Here we have considered a system ABC (a fictional name). System ABC is a combined hardware and software system that is used by administration personnel of the digital switching system. The actual architecture of system ABC is complex, but for four purposes we can treat it as a black box containing a computer. The user connects to this computer with a standard asynchronous terminal, and interacts with the system via a "Dialog" (defined here as everything the user types in and everything the system prints out).

The design of the human-computer interface is constrained by several factors:

1. a wide variety of different terminals (CRT and hardcopy) would be used

2. system ABC has to be compatible with several other related systems
3. training and start-up time for users has to be minimized
4. memorization has to be reduced
5. system response time has to be fast and
6. the dialogue had to be designed in accordance with the
   Man-Machine Language (MML) standard of the International
   Consultative Committee for Telephony and Telegraphy
   (CCITT)[3].

It has been made sure that the system would meet the needs
of users, with special emphasis on minimizing training
needs, maximizing initial user performance, and contributing
to job satisfaction.

We have discussed the constraints in designing
human-computer interface in first section. This is followed
by design requirement of HCI and the role of standards and
guidelines for this topic. The MML model, problems with
design guidelines are the central point of this section.
Following section presents a structure of an interactive
system. Proposed model for Human-computer interface has
been shown in next section. Finally this chapter is
concluded with interface characteristics. In this section
four different types of dialogue styles have been discussed.

Human factors specialists concerned with the human-computer
interface live in two worlds. In the world of theory we are
concerned with the properties of the ideal interface -- one
that is easy to learn and use, and results in performance that is efficient and error-free. In the world of practice we must design working systems in the presence of many constraints.

4.1 DESIGN REQUIREMENTS

To determine design requirements for the ABC dialogue, we first tried to discover as much as possible about the potential users, the tasks they would be performing, and the environment in which they would be using the system. For this purpose investigations were made to understand the problem fully. The investigations revealed that the typical users of System ABC would be skilled technicians, male or female, aged 25-55, with 12 years of education, no typing skills, and little or no prior exposure to computerized systems. In a typical day, these technicians would perform a variety of tasks, only a fraction of which would involve work with System ABC. This work would take place in telephone central offices, which are often noisy and crowded with equipment, on a terminal located close to the physical equipment for System ABC. In light of these facts, a structured dialogue with menus and prompts seemed most appropriate.

The main pitfall of this analysis, however, is that it fails to take into account the impact the introduction of System ABC would have on the users, the work, and the environment.
By automating certain tasks previously performed manually and allowing equipment to be controlled remotely. System ABC encourages changes in telephone company work operations. As a result, certain technicians may specialize in the operation of the system, thereby becoming frequent rather than occasional users. Ultimately, the more routine operations may be performed from a distance in regional centers, where the typical user might be an entry level employee, male or female, aged 19-30, with 12 or more years of education, good typing skills, and trained in the operation of computerized systems.

Looking to the future, we thought it important to design a user interface that could be easily modified in later versions of the system. We also thought it wise to provide the system with an interface that would allow the user to move in stages from a highly structured, menu- and prompt-driven dialogue to a more efficient, command-style dialogue.

4.2 THE ROLE OF STANDARDS AND GUIDELINES

The design of System ABC is influenced by the CCITT MML Standard and by various design guidelines. Ideally, standards and guidelines bridge the gap between theory and practice, and allow us to design better interface in a shorter period of time.
4.2.1 Problems With The MML Standard

The purpose of the CCITT MML standard is to provide a syntax, a vocabulary, and a set of formatting conventions for all languages used for human-machine interaction in the operation or maintenance of a telephone network. Because today's telephone networks are supported by a variety of computerized systems, such standardization is desirable as an effective means of reducing operator training time and minimizing errors caused by "negative transfer" of learning between systems.

In view of these benefits, we attempted to implement the MML standard in the design of System ABC. In doing this, we encountered a number of difficulties that are characteristic of standardization efforts.

4.2.1.1 The System ABC Model -

The ABC model defines four levels of user input:

* characters
* fields
* lines and
* procedures

The most basic of these is the field, which consists of a string of characters followed by a field terminator (e.g., a comma or a carriage return). To provide a structured dialog, the ABC system prompts the user for each field of
input. Associated with each prompt are:

* An optional menu.
* A series of validation routines and error messages, and
* A set of branching instructions that interpret the current field and use it to determine the next prompt.

In keeping with this model, the ABC system examines the user's input one character at a time. If character string is special (e.g., "help") the system immediately takes an appropriate action. Otherwise, the input is buffered until the user enters a line terminator (e.g., carriage return). A line of input may consist of a single field, or the user may enter several fields. If a line of input contains several fields, the system evaluates it one field at a time. The ability to enter several fields on a line allows users to "chunk" together input in a flexible way. As the user becomes experienced in the use of the system, we would expect those chunks to map onto "procedures", here defined as a series of input fields containing sufficient information to initiate a specific system action. In this way, the user moves from a structured, menu-and prompt-driven dialog to a more efficient, command-style dialog.
4.2.1.2 The MML Model -

The MML model defines three levels of user input: characters, lines, and commands. The basic level is the command, which consists of a command code plus parameters. The command code, in turn, consists of a command verb plus one or two optional command qualifiers. Parameters may be defined positionally, in which case they simply consist of a parameter value, or they may be defined by keyword, in which case they consist of a name plus a value. The syntax of MML requires that the command verb and qualifiers be separated from one another by hyphens, and that the command code be separated from the parameters by a colon. In addition, parameters are separated from one another by commas, and the equal sign is used to separate parameter names from parameter values. As this description illustrates, the structure of a MML command is potentially rather complex. The MML model assumes that in the typical case a line of input will correspond to a command. However, the standard also allows a command to be assembled from several lines of input. Furthermore, help messages, prompts, menus and form-fill blocks can be provided to aid the user in constructing a command.

4.2.1.3 The Consequences Of Model Incompatibility -

At first glance it may seem that the two models described above are similar enough to allow the MML standard to be
applied to the ABC dialog with little difficulty. In practice, this turned out not to be the case. The nature of our difficulties is illustrated by problems we had concerning the use of the "help" character string.

The MML standard specifies the use of the question mark as a character to request an explanatory help message, a menu, a prompt, or a form-fill block. In the context of the MML model, this use of the question mark is coherent and consistent. In the context of the ABC model, however, it makes little sense to use the question mark to request menus or prompts since these are automatically provided. Nevertheless, a strict reading of the standard would seem to mandate the use of a question mark to terminate each line of input, unless the last field in that line marks the end of a procedure. Doing this, however, makes it impossible to respond immediately to the "help" character with a help message. It also makes it impossible to have a null field indicate acceptance of the default value for a prompt. Consequently, we choose to use the carriage return as a line terminator, a choice that may not be consistent with the MML standard.

Needless to say, we were not aware of the nature or consequences of the model incompatibility described above when we formulated our initial design requirements. Had we been, we might have attempted to make our design fit the MML model. However, it is not clear that we would have been
able to meet the needs of our user population as well with a
dialog based on this model.

4.2.1.4 Problems With Design Guidelines -

There is no doubt that some of the guidelines available to
us were quite helpful[4,5,6]. Nevertheless, we experienced
some difficulties in using these and other guidelines, the
most significant of which are described below.

4.2.1.5 Vague Or Unsupported Advice -

A number of design guidelines advise that a user interface
should be easy to learn, easy to use, powerful, flexible and
fun to work with, that error messages should be informative
but short, and that help messages should be designed to
answer whatever question a user has in mind at a given time.
While these are certainly nice principles, such advice is so
vague as to be useless in the context of real design work.
At the other extreme, guidelines sometimes offer advice that
is very specific but unsupported.

4.3 STRUCTURE OF AN INTERACTIVE SYSTEM

The structure of a human-computer interaction can be divided
into three components as shown in figure 4.1. The set of
all Dialogue-Computation (D-C) functions, plus the highest
level pure dialogue and pure computation function comprises
the control structure of the entire system. The dialogue
FIG. 4.1 STRUCTURE OF AN APPLICATION SYSTEM
component is the set of all dialogue functions (circles) and the computational component is the set of all purely computational functions (Boxes). The control component and the dialogue component, taken together, are the behavioral structure which determines the form, content, and sequencing of external dialogue at the human-computer interface. The computational component, via internal dialogue, sends data for display and receives input values for computation. The behavioral structure can be prototyped, and system behavior demonstrated, before the computational component is implemented.

The design life cycle begins with a conceptualization of the problem and requirements specification. As the need for dialogue and computational functions becomes known, these are developed independently. Dialogue independence allows these two components to be developed separately from each other and from the control structure.

4.4 HUMAN-COMPUTER INTERFACE MODEL

Every exchange of information between a computer and its user follows a specific sequence. This exchange is called a dialogue transaction and a human-computer interfaces is composed of many transactions. The developed model for HCI has been shown in figure 4.2.

