MANAGEMENT SCIENCE
Vol. 14, No. 2, October, 1967
Printed in U.S.A.

Information Systems in Management Science

Languages are important in any discussion of information systems. It is through the medium of a machine-oriented (as opposed to natural) language that the user must communicate with the computer. In order to use the earliest computers, it was necessary to describe the problem in the detailed computer hardware language. Then higher level languages were evolved so that instead of describing a solution in machine language, it was possible to describe it in a more "natural" language. These languages (FORTRAN, COBOL, etc.) are general in the sense that it is possible to solve almost any problem that is solvable in machine language using them.

Now we have a group of languages in which it is possible to solve only the specific problem that each was intended to solve. The following letter describes such a language and its use in a simulated management environment.

Dear Sir:

The Syracuse University Research Corporation during the past three years has been developing a computer program (SMOLDS) that will allow the lay manager to personally operate computer equipment for data retrieval, numerical computation, and data and numerical manipulation, actually producing hard copy for use back at his desk. This use possibility for the manager, his assistants, and other clerical help could make the computer an integral on-line tool for everyday use. (SMOLDS is the acronym for Syracuse Managerial On-Line Data System. This project was financed under a contract with the Rome Air Development Center, Griffiss Air Force Base, Rome, New York.)

The following paragraphs are a narrative of my first attempt to use the SMOLDS program (and, in fact, any computer program). No attempt has

been made to over-generalize in the narrative. All the small, bothersome problems that computer technicians must solve before such a program becomes acceptably operative are reflected in the account, because to the manager they become major annoyances.

The first thing that had to be done was to decide what information could best be used in an on-line system. I decided to include two types of information: contract cost analysis and personnel. The contract cost analysis information seemed particularly well suited because it is organized into discrete classifications, which are few in number and are historical, on a monthly basis, from the time a contract begins until the contract ends. The information is used frequently to figure the rate at which moneys are being spent, on what cost classifications moneys are being expended, how long the contract will last at present rates of expenditures, etc. These computations could be made very rapidly on the computer because all contracts could be stored at the same time.

Although no computation is performed on the personnel data, I felt that it would be interesting to use for information retrieval. This information consisted of ten significant parameters for each individual, including, among other items, job title, name, and college degrees. The personnel data proved interesting, not only for information retrieval, but because it could be related to and manipulated with particular items of the cost-analysis data. In other words, the two data sets could be linked.

After receiving a brief introduction to the computer, my first task was to learn how to communicate with the machine using the SMOLDS language. During the learning process I was, for the first three days, in a kind of vague relationship to the whole program. By following directions literally, it appeared as though I was learning quite rapidly, and I suppose I was, except that I didn't quite know it. The thing that helped the most to make me feel comfortable was having someone explain to me exactly what was happening in the machine.

Although the SMOLDS language itself is quite simple and can be learned, I would say that it is necessary for a manager using SMOLDS to have an understanding of just what each command is causing the machine to do. To me, this is equivalent to knowing how a clerical person will react and what thoughts he has when a particular request is made. By knowing this, the manager maintains control of the work being performed because he knows what to expect. The analogy breaks down, of course, because the clerical person is more flexible than the computer, and will modify and interpret commands, whereas the computer will do what it is instructed to do, for better or worse.

My modus operandi was to work for about one hour a day; this practice consisted of (a) having in mind the information I wanted to derive from the data base, (b) outlining the commands that were necessary to obtain this information, and (c) actually addressing myself to the typewriter and issuing the commands. For about three months this was the pattern of my work schedule. However, several intervening disturbing factors made this schedule less than perfect; for instance:

(1) demands on my time and attention from other duties sometimes prevented me from the daily stint
(2) many times during those three months the machine was inoperable, due to typewriter or other computer equipment failure, with the attendant having to open the machine in order for IBM technicians to make adjustments
(3) on two or three occasions my scheduled time was pre-empted by other programs.

On several occasions my experience during my practice time was made very ineffective by the faulty operation of equipment, especially the typewriter. By locking itself in the upper case position, it would often cause the machine to hang up. This would mean that the entire program had to be reloaded from the disk, an operation which required about seven minutes.

On a few occasions the internal operation of the computer program itself was faulty, causing an entire sequence of commands to be cancelled out. This required a correction of the machine programming.

Mistakes which I made in giving the commands were sometimes correctable by certain key operations on the typewriter. Other commands in which there were mistakes, if not caught by me before actuating the machine, caused the computer hang up, sometimes resulting in the loss of the entire program. The computer specialist had programmed the machine to inhibit many of these mistakes and the only loss in such cases was the immediate preceding command; however, many of my invalid command sequences had not been anticipated. In these cases, the machine would hang up and the entire program was lost. This again necessitates reloading the program into memory.

