A Message System for Interactive Dialog

G. C. PATTON

Bell Telephone Laboratories, Incorporated, Whippany, New Jersey

I. Introduction

When writing programs for interactive computer systems, programmers often find that it is difficult to develop the interactive dialog these programs require. The difficulty arises for two reasons. First, because interactive terminals are relatively new, programmers are not accustomed to planning and developing computer dialogs. Second, there is often no easy way to program the dialog once it has been developed. The solution to the first problem is heavily dependent on the programmer, his training and his background, and does not lend itself to a computerized solution. The solution to the second problem, however, can be eased by providing a higher-level, dialog construction language. The basic features for such a language appear in several systems used for computer-aided instruction [1–4]. The Message System builds on these features to provide a dialog construction language for general use.

II. Using the System

The Message System is used where interactive dialog is required. In a typical FORTRAN program this is accomplished by writing a CALL to a subroutine named MESSAG (Fig. 1) each time interactive dialog is required. After the program is completely written, the programmer constructs the dialog associated with the calls to MESSAG using a program called MFILE (Fig. 2). MFILE is an interactive, self-teaching program which collects information from the programmer about the dialog and packs it into a message file on disk or tape. The message file can later be corrected or changed, using the edit features of MFILE.

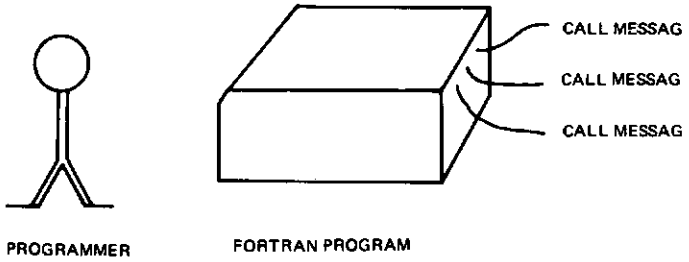

FIG. 1. Writing a program.

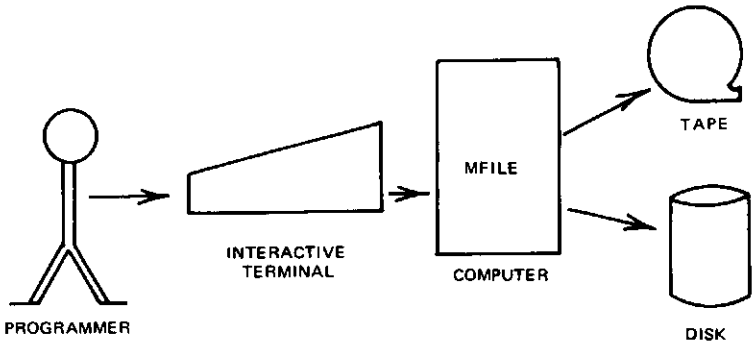

FIG. 2. Constructing a message file.

After the program and its dialog have been written, the program is ready for interactive use. The interaction is controlled by the routine MESSAG, which is a general program that asks questions and receives and evaluates answers. MESSAG is "driven" by information previously placed in a message file by MFILE (Fig. 3).

Although a programmer converses with MFILE using a higher-level, dialog construction language, the concept of the Message System is quite

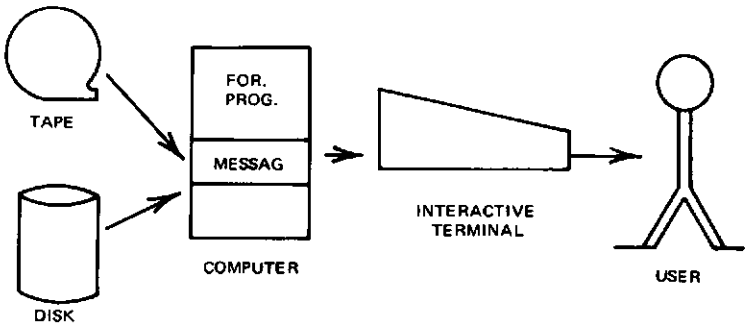

FIG. 3. Using a message file.

simple. MFILE is not a compiler and does not assemble machine instructions. MFILE is a collector, gathering information about a program's dialog; a translator, translating this data into standard units called messages which are placed in a message file; and a teacher, providing the programmer with self-teaching sequences when he requests assistance or when errors are detected.

