NONPROCEDURAL COMMUNICATION BETWEEN USER AND APPLICATION SOFTWARE

Borivoj Melichar

September 1979
WP-79-115

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

B. MELICHAR is Assistant Professor at the Czech University of Technology, Department of Computers, Karlovo namesti 13, 12000 PRAGUE 2, Czechoslovakia.

PREFACE

Borivoj Melichar participated in IIASA Junior Scientist Program three months during the summer 1979. This paper is one of the results of his stay at IIASA where he joined the Management and Technology Research Area. He was associated with the study of the impact of small scale computers on managerial tasks. Within this task the future of software development has been identified as one of possible bottlenecks to a healthy balance between cheap hardware and user-friendly software. Nonprocedural languages are often mentioned as one major way of reducing the potential problems. A comprehensive survey of this field seems not so far to be available. The present survey on nonprocedural communication between user and application software is a contribution to fill this gap.

Laxenburg,October 1979
Goeran Fick

SUMMARY

The present paper is a survey of nonprocedural communication between a user and application software in interactive data processing systems. It includes a description of the main features of interactive sytems, a classification of potential users of application software,and a definition of the nonprocedural interface. An annotated classification of the main types of nonprocedural languages is presented. Future trends in user-computer interfaces and possible developments of languages for managers are mentioned as well.

CONTENTS

Nonprocedural Communication Between User and Application Software

Borivoj Melichar

1. Introduction

1.1 Basic Concepts

Advances in semiconductor technology during the past decade have dramatically increased the availability of low cost computer hardware. One of the results of this expanded availability has been the development of cheap but powerful small scale computer systems.

According to Fick (1979a) the power of computer systems has more than doubled every two years, while the cost has remained constant. With the "real" cost of computing capability declining, it is, nonetheless, apparent that low cost computer hardware does not necessarily mean "cheap" computing. Computer hardware is only one part of computer systems . The other part is software. Computer software is a labor-intensive product, mainly customized for a small group of users or even for an individual user (Fick 1979b). Therefore, traditional software is more expensive than mass-producible hardware. There is a number of problems with software for low cost computer hardware. The main areas of problems associated with software design, implementation, and utilization are as follows:

a. Development of theoretical and methodological tools useful for software design in different fields of computer applications.
b. Development of tools for software realization (programming languages, automatic program generation, program debugging and verification, etc.).

c. Development and production of media for software distribution (semiconductor read only memories, magnetic tapes, magnetic discs, punched cards, punched paper tapes, books, journals, etc.)
d. Development of means for communication between user and the application software (input/output devices, communication languages, etc.)

In this article we survey problems of communication between a user and the application software. The manner of communication between a user and the application software is highly dependent on the method of access to the computer system. In recent years there was much discussion of the issue of "indirect access" vs. "direct access," e.g., batch processing systems vs. time sharing systems. The communication between user and computer is very slow in a batch processing system. The user can neither influence the way of running the program nor intervene during the running of the program. Therefore, the issue of batch processing systems vs. time sharing systems has been resolved in favor of time sharing systems. This means that communication between user and computer has an interactive nature. Current discussions now usually assume interactive systems and are directed to features of such systems.

1.2 Interactive Systems

Many problems of batch processing systems are overcome by the opportunity to communicate directly with the computer through an interactive dialogue. However, some new problems have appeared during the period of utilization of interactive systems. These problems are being broadly discussed, and requirements for basic features of interactive systems and the basic principles of the interactive dialogue are being debated (Miller and Thomas 1977, Watson 1976, Fitter 1979, Gaines and Facey 1976). Here we offer a list of the most important features of interactive systems and some principles of interactive dialogue. Some of the following points may be also relevant to a batch processing (non-interactive) environment, but we focus only on interactive systems.

a. Systems Response Time

System response time is the time spent in processing of user's input and in displaying response. The question of how fast or slow response times should be is a difficult one. There is no common agreement about the required absolute magnitude of the system response time. There are several arguments for fast response times:

+ human being works on mean of two seconds;
+ slow response times decrease throughput;
+ long delays are usually disruptive and disturbing.

On the other hand, arguments exist against fast response times:

- fast response times mean high investment in the system;
- for more complex tasks slow response times might be helpful;
- fast responses may cause users to expect level of service all the time.

It has been observed that the variability of the system response time is one of the most distressing factors for users.

Rohlfs(1979) proposes,that systems should be designed so that their response time is adjusted to user's individual activity:

>15 sec	intolerable,
> 4 sec	too long in most cases,possibly tolerable after termination of major work step,
> 2 sec	too long for high concentration work,
< 2 sec	necessary for work consisting of more than one step,
< 1 sec	immediate reaction.

b. Availability and Reliability

A computer system may be available at any time for users. This feature would be realized by personal computers. Because the user will be unhappy with any system performance error or degradation regardless of good normal performance, reliability is also a very important feature of an interactive system. For many computer applications almost no loss in availability or degradation can be tolerated.

c. Commonality

A software system is usually composed of a varying number of subsystems. It may be assumed that in all subsystems terminology and operations are applied consistently. This means that the input language of each subsystem is an extension of the common base language. Thus, the user will learn to use only additional functions or statements when using a new subsystem and not have to learn new "foreign" language. When the user is in trouble, he/she can use help functions in a standardized way.

d. Adaptability to User Proficiency

The interface between user and computer may be oriented to users with different amounts of knowledge about a particular subsystem. A sophisticated user may prefer to use mathematical or formalized notation in his/her dialogue. On the other hand,

a novice user is likely to prefer less formalized notation and use simpler system functions.

Recent systems have been oriented to users with various levels of proficiency by designing different levels of user interfaces. As soon as the user becomes more proficient, he/she can use more sophisticated functions or a more formalized interface.

e. Immediate Feedback

A system may give the user feedback for each of his/her requests by making an unambiguous response. This should be sufficient to identify the activity of the system. In situations where system response times are longer than usual, it is highly desirable to confirm receipt of the user's command immediately. It is very useful to display time information (for instance,by a countdown clock) on the terminal to inform the user of when the computer will produce a response.

f. Observability and Controlability

A system may be regarded as an automaton. When the user's input is processed, than the following process is regarded as a transition from one state to another. The user can be informed about the current state of the system. It is important that the user feels in control of the system, and in order to make this control possible he/she must have knowledge about the current state of the system.

g. Use the User's Model

All people rationalize their experience in their own terms, and each user will model a computer system according to his/her experience of it. This cannot be prevented and should be made as easy as possible.A system should use a model of the activity which corresponds to that of the user. Interactive dialogue should resemble a conversation between two users accepting the same model. Given that we can somehow determine the user's model we should make the underlying processes reflect it, and design the dialogue to reveal it as clearly as possible.

h. Validation

All input commands and data must be validated by checking syntax, semantics, and, if possible, values. The system must clearly inform the user about errors and ambiguities in input data and let the user update those in question before the system acts upon them.

i. Query-in-depth

Information and tutorial and help material can be divided according to possible user requests and accessed by the user through a simple uniform mechanism.

j. Extensibility

There will never be enough professional programmers and system developers to build all the tools that users may desire for their work. Therefore, it shold be possible for users to add new tools or extend the interface functions.

k. Written Documentation

In some cases it is necessary to produce high-quality documents as a result of terminal interaction. This is especially true for text processing.

l. System Activities

To maintain records of system performance and user's activities, it is necessary to evaluate and improve the behavior of the system.

1.3 Interface between the User and Application Software

The nature of a user-application software interface is mainly determined by the medium used for communication. As basic media for this communication we can use

-- alphanumeric texts, or
-- graphics.

More advanced media for communication,such as speech, eye movement, brain wave control, and hand written script (Watson 1976), are being researched.

In this survey we focus on alphanumeric texts as a medium for user-computer communication. It is supposed, that a usual keybord and alphanumeric display (with/without hardcopy) are available for the user.

The user-software interface has two sides (Watson 1976, Sprague 1979):

-- the input side by which the user inputs information by means of an action language;
-- the output side by which the computer provides feedback and other assistance to the user by means of a screen language.

First let us survey the action languages. A wide range of action languages have been designed to accommodate the variety of users. The selection of a particular action language determines the communication mode that is used. We can classify action language and/or communication modes as follows:

a. low level machine-like programming languages,
b. high level universal programming languages,
c. high level programming languages with embedded new syntax and semantic forms (Such languages can be used as special purpose languages.),
d. self contained special purpose languages,
e. answer languages,
f. command languages,
g. query languages,
h. natural languages,and
i. two-dimensional positional languages.

This classification of communication modes follows the range available from the artificial machine-oriented languages to the natural human-oriented languages.

Because diversity of user types may require a spectrum of communication modes, it is very important during software development and implementation to select communication modes which are appropriate to the end-users.

According to Schneiderman (1978) we can categorize users with respect to their skills, frequency of using application software, and according to their professional fields as follows:

a. Non-trained intermittent users who infrequently use application software. The user from this category is called a "casual user" (Codd 1977) or "general user" (Miller and Thomas 1977). These people are not computer professionals, have no syntactic knowledge, and have little knowledge of the application software organization. At the same time it is assumed, however, that these users have sound professional knowledge in their fields and, therefore, that the system should allow them to express themselves in the terminology of their respective professional fields (Lehmann, 1978).
b. Skilled frequent users who make daily use of application software. These users can learn a simple syntax of communication action language, but they are more interested in their own work than in computer programming. This category includes skilled secretaries, engineers, managers, etc.
c. Professional users whose main role is to develop and maintain the application software. They apply their long experience and accept substantial training and are concerned with the efficiency and the quality of the computer system work. This category includes data base administrators, software system programmers, etc.

Although programming and communication with a computer in high level programming languages like ALGOL, FORTRAN, COBOL, PL/1 and PASCAL are a major advance over earlier methods (programming in the machine-like low level programming languages),such languages are not sufficient for current and future software, given a cheap hardware. With the great diffusion of cheap computer hardware it is expected that the categories of the non-trained intermittent and the skilled frequent users will expand very rapidly. These users will have little or no data processing background. For such people it is very hard and very time consuming to learn a method of construction and description of an algorithm in programming procedure oriented languages. Therefore, it is highly desirable to make it possible for such categories of users to communicate with an application software in a non-procedural manner.

There is very strong interest in the development of non-procedural languages not only to facilitate communication between the user and the application software, but also from the point of view of application software implementation.

According to Winograd (1979) it is useful to distinguish three types of specifications of computational processes:

a. Program specification. A formal structure which can be interpreted as a set of instructions for a given computer. This is the imperative style of traditional procedure oriented programming languages.
b. Result specification. A process-independent specification of the relationship between the inputs (or initial state), internal variables, and outputs (or resulting state) of the program.
c. Behavior specification. A formal description of the time-course of activity of a computer. Any such description selects certain features of the machine state and action without specifying in full detail the mechanisms which generate these.

If we attempt to analyze an algorithm, we can regard it as consisting of two components (Kowalski 1979), a

-- logic component, which specifies the knowledge to be used in solving the problem and a
-- control component, which determines the problem solving strategy by means of which the knowledge will be used.

An algorithm for computing factorials offers a simple example. The definition of a factorial constitutes the logic component of the algorithm:

1 is the factorial of 0;
u is the factorial of x if v is the factorial of x-1 and u is v times x.

This is a result specification of the computational process (the knowledge which can be used to compute the factorial).

For comparison we introduce a procedure in ALGOL 60 as a program specification to compute the factorial.

```
procedure factorial (x); value x; integer x;

if x = 0 then factorial : = 1 else
     factorial : = factorial (x-1) * x;
```

In this procedure the logic component is blended with the control component.

According to McCracken's (1978) ideas, we can characterize non-procedural languages as follows:

a. The user cannot take any care of storing data. Decisions that relate only indirectly to the calculation are considered to be part of the internal functioning of the system. These include decisions about internal representation of numbers (fixed point, floating point, octal, decimal), dimensions of quantities that occur only as intermediate results, input and output formats, etc. The representation of data is selected by the system itself, and the description of the data representation is stored with the data. This is called data independence.

b. The user cannot tell the computer how to do a process to obtain desirable results.Rather, he tells the computer only what he wants. This means that user input does not involve the loops and branches which make up most of the computational steps in a program written in procedural language. This we can call control independence.