The commands themselves are easy to learn. The command word symbols used are easily relatable to the actual function that is being performed so that there is a built-in memory aid. I had more trouble remembering to label information so that it could be called out for use in many subsequent operations. The part of the command that involves identifying the information on which one wants an operation performed is really the most difficult part of the learning process. This is the part of the command involving the "real business" which is on the manager's mind. The manager must know the content of the answer he wants, and must know how the machine will get it for him.

Because the machine is programmed to handle information in the logical sequence determined by the programmer, it is necessary to know the command capability by heart; otherwise, the wrong computation may be performed on any given command. For instance, when performing arithmetic functions of subtract and divide, the operator must know that the key works are "take away" and "by" in order for the operands to then be typed in the proper sequence. To answer the question: "How long will contract monies last at current rate of expenditure?" involves dividing *current rate* into *amount open*. This answer is obtained through the following commands:

	COMMAND	LABEL	FORM	BLOCK	RELATIONSHIP
I	KNOWN	CONT	1 /	1 /	NE / - - - /
II	FETCH	TOTAL	CONT/	46 /	
III	FETCH	OPEN	CONT/	50 /	
IV	DIVIDE	ANSWER	OPEN/TOTAL/		
V	PRINT	ANSWER			

The crucial point is in the DIVIDE command. The machine is programmed to divide the first label *by* the second label or, schematically, this *by* this, not, this *into* this. Similarly, if the result wanted was the open amount less the current rate, the command is:

SUBTRACT LABEL OPEN / TOTAL /

The schematic command is, *take away* this, not, this *from* this. To state this point again, the operator must know the program vernacular and computational lingo cold in order to have a satisfying and successful conversation with the computer.

Another learning problem was that of knowing the programming well enough so that as desired output was being developed, you could give the proper commands to the machine without having to have constructed a set series or sequence of commands ahead of time. Having this kind of insight gave me the flexibility of arriving at an answer by using different series of commands. While it is true that this type of free manipulation is limited, I often got new ideas while working with figures or information and did have some flexibility while using the computer. This enabled me to divert from a given series of commands in order to get the newly wanted piece of information and then be able to pick up the original series of commands without starting from the beginning again.

An added fillip to my learning was the capability of inserting information into the machine not previously a part of the data base. This is done by means of a programmed command entitled STORE. This stored information can be in either alphabetic or numeric form. For instance, one of the practice sessions was spent in manipulating the personnel information in various ways: the question was asked, "What is the average age of professional people in our division?" Even though birth dates were not in the computer, it was possible to enter them, compute ages, and average them.

It was an exhilarating experience to get this answer by inserting information; the horizons for use of the computer seem to extend indefinitely. Inserted information can also be manipulated with data extracted from the existing data. This feature is particularly exciting, because the operator can use the equipment without being completely dependent on the existing data. In other words, he can *conjecture* and *probe* fairly widely, within the language limits of the computer. He can think creatively, and seek information at will.

In retrospect, my time on the managerial on-line data program has been both

fascinating and valuable. My interest did go through periods of waxing and waning. Initially, the speed of fact acquisition and manipulation was a great stimulation to learning the on-line skills. My expectations were satisfied. There were, however, flaws in the possibilities. As my interest in current continuous use of the data rose, so did the realization that the supporting information flow was inadequate. I felt like General Patton on reaching the Rhine—I had no gasoline left. Updating the data was a month behind. So there was a waning in my enthusiasm—instantaneous access to stale information is of no help to a manager. After mastering the language and becoming somewhat *a virtuoso* (people no longer laughed when I sat down to play) this lack of current information sparked a waning period in my interest.

However, continued practice with the system seems to buoy up my interest and enthusiasm. Projecting my experience into a conjectural future where supporting information flow is commensarate with the data demands for current information fires my enthusiasm to a higher degree than at the start. Now I know that as a manager I will be able to approach the computer and deal with either a specific problem or conjecture with various possibilities *in a matter of minutes.*

David M. LaMar
Administrative Officer
Syracuse University Research Corporation

Although I feel that most managers will not use a typewriter (see my last two columns), Mr. LaMar's letter is noteworthy because of his description of the SMOLDS language, and because of the description that he gives of learning about computers. In general, I find that people who have had no experience with computers have a somewhat mystical feeling about them. I feel that many people are shocked to find that a question must be phrased in great detail and the form of the answer known before it is possible to get an answer.

Mr. LaMar spent a great deal of energy learning the language. Most managers are either unwilling or unable to give up large amounts of time learning to program. It is possible to cut the capability of the system to the point where all of the communication is done by pressing labeled buttons (as in the brokerage system). If an objective of the system is to keep language flexibility and yet make the system simple to use, then the computer itself must lead the user by showing him alternatives.

