pretty printing")

EMACS is well interfaced with the MACLISP programming system. EMACS and MACLISP are kept in the machine as parallel jobs which is a necessary requirement to switch back and forth with a few keystrokes. This is quite different from the editing philosophy of the INTERLISP system (TEITELMAN 1978) where the editor is an integral part of the INTERLISP system itself. The advantages and disadvantages of these two approaches ("source-file" versus "residential" systems) are thoroughly discussed in SANDEWALL (1978).

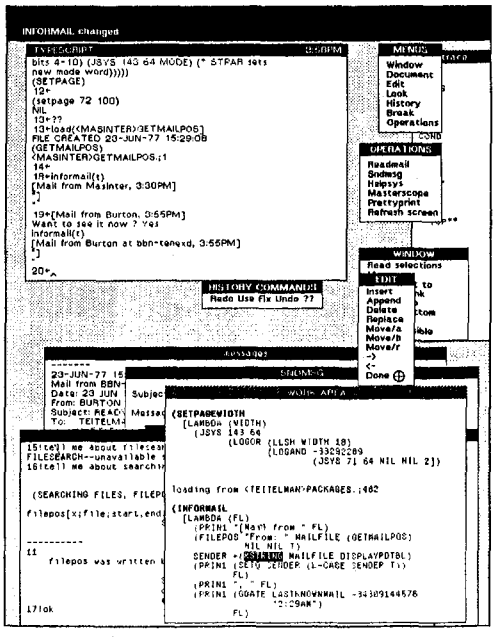
Powerful **personal computer systems** (like the LISP machine) contribute to the extensibility and modifiability of an information manipulation system because they make the entire software system interactively extensible by writing it in a higher level language (eg LISP) and allowing the user to redefine the functions composing the innards of the system (ie they provide the linguistic uniformity which we have mentioned in 1.2).

2.2 SMALLTALK and DLISP

SMALLTALK (KAY 1977) and **DLISP** (TEITELMAN 1977) are systems at Xerox Palo Alto Research Center, which rely heavily on a high resolution bit map display, a mouse as a pointing device and excellent software, which supports multiple windows with associated menues and multiple fonts (see Diagram 3 for an example); through their iconic representations and their menues they only requires the recognition of commands (ie no recall). These environments provide prototypes for man-machine interfaces which are heavily based on graphics. The problem of not having enough space on the screen is solved by allowing the windows to overlap. The resulting configuration considerably increases the user's effective work space and it contributes to the illusion that the user is viewing a desk top containing a number of sheets of paper which he can manipulate in various ways.

Diagram 3: The DLISP display facilities
(from Teitelman 1977)

the display shows the following features:
- several menues (which are context dependent and therefore can be kept small in size; they allow the recognition of commands and do not require a recall)
- windows to receive and send messages
- "WORK AREA" window which allows additional communication with the system
- selected text is indicated by reversing the color of the screen
- the virtual size of the screen is increased because the windows can overlap



Abilities like suspending an operation, performing other operations (eg to answer quickly to an urgent request received through the mail system) and then return without loss of context have turned out to be essential for many problem solving activities. The technique of using

different windows for different tasks does make this switching of contexts easy and painless.

These systems combine the best features of display and hardcopy terminals. A standard complaint with conventional display terminals is that material that the user wants to refer to repeatedly (eg the text of a function, the trace of a program execution) is displaced by subsequent, incidental interactions with the system. In a situation like this when using a hard copy terminal the user tears off the part he is interested in. The equivalent action in a window system is to freeze the relevant portion of the interaction in a seperate window (eg like the "WORK AREA" window in Diagram 3) whose content will not be affected by the following interactions (see TEITELMAN, 1977).

The graphical orientation of these systems has inspired research (eg SMITH 1977 and BORNING 1979) to create programming systems where more and more symbolic descriptions can be replaced by iconic descriptions. These efforts have the goal to integrate some of the features which have made display-oriented editing systems so successful into programming environments. Teletype-oriented editors require sequences of commands like "4DOWN 12LEFT 4DELETE" to delete four characters somewhere in a buffer. In a display-oriented environment we see the content of the buffer on the screen and can move with the cursor (supported by continuous visual feedback) to the object to be manipulated. An example of symbolic versus iconic programming is given in Diagram 4. The operation to be performed is to change the value of the third element of an array. In the symbolic case we have to "tell" the computer that we want to assign a new value to the third element of an array, whereas in an iconic programming environment the array would be displayed on the screen and changed directly.

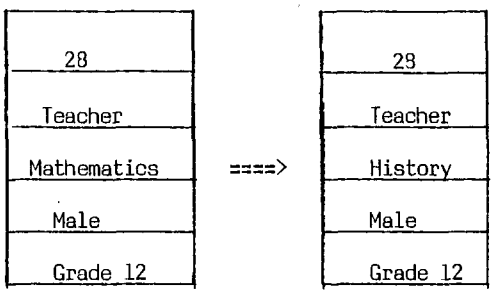
Diagram 4: Symbolic versus Iconic Programming

symbolic programming:

```
print x<3>
Mathematics

x<3> <-- "History
print x<3>
History
```

iconic programming ("you get what you see"; emphasis is on doing rather than telling):



2.3 TINTE and LOGO

TINTE (RATHKE 1979), also a real-time display oriented editor, has a limited set of user-definable keys. It runs on a **small personal computer** and through its **incremental design** it can be used by users of all levels of expertise. Incremental design is one of the most essential features of a complex system, because the novice user of any reasonably powerful computer system is more confused than assisted by a full display of the information and options available to him. (**Note:**This paper was written with the help of TINTE).

The interface between TINTE and the programming system LOGO (BOECKER 1977) is not as smooth as in the systems described above. The main reason for this is the limited memory space available in small computers which does not allow to load the two systems as parallel jobs.

Our research during the last few years has not only been concerned with the technology of providing interactive computer service, but also with changes in **conceptualizing, visualizing and organizing work and research** with these systems and in procedures and methods for working individually and cooperatively (FISCHER 1979; FISCHER, BROWN and BURTON, 1978; FISCHER and LAUBSCH, 1980).

3. Theoretical considerations for the design of an IMS

If a civil engineer has to design a bridge, he acquires a detailed knowledge of the country side in which the bridge will be built and he will use the laws of physics to come up with a plan.

Unfortunately things are not quite as easy with respect to the design of an **IMS.** Changing hardware is the smallest problem; the major obstacle is that there is **no cognitive theory of sufficient predictive power** (fulfilling the same function as the laws of physics for the bridge) which would be specific enough to provide a complete set of design criteria for an **IMS.** NEWELL and SIMON (1976) argue convincingly why research like the design of an IMS has to be carried out as "empirical inquiry".

Despite the lack of a complete cognitive theory, work in Cognitive Science has accumulated a substantial body of knowledge which is important for integrated **IMSs.** The scope of this paper does not allow us to give a detailed description of our theoretical framework; therefore we mention only the most important aspects:

1) research in AI and cognitive psychology has shown that **knowledge** is at the basis of all problem solving; the active research in the development of knowledge representation languages (like KRL and FRL) is based on this insight

2) psychological research and empirical evidence supports the hypothesis that **"thinking always begins with suggestive but imperfect plans and images;** these are progressively replaced by better, but usually still imperfect plans". This hypothesis indicates the evolutionary character of complex systems, it implies that linear approximation is an important methodology and that debugging processes have to be understood thoroughly

3) **problem solving theories** about planning (which operates in a simplified abstraction space), analogy (which forms the basis for recognition methods), debugging (see previous point) and multiple representations (see **3.2**) are not any more only directed towards the understanding of abstract and well-structured problems but investigate ill-structured problems in semantically rich domains (SIMON 1978)

4) knowledge about **human information processig capabilities** (eg about the limited capacity of our short term memory) shows that for complex systems there exists a need to prefold information for the user so that more pieces of the whole picture can be maintained in the user's immediate attention at once

5) SIMON (1969) has provided an insightful analysis of the structure of **complex systems** (by showing their hierarchical structure, their property of being "nearly decomposable" etc)

3.1 Generative Processes

One purpose of an **IMS** is to support the creative aspects of the writing and programming process. Writing and programming often means to make a rough draft and refine it over a considerable period of time (in other words: it subsumes all the processes required to go from the first idea to the final product; see FLOWER and HAYES, 1979). It includes the expectation of an **"unacceptable first draft"** which will develop through successive changes into a presentable form. An important general characteristic of computers is that they allow us to build quickly low-cost modifiable models which we can analyze and experiment with. We believe, contrary to the formal, verification oriented group in the Structured Programming community, that this view is as adequate for programming as it is for writing.

Text processing and programming are examples of generative processes which are best understood as problem solving. Inadequate technologies (eg a typewriter, a batch system) force the writer or programmer to limit himself to a small set of strategies. For example he has to proceed in a serial fashion, whereby the form of the written word imposes restrictions on the generation of language and ideas. On the other hand it is well known that knowledge is not simply additive which implies that a new insight or idea may require a major restructuring of what has been done before.

Creative writing and programming is an **ill-structured problem** (SIMON 1978). In these problem solving situations the problem solver has to contribute actively to the exact specification of the problem and he has to define criteria what will be accepted as a solution.

3.2 Multiple Perspectives

The computer as an active medium offers more posibilities than paper for a person who wants to write, understand or read a report or a program. For complex descriptions it is often a big advantage to be able to generate **multiple perspectives** which facilitate or highlight certain aspects of a system. Multiple perspectives are able to resolve the basic conflict that symbols, which are ordered in one fixed order (eg on a printed page), serve as pointers to concepts which form a highly interrelated network. This implies that no single linear order is adequate. The value of multiple perspectives can be illustrated in a nice way using maps as an example: there may be many different maps for the same territory using large and small scales, showing the percipitation, the population density, the economical structure and any other relevant criteria.

In reading text can be selected according to the wishes or needs of the reader (to allow "**dynamic reading**"; a display screen can be regarded as a dynamic blackboard):

1) for the novice and the expert, different parts may be left out or included

2) to get a global overview, we can generate a table of contents at arbitrary levels of abstraction

3) information can be reordered such that all occurences of a certain concept are selected (which occur in other representations at arbitrary places)

Similar possibilities exist for the representation of programs:

1) certain modules of the program can be listed selectively (eg all the data accessing functions, all declarative information, all procedures which achieve a specific subtask); procedures can be listed in different orders (eg alphabetically or according to the calling structure)

2) the calling structure which shows the connectivity structure between different procedures can be displayed at arbitrary levels of detail; the user should be allowed to define a "view specification"

3) symbol tables give a recoding of information according to a different criterion

3.3 Problem solving versus routine skill

An IMS should also support the **routine skill** (CARD 1978) of editing a manuscript or coding an known algorithm in the syntax of a programming language. In this case it helps to eliminate the boring, time-consuming and unproductive work of secretaries who have to spend long hours to retype manuscripts, to make only trivial changes to a prototype of a letter but still have to retype it as a whole and who become greater experts in using scissors and glue than in anything else. A routine cognitive skill means that the methods to be used are well known and that the sequence of actions which occur are of a modest variety (therefore there is little search to find out what to do next).

4. Implications for the process of system design

"Truth emerges more readily from error than from confusion".

4.1 The necessity for empirical investigations

It is generally accepted that when a program is to be written, specifications should be designed in advance. But for real design tasks or ill-structured problems (see 3.1) this is more wishful thinking than a realistic goal. The history of the development of text editors is a good example for this assertion (another example would be timesharing systems; see NEWELL/SIMON (1976) for an insightful analysis of this topic) and provides a good illustration of the **co-evolution** of implementations and interface specifications. As experience accumulates in using an implementation, more of the real needs and benefits are discovered causing the partial interface specifications to change. The chain of necessary steps leading to one of the systems described in section 2 starting with the availibilty of the display processors would have been simply too long for anyone to have imagined the final result before the first step had been taken (for a general discussion of these issues see FISCHER, BROWN and BURTON, 1978).

4.2 A design conflict

In the initial phase of using a text processing system it is very important that the introduction of the computer system changes the tasks performed as little as possible. For computer naive user it is a traumatic experience anyway to change the tangebility of a piece of paper by the illusiveness of electronic documents and files. It is a step that drastically alters the appearance of their tools.

As users become more experienced and more familiar, the systems should take advantage of the new medium. Strict adherence to normal typing conventions in an **IMS** is not always advantageous (eg good text processing systems do not require that the user pays attention to the end of a line, they allow him to define abbreviations, to experiment easily with the layout, they take care automatically for constraints, etc). Lack of attention to this essential phenomena is one of the reasons that many innovations fail.

Regarding the efficient use of an **IMS** as a **skill** which develops over a long period of time and which gets used repeatedly (FISCHER, BROWN and BURTON, 1978) implies that we have to pay attention to the following design issues:

1) **time:** how long does it take to accomplish a task?
2) **errors:** what kind and how many errors does a user make and how serious are they?
3) **learning:** how long does it take a novice user to learn to use the system (for a secretary, for a trained computer scientist)?
4) **functionality:** what range of tasks can a user perform with the system? How can it be made extensible to take care for unforeseen requirements?
5) **recall:** how easy is it for a user to recall how to use the system for a task that he has not done for some time??