As an illustration of the non-procedural approach here is a query on the data stored in a data base.

```
RETRIEVE (AGE>40 AND <65) AND SALARY >3,000;
FOR EACH
     IF WEIGHT > TABLE (HEIGHT -50)
     THEN SET OVERWEIGHT = "TRUE"
     PENSION = SALARY/3;
     ELSE SET PENSION = SALARY/2;
```

On the basis of the ideas mentioned above we can give a working definition of non-procedural language:

In a non-procedural language the computational process is specified by the desired result (or behavior). This specification is data independent and control independent.

We shall consider as a non-procedural oriented language such a language which does not fully satisfy all conditions of the definition of the non-procedural language, but which does not require program specification.

From the classification of communication modes listed above we can consider answer languages, command languages, query languages, natural languages, two-dimensional positional languages, and partly special purpose languages as non-procedural oriented action languages.

In the next section we introduce an annotated classification and give examples of non-procedural action languages available for the communication between non-trained intermittent or skilled frequent users and application software.

Communication in the direction from the system to the user is realized by means of screen languages.

The description of the main features of the screen languages is contained in the third section.

2. Nonprocedural Oriented Action Languages

In the last section we began to define the concept of nonprocedural action languages and sketched the arguments for using them for user-application software communication. In this section we shall give the basic characteristics of each language type and introduce examples of them. Literature references were chosen to provide examples of relevant concepts, but the number of references is deliberately low. This section is structured according to the following list of the language types:

a) answer languages,
b) command languages,
c) query languages,
d) natural languages,
e) special purpose languages,
f) two-dimensional positional languages.

2.1 Answer Languages

Answer language is the set of words, phrases, or sentences, which may be used to answer questions asked by the computer. This type of language we introduce first, because it is the simplest language for the user-computer communication.

The answer languages used as action languages are very closely related to screen languages. The screen language in this case contains, among others, the set of questions which are asked and which the user is obliged to answer. With respect to the complexity of answers we can divide the action answer languages into the following types:

a. binary answer language,
b. menu selection systems,
c. instruction and response systems,
d. displayed format systems.

Binary answer language is composed of two answers, YES and NO, often represented by their abbreviations (for example Y, N).

The binary answer languages are used in software systems in which the internal structure corresponds to a binary oriented graph. In the binary oriented graph two oriented branches leave each edge. Edges correspond to states of the system, and in each such state the system asks a question, and according to the answer (NO or YES) the first or the second branch is selected by which to depart from the edge and reach a new one. The binary answer language is used as an action language mainly in simple systems like computer games.

As an example we use the popular game Black Jack (Thompson and Ritchie 1975). The dealer (simulated by computer) might ask the following questions:

New game?
Hit?
Insurance?
Split pair?
Double down?

Each question is answered by YES or NO.

From the above description it is clear that binary answer language may be used only in systems in which a limited number of questions may be asked.

In the case when the answers YES or NO are not sufficient for answering all questions, we can use a menu selection system (n-ary answer language).

In a menu selection system the set of possible answers to each question is defined. Each set of answers must be finite, and from a practical point of view should be a small cardinal number.

The set of answers to a particular question is called the "menu". There are two ways for presentation of the menu to the user. First, the definition of the menu for each question may be contained in the description of the software system, and the user is thus obliged to learn these menus before using the system.

Much better is the second way, in which the menu is contained in the question asked. This method fulfils a number of important functions. The user need not learn the menus for all possible questions, if a repetoire of answers from which to

choose is provided to the user as a part of the question. With this method the user may not remember the answers or may not be aware of the existence of possible answers.

Menu selection systems, analogously to binary answer languages, are used in software systems in which the internal structure corresponds to an ordinary oriented graph. In such a graph a varying number of branches leave from each edge. Each branch corresponds to one possible answer to the question which corresponds to the edge from which the branch is leaving.

As an example we use some menus from Teitelman (1979). In this paper is described a system which provides the user assistance in the development of programs. As a part of the user interface the menu selection system is used.

In this system, for example, the following question may be asked:

```
MENUS:  WINDOW
        DOCUMENT
        EDIT
        LOOK
        HISTORY
        BREAK
        OPERATIONS
(This menu is used to select further menus.)

WINDOW:  READ
         MOVE
         GROW
         SHRINK
         PUT ON TOP
         PUT ON BOTTOM
         KILL
         MAKE INVISIBLE

EDIT:    INSERT
         APPEND
         DELETE
         REPLACE
         MOVE
         ---->
         <----
         DONE
```

Questions with menus are displayed on a screen, and the user can select an answer by means of the cursor.

When the number of answers to a particular question grows, it is inefficient (or sometimes impossible) to display all possible answers as a part of the question. We meet such a situation if the answer contains a number. In such cases we may use an instruction and response system.

In an instruction and response system an explanation of the content of an answer is a part of the question. The

following example of an instruction and response dialogue is from Hebditch (1979).

```
ORDER OR CREDIT? O
CUSTOMER NUMBER? 848923
CUSTOMER IS BROWN'S STATIONERS LTD
            HIGH STREET
            WATFORD
PLEASE CONFIRM (Y/N)? Y
DELIVERY ADDRESS IS AS ABOVE
PLEASE CONFIRM (Y/N)? Y
ORDER NUMBER? 77/34
DISCOUNT? 12.5
***12.5 PERCENT IS HIGHER THAN NORMAL TERMS
PLEASE CONFIRM BY RE-ENTRY? 12.5
ENTER PRODUCT CODE. QUANTITY (END AFTER LAST ITEM)
?BO4,24  24 DOZ PENCILS (HB)
?A68,1O  1Ø REAMS BANK PAPER
?B61,36  BALL-POINT PENS
***36 DOZ IS ABNORMAL QUANTITY FOR THIS ITEM.  PLEASE CONFIRM
?B61,3   3 DOZ BALL-POINT PENS (BLUE)
?Z15,1   1 DISPLAY STAND (BALL-POINT PENS)
?END     ORDER COMPLETED (4 ITEMS)
DO YOU WISH TO SEE INVOICE PRIOR TO PRINTING (Y/N)? N
```

In systems, which require a fixed format of an answer, the description of the format may be contained in the question. Such a system is called a displayed format system.