The dialogue of human-computer interfaces is typically
FIG. 4.2 MODEL OF HUMAN-COMPUTER INTERACTION SYSTEM
conducted by means of a transaction sequence, based on an input-process-output configuration[10]. A dialogue input transaction prompts the user and validates the resultant input. That input is passed to the computational component, where it is processed, and the results of this processing are displayed to the user via a dialogue output transaction. Each circle is a dialogue transaction of one of the types shown in the model. The collection of all dialogue transactions for an application system is its dialogue component.

A transaction is a sequence of one or more interactions to extract an input and/or to produce an output. An input interaction consists of following parts:

* System Prompt
* Human language input
* System syntactic configuration.

Any of these parts may be implicit in a given interaction. Each prompt is a general display part and can be made up of various pieces, including pieces having these styles:

* List menu
* Labeled keypad outline
* Text
* A form to be filled in.

Correspondingly, the language input definition can be composed of pieces featuring menu selection, keypad selection, command string input, form-filling.
Various components of the model has been discussed in following sections.

4.4.1 Login

As soon as a user inputs his/her identity (user name and password), validity check is performed for the user's authorization. Here standard UNIX login procedure has been implemented. For unauthorized user system gives some informative message. User can try to login as many time he/she wishes. This check has been built to ensure that only authorize user may access the system's database. The authorization check is done by the help of user and password table. Details of this data structure has been discussed below.

<table>
<thead>
<tr>
<th>Name</th>
<th>User table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>To check the validity of the user for accessing the system.</td>
</tr>
<tr>
<td>Location</td>
<td>Main memory</td>
</tr>
<tr>
<td>Organisation</td>
<td>Sequential</td>
</tr>
<tr>
<td>Access mechanism</td>
<td>Sequential</td>
</tr>
<tr>
<td>Initialisation</td>
<td>At system initialisation time from disk.</td>
</tr>
<tr>
<td>Note</td>
<td>None</td>
</tr>
</tbody>
</table>
Structure:

```c
struct {
    char user_name[NAM_SIZE];
    /* Max. no. of char. allowed for user_name */
    char password[PASWRD_SIZE];
    char commd_clas[ALOWED_CLS];
} user[MAX_SYS_USER];
```

4.4.2 Data-initialization

After a successful login, some relevant data is loaded into memory of the terminal. These are command class allowed for a user from a terminal.

A user may be allowed to work for several command classes. A terminal, from which a user is working, may support some command classes only. So to check the eligibility to work with terminal is checked for every command. This is very frequently used data. This data is stored in local memory of the terminal for quick reference. Intersection is performed between data of terminal command class and user command class, which are globally stored in memory of the system for every terminal and user. Mathematically it would be represented as

\[ X = A \cap B \]

Where \( A \) = All command classes allowed from a
\begin{align*}
\text{B} &= \text{All command classes allowed for a user.} \\
\text{X} &= \text{All command classes allowed for a user from the working terminal.} 
\end{align*}

4.4.3 Syntactic Confirmation & Command Validation

Here the syntax of the command is checked and proper command validation is done. If syntactically the given command is correct, the range check, interfeature restriction, user eligibility to work with the issued command, command eligibility with that terminal etc are checked. After command syntax is checked and validated properly, on successful condition it would be passed to computational part for further processing. In case of unsuccessful validation proper error message would be passed to the operator and prompt would be displayed for his/her further action. It is upto user to decide, what to do further, what mode of input he/she wants to select or wants to quit the session etc.

Basically user's input is validated & interpreted at this level of model. Human-Computer interface is designed in such a way that errors in commands or control actions do not cause the system to stop or unduly alter the system configuration. Commands errors and parameters related errors are detected at the validation level only and are indicated by outputting proper error messages or error
codes. Such commands are not passed to the computing processes to avoid overloading of the computing processes. Syntax of the Human-Computer language would be checked according to syntax specified in Appendix B. Whereas input (command) validation would be done through various data structures. Relevant data structures have been discussed in this section.

4.4.3.1 Command-class Table -

| Name       | User table          |
| Purpose    | To get the command code of a command class |
| Location   | Main memory         |
| Organisation | Sequential         |
| Access mechanism | Sequential          |
| Initialisation | At system initialisation time from disk. |
| Note       | None                |
| Structure  | struct              |

    char    no_comnd_clas;
    /* no. of command classes */
    char   comnd_clas[ALLOWED_CLS];
    comnd_code *comnd_cod_ptr;
    } comnd_class[MAX_COMND_CLAS];
4.4.3.2 Command-code -

Name : command code
Purpose : This data structure is used to match the command code input by user and to get the parameter related information for guidance.
Location : Main memory
Organisation : Sequential
Access mechanism : Indexed
Initialisation : At system initialisation time from disk.
Note : None
Structure :

```c
struct {
    char des_length;
    /* description length of command */
    char des_comnd_clas;
    char *relation_count;
    char *relation;
    char comnd_code;
    int *ptr_parameter_table;
} comnd_code_info[MAX_COMND_CODE];
```

4.4.3.3 Parameter Table -

Name : parameter table
Purpose : To get parameter related information corresponding to every command.

Location : Main memory

Organisation : Sequential

Access mechanism : Indexed in the big table then sequential in small table meant for that command.

Initialisation : At system initialisation time from disk.

Note : This structure is variable. Number of parameter varies command to command.

Structure :

```c
struct {
    char    des_length;
    /* description length of parameter */
    char    des_commd_code[20];
    /* description of command code, where it has been referred */
    char    par_count;
    /* parameter count */
    char    par_type;
    /* type of parameter */
    char    par_length;
    /* length of parameter */
};
```

- 90 -
char par_name[20];
    /* parameter name */
char def_value;
    /* default value */
} parmtr_detail;

4.4.3.4 Terminal & Command-class Relation -

Name : Terminal & command-class relation

Purpose : To check the eligibility of a command from a terminal.

Location : Main memory

Organisation : Sequential

Access mechanism : Sequential

Initialisation : At system initialisation time

Note : This structure is variable. Number of command classes allowed from a terminal are varying.

Structure :

Structure
{
    char no_cls_eligible;
        /* no of eligible command classes */
    char cls_eligible[MAX_CLS_ALWD];
}
4.4.3.5 Combination Table -

Name : Combination table
Purpose : To check the validity of the combination of values allowed for a parameter.
Location : Main memory
Organisation : Bit map structure
Access mechanism : Indexed to get bit.
Initialisation : At system initialisation time from disk.
Note : Referenced by bit position.
Structure :

```c
struct {
  char comb_value[10];
} combination_value;
```

4.4.4 Algorithms

There are many algorithms, but some important one, which are involved in syntax check have been discussed here.

4.4.4.1 User Access Algorithm -

Name : User access algorithm
Input : user name & password.
Result : Authorized/Unauthorized user.
occasion : At every login time.
Steps :
Read user name and password in buffer;
compare this input from user table of system;
  /* user table keeps list of all valid
   users of the system */
IF ( entry exists for this input in user table)
  read all command classes allowed for the user;
  /* This would be read from the user-command
   class relation of the memory */
read all command classes allowed from a terminal;
  /* This would be read from the terminal-command
   class relation of the memory */
compare allowed command classes of user from the
allowed command classes of terminal one by one;
IF( command classes are matching)
  store these command classes in a local memory
  of terminal;
  display successful login;
  display header; /* work station identity, date
  & time of the day */
  display system prompt;
  /* now system is ready to accept the user
  input */
ELSE
  display access not allowed;
  terminate the session;
  - 93 -
display login prompt;
ENDIF
ELSE
   display unauthorised user;
   terminate the session;
   display login prompt;
   /* now system is ready to accept again
   user identification. */
ENDIF

4.4.4.2 Command Code Eligibility & Syntax Check -

<table>
<thead>
<tr>
<th>Name</th>
<th>Command code eligibility &amp; syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>command code with parameter details</td>
</tr>
<tr>
<td>Result</td>
<td>command code allowed or not allowed to work.</td>
</tr>
<tr>
<td>occasion</td>
<td>At the time of starting a command session.</td>
</tr>
</tbody>
</table>

Steps :

Find command class for chosen command code;
search for this command class in terminal-command class entry;
   /* terminal-command class-user entry is stored information of intersection of terminal allowed classes and user allowed classes. */
IF ( entry found )
user is eligible to issue a command;
syntax check is performed for the issued
command;
IF ( command is syntactically correct )
send the command & its details to
computational part of the model;
wait for result;
ELSE
display syntax error;
terminate the command session;
display the system prompt for next
command input;
ENDIF
ELSE
display access not allowed;
terminate the command session;
display the system prompt for next
command input;
ENDIF