MESSAG is also conceptually simple. It is merely a routine to associate a given message file with a specific program. MESSAG is called during execution of a program whenever dialog is required. When called, it searches a specified message file for the requested message and interprets the commands in the message. The sequence in which the commands appear governs the communication with the user of the program.

III. Description of a Message

The message files constructed by MFILE and used by MESSAG, are composed of messages, the smallest set of information needed for one interaction. A message may contain four parts as listed in Table I. These parts are combined to form each message.

TABLE I

PARTS OF A MESSAGE

Part no.	*Part name*	*Purpose*	*Optional*
1	Identification	Contains a number used for reference and an indication of the type of message	No
2	Text	Information presented to the user	Yes
3	Choices	Possible replies from the user	Yes
4	Branches	Directions telling the program what to do next	At least one necessary

A typical message would thus contain identification such as

```
Message Typed/1/[1]
```

where "Message Typed" specifies that the text of the message is to be presented on a typewriter or a visual terminal, and "1" is the number assigned to the message. For convenience, system commands (message codes) may be replaced by their abbreviations. For example, the abbreviation for "Message Typed" is "MT." Such abbreviations will be used throughout the remainder of this paper.

[1] Slashes will be used to separate the parts of a message in the example.

The text of the message follows the identification as shown below.

```
MT/1/On-line math library. Do you want instructions./
```

The text portion is the part of the message seen by the user. The text is normally followed by a series of possible replies (choices) which a user might be expected to type after seeing the text. Associated with each choice is a branch which gives the program action to be taken when a user types a specific choice. A typical series of choices and branches is shown below.

```
MT/1/On-line math library. Do you want instructions.
/CS/yes/BM/2/CS/no/BM/3/BM/4/
```

In this example the abbreviation CS stands for choice and BM is the abbreviation for branch to message. Thus the series could be interpreted: "If the user

```
LOAD LIMITS 14202 16677

THIS IS MFILE-01. WOULD YOU LIKE
INSTRUCTIONS. TYPE YES OR NO, A SLASH,
AND THE KEY MARKED RETURN.
:=NO/(CR)

WHICH CONSOLE ARE YOU USING

     1. TELETYPE
     2. SANDERS
     3. IBM
     4. OTHER
:=TELETYPE/(CR)

PROVIDE INDEX RECORD NUMBER.
:=1000/(CR)

IS THIS AN INITIAL RUN.
:=YES/(CR)

MESSAGE TYPED OR END OF FILE:=MESSAGE TYPED/(CR)

MESSAGE NUMBER:=1/(CR)

TYPE TEXT
:=ON-LINE MATH LIBRARY.  DO YOU WANT INSTRUCTIONS./(CR)

FORMAT, BRANCH, RETURN, DELETE, READ, OR CHOICE
:=CHOICE/(CR)

TYPE CHOICE
:=YES/(CR)

BRANCH, RETURN, READ
:=BRANCH/(CR)

YOU HAVE ERRED AFTER SUPPLYING A CHOICE.  DO YOU NEED HELP.
:=YES, I GUESS SO/(CR)
```

FIG. 4. Interacting with MFILE.

replies yes, execute Message 2 next; if he replies no, execute Message 3; if he replies neither yes nor no, execute Message 4."

IV. Interaction with MFILE

A programmer constructs messages for his application program through interaction with a System program called MFILE. Using a series of questions and answers about message parts, MFILE helps the programmer build his dialog. This information is then put into a message file on tape or disk. A typical interaction between a programmer constructing dialog and MFILE is shown in Fig. 4. The programmer's responses are underlined in this and subsequent figures and a carriage return is indicated by the symbol CR (circled in the figures).

```
POSSIBLE REPLIES ARE:

     1. BRANCH CODES, BM, BS, AND BD
     2. RETURN CODES, RT AND RC
     3. READ CODES RA AND RN.

WHICH OF THESE CODES WOULD YOU LIKE TO KNOW MORE ABOUT.
:=NONE/(CR)

BRANCH, RETURN, READ
:=BRANCH MESSAGE/(CR)

TYPE MESSAGE NUMBER FOR BRANCH:=2/(CR)

BRANCH, RETURN, READ, CHOICE, DELETE INPUT, MESS. TYPED,
OR END OF FILE
:=CHOICE/(CR)

TYPE CHOICE
:=NO/(CR)

BRANCH, RETURN, READ
:=BRANCH MESSAGE/(CR)

TYPE MESSAGE NUMBER FOR BRANCH:=4/(CR)

BRANCH, RETURN, READ, CHOICE, DELETE INPUT, MESS. TYPED,
OR END OF FILE
:=BM/(CR)

TYPE MESSAGE NUMBER FOR BRANCH:=3/(CR)

MESSAGE TYPED OR END OF FILE:=MI/(CR)

.

.

.
```

FIG. 4. (continued)

The first four questions asked by MFILE in this example are used to define System parameters. For example, the index record number supplied in answer to the third question is used to identify the message file to be constructed or modified. The answers to the fifth and following questions supply MFILE with the message parts described in Table I. Note that halfway through the example the programmer has made an error by not specifying which type of branch he wants (there are several types of branches and the question is supplied as a reminder rather than as a complete list of possible replies). The System discovers the error and is ready to help to the extent desired by the programmer. In the example shown, the programmer realizes his error immediately upon seeing the possible replies which he could have made (in their abbreviated form). Thus, he returns quickly to the normal series of questions without using the help sequence to its fullest. Two of the last three replies show how the MFILE program accommodates the more experienced programmer, allowing him to reply with the abbreviations BM and MT. Actually the programmer could have abbreviated all the system commands if he had desired.

Figure 5 shows MFILE used by an experienced programmer who is able to anticipate the order in which MFILE will ask questions. He can thus take

```
LOAD LIMITS 14202 16677

THIS IS MFILE-D1. WOULD YOU LIKE
INSTRUCTIONS. TYPE YES OR NO, A SLASH,
AND THE KEY MARKED RETURN.
:=NO/TELETYPE/1000/YES/(CR)

MESSAGE TYPED OR END OF FILE:=MT/1/ON LINE MATH(CR)
:=LIBRARY.  DO YOU WANT INSTRUCTIONS./CS/YES/BM/2/CS(CR)
:=NO/BM/4/BM/3/(CR)

MESSAGE TYPED OR END OF FILE:=MT/3/INVALID REPLY.  TYPE YES(CR)
:=OR NO./S/1//RC/1/(CR)

MESSAGE TYPED OR END OF FILE:=MT/4/CHOOSE A PROGRAM:(CR)
:=/S/1/     1. ROOTS OF POLYNOMIALS/S/0/     2. CURVE(CR)
:=FITTING/S/1//CS/1/BM/5/CS/ROOTS/BM/5  /CS/2/BM/20(CR)
:=/CS/CURVE/BM/20/CS/HELP/BD/6/CS/NONE/BM/7/BD/8/(CR)

MESSAGE TYPED OR END OF FILE:=EF/(CR)
```

FIG. 5. Expert's interaction with MFILE.

advantage of the feature of MFILE which allows him to answer a series of anticipated questions by typing a series of replies on a single line separated by slashes. In this case the slashes serve to delimit the answers to the questions asked by MFILE, and the carriage return serves only to transmit the line of information to the computer.

In instances where multiple answers are specified on a single line, MFILE does not print the questions which have been answered. The next unanswered

question will be printed only if the line ends with a slash and a carriage return. The absence of a slash before the carriage return suppresses the printing of the next question.

The abbreviated message codes appearing in the example are described in Table II. These codes are essentially self-explanatory. The additional explanation provided in the following paragraph, however, gives an indication of the flexibility of the dialog.

TABLE II

DESCRIPTION OF MESSAGE CODES

Code	*Description*
MT	Message to be typed
CS	Choice or possible reply
BS/X	Branch to message X and save reply made
BM/X	Branch to message X
BD/X	Branch down to message X (the program remembers the current message so that it can be returned to later)
S/X	Skip X lines
RC/X	Return to choices of message X
RT	Return to message containing the last BD code or to the main program
RA	Read characters up to the next break character
RN	Read numbers (separated by blanks) up to the next break character
EF	End this message file

During execution of an application program, messages are printed when the MESSAG subroutine is called (MT). After a question is printed, a reply may be requested (CS or RC, where RC specifies the choices in a previous message). The replies are then evaluated in terms of the information in the message file. Thus, a reply might cause the printing of text from a new message (BM), or an immediate return to the main program (RT). When transfers are made between messages without returning to the calling program, one or more replies may be saved (BS, RA, RN) in a buffer for later evaluation by the calling program when it regains control.

V. Calling Sequence to MESSAG

Once a message file has been constructed it is used by subroutine MESSAG. MESSAG is called by the FORTRAN program which will interact with a user. The call sequence to MESSAG is shown below.

```
CALLMESSAG(MESGNO,NDATA,IW,NOC)
```

The parameters of this call sequence are defined in Table III.

TABLE III

PARAMETERS OF MESSAG CALL

Parameter	*Description*
MESGNO	The message number of the first message to be printed (printing of subsequent messages is dependent on a user's replies)
NDATA	A block of data in which the first word contains the number of the last message to be printed before the return to the calling program. The remaining words of the block hold any user replies which must be returned to the calling program
IW	The line width of the text to be printed
NOC	The number of the choice which matched the user's reply to the last question printed

Two arguments in this sequence, MESGNO and IW, are used to transfer information into the subroutine. The remaining two arguments serve to pass information back to the calling program about the interaction. This information is in a form suitable for testing to determine the further course of the program (see Patton [5] for the exact format). For example, the number of the reply to the last message printed (NOC) could be used as the index in a computed GOTO statement or could be tested in a series of IF statements.

VI. Interaction with MESSAG

To a person interacting with the Message System via subroutine MESSAG, the construction of the message files and the calls to the subroutine are not apparent. In fact, if the dialog is adequately constructed by the programmer, a person feels that he is talking to a computer which understands him, since questions are asked in a natural language and he replies to the questions in the most natural way. A typical interaction using the message file developed by the programmer via the interaction shown in Fig. 5 is shown in Fig. 6.

In this example the user interacts with the computer in a natural language. When the need for help is indicated by either a direct request from a user or by the System's discovery of a wrong or invalid reply, a sophisticated help sequence, which the programmer planned, can be initiated.

When the same program is used repeatedly by one person, this approach (requiring detailed dialog) becomes inadequate since the user has to wait while the lengthy dialog is printed. To remedy this situation the Message System allows abbreviated dialog to be substituted easily in place of the

```
ON-LINE MATH LIBRARY, DO YOU WANT INSTRUCTIONS.
:=NO/(CR)

CHOOSE A PROGRAM:

     1. ROOTS OF POLYNOMIALS
     2. CURVE FITTING

:=ROOTS OF POLYNOMIALS/(CR)

DEGREE OF POLYNOMIAL:=4/(CR)

TYPE COEFFICIENTS, LEAVE SPACES BETWEEN THEM
:=HELP/(CR)

TYPE COEFFICIENTS STARTING WITH COEFFICIENT FOR
HIGHEST POWER OF THE VARIABLE.  CONTINUE ON
SUCCEEDING LINES IF NECESSARY.  NEED MORE HELP.
:=NO/(CR)

TYPE COEFFICIENTS.
:=
```

FIG. 6. Using a message file.

original lengthy text. This is accomplished by constructing a second message file with abbreviated messages and allowing the user to request this file in place of the original one at run time. For example, a math assistant repeatedly using the program to find the roots of a number of polynomials may request that the program use a message file containing an abbreviated dialog. The result of his interaction with the abbreviated messages is shown in Fig. 7. The answer to the first question indicates that abbreviated messages are to be used. Note that the extended help sequence is still available when it is needed.

Now consider the same math assistant finding the roots of his one hundredth polynomial. After using the root finding program 99 times he does not need even the abbreviated dialog for he has, by repeated use, memorized the sequence of questions. In such cases, the Message System allows him to anticipate the questions and put multiple answers on a single line if they are

```
ON-LINE MATH LIBRARY, DO YOU WANT INSTRUCTIONS.
:=SHORT/(CR)

PROGRAM ROP, CF.:=ROP/(CR)

DEG.:=4/(CR)

COEF.:=HELP/(CR)

TYPE COEFFICIENTS STARTING WITH COEFFICIENT FOR
HIGHEST POWER OF THE VARIABLE.  CONTINUE ON
SUCCEEDING LINES IF NECESSARY.  NEED MORE HELP.
:=NO/(CR)

COEF.:=
```

FIG. 7. Using abbreviated dialog.

separated by slashes. The resulting dialog is shown in Fig. 8. Note that the programmer has had to do no extra programming to add the multiple answer capability, since the system always looks for multiple answers and suppresses the already answered questions. Thus, when specifying answers in this fashion, the math assistant has the dialog and help facilities of the System available.

```
ON-LINE MATH LIBRARY. DO YOU WANT INSTRUCTIONS.
:=SHORT/ROP/4/(CR)

COEF.:=HELP/(CR)

TYPE COEFFICIENTS STARTING WITH COEFFICIENT FOR
HIGHEST POWER OF . . . . .
```

FIG. 8. Interaction anticipating questions.

For example, if he ends a line with a slash, the next message to be answered is printed to remind him of the next reply. If the line ends without a slash, the next question is not presented and he may type the next reply or series of replies. This flexibility in interacting with a message file through MESSAG is identical to the flexibility offered to the programmer who originally made up the message file while interacting with MFILE.

VII. Special Message System Features

The techniques used in the Message System apply equally well to languages other than English. Thus, the same FORTRAN program could be used in many languages by changing only its message file. In fact, the first question asked by a program could determine the language of interaction, and the appropriate message file could be selected at run time. Figure 9 illustrates this feature by showing the previous program using a message file translated into German.

In a similar manner a message file could be constructed in which the text was replaced by instructions for running an audio output unit (preceded of course by an MA [message audio] command in place of an MT command). At run time a decision could be made as to the type of terminal, teletypewriter or audio, and by using the proper message file, the same program could interact with users at either console.

Another feature of the Message System is the ability to change dynamically the width of a page without rewriting the dialog. For example, a change in the line width from the 72 characters of a teletypewriter to the 40 characters of a visual display, requires a change of only one parameter in the MESSAG calling sequence.

A detailed description of other features, system programs, and the format of the message files is given by Patton [5].

```
ON-LINE MATHEMATIK-PROGRAMM-BIBLIOTHEK.  WOLLEN SIE
INSTRUKTIONEN.
:=NEIN/ (CR)

NENNEN SIE EINES VON DIESEN PROGRAMMEN:

     1. WURZELN VON POLYNOMEN
     2. KURVENANPASSUNG

:=WURZELN VON POLYNOMEN/(CR)

GRAD DES POLYNOMS:=4/(CR)

NENNEN SIE DIE KOEFFIZIENTEN, GETRENNT DURCH
ZWISCHENRAUM.
:=HILFE/(CR)

NENNEN SIE DIE KOEFFIZIENTEN IN DER REIHENFOLGE
ABSTEIGENDER POTENZEN DER VARIABLEN.  BRAUCHEN
SIE MEHR HILFE.
:=NEIN/(CR)

NENNEN SIE DIE KOEFFIZIENTEN.
:=
```

FIG. 9. Interaction using German message file.

VIII. Evaluation of the Present System

In over a year's experience at the Bell Telephone Laboratories, the most important feature of the System was found to be the ease with which dialog could be constructed. This allowed the development of dialog which would otherwise not have been attempted, and therefore, of more meaningful programs which could be used by people with little computer experience and little knowledge of the programs they were using.

Another important feature was that a user could talk to a computer in a natural language. This, along with the System's ability to recognize errors and provide help sequences, allowed people who had never used a computer before to interact with one without any previous instruction.

Because most business and scientific programs are used over and over again by the same people, the feature which allows input of multiple answers on one line, when the sequence of questions is known, was also found to be very useful. Once the main series of questions was mastered, a user could eliminate the long waits while lengthy messages were typed out, and even more important, not worry about the time-sharing delays between each interaction. These delays, 7 seconds to print an average message, and 10 seconds minimum between each interaction, do not appear large at first. The seconds quickly add up, however, even for short programs, and soon become a frustrating,

psychological bottleneck. This frustration is especially serious when the user knows in advance the series of questions which is going to appear.

The ability to store the dialog on secondary tape or disk storage was another feature which proved quite useful, especially during the System's early development which took place on a small time-sharing system. This secondary storage feature aided development of the extensive dialog needed for adequate interaction, since the only core required for interaction was the 2000 locations containing the MESSAG subroutine.

At first 2000 locations seem excessive, but when compared to the storage required to hold only the dialog (not the reply evaluation logic) it is quite realistic. For example, a small-to-average program requires from 50 to 100 messages, and the average message contains two lines of text, each about 72 characters long. Thus, a minimum storage of 7200 characters is needed just to store the text. On a small machine with 3 characters in a computer word, 2400 words of core would be required to hold the smallest set of messages.

IX. System Improvements and Uses

Although the present version of the Message System has been quite adequate for our purposes, there are several improvements which could be made. For example, in the future we plan to implement system commands which will result in greater flexibility when specifying possible replies, more flexible output of data within messages, and more sophisticated formatting of messages, especially for visual display equipment.

The expected uses of the System are numerous. In addition to its present use with business, scientific, and military programs, it can be used alone as a teaching machine. After the anticipated system commands for formatting and line-by-line editing of message texts are added, the System could be used for composing, context editing, and other publication tasks.

Since the System is written completely in FORTRAN, it can be used with only slight changes on a wide range of computers. To date it has been used on computers having both 3 and 6 characters per word, and work is presently going on to adapt it to work with 9-bit instead of 6-bit characters.

The System is both flexible and easy to use. It is easy to construct messages and to interact with the messages once they have been constructed. The System is flexible in the construction of messages, terminals on which the messages are displayed, and the machines on which the System can be used. Thus, it provides a solution to the majority of problems associated with interaction dialog.

REFERENCES

1. FEINGOLD, S. L., and FRYE, C. H., User's Guide to PLANIT—Programming Language for Interactive Teaching, System Development Corp., Santa Monica, California, 1966.
2. ANONYMOUS, IBM 1401, 1440, or 1460 Operating System Computer Assisted Instruction, Form C24-3253-1, IBM, Endicott, New York, 1965.
3. SILVERN, G. M., and SILVERN, L. C., Computer-Assisted Instruction: Specification of Attributes for CAI Programs and Programmers, *Proc. of the 21st Natl. Conf., ACM, 1966*, pp. 57–62.5. Thompson Books, Washington, D.C., 1966.
4. UHR, L., The Compilation of Natural Language Text Into Teaching Machine Programs, *Proc. Fall Joint Comput. Conf., 1964* **26**, 35–44. Spartan Books, Washington, D.C., 1964.
5. PATTON, G. C., The Message System, Master's Thesis, Newark College of Engineering, Newark, New Jersey, 1967.