The following simple and self explanatory example (Hebditch 1979) shows the question with the format description and answer.

```
                        BOOK ORDER

ENTER  AUTHOR / TITLE / PUBLISHER / ISBN / NO. OF COPIES /
       CUSTOMER NAME / CUSTOMER ADDRESS / POST OR COLLECT?

HEBDITCH/DATA COMMUNICATIONS:  AN INTRODUCTORY GUIDE/
ELEK SCIENCE LTD/NK/4/A WISEMAN/NA/COLLECT
```

2.2 Command Languages

Command languages in one form or another have been in use since the earliest operating systems of the late 1950's. The name command languages was used in the past for job control languages used as the interface between users and operating systems. Today command languages encompass a wide class of languages used for user-interfaces in many kinds of software systems.

The command language consists of a set of commands. A command of a typical command language is composed of the following parts:

a) command prefix,
b) operation specification,
c) parameter part, and
d) command completion.

The first part of the command, the command prefix is used as

-- command indication (a symbol or string of symbols to distinguish the command from other inputs),
-- command identification (a label or number used for reference purposes in other commands), and
-- condition, which must be satisfied to execute the command (for example: IF TIME < MAXTIME THEN)

As an operation specification a reserved command word is frequently used (Miller and Thomas 1977).

Watson (1976) proposes an operation specification of the form verb-noun pair. In this case we obtain a verb-noun matrix as for example in an editing system:

	character	line	page
delete			
insert			
change			
move			

Each element of such a matrix is a normal English imperative mood created according to the paradigm

do this.

Layout of the operation specification in this form seems to be very useful for users with no data processing background (Keen and Hackathorn 1979). Hebditch (1973) proposes a more structured operation specification layout using adverb and adjective to create an operation specification with three forms (V-verb, N-noun, A-adverb, J-adjective):

```
V A N     (PRINT CONDENSED RECORD)
V N J     (FIND EMPLOYEES WITH A DEGREE)
V A N J   (PRINT ALL LINES BEGINING WITH +)
```

Further, Hebditch (1973) proposes that a set of basic operation specification verbs of the command language be used as an interface to a data base:

Function	Operation specification verb	Short form	Alternatives
Initiation	START	S	Begin, Sign-on, Initiate, Go, Set up, Evoke
Location of logical record	FIND	F	Locate, Get, Search, Read, Obtain, Pick (good for inventory data base?)
Display of data item	PRINT (for hard copy)	P	DISPLAY (for VDUs), Show, Query, Give, List, Present
Amendment of data item	ALTER	A	Change, Amend, Modify, Replace, Convert, Set
Addition of new record or item	INSERT	I	Add, New, Assign, Include Originate, Form
Movement of record (or data) from one logical location to another	MOVE	M	Transfer, Shift, Relocate, Convey, Reallocate, Transpose
Obtain assistance	EXPLAIN	E	Assist, Why?, Expand, Clarify, Help, Interpret
Termination	HALT	H	End, Finish, Done, Close Terminate, Conclude

The parameter part involves operand specifications and various options or alternatives for the execution of the particular command. There are two distinct methods of formatting the arguments for commands: positional format or keyword format.

If the positional format is used, then specific kinds of information must appear in a fixed, relative, or absolute position in the parameter part.

In the case of the keyword format the parameter part is a permutable string of arguments and each argument contains the special word, keyword, indicating the argument type and, sometimes, its value.

Both types of argument format occur in today's systems. From the user's point of view the keyword format is more acceptable, because the values of the arguments must be remembered in both cases, and because remembering the position in the case of the positional format is additional memory load for the user.

The arguments in the parameter part are composed from several types of items:

-- keywords,
-- constants (numeric, boolean,...),
-- text strings,and
-- expressions (regular, arithmetic, boolean,...).

There remains the question of what to do when the user does not specify some arguments or any other information that either could or should have been provided. There are several options for prompting the user for missing information:

a) listing of the missing argument names with all potential values for each of them to allow the user to choose from;
b) assigning a default value automatically to some of the missing arguments and asking the user for agreement;
c) supplying missing information on the basis of the arguments supplied to previous commands.

Related to argument specification is the problem of choosing argument delimiters. The following are delimiting functions (Watson 1976):

a) delimiting command words,
b) delimiting arguments,and
c) delimiting optional arguments or arguments with default values.

It is recommended to use the same symbol for delimiting command words and arguments and to use a different symbol to delimite optional or default arguments.

The last part of the command is a command completion. According to Watson (1976) there are three types of command completions.

a) Command accept: Completion of the command indicating the execution of the command and the return to the base state to receive the next command.
b) Repeat: Completion of the command and return to an intermediate command state for quick repetion of the command with or without request. This mode is useful when an operation is repeated several times.
c) Insert: Completion of the command and entry to insert-command mode for insertion of some new parameters and repeat.

For each type of command completion a different symbol has to be used.

2.3 Query Languages

Many software products include parts consisting of data bases. The main operations that users wish to perform on data bases can be listed as follows (Schneiderman 1978):

a) insertion of one or more items of information;
b) deletion of one or more items of information;
c) retrieval of information from data base;
d) locking and unlocking of items to provide integrity during concurrent processing;
e) privacy check to ensure that user is permitted to perform the operation requested;
f) data definition to create a schema or subschema of the data base; and,
g) utility functions including administration operations such as initial load, physical reorganization, logical restructuring, data translation, performance statistics collection, and data validation.

According to Olle (1973) there are four levels of user interface with a data base:

a) At the top level is the data administrator, that is, the person responsible for the data base.
b) At the conventional level, there is the applications programmer.
c) At the next level is the non-programmer, who is the individual who understands how to formulate questions about data stored in data base.
d) At the lowest level is the individual, who does not know how to formulate a question.

Data administrators and applications programmers use mainly programming languages in their work. For users who do not know how to formulate a question the answer languages are available.

For users who are non-programmers but understand how to formulate questions the query languages are avilable.

Query languages are high level non-procedural data base languages, which provide the user with a simple interface to express operations on the data base, namely operations like insertion, deletion, and retrieval. Strong emphasis is placed on the retrieval operations on the data base.