4.4.4.3 Menu Selection Algorithm -

Name : Menu selection algorithm
Input : menu selection option.
Result : display of menu if access allowed.
occasion : At the time of selecting menu option.
Steps :
Accept menu option of the user;
IF ( menu option is for command class)
   /* check whether user is allowed for
      that command class or not */
   IF ( user is allowed for the command class)
      IF ( terminal supports that command class)
         display menu with their selection option;
         be ready to accept the menu option input
         by user;
      ELSE
         display access not allowed;
         display system prompt;
         be ready to accept new input;
      ENDIF
   ELSE
      ENDIF
ELSE
   IF (menu selection is for command)
      find command class for the command code input;
      check for eligibility of the user for the
      desired command;
      check for the eligibility of the terminal for the
      issued command (command-class);
      IF ( both are matching)
         display details of the menu selection;
      ELSE
         access not permitted;
      ENDIF
   ELSE
      ENDIF
ELSE
4.4.4.4 Parameter File Name Assignment -

Name : Parameter file name assignment
Input : file name
Result : Assignment of parameter value
occasion : While inputting a command, instead of giving list of values directly, a file name is assigned. List of values are read from file and assigned to the parameter.

Note : While assigning file name, a dollar($) symbol is put before file name. This is made to understand that values for the parameter would be read from a file.

Steps :

IF ( parameter is allowed to take value from file)
   IF ( file name exits)
      IF ( file empty )
         move cursor to the place, where file name was assigned;
         /* This is done to allow user to give
some valid value directly */

ELSE

REPEAT

read value from file;
check the validity of the value for parameter;

/* here type,length,range would be checked */

UNTIL ( EOF is not found OR value is not valid)

IF ( EOF is encountered)
    move cursor to the next parameter;
ELSE

IF ( value is not valid)
    display error;
    move cursor to the place, where file name was assigned;

ENDIF

ENDIF

ENDIF

ENDIF

move cursor in front of the parameter for which file name was assigned;
4.4.4.5 Validation Of Parameter Range Set Values -

Name : Parameter value validation
Input : Range or set value
Result : Valid or invalid value
occasion : While inputting a command, a range or set of values can be given as parameter value.

Note : None

Steps :

IF ( more than one value assigned to a parameter)
  IF( set value allowed )
    IF(range value assigned)
      IF(range value allowed)
        IF( 1st range value is LESS THAN 2nd)
          XYZ : REPEAT

          IF(no of calculated values > 1)
            IF(valid value)
              decrement number of values by one;
            ELSE
              display error;
              position cursor-function;
              /* This would position the cursor in front of parameter, for which error occurred */
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
ENDIF
ELSE

IF(number of parameter Equals total parameter)
wait for user input;
ELSE
increment parameter number of command by one;
position cursor for the parameter;
wait for user input;
ENDIF
ENDIF

UNTIL(range or set values are found proper)
ELSE
display error;
position cursor infront of the same parameter, for which error occured;
system waits for user input;
ENDIF
ELSE
calculate the number of values;
/* exact values would be known */
goto XYZ;
ENDIF
ENDIF
ENDIF

display error;
ENDIF
4.4.5 Result Display

Once the computation of the command is over, result is passed to the result display unit. If complete computation time is long, partial result is passed to the operator, so that he/she would feel about the execution of his/her command. Specially for novice user partial result display is very much necessary. They may be anxious about the result of the command.

After completing display of the result of a command, again prompt is displayed to the user.

4.4.6 Dialogue Executors

At execution time, the form of the dialogue transaction model is built into the control structure of a transaction executor. This transaction executor is data driven at run time to instantiate a dialogue transaction, and to process the dialogue transactions of an applicative system interface. Each part of the transaction has its own executor, called by transaction executor: a display executor to interpret the display definition and to produce a display; a language executor to interpret the language input definition and to accept, parse, and validate the user's input; and a confirmation executor to interpret the confirmation definition and to produce the system
confirmation. Invoked by the transaction executor, a
language executor uses the stored language input
definitions, which were created at the design time, to
process each character which the end user enters at run
time, with appropriate validation checks.

The approach taken dictates that wherever possible, validity
checking is specified in the dialogue component as part of
the syntax of the language specification, rather than in
semantic routines in the computational component. That
achieves two goals: validity checking is dynamic as a
transaction progresses, and no programming is required.
While validating token completion or spelling correction are
extremely sophisticated and beyond the normal skills of most
programmers and dialogue designers. Others such as format
and range checking, are included because these capabilities
are so frequently needed in Human-computer interaction.