5. Empirical findings

Observing many people how they use **IMS** and taking into account empirical data based on interviews and questionaires, has revealed the following:

1) the systems can reduce the **psychological stress** of doing something wrong (because wrong things can be easily corrected)

2) they increase the willingness to **experiment** with new and different ideas

3) the small amount of effort to change things in a non-trivial way (eg to find a major rearrangenment of a text or a more modular solution to a programming problem) leads in many cases to an improvement not only in form but also in **content**

Much more empirical work is needed to develop a detailed requirement analysis which can serve as a guideline for the design of the next generation of information manipulation system. Unfortunately the verdict of users is not particulary reliable: as usual, users of the respective systems tend to prefer what they are used to.

6. Conclusions

In the 1980's there will be a massive attempt to introduce information manipulation systems into universities, offices, clerical operations and the home. The well-being of many workes as well as the technical success of the systems themselves will depend on how much the design pays attention to cognitive dimensions.

One of the major research goals for the future will be to build totally integrated **IMS** allowing to make computer systems accessible to many more people and to make computer systems do many more things for people.

Acknowlegdements

I am indebted to the members of several research groups at Xerox Palo Alto Research Center and to many members of the MIT AI and LOGO Lab for giving me a chance to visit both places several times over a longer period of time and letting me explore and work with their systems. H.-D. Boecker has made substantial contributions to this paper.

References

Boecker, H.-D. (1977): "LOGO Manual", Forschungsgruppe CUU, Projekt PROKOP, Darmstadt

Borning, A. (1979): "Thinglab -- A Constraint-oriented Simulation Laboratory", SSL-79-3, July 1979, Xerox Palo Alto Research Center, Calif

Card, S. E. (1978): "Studies in the Psychology of Computer Text Editing Sytems", SSL-78-1, Xerox Palo Alto Research Center, Calif

Engelbart, D. C. and W.K. English (1968): "A research center for augmenting the human intellect", AFIPS FJCC, pp 395-400

Fischer, G. (1979): "Powerful ideas in Computational Linguistics - Implications for Problem Solving and Education", in Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics, San Diego, pp 111-125

Fischer, G. (1980): "Cognitive Dimensions of Information Manipulation Systems", in P.R. Wossidlo (ed): "Textverarbeitung und Informatik", Informatik Fachberichte Vol 30, Springer Verlag, pp 17-31

Fischer, G., J.S. Brown, R. Burton (1978): "Aspects of a theory of simplification, debugging and coaching", in Proceedings of the 2nd Conference of the Canadian Society for Computational Studies of Intelligence, Toronto, July 1978, pp 139-145

Fischer, G. and J. Laubsch (1980): "LISP-basierte Programmentwicklungssysteme zur Unterstuetzung des Problemloesungsprozesses", in Heft 3 der Notizen zum Interaktiven Programmieren, Fachausschuss 2 der Gesellschaft fuer Informatik, Darmstadt, Maerz 1980

Flower, L. S. and J. R. Hayes (1979): "Problem solving and the cognitive process of writing", in J. Lochhead and J. Clement (eds): "Cognitive process instruction", The Franklin Institute, Philadelphia

Kay, A. (1977): "Microelectronics and the personal computer", Scientific America, September 1977, pp 231-244

Laubsch, J., G. Fischer and H.-D. Boecker (1979): "LISP-based systems for educational applications", BYTE, Vol. 4, No. 8, August 1979, pp 18-25

Newell, A. and H. Simon (1976): "Computer Science as Empirical Inquiry: Symbols and Search", CACM, Vol 19, No 3, March 1976, pp 113-126

Rathke, C. (1979): "TINTE - ein interaktiver Texteditor", MMK Memo 16, Institut fuer Informatik, Universitaet Stuttgart

Sandewall, E. (1978): "Programming in an interactive environment: The LISP experience", ACM Computing Surveys, Vol 10, No 1, March 1978, pp 35-71

Simon, H. (1969): "The Sciences of the Artificial", MIT Press, Cambridge, Ma

Simon, H. (1978): "The structure of ill-stuctured problems", in H. Simon: "Models of Discovery", D. Reidel Pulishing Co, Boston, Ma, pp 304-325

Smith, D. (1977): "Pygmalion - A Computer Program to Model and Stimulate Creative Thought", Birkhaeuser Verlag, Basel und Stuttgart

Stallman, R. (1979): "EMACS - the extensible, customizable, self-documenting display editor", MIT AI Lab, Memo 519, Cambridge, Ma

Teitelman, W. (1978): "INTERLISP Reference Manual", Xerox Palo Alto Research Center, Palo Alto, Ca

Teitelman, W. (1977): "A Display-oriented Programmer's Assistant", in Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, Ma, pp 905-915

Weinreb, D. and D. Moon (1979): "LISP Machine Manual", 2nd preliminary version, January 1979, MIT AI Lab, Cambridge, Ma