The SMOLDS language is an example of a "Problem-Oriented Language" (POL). Most computer languages (FORTRAN, ALGOL, COBOL, AUTOCODER, etc.) are structured in such a way that the program controls the data, moves and manipulates it. For example, a Linear Programming Code (i.e., a program that solves L.P. problems) in FORTRAN would search through the column vectors for the proper one to introduce next. In doing this, the program manipulates the data, keeping complete control over it.

In contrast to most other computer languages, the POL passes a set of instructions across a set of data; the data—rather than the program—is of paramount

importance. In fact, in a POL, the program is of so little importance that it is often used once, and then discarded. Statements in the language take the form:

Instruction, Data List

The "instruction" describes the operation to be performed upon the data, and the "data list" specifies the data to be used. The language performs a series of calculations upon the data with the data the only link between statements.

The first of these languages to be widely used is COGO, designed and written for an IBM 1620 by Professor Charles Miller, chairman of the MIT Civil Engineering Department. The data set used by a COGO Language Program is a table containing the coordinates of points. To enter points numbered 1, 2, 3, and 4 with coordinates of (13.5, 18.1), (12.2, 17.1), (13.6, 18.2), (17.6, 8.7), the language statement would be

KNOWN 4, 1, 13.5, 18.1, 2, 12.2, 17.1, 3, 13.6, 18.2, 4, 17.6, 8.7

KNOWN is the instruction, and the data list has the number of points to be entered (4), followed by the identification of the first point (1), its coordinates (13.5, 18.1), the identification of the second point (2), etc. Having entered points, it is possible to create new points.

POINTS 5, 1, 3, 2, 4

is a statement which finds the point at the intersection of the lines defined by points 1 and 3 and by points 2 and 4 and enters its coordinates in the table, identifying it as 5.

It is possible to use the information in the table. In order to find the area enclosed by the five points, 1, 2, 3, 4, 5, a command would be given:

AREA 5, 1, 2, 3, 4, 5

where the first five (5) is the number of vertices in the figure, and the 1, 2, 3, 4, 5 are the location of the vertices (the figure defined by 1, 4, 2, 3, 5 would have a different area). For a complete discussion of the language, see the IBM 1620 COGO Manual, H20-0143, available from any IBM office.

COGO was designed to be used from a typewriter by an engineer with a sketch who would ask questions of the computer to help him complete that sketch. Because of this, the program automatically prints the results of any calculation made. When the engineer finally asks enough questions, his sketch is complete, and he leaves the typewriter.

A POL is an excellent tool where there is a specific problem to be solved, where the description of the problem leads directly to the solution (such as in COGO, where the description of the geometry leads to the solution of the problem), and where the problem solver is willing to describe the solution to the problem in detail. In the case of COGO, the engineer would have had to do the calculations by hand, and so is given a time saving tool by having a computer and language at his disposal. It would be possible for the engineer to solve any problem that is solvable in COGO in FORTRAN, but it would take a great deal more time

for the engineer to learn FORTRAN, to program and to debug the program, making it more economical to do it by hand than to use the computer.

Some of the characteristics of a problem that should be taken into consideration when deciding whether it is amenable to solution by using a POL are:

1. there must be a well-defined set of data that is made available to an operator;
2. in order to get answers through the medium of a POL, it is necessary to be able to ask questions as a single statement (such as in the instruction AREA), or as changes to the data set and then the question (such as the instruction POINTS, followed by the instruction AREA);
3. the operator must be willing and able to break the question into its parts, and ask it in detail, using a typewriter.

Where the POL is applicable, it is a capable tool because it is easy to learn and use, is extremely powerful, and lends itself to a man/machine communication environment. Although the POL can be expanded to handle general language problems (by adding statement names, logical statements, and indexing), I have seen no attempts to do so that have been completely successful, and I would suggest using it only where it obviously fits.

There is at least one computer program that allows the user to specify his own language, treating the description of a problem-oriented language as data. The program (PLAN) is included as part of the IBM 1130 Data Presentation System (1130-CX-14X).

The choice of language is extremely important where non-programmers will be expected to use the computer. Although Mr. LaMar was willing to spend the time to learn a detailed language, his situation was not typical because he was part of an experiment in a university environment. Generally, you can expect a nonprogrammer to learn only as much as will benefit him, or he is forced to by the nature of his job. It is often unnecessary to have a non-programmer learn a general-purpose language, because problem-specific languages exist for many problem/machine combinations.

For example, there are innumerable languages which allow the analogue computer expert to program a digital computer using the symbols that he is familiar with. There are languages that handle the network, server, queue problems. The proliferation of languages is almost unbelievable, and the use of these languages can add to the value of the management science effort.

The way the operator communicates with the computer is crucial to the use of the computer where the operator is not a programmer. I am actively soliciting contributions about man/machine communications, as well as other areas relating to information systems. Please send contributions to:

Mr. Harry Stern
General Foods Corporation, NG-3
250 North Street
White Plains, New York 10602