1987

A METHODOLOGY FOR ADAPTIVE USER INTERFACE DESIGN

Feng-Yang Kuo
University of Colorado, Denver

Jahangir Karimi
University of Colorado, Denver

A METHODOLOