Automatica, Vol. 19, No. 6, pp. 627–637, 1983
Printed in Great Britain.

0005-1098/83 $3.00 + 0.00
Pergamon Press Ltd.

Symbiotic, Knowledge-based Computer Support Systems*

GERHARD FISCHER†

A prototype knowledge-based information manipulation system (IMS) provides directions for designing future symbiotic systems which will expand human potential and productivity with human-oriented computers having a suitable blend of human and computer intelligence.

Key Words—Cognitive ergonomics; convivial tools; human–computer communication; knowledge-based systems; information manipulation systems; symbiotic systems; user interface.

Abstract—The full benefit of computers as tools of thought can come only when we learn to divide intelligence into a portion which is best suited to the human being and a portion which is best suited to the computer and then find a way to combine the process. The limiting resource in future human–computer systems will be the human (with respect to time, attention, complexity management) and not the computer. There is a need for symbiotic, knowledge-based computer support systems which will make communication with computers easier, more rewarding and turn the computer into a convivial tool. Theories, methodologies and tools are neded to make systems of this sort broadly available. In our research project INFORM we are designing, implementing and evaluating a knowledge-based information manipulation system (IMS) as a prototypical development.

INTRODUCTION

THE GOAL of the work described in this paper is to design and implement symbiotic systems which are convivial tools for the people who use them. It will be shown that a system of this sort has to be a knowledge-based system. Some of the underlying theoretical aspects which are regarded as relevant for this research will be described. Ideas will be illustrated with examples taken from the work in the research project INFORM (Boecker, Fischer and Gunzenhaeuser, 1980; Bauer and co-workers, 1982).

* Received 8 February 1983; revised 2 September 1983. The original version of this paper was presented at the IFAC/IFIP/IFORS/IEA Conference on Analysis, Design, and Evaluation of Man–Machine Systems which was held in Baden-Baden, F.R.G. during September 1982. The published proceedings of this IFAC meeting may be ordered from Pergamon Press Ltd, Headington Hill Hall, Oxford OX3 0BW, U.K. This paper was recommended for publication in revised form by editor A. Sage.

† Research Group on Human–Computer Communication and Knowledge-based Systems, Department of Computer Science, University of Stuttgart, Herdweg 51, D-7000 Stuttgart, F.R. Germany.

COMPUTER-BASED MAN–MACHINE SYSTEMS—HOW THEY ARE AND HOW THEY SHOULD BE

The purpose of computer-based man–machine systems is to direct the computational power of the digital computer to the use and convenience of man. There is no doubt that there has been great progress in some fields of computer science. The dramatic price reduction in hardware has opened up totally new possibilities. But other aspects have not kept pace with this progress, especially how easy it is—not only for the expert but also for the novice and the occasional user—to take advantage of the available computational power to use the computer for a purpose they have chosen.

Most computer users feel that computer systems are unfriendly, not cooperative and that it takes too much time and too much effort to get something done. They feel that they are dependent on specialists, they notice that software is not soft (i.e. the behaviour of a system cannot be changed without a major reprogramming of it) and the casual users finds him/herself in a situation as in instrument flying: he/she needs lessons (relearning) after not using a system for a long time.

The goals are to create symbiotic, knowledge-based computer support systems which

- handle all of their owner's information-related needs; these needs will be quite different for different groups of users;
- make computer systems accessible to many more people and make computer systems do many more things for people;
- help us to gain a better understanding of the cognitive dimensions and tasks structures.

To achieve these goals we need beside technological expertise

- *theories*, which take knowledge and insights from computer science, psychology, linguistics and sociology into account and help us to define new design critera; allow

- *methods* which are based on those theories; and
- *tools* which use the computational power of the computer to support the user.

SYMBIOTIC SYSTEMS

Symbiotic systems are based on a successful combination of human skills and computing power in carrying out a task which cannot be done either by human or by computer alone. The conception of symbiotic systems is illustrated by giving examples in different domains.

Software engineering

We need systems which provide more assistance in all phases of the process (e.g. problem formulation, design, specification, implementation, testing, verification, documentation and modification) and more automation. This would free programmers from the clerical parts of their task so that they can concentrate more on the difficult aspects of problems.

There is no doubt that we have made big progress in the area of programming [by having available Assembler, Compiler for high-level languages, interactive programming environments (as in LISP and SMALLTALK)] so that the labour of programming was reduced by several orders of magnitude during the last 20 years. Yet despite all this progress, programming a computer to perform a non-trivial task remains a difficult problem which is much more complicated and tedious than instructing an intelligent and trained human who would help us as an assistant.

Help systems

Help systems (which should be an integral part of every computer system) can be used to illustrate a basic misunderstanding of the real limiting resource in designing computer systems. The important function of computers is not to multiply information but to analyse it so it can be filtered, compressed and diffused selectively. Most online assistance systems offer a huge static, tree-structured information network where it is very time consuming to find a relevant piece of information. What the user really needs are answers to questions like:

how can I do X?
what happens if ...
why did *Y* occur?
can I undo the effects of *Z*?

To be able to give answers of this sort a computer system must have self-knowledge about its own functionality and about the dynamic execution state, it must offer a descriptive language so the user can communicate with it and it must provide explanation facilities which are based on a model of the user. A further important characteristic of a symbiotic system is that it should be non-intrusive, i.e. it should not get in our way if we do not need it.

Further examples of symbiotic computer systems

Computerized axial tomography (CAT scanning; McCracken, 1979) is based on a partnership between doctor and computer; the necessary inverse Fourier transformations involve an immense amount of computation and cannot be done without the help of a computer—and the interpretation of the data requires discrimination between subtle differences in density which is beyond current capabilities in image processing.

Kay (1980) proposes a symbiotic machine translation system that is always under the tight control of the translator. The system is there to help increase his productivity and not to supplant him. The fully automatic approach has failed badly in the past.

Personal computers (e.g. such as the DYNABOOK; Goldberg, 1981) have made it possible to turn everyone into an artist and produce animated pictures, music and other artifacts which in the past could only be produced by the professional because the process was too time consuming and too costly.

In the research project INFORM application systems have been developed which serve as prototypes for thinking about symbiotic systems:

1. A system FINANZ which helps in filling out finance plans for research proposals; the systems assists in the planning process, takes over all the clerical details (e.g. to compute repeatedly the sums of some numbers after one of the numbers has changed), it changes input data to the right format, it warns if we have made a mistake and it provides explanations about the origin of pieces of data.

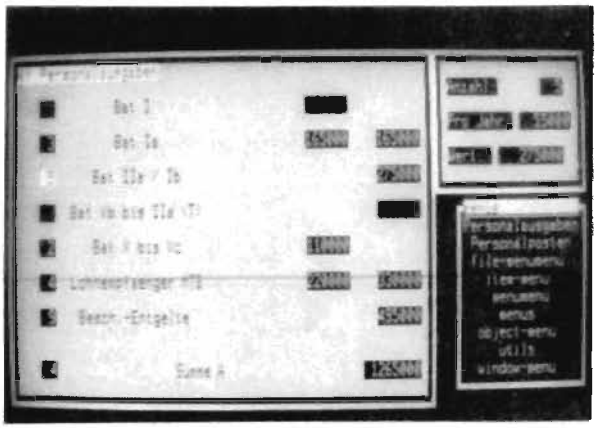

FIG. 1. Screen layout for a financial planning system (Rathke, 1983). The screen shows menues (the commands contained in them can be activated by a pointing device) and windows (which are viewers into the same knowledge base); the program can be considered as a knowledge-based version of Visicalc.

2. A system PLANER which helps a graduate student in our department to plan his studies; it generates proposals for timetables under constraints *formulated by the user*, it indicates conflicts which will arise in later years by making a specific choice for the next semester and it helps us to resolve these conflicts.

CONVIVIAL SYSTEMS

Illich (1973) has introduced the notion of 'convivial tools' which I regard as one of the most important aspects of symbiotic systems. He defines them as follows:

> 'Tools are intrinsic to social relationships. An individual relates himself in action to his society through the use of tools which he actively masters, or by which he is passively acted upon. To the degree that he masters his tools, he can invest the world with his meaning; to the degree that he is mastered by his tools, the shape of the tool determines his own self-image. Convivial tools are those which give each person who uses them the greatest opportunity to enrich the environment with the fruits of his or her vision.
>
> 'Tools foster conviviality to the extent to which they can be easily used, by anybody, as often or as seldom as desired, for the accomplishment of a purpose chosen by the user.'

Illich's thinking is much broader and tries to show alternatives for future technology-based developments and their integration into society. We have applied his thoughts to information processing technologies and systems and believe that *conviviality is a dimension which sets computers apart from other communication technologies.* All other communication and information technologies (e.g. television, videodiscs, interactive videotex) are passive, i.e. the user has little influence to shape them to his own taste and his own tasks. He has some selective power but there is no way that he can extend system capabilities in ways which the designer of those systems did not foresee.

I do not claim that current existing computer systems are convivial. Most systems belong to one of the following two classes (which constitute opposite ends along the dimension of conviviality): (a) *turn-key systems*: they are easy to use, no special training is required but they can not be modified by the user; (b) *general purpose programming languages*: they are hard to learn, they are often too far away from the conceptual structure of the problem to be solved and it takes too long to get a task or a problem solved.

There are promising ways starting from both ends to make systems more convivial. Good turn-key systems contain features which make them modifiable by the user without a necessity to change the internal structures. Good text processing systems allow the user to define his own keys ('keyboard macros'; Fischer, 1980) and good display systems allow the user to create and manipulate windows at an abstract and easy to learn level (Bauer, Boecker and Fischer, 1981).

We believe that the development of convivial tools will break down an old distinction: there will no longer be a sharp borderline between programming and using programs—a distinction which has been a major obstacle for the usefulness of computers. Convivial tools will take away the

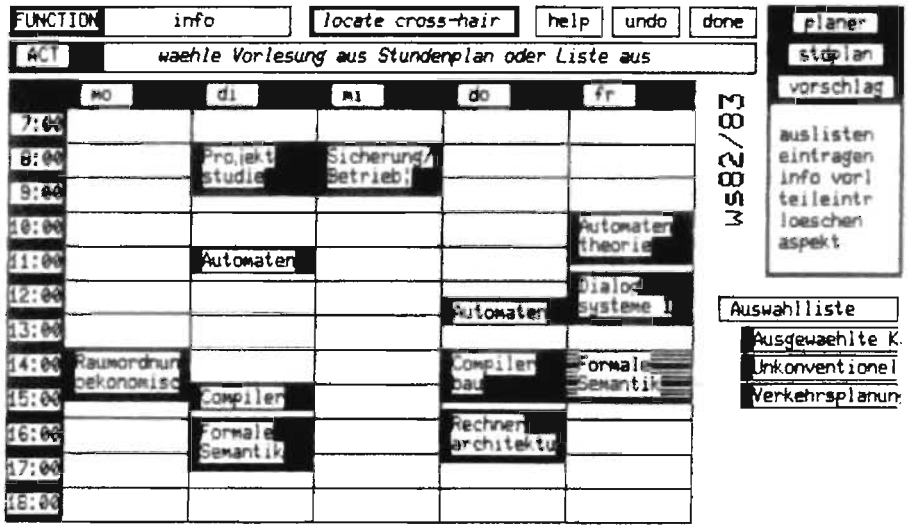

FIG. 2. PLANER—a computer-assisted planing program (Maier, 1983). The figure shows (in colour on the screen): (a) a timetable (constructed by the system based on general predicates given by the user); (b) a menu (upper right corner) which lists the current commands for selection and in the top part the path into the hierarchical structure; (c) general windows for the interaction (e.g. a undo and a help key); (d) several windows which provide information what to do next and which lecture can be selected to be inserted in the timetable.

impossible task from the 'meta-designer' (i.e. the person who designs design-tools for other people) that he has to anticipate all possible uses and all people's needs. Convivial tools encourage the user to be actively engaged and generate creative extensions to the artifacts given to him and we hope that they will be tools for everyone and not only the private domain of a few highly educated people.

Another idea related to the concept of convivial systems is a 'toolkit' which provides a set of components and set of tools (by means of which these components can be viewed and manipulated) that can be used to create many different but related things. This approach has been successfully exploited in other areas (see technical construction systems, for example; Fischer and Boecker, 1983), and it has a goal to protect the user from the full complexities of a general purpose system. Examples in the world of information processing systems are developments from the Learning Research Group at Xerox with SMALLTALK (Goldberg, 1981) and the operating system UNIX which allows the user to create complex procedures not by writing large programs from scratch, but by interconnecting relatively small predefined components.

There is a crucial difference whether a computer user feels that he is computerized or whether he is using a computer as a powerful tool. Decision support systems ('expert systems') in medicine (e.g. Mycin; see Shortliffe, 1976) can be seen by a physician either as a robot doctor which replaces him or as a tool (such as an X-ray machine, a more easily consulted reference book or a computer tomograph).

What it really is lies mainly in the mind of the physician, but we (as system designers, who work out the theoretical foundations for them) should do our best that our systems are convivial tools and their users have the possibility and the feeling that they are the controlling agents.

KNOWLEDGE-BASED SYSTEMS

There is a difference between databases and knowledge bases. In a *database* the conceptual organization is simple and the main problems are size and efficiency (an example would be all the records of a company with 100 000 employees). For a *knowledge base* there is no definition what would be included and under which circumstances and in which form the stored knowledge becomes relevant (examples would be the knowledge of a chess master to make a move, the knowledge of a doctor to find a diagnosis and the knowledge of a programmer to explain what his program does).

Knowledge-based systems are a very active research area (see the plans in Japan for a 10 year research project to develop 'fifth generation computer systems' as knowledge-based systems). Our own efforts are centred around the question how knowledge-based systems can be used to improve human–computer communication and contribute towards the creation of symbiotic systems.

Traditionally human–computer communication has relied on the model shown in Fig. 3. There is no way to achieve the characteristics of symbiotic systems (as described above) with this model. Human communication and cooperation can serve as a model for a symbiotic relationship. What can humans do that most current computer systems cannot do? Human communication partners (e.g. a programming assistant or a teacher):

- do not have the literalness of mind which implies that not all communication has to be explicit; our communication partner can supply additional information which we have forgotten and he can correct simple mistakes; see the DWIM ('do what I mean') facility in INTERLISP (Teitelman and Masinter, 1981) for a first step in this direction;
- can apply their problem solving power to fill in details if we give statements of objectives in broad functional terms; the development in programming from 'how' (e.g. a detailed and exact description in an assembler language *how* to evaluate an algebraic expression) to 'what' (e.g. to write down this expression in PASCAL) indicates in which direction further progress is needed;
- can interpret meaning and intent, when confronted with the vagueness and informality of natural language;
- can articulate their own understanding and misunderstanding; and
- can provide explanations.

Knowledge-based systems are one promising approach to equip machines with some of these human communication capabilities. Based on an analysis of human communication processes the model shown in Fig. 4 has been developed. It contains a knowledge base which can be accessed by both communication partners; this implies that the necessity to exchange all information explicitly does not exist any more.

The system architecture in Fig. 4 contains two major improvements (compared to Fig. 3): (a) the explicit communication channel is widened. The interfaces use windows with associated menus, pointing devices, colour and iconic representations; the screen is used as a design space which can be manipulated directly (see Figs 1 and 2); (b)

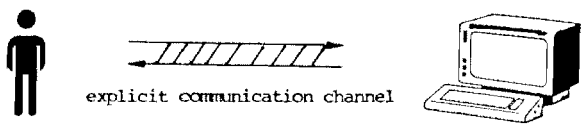

FIG. 3. The traditional model of human–computer communication.

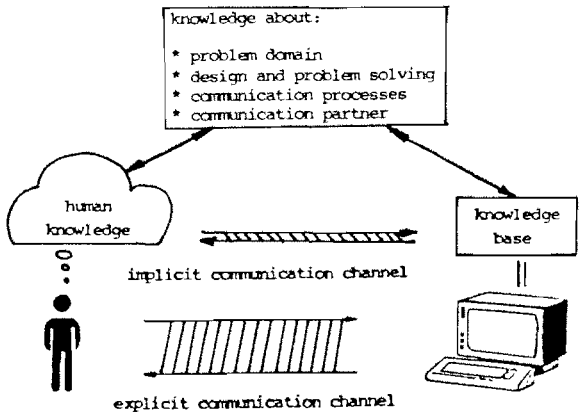

FIG. 4. Extended model of human–computer communication.

information can be exchanged over the implicit communication channel. Both communication partners have knowledge which eliminates the necessity that all information has to be exchanged explicitly. The four domains of knowledge shown in Fig. 4 have the following relevance:

1. *Knowledge of the problem domain*: research in artificial intelligence has shown that intelligent behaviour builds upon large amounts of knowledge about specific domains (which manifests itself in the current research effort surrounding expert systems).

2. *Knowledge about design and problem solving*: it is important not only to retain the finished product but the important parts of the development process; we should be able to explore alternative design in different contexts and merge them if necessary (Fischer and Boecker, 1983).

3. *Knowledge about communication processes*: the information structure which controls the communication should be made explicit, so the user can manipulate it; e.g. scripts (actions which are in general carried out in a sequence) should be available to structure the communication process.

4. *Knowledge about the communication partner*: the designer of a program wants to see quite different parts compared to another programmer who wants to use the module only as a package. For a user, information will be relevant which helps him to create a consistent model of the system, to know how to invoke certain subsystems and to link the behaviour of the system to the underlying design rationale.

The limitations and the possibilities of these two models are illustrated with an example of a computer program which could serve as a travel assistant (Fauser and Rathke, 1981):

Computer: What time do you want to leave?
User: I must be in Berlin before 10 a.m.

At the surface level the answer has nothing to do with the question but it is definitely relevant for the question. The computer needs the following knowledge to cope with it: being somewhere is the consequence of going somewhere and the time of leaving determines in a systematic way the time of arrival.

Another aspect of this dialog fragment is to understand the meaning of the word '*must*' in the answer. In case there is no plane, should a private plane be chartered? What alternatives can be suggested to the user? Without a user model (e.g. is the traveller a student or the president of a company) and a large amount of commonsense knowledge there is no chance that the program gets a real understanding of the implications of the answer. It is obvious that a system based on our second model is needed to cope in a meaningful way with this situation.

INFORMATION MANIPULATION SYSTEMS (IMS)

This term has been used to indicate that the original meaning of 'programming' as the art of finding and coding of algorithms is too restricted to cover the many possible uses of a computer by a human being. One of the obstacles computer systems prevent 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, e.g.:

- 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 the idiosyncracies of the different systems mentioned above. An IMS offers uniformity in two dimensions to cope with this problem:

Linguistic uniformity: all tools (e.g. 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 knowledge. Each module in the system can be regarded as a 'glass-box', i.e. 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 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 (i.e. it provides a similar support like pencil and paper for the multiplication of two five digit numbers).