In view of this fact, a finer categorization of retrieval operations is appropriate (Schneiderman 1978):

a) Simple verification of the presence, absence, or acceptability of a specified item.
b) Single record retrieval when a key is provided.
c) Record collection retrieval when a key or boolean predicate is provided.
d) Total report listing of all information stored.

For the same reason, that the retrieval operations are most frequently used, we list query features (Schneiderman 1978):

a) Simple mapping returns data values when a known data value for another field is supplied. Example: Find the names of employees in department 50.
b) Selection of all data values associated with a specified key value. Example: Give the entire record for the employee whose name is John Jones.
c) Projection in the relational model is an entire column or domain of a relation. Example: Print the names of all employees.
d) Boolean queries are those which permit AND/OR/NOT connections. Example: Find the names of employees who work for Smith and are not in department 50.
e) Set operation queries are those which permit set operators like intersection, union, symmetric difference, etc.
f) Built-in functions provide special-purpose functions to aid in question formulation. These are functions like MAXIMUM, MINIMUM, AVERAGE, SUM,... Example: Print the sum of salaries in department 50.
g) Combination (composition) queries are the result of using the output of one query as the input for another. Example: Find the names of all departments which have more than 30 employees and then find the names of the department managers.
h) Grouping of items with a common domain value. Example: Print the names of departments where the average salary is greater than 15,000.
i) Universal quantification corresponding to the "for all" concept of the first-order predicate calculus.

The query features listed are available in most query languages that have been designed for data bases with various models of data (relational, hierarchical, network).

In order to provide examples of queries we introduce an example of the data base with a relational model of data (Chamberlin 1976):

PRESIDENTS

NAME	PARTY	HOME-STATE
Eisenhower	Republican	Texas
Kennedy	Democrat	Mass.
Johnson	Democrat	Texas
Nixon	Republican	Calif.
Carter	Democrat	Georgia

This relation PRESIDENTS has domains NAME, PARTY and HOME-STATE.

ELECTIONS-WON

YEAR	WINNER-NAME
1952	Eisenhower
1956	Eisenhower
1960	Kennedy
1964	Johnson
1968	Nixon
1972	Nixon
1976	Carter

This relation ELECTIONS-WON has domains YEAR and WINNER-NAME.

The two relations PRESIDENTS and ELECTIONS-WON are the only relations in our example data base.

According to Chamberlin (1976) we can illustrate the variety of the query languages by presenting examples of four classes of them:

a. relational calculus based languages,
b. relational algebra based query languages,
c. mapping-oriented languages,and
d. graphics-oriented languages.

An example of a relational calculus based query language is the language QUEL (Stonebracker et al. 1976).

A typical query in QUEL has three parts:

-- result name, which is the name of the relation which will be retrieved into,
-- target list, which specifies the particular domains of the relation which are to be returned,
-- qualification which selects particular tuples from the target relation by giving a condition which they must satisfy.

A QUEL interaction includes at least one RANGE statement to specify the relation over which each variable ranges.

Two examples of queries are:

What was the home state of President Kennedy?

```
RANGE OF P IS PRESIDNETS
RETRIEVE INTO X (STATE = P.HOME-STATE)
WHERE P.NAME = "KENNEDY"
```

List the election years in which a Republican from Ilinois was elected?

```
RANGE OF E IS ELECTIONS-WON
RANGE OF P IS PRESIDENTS
RETRIEVE INTO Y (YEARS = E. YEAR)
WHERE P.PARTY = "REPUBLICAN"
  AND P.HOME-STATE = "ILLINOIS"
  AND P.NAME = E.WINNER-NAME
```

Relational algebra based query languages use a collection of operators that deal with relations, yielding new relations as a result. The major operators of relational algebra incluce the following:

-- projection,
-- restriction, and
-- set-theoretic operators (union, intersection and symetric difference).

The same two examples of queries as above in relational algebra based query language are:

What was the home state of President Kennedy?

```
PRESIDENTS [NAME = "KENNEDY"] [HOME-STATE]
```

In the above example we use projection and restriction operations.

List the election years in which a Republican from Illinois was elected!

```
(ELECTIONS-WON [WINNER-NAME = NAME] PRESIDENTS)
[PARTY = "REPUBLICAN"] [HOME-STATE = "ILLINOIS"] [YEAR]
```

In this example we use join, projection and restriction operations.

The basic building block of mapping oriented query languages is the "mapping" which maps a known domain or set of domains into a desired domain or set of domains by means of some relation. The same two examples as above are in language SEQUEL (Astrahan et al. 1976).

What was the home-state of President Kennedy?

```
SELECT HOME-STATE
FROM PRESIDENTS
WHERE NAME = "KENNEDY"
```

List the election years in which Republican from Illinois was elected!

```
SELECT YEAR
FROM ELECTIONS-WON
WHERE WINNER-NAME =
      SELECT NAME
      FROM PRESIDENTS
```

```
WHERE PARTY = "REPUBLICAN"
AND HOME-STATE = "ILLINOIS"
```

Graphics oriented query languages are in this survey mentioned are in this survey mentioned later in the section dealing with two-dimensional positional languages.

The mapping and graphics oriented query languages are directed to the users with no data processing background and offer power equivalent to relational algebra or relational calculus based languages while avoiding mathematical concepts such as quantifiers and operations on relations.

2.4 Natural Languages

Natural language communication with computers has provoked discussion from the early years of machine translation. While the concept of natural language interface with computers is obviously attractive, to implement it is another matter.

According to Addis (1977) natural language is the technique of verbal communicaton betwen people. Natural language is extremely complex in its structure because it reflects the way men think.

The utilization of natural language for man-computer communication has several principle advantages (Man/Computer Communication 1979):

a) It provides an already familiar way of forming questions. This means that the natural language interface is available to a large number of users without special training or learning even of a high level non-procedural formal language.
b) There are often many ways to extract the same data. The user can usually communicate his/her knowledge in natural language augumented by specialized notations and vocabularies particular to his/her domain.
c) The formulation of more complicated queries may be easier compared with formal languages or menu selection methods.
d) The user does not have to learn a formal syntax and his/her departures from accepted grammar may be tolerated without comments.