4.5 INTERFACE CHARACTERISTICS

4.5.1 Dialogue Style

We propose here a classification in which user-system
dialogue styles can be viewed as differing basically in
terms of two independent characteristics:

(1) Whether the interchange is guided overall by the
user or the system, and
(2) whether the user has to make a choice of his
input from a set of alternatives presented by the system or else is able to provide a free response.

The party that is the "guide" is the one that takes the initiative during the course of the exchange and also decides on a satisfactory termination point.

If in a questionnaire situation, the system phrased questions in the form "PLEASE ENTER YOUR AGE: ", the user would be unconstrained as to input and would be in the free response mode. However, if the question were phrased: "PLEASE INDICATE YOUR AGE; (1) LESS THAN 18, (2) BETWEEN 18 AND 30, AND (3) OVER 30", the user would be in the forced choice mode. Although it might appear that when users select from the systems menu the system must be guiding the interaction, this forced-choice Vs free-response distinction is independent of the question of who is the guide. For example for a computer questionnaire, the computer creates a situation in which (1) the system is the guide, and (2) the user is in the forced-choice mode. However, in another situation the user may, for example have invoked some on-line Help facility and is presented with a menu of selections from which to choose the closest to his information requirement. Here the system is definitely not the guide, but the user is still in the forced-choice mode.

Since the two distinctions are independent of each other,
there are four basic types of dialogues, and each has different advantages.

4.5.1.1 System Guides/user Has Forced-choice -

This is most appropriate for routine task activities, where it is appropriate to step the user through a fixed procedure, providing menu selection at each step. Not only may this enhance the speed of accomplishing the task, but also the restriction of input to a small set of alternative greatly reduces the possibilities of error. When the user enters a number as part of the guided task, the forced-choice concept is still considered to apply, since (1) the user is forced to pick a number rather than anything else, and (2) quite often the number will be checked against expectations and rejected if out of bounds etc. This guided/forced situation is also appropriate for structured interviews, surveys or any information-gathering activity in which the topics and categories of interest can be specified ahead of time.

4.5.1.2 System Guides/user Has Free-response -

This dialogue is particularly appropriate when the overall objective is unstructured information-gathering such as an interview in which user is asked general questions (e.g. what are your life goals) where providing structure might actually interfere.
4.5.1.3 User Guides/user Has Forced-choice -

This combination is desirable when the user is at least somewhat knowledgeable about possible requests he could make of the system, or when the requirement to make a free-choice would be an unnecessary burden. This dialogue style is often chosen for allowing a user to select desirable system alternatives or information possibilities.

4.5.1.4 User Guides/user Has Free-response -

This provides the maximum latitude for the user and is therefore most appropriate for experienced and confident users performing complex tasks. This situation also is the least structured and, with current technology, allows the maximum opportunity for miscommunication between user and system.

Heuristic for shifting from one to another of these modes can easily be imagined as a function of the user’s difficulties and desire during the session.

For example, in a user guided interaction, the dialogue could be shifted from that of free-response to forced choice either by signal of the user or on the basis of the system’s detection of inordinate user difficulties.

In designing the actual input of responses and commands from the user, ease of use has been given top priority. First, the need of memorization by users has been minimized. In
In first and second type of dialogue style, we have considered a user, who is neither novice nor expert. To a novice user full details of a command is displayed and option is open for him to select. A typical example has been shown for this purpose:

<table>
<thead>
<tr>
<th>DATE TIME</th>
<th>WORK-STATION</th>
<th>IDENTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_SUB_LINE</td>
<td>CREATE A NEW SUBSCRIBER LINE</td>
<td></td>
</tr>
</tbody>
</table>

```
+------------------------------------------------+
<table>
<thead>
<tr>
<th>DATE TIME</th>
<th>WORK-STATION</th>
<th>IDENTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_SUB_LINE</td>
<td>CREATE A NEW SUBSCRIBER LINE</td>
<td></td>
</tr>
<tr>
<td>DIR[NUMJ {6DIGIT}</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>BM_NO [NUMJ {1 TO 4}</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>FAC [ALPHAJ {8char</td>
<td>} = HOT_LINE</td>
<td></td>
</tr>
</tbody>
</table>
+------------------------------------------------+
```

[ ] indicates type of parameter
[ ] possible values

For a novice user the above mode of interaction would be most suitable. Because such class of user would need help at every level. Instead of invoking help facility by user, we are providing the required information to be used while in session.

For a user, who has got little experience, this mode of interaction would be irritating for him. He knows little about the commands, so providing parameter type and possible values at every level would be verbose. If he needs any help he/she can invoke the help facility at any level. For this type of user the format most suitable is shown here.