The structure of an IMS is shown in Fig. 5. The most crucial issue in the design is the *integration* of the different components with each other.

I regard IMSs as prototypical systems which are an effort towards the goal to construct truly symbiotic systems. IMSs can be used as software production systems, as application systems for end users (e.g. in office automation) and as testbeds for the construction of modules to improve human–computer communication.

COGNITIVE SCIENCE

To make full use of a symbiotic relationship implies that each partner can take advantage of his strength and gain some support for his weaknesses. Cognitive science (Norman, 1981) is an interdisciplinary research discipline which tries to understand the principles of intelligent systems (independently where and how these systems exist).

In the last few years a substantial body of knowledge has been accumulated which can serve as design guidelines for symbiotic systems (this body of knowledge provides a starting point for the new discipline of cognitive engineering which tries to construct artifacts along cognitive dimensions). Some of the strong parts as well as the mental processing limits of the human information system have been identified and prototypical systems have been constructed which try to acknowledge these contributions. Examples are (Newell and Simon, 1972; and Simon, 1981):

- the visual system is the most efficient chunking method in the human information processing system; this means that two-dimensional representations, iconic instead of symbolic programming (e.g. screen-oriented editing), forms which support content (e.g. by using different fonts for different things) are important characteristics in system design;
- recognition memory is greater than recall memory; this provides an incentive to construct menu-based systems instead of keystroke systems;
- the scarce resource in human–computer systems is not information but human attention; this implies that techniques for intelligent summarizing, prefolding of information, etc. have to be developed;
- our artifacts have reached a complexity that they can not be understood any more without adequate tools. These tools must be better than pencil and paper and the challenge is to achieve this goal for computer systems. Very modest beginnings of this claim have been demonstrated, e.g. a computer can generate dynamic views of a document (a reference manual and a primer are different, but they can be generated from similar knowledge structures), which eliminates the necessity of a static predetermined view;
- our short-term memory is limited, i.e. there is a very finite number of things which we can keep in mind at one point of time. To overcome these limitations we have to construct systems which give us not only reasoning support but also memory support. Levels of abstraction and mnemonic names are able to reduce the cognitive burden of the user;
- our systems must be consistent and uniform for the user. Consistency must be based on an

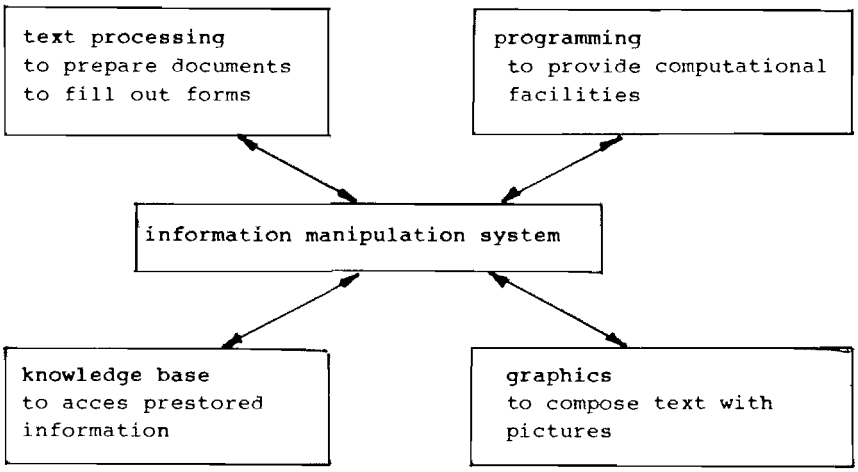

FIG. 5. The structure of an information manipulation system (IMS).

adequate model of how a system will be used.

To develop this preliminary collection further and to demonstrate how to make all these principles operational is an important topic for future research in human–computer communication.

COGNITIVE ERGONOMICS

Research in ergonomics investigates and analyses the effect of technologies for human work and it tries to develop adequate tools to make life easier. In the past, properties of systems were investigated which could be measured with methods from physics (e.g. the design and layout of a keyboard). This approach is not sufficient to evaluate information processing technologies (Moran, 1981). Modern computer systems try to support the human in decision making, planning, design and other cognitive activities. To evaluate these intelligent tools we must extend ergonomics research to pay attention to the conceptual and cognitive skills of people.

At the current stage of development, research in cognitive ergonomics should not be restricted to the comparison of finished products, but it should take an active part in the design and integration of cognitive dimensions in our information manipulation systems.

PROJECT INFORM

In the research project INFORM (Boecker, Fischer and Gunzenhaeuser, 1980; Bauer and co-workers, 1982) an integrated information manipulation system is being designed, implemented and evaluated. It should serve as a prototype for a symbiotic, knowledge-based computer support system which is based on the framework outlined in the previous section. In this paper we choose knowledge representation and visualization as two examples of our work and show how they are relevant for the construction of a specific application program: a system to support program synthesis and analysis (Fischer and co-workers, 1981).