At the same time we must note the following disadvantages:

a) The use of natural language interface encourages an unrealistic expectation of the system power.
b) The linguistic limitations of such a system are not as well defined as they are with a formal language. They can arise as a result of an unknown word, a grammatical construction, or a misunderstanding.
c) In natural languages sentences are frequently ambiguous. Implementation is difficult if all possibilities of meaning of a sentence must be

considered.

d) Because much of the vocabulary may be specific to the particular domain, the system has to be partially recast for each new domain of discourse.

e) The system with a natural language interface is inherently much more complicated to implement to provide an acceptable interface.

These disadvantages do not imply that natural language interface is useless, only that its domain of application is less broad than is common opinion. Moreover, natural language interface may not be preferable in every situation. Schneiderman (1978) describes a "natural versus artificial query language experiment" concerning communication with data bases and concludes that user knowledge of the applicaton domain is critical. Without this prerequisite natural language usage would be difficult. The user has to know the semantics of the information of the data base. When a user learns a query language, he/she at the same time learns the semantics of the information stored in the data base.

On the other hand, Newsted and Wynne (1976) describe a decision system support called AIDS with a user interface of the natural language type, and they state that the system was successfully applied to a wide range of topics.

2.5 Special Purpose Languages

Specific kinds of formalisms are defined in some areas to provide a formalized description of the problems. It is a good idea to use such formalisms directly as special purpose languages for an interface with specialized software systems.

We introduce an example of one class of such a formalism widely used for language design and implementation. Special purpose languages based on such a formalism are used as interfaces in translator writing systems. The notion of a context-free grammar is a base for these. A context-free grammar is a set of rules of the form:

left part : right part

where left part is one nonterminal symbol called a syntactic category or name of a syntactic structure and the right part is a string of nonterminal and terminal symbols.

The way to derive sentences from one distinguished nonterminal symbol, start symbol, is defined.Sentences are composed from terminal symbols. The set of all sentences which can be derived from the start symbol is a formal language generated by the grammar.

To illustrate a notion of a context-free grammar we provide a simple example. In this example nonterminal symbols

are mnemonic names between symbols < and >. The terminal symbols are: 0,1,2,3,4,5,6,7,8,9, and /. The rules are:

```
<date>   : <number>/<number>/<number>
<number> : <digit>
<number> : <number><digit>
<digit>  : 1
<digit>  : 2
<digit>  : 3
      .
      .
      .
<digit>  : 9
<digit>  : 10
```

The start symbol of this grammar is <date>.

An example of a derivation using the above grammar is (the symbol => is used for one step in the derivation):

```
<date> => <number>/<number>/<number>
       => <digit>/<number>/<number>
       =>    1   /<number>/<number>
       =>    1   /<digit>/<number>
       =>    1   /   2   /<number>
       =>    1   /   2   /<number><digit>
       =>    1   /   2   /<digit><digit>
       =>    1   /   2   /   7<digit>
       =>    1   /   2   /   79
```

A language generated by this grammar is a set of sentences of the form number/number/number/, which can be read as dates.

Context-free grammars are often used for describing syntax structures of formal and natural languages. Moreover, they can be used as a base for special purpose languages used in translator writing systems.

The following example is the input of the YACC translator writing system (Johnson and Lesk 1978):

```
% token DIGIT
% %
date: number '/' number '/' number
        {date ($1, $3, $5);};
number: DIGIT
        {$$ = $1;};
number: number DIGIT
        {$$ = 10 * $1 + $2;};
```

The nonterminal symbols of this grammar are date and number. Terminal symbols are DIGIT and /. A program fragment is appended to each rule of the grammar. These program fragments compute the meaning or value of the nonterminal symbols. The variables $$, $1, $2, $3, etc. refer to the left side nonterminal of the rule, first, second, etc. symbol of the right side of the rule, respectively.

If this input text is processed by YACC parser generator, a program is generated which is able to read dates, convert them and store in the computer, provided that digits are read by another program returning the value of each digit.

2.6 Two-Dimensional Positional Languages

Two dimensional positional languages are languages in which input information contains indication of position in two-dimensional space. This space is displayed on a screen. One inputs the position information by means of a cursor controlled by a keyboard, throgh a joystick, or a mouse. Other ways include using a lightpen or finger touching of the screen.

Two-dimensional positional languages are useful in many applications. We may mention here some of the more important systems:

a. form filling systems,
b. screen oriented editors, and
c. two-dimensional query systems.

In form filling systems the user is provided by format map displayed on a screen and can then key data in free areas. The format map is protected and cannot be inadvertently altered from the keyboard. After filling in the form the user presses a special key and all input data are transmitted to the computer. Such techniques are very easy to use. As an example we take the following form:

```
NAME [                ]
FIRST NAME [          ]
BIRTHDATE
    DAY [    ] MONTH [            ] YEAR [    ]
PERSONNEL CODE [          ]
```

In such a form the user can put data between symbols [and] only.

The second type of two-dimensional positional system is the screen oriented editor. A screen oriented editor displays a portion of the file on the user's screen and allows him/her to make changes in the place where the cursor is. There are three principal groups of commands in a typical screen oriented editor action language (Pearson 1978):

a. cursor movement commands,
b. text movement commands, and
c. text modification commands.

In some cases cursor movement commands can be substituted by special keys on the keyboard (Altair word processing package 1977).

The last two-dimensional positional system to be mentioned here is two-dimensional query system. An example of such a system is the Query-by-example (Zloof 1976). This system is

used for user-interface with a data base. In order to query the data base the user fills in an example of a possible answer in a skeleton of the logical structure of the data base displayed on the screen.

An example of the skeleton structure for the data base is as follows:

PRESIDENTS	NAME	PARTY	HOME-STATE

where PRESIDENTS is the name of relation, and
NAME, PARTY,and HOME-STATE are the names of the domains.

The user can fill in such a skeleton of a relation by using the following two entities:

a. The "example element" (variable) which must be underlined and
b. the "constant element" which should not be underlined.

In addition, the function denoted ty "P." stands for "print," and the user inserts "P." before the data he/she wishes to output.

Here is an example of the user's query in the relation PRESIDENTS mentioned above:

PRESIDENTS	NAME	PARTY	HOME-STATE
	P. NIXON	DEMOCRAT	

The user needs only to fill in the skeleton with P. NIXON and DEMOCRAT. The answer of the system may be:

ITEM
KENNEDY
JOHNSON
CARTER

3. Main Features of Screen Languages

In this section we focus on the output side of the user-application software interface. The information produced by the computer to provide the user with feedback and other assistance we shall call the screen language.

The screen language is composed of several kinds of text. We can list the main parts of the screen language as follows:

a. formatting of the dialogue document;
b. assistance to the user during the specification of an input;
c. regular feedback to the user input;
d. error messages;
e. help facilities.

In this section we dicuss desirable features of the parts of the screen languages listed above.

3.1 Formatting of the Dialogue Document

First we introduce factors which contribute to a good design of dialogue document formats (Hebditch 1979):

-- Logical sequencing. The dialogue document contains different kinds of information. The sequence of the information should be as logical as possible. An example of bad sequencing is the blending of input and output text.
-- Distinguishing of input and output. It is very useful and improves legibility to distinguish inputs (action language phrases) and outputs (screen language phrases). The methods to do this depend on the type of terminal used. Methods which can be used include lower case-upper case, underlining, different colors, different densities, etc.
-- Spaciousness. The whole two-dimensional space of the dialog document can be used for output. Use of a table format improves legibility. If, for example, a menu is a part of an output, it could be presented in a table format.

3.2 Assistance to the User During the Specification of an Input

Assistance to the user may be of different kinds in various user-computer interface systems. If an answer language is used as an action language, the specification of the desired input is contained in the question produced by the computer.

In the case where an input language contains keywords, the system can assist during their specification.

Five forms of keyword recognition could be provided.

a. A whole keyword mode. The user must type the whole keyword.
b. An anticipatory mode. This mode requires the user to type just enough characters for the command to be uniquely specified. The system then automatically fills in the remainder of the keyword.
c. A fixed mode. Keywords in the system are picked in such a manner that the recognition of the keyword is possible on entry of a known number of characters.
d. A demand mode. This mode require a special character to initiate recognition after typing the initial part of the keyword.
e. A single-character mode. This mode allows high speed single-character recognition of the most commonly used keywords. This mode may be used only when unique first characters of keywords are defined.

Another method of system assistance is the presentation of noise words. When the system recognizes the first part of the user input phrase, it can generate some words, called noise words, so the user can distinguish between what he/she has entered and what is awaited by the system. For example in the input command

CREATE LINE from x1 to x2

the words from and to could be generated by the system as the noise words. The noise words aid the user to remember what to do next.

When the system assigns default values to some missing arguments, or supplies missing information on the basis of previous commands, it should inform the user about values assigned and ask him/her for confirmation.

3.3 Regular Feedback to the User Input

Regular feedback to the user input is the reaction of the system on valid requests. We can specify the following points concerning the contents of the system response:

a. Confirmation of input receipt. The system should confirm that the user input is valid and accepted. In cases where misunderstanding is possible, as for example with natural language interfaces, the system can output a question and ask the user for confirmation.
b. Information about the non-availability of resources. If a process, which is initialized, needs some resources like files, peripheral devices, etc., the user must be informed that resources desired are not available and why they are not.
c. Output data of the process. The output data can

either be presented on the user terminal or by means of some other output devices. In the latter case the user needs information about where and how to obtain his/her results.

d. End information. If the process created as a result of the user input finishes, information about the mode of termination (normal, failure of the system, error in input, etc.) will be useful.

3.4 Error Messages

In every piece of the input the system must anticipate errors. Therefore, sophisticated techniques must be used to handle user's errors. Three levels of error handling can be identified:

a. Error detection. The system must take great care to ensure the detection of any error.
b. Error recovery. After the detection of the error in an input text, it is desirable to continue the processing of the remainder of the input without "pseudo-error" indications.
c. Error correction. Some kinds of errors may be corrected automatically, but in any case of automatic correction the system must ask the user for agreement, because automatic error correction may introduce an unsolvable problems.

After error detection the system must report a message, which displays exactly and clear the character of the error.

Hebditch (1979) provides some guidelines on devising error reporting technique:

a. Avoiding coding error messages and the need to refer to manuals.
b. Making error messages as self-explanatory as possible.
c. Error messages should be pre-specified by the system designer and checked for understanding and effectiveness with the potential users.
d. Detection of errors must be as early as possible.
e. Avoiding the need to re-key valid inputs during the error correction process.
f. Rechecking everything after correction.

3.5 Help Facilities

Any software system must be properly documented in order to be usable (Cohen 1976). Documentation for a large system is not an easy task. It is still worse if the system is designed to be extended by means its users. Any printed documentation for such a system would be outdated before it is published. therefore, the system itself has to be capable of providing facilities that supply documentation in a way that is guaranteed to be up to date.

There are three main areas (Watson 1976) in which the user needs information to help him/her:

a. to know where he/she has been,
b. to know where he/she is, and
c. to know where he/she can go from here.

To keep the user informed in these three areas, it is desirable to offer information in the following spaces:

a. Information space. The user needs to know where he/she is in information space and which part of the many possible information is being displayed to him/her. The user arrived at his/her present position from previous points, and he may want to be able to return to previous points or views as well as to move on. It is possible to achieve this goal by organizing help facilities in a tree structure. Each information node in the tree contains a text document which explains a specific part of the system. The tree structure provides quick access to information about a specific topic.
b. Subsystem or tool space. In systems containing many tools and commands, the user needs to know which tools are active, which ones he/she used previously, and which ones he/she can enter from here. Each subsystem has a name and contains a number of related commands. It is very useful for all the tools to work on information in the same file structure in order to facilitate moving from one tool to another. The user need to be informed about subsystems available to him/her and about the subsystems in which he/she has previously been, as well as about the name of the current subsystem.
c. Input syntax space. Several types of assistance to the user in formulating input have been described above as a help to the user during the specification of an input. If the facilities are not sufficient beause of uncertainty about the basic concepts or the vocabulary, the user can enter the help facilities described above by specifying either a specific concept of interest or the general need of help only. In the latter case the system could utilize the information, which has been input up to this point in order to select information which the user needs.

In data base management systems it is useful to keep the user informed about the semantics of the data stored in the data base.

In the data base management system INGRES (Stonebraker, et al. 1976) information about relations is included and may be used in the same way as other help facilities specifying the names of the relations only.

4. Conclusion

Although we have discussed many issues concerning a user-application software interface, there are numerous aspects we have not mentioned. We have focused on such an interface only, in which alphanumeric texts are used as a medium for communication. The main problem in a communication with a computer using alphanumeric texts is a great difference between speed of an input and output. While the speed of an output can be very high (thousands of characters per second), the speed of the input via keyboard is very low (some characters per second). This drawback can be partly reduced by using single letters in an action language; for example, 'f' for FIND, 'p' for PRINT, and so on. But this is in conflict with the legibility of a dialog document and can be used for the frequent users only.

Another promising medium for interfaces is graphics. Graphics can be used as a part of a screen language or an action language or both as well. We have mentioned one type of systems, two-dimensional positional languages, in which a simple graphic is used as a part of a screen language. A communication in such systems is not only more simple but it is also faster.This fact could be seen from comparison of a line orieted editor with a screen oriented editor.

The second problem with a communication with an application software is the selection of an appropriate language type and its design. Computers and, especialy, small scale computers are used and/or will be used, among else, as a tool for office automation, for business systems, and for managerial systems. Most people, who use and/or will use such systems, have no or little data processing background. Therefore, it is desirable to design such software systems with a non-procedural interface. In such a situation the use of natural language seems to be most appropriate. But because problems to implement natural language interfaces even on small scale computers, we must suppose that formal languages will still be widely used in the future. In those cases it is very important to design the language in such a way that it follows a natural language as close as posible. Some remarks on the user language design of an office automation automation can be found in Rohlfs (1979). Keen and Hackathorn (1979) and Blanning (1979) describe ways to design languages for managerial systems.

Acknowledgements

I would like to thank Goeran Fick for many useful and helpful comments. Thanks also to Miyoko Yamada for her help in preparation of this paper.

REFERENCES

Addis, T.R. (1977) Machine understanding of natural language. International Journal of Man-Machine Studies 9:207-222.

Altair Word Processing Package (1977) Atlanta, Georgia:Altair Software Distribution Company.

Astrahan,M.M. et al.(1976)System R: Relational approach to database management. ACM Transactions on Database Systems 1(2):97-137.

Blanning, R.W. (1979) A language for describing decision support systems. Informal Workshop on Decision Support, Departement of Decision Sciences, The Wharton School, University of Pensylvania.

Chamberlin,D.D.(1976) Relational data-base management systems.Computing Surveys 8(1):43-63.

Codd, E.F. (1977) Seven steps to rendezvous with the casual user. Pages 179-199, Data Base Management, edited by J.W. Klimbie and K.L. Koffeman. Amsterdam: North-Holland Pub. Co.

Cohen, S. (1976) Speakeasy--A window into a computer. Pages 1039-1047, AFIPS Conference Proceedings.

Fick, G. (1979a) The challenge of low cost computers to the organization. Introduction to a Working Group. Second IFIP Conference on Human Choice and Computers, June 4-8,1979. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Fick, G. (1979b) The challenge of low cost computers to the organization. Report from Working Group No.13, Second IFIP Conference on Human Choice and Computers. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Fibber, M. (1979) Toward more "natural" interaction systems. International Journal of Man-Machine Studies 11:339-350.

Gaines, B.R. and P.V. Facey (1976) Programming interactive dialogues. Computing and People. Papers submitted to the conference at Leicester Polytechnic. Leicester:Edward Arnold.

Hebditch, D.L. (1973) Terminal languages for data base access. Data Base Management. Infotech State of the Art Report 15. Maidenhead, Berkshire, England:Infotech International Limited.

Hebditch, D.L. (1979) Design of dialogues for interactive commercial applications. Pages 973-992, Man/Computer Communication, Infotech State of the Art Report. Berkshire, England:Infotech International Limited.

Keen, P.G. and R.D. Hackathorn (1979) Decision support systems and personal computing. Working Paper 79-01-03. Department of Decision Sciences, The Wharton School, University of Pennsylvania.

Kowalski, R. (1979) Algorithm = Logic + Control. Communication of the ACM 22(7):424-436.

Lehmann, H. (1978) Interpretation of natural language in an information system. IBM Journal on Research and Development 22:560-572.

Man/Computer Communication (1979) Infotech state of the art report. Vol.1: Analysis and bibliography. Berkshire, England:Infotech International Limited.

McCracken, D.D. (1978) The changing face of applications programming. Datamation (November 15):25-30.

Miller, L.A. and J.C. Thomas (1977) Behavioral issues in the use of interactive systems. International Journal on Man-Machine Studies 9:509-536.

Newsted, P.R. and B.E. Wynne (1976) Argumenting man's judgment with interactive computer systems. International Journal of Man-Machine Studies 8:29-59.

Olle, T.W. (1973) A summary of the state of the art in data base management. Pages 215-233, Data Base Management, Infotech State of the Art Report. Maidenhead, Berkshire, England:Infotech Information Limited.

Pearson, M. (1978) A users' guide to "edx". Laxenburg, Austria:International Institute for Applied Systems Analysis.

Rohlfs,S.(1979) User interface requirements.Pages 165-199, Convergence:Computers, Communications and Office Automation, Infotech State of the Art Report.Maidenhead, Berkshire, England: Infotech International Limited.

Schneiderman, B. (1978) Improving the human factors aspect of data base interactions. ACM Transactions on Database Systems 3(4):417-439.

Sprague, R.H. (1979) A conceptual model of information technology support for managerial tasks. Seminar given in Management and Technology Area, International Institute for Applied Systems Analysis, July 1979, Laxenburg, Austria.

Stonebraker, M., E. Wong, P. Kreps and G. Held (1976) The design and implementation of INGRES. ACM Transactions on Database Systems 1(3):189-222.

Teitelman, W. (1979) A display oriented programmer's assistant. International Journal of Man-machine Studies 11:157-187.

Thompson, K. and D.M. Ritchie (1975) Unix programmer's manual.

Watson, R.W. (1976) User interface design issues for a large interactive systems. Pages 357-369, AFIPS Conference Proceedings.

Winograd, T. (1979) Beyond Programming Languages. Communications of the ACM 22(7):391-407.

Zloof, M.M. (1976) Query-by-example--Operations on hierarchical data basis. Pages 845-853, AFIPS Conference Proceedings.