Knowledge representation

Our work on knowledge representation is based on OBJTALK, an object-oriented language (Fischer and Laubsch, 1979; Rathke, 1983) which is modelled after SMALLTALK (Goldberg, 1981). OBJTALK is built on top of LISP and allows the user to define objects which can be grouped into classes. To take advantage of the fact that most complex systems are hierarchical systems and therefore the knowledge to describe them can best be organized as hierarchical framework, OBJTALK has an inheritance mechanism between classes.

This knowledge representation machinery provides a good framework for the implementation of the knowledge base of a computer system to support program synthesis and analysis. There are different sources of knowledge which are important for program synthesis and analysis:

- knowledge about the programming language (which may be used for syntax-driven program construction, for real-time identing of the program text, etc.);
- knowledge about programming constructs and techniques (to give assistance in building adequate data structures);
- knowledge about complex artifacts (e.g. what is the relationship between different parts; this information can be used to drive the editor and to check the consistency after changes have been made);
- knowledge about the design process (to assist in exploring alternative worlds, to retain the whole development process and not only the finished product);
- knowledge about the semantics of the problem domain (which can provide links between semantic names and the concepts which they denote).

In the example given, the basic knowledge units are organized around the concept of a function (since we work in a functional language like LISP). Figure 6 shows a *simplified* version of a knowledge structure created for a function which is used in our editor BISY.

Most of this context structured knowledge base is *simultaneously* compiled while the user is working interactively on his problem. In addition it includes information supplied by the user, e.g. a purpose slot and a natural language description of the algorithm; these slots are not used by the interpreter but they serve an important role in providing memory support for the human programmer.

After a function is defined it is translated immediately into the internal representation of Fig. 6 which is used for all further analysis. Therefore it is important that the internal representation contains all the information. From this example it should be quite obvious that in a knowledge-based system a program is more than its listing.

Visualization

One of the greatest steps forward in human–computer communication was the possibility to use the display as a truly two-dimensional medium. In Fig. 7 a visual representation of a LISP data structure which is generated automatically from the symbolic representation is shown.

New innovations introduce new problems. The modern way of using screens confronts us with a new set of display management problems that did not arise with a teletype terminal: what information should be displayed? How should it be displayed? How can we direct the users attention so he will

```
Conventions:
  (1) shadow              : slot names of our frame-like structure
  (2) underlined          : user-provided information and commentaries
  (3) normal font         : data supplied by the system
  (4) CAPITALS            : "guesses" of the system, i. e. derived
                            information which is incomplete and uncertain
  (5) bold and underlined : inherited information


(Function: read-lisp-buffer
 (comment: "BISY means: bildschirm- und syntaxorientierter Editor;
            eobp means: end-of-buffer-predicate")
 (purpose: "makes a list of LISP-lists and -atoms out of the BISY-buffer and
            returns it as value")
 (algorithm: "conses all characters of the buffer together, applies a
              readlist on them and lets this list begin with a progn")
 (Status: defined)
 (Code: (def read-lisp-buffer
           (lambda nil
                   (prog2 (progn (push-position)
                                 (set-position 1 1))
                          (do ((source nil)
                               (ch (get-char&att)
                                   (get-char&att)))
                              ((eobp)
                               (car
                                (errset
                                 (readlist
                                  (cons "("
                                        (nreverse (cons ")" source)))))))
                              (or ch (setq ch 10))
                              (setq source
                                    (cons (char ch) source))
                              (rightc))
                          (pop-position)))))
 (Package: BISY - Userfunction)
 (iscalled by: (bisy bisy-reader eval-buffer))
 (calls: (as command:   push-position set-position rightc)
         (as function:  get-char&att char pop-position)
         (as predicate: eobp))
 (type: function)
 (parameters: 0)
 (local variables: (ch     (type: fixnum)
                           (used as: (+ ASCII-Code (* 256 Attribut)))
                           (possible values: from-0-to-65536))
                   (source (type: list-of-fixnums)
                           (used as: input-stream)
                           (possible values: all-lists-of-fixnums)))
 (free variables: ())
 (history: (defined: 20/1/1982)
           (modified: 10/6/1982)
           (programmer: JOBA)
           (reasons for change: "error while reading lisp comment lines"))
 (side effects: ())
 (error routines: errset))
```

FIG. 6. Example of a knowledge structure for a LISP function.

notice an important message? Which new techniques are possible?

It is quite obvious from the knowledge structures in Fig. 6 that it is not useful to display the information in this form. We want to have a system which supports multiple windows (Bauer and coworkers, 1981), so we can see selected views of the complex structure (see Fig. 8). To be convivial the user should have control to define filters so he can determine the relevancy of the information. Generating context-sensitive, multiple perspectives each of reduced complexity is a common technique among designers which is used to dissect otherwise intractable entities (compare the use of maps that picture economic, political and hydrological perspectives of a country). The use of windows supports this approach and allows us to regard a program as a complex information structure, of which only those parts which are of interest to the programmer are displayed at a certain time (this principle is sometimes called 'progressive disclosure').

A window system is also a prerequisite to implement a good browser (Goldstein and Bobrow, 1981). A browser is a display-based interface which allows a user to examine a complex environment *without* prior knowledge of its exact structure. It allows the user to navigate in a complex space and to focus on his area of interest. Browsing capabilities can be used to filter information and summarize it into semantic units.

In Fig. 9 we show one of our browsers (for details

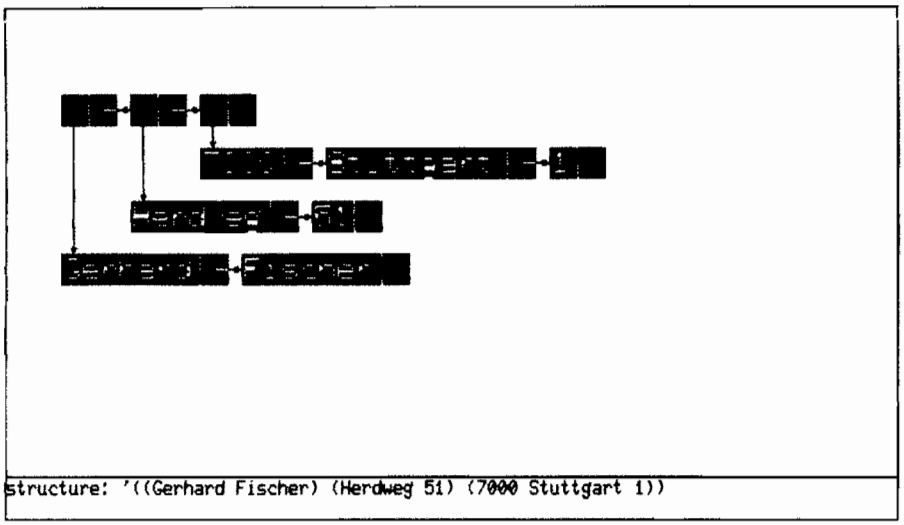

FIG. 7. Visual representation of list structures (Nieper, 1983). The two-dimensional representation is generated from the symbolic description. Circular lists can easily be represented and editing can be done by manipulating directly (with a pointing device) the graphical representation.

see Fischer and co-workers, 1981). With the help of this browser we can traverse a hierarchical network by sequentially selecting items starting in the leftmost top window (selection is shown by underlining). Based on our selection the contents of the next window changes accordingly.

User interfaces are a specific, but important issue in our research on human–computer communication within the project INFORM. One empirical finding of our work so far is that a knowledge based system is of little use if the information cannot be delivered to the user in a way which takes the cognitive limitations of the human into account.

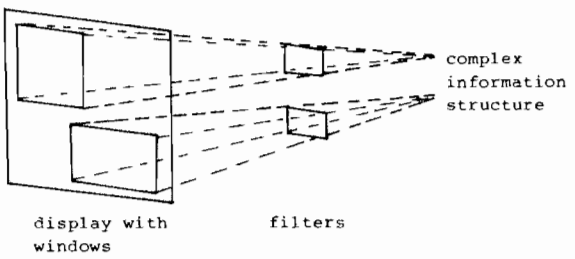

FIG. 8. Example for multiple perspectives using multiple windows and filters.

CONCLUSIONS

Symbiotic, knowledge-based systems are valuable tools which enhance human capabilities in many dimensions (e.g. reducing the mental load; pushing off clerical details to the machine so we can concentrate on the difficult conceptual parts of a problem; enlarging our strategies to tackle a complex problem) and our research has provided us with a deeper understanding of the design issues for future systems of this sort. The substantial computing power which will be available in future systems should be used to expand human potential and magnify human productivity. Work in

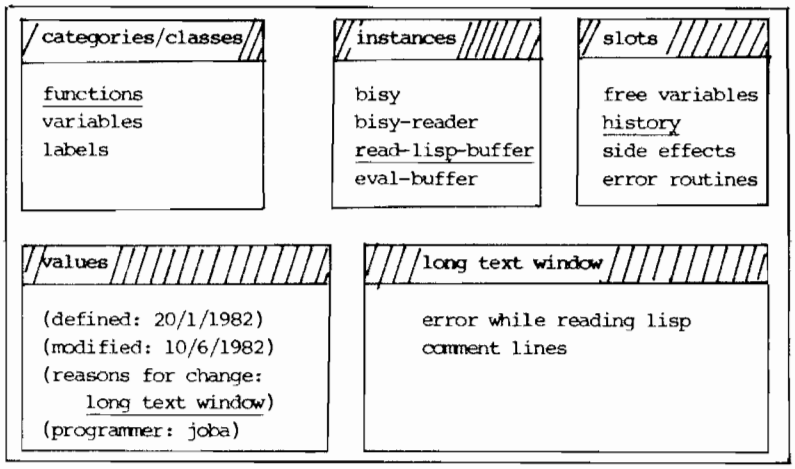

FIG. 9. An example of one of our browsers.

human–computer communication should contribute to the goal that we are able to construct human-oriented computers to avoid that humans must become computer-oriented persons.

REFERENCES

Bauer, J., H.-D. Boecker and G. Fischer (1981). Entwurf und Implementation eines Systems multipler Fenster. In *Helft 6 der Notitzen zum Interaktiven Programmieren, Fachgruppe Interaktives Programmieren in der Gesellschaft fuer Informatik*, Oldenburg, pp. 90–99.

Bauer, J., H.-D. Boecker, F. Fabian, G. Fischer, R. Gunzenhaeuser and C. Rathke (1982). Wissensbasierte Systeme zur Verbesserung der Mensch-Maschine-Kommunikation. Antrag an das BMFT.

Boecker, H.-D., G. Fischer and R. Gunzenhaeuser (1980). Die Funktion von integrierten Informationsmanipulationssystemen in der Mensch-Maschine Kommunikation. MMK-Memo, Instut für Informatik der Universitaet Stuttgart.

Fauser, A. and C. Rathke (1981). Studie zum Stand der Forschung ueber natuerlichsprachliche Frage/Ant-Wortsysteme. BMFT-FB-ID 81-006, Institut für Informatik, Universitaet Stuttgart.

Fischer, G. (1980). Cognitive dimensions of information manipulation systems. In R. Wossidlo (Ed), *Textverarbeitung und Informatik*. Springer, pp. 17–31.

Fischer, G. and H.-D. Boecker (1983). The nature of design processes and how computer systems can support them. In P. Degano and E. Sandewall (Eds), *Integrated Interactive Computing Systems*. North Holland, Amsterdam, pp. 73–86.

Fischer, G. and J. Laubsch (1979). Object-oriented programming. In *Heft 2 der Notitzen zum Interaktiven Programmieren*, Fachgruppe Interaktives Programmieren der Gesellschaft für Informatik, Stuttgart, pp. 121–140.

Fischer, G., J. Failenschmid, W. Maier and H. Straub (1981). Symbiotic systems for program development and analysis. MMK-Memo, Institut für Informatik der Universitaet Stuttgart.

Goldberg, A. (Ed.) (1981). SMALLTALK. Special issue, *BYTE*, **6**, No. 8.

Goldstein, I. and D. G. Bobrow (1981). Browsing in a programming environment. In *Proceedings of the 14th Hawaii Conference on System Science*.

Illich, I. (1973). *Tools for Conviviality*. Harper & Row, New York.

Kay, M. (1980). The proper place of man and machines in language translation. CSL-80-11, Xerox Parc. Palo Alto, CA.

Maier, D. (1983). Computerunterstuetzte Planungsprozesse am Beispiel des zweiten Studienabschnittes des Informatikstudiums Diplomarbeit Nr. 227, Institut für Informatik der Universitaet Stuttgart.

McCracken, D. (1979). Man + computer: a new symbiosis. *CACM*, **22**, 587.

Moran, T. (Ed.) (1981). The psychology of human–computer interaction. *ACM Computing Surveys*, **13**, No. 1.

Newell, A. and H. Simon (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ.

Nieper, H. (1983). KAESTLE. Ein graphischer Editor für LISP-Datenstrukturen. Studienarbeit Nr. 347, Institut für Informatik der Universitaet Stuttgart.

Norman, D. (1981). *Perspectives on Cognitive Science*. Lawrence Erlbaum, Hillsdale, NJ.

Rathke, C. (1983). Wissensbasierte Systeme: Mehr als eine attraktive Bildschirmgestaltung. *Computer Magazin*. No. 3,40.

Rathke, C. and J. Laubsch (1983). OBJTALK, eine Erweiterung von LISP zum objekt-orientierten Programmieren. In H. Stoyan and H. Wedekind (Eds), *Objektorientierte Software- und Hardwarearchitekturen*. Teubner, Stuttgart, pp. 60–75.

Shortliffe, E. H. (1976). *Computer-based Medical Consultations: MYCIN*. Elsevier, New York.

Simon, H. (1981). *The Sciences of the Artificial*. MIT Press, Cambridge, MA, 2nd edition.

Teitelman, W. and L. Masinter (1981). The Interlisp programming environment. *Computer*, April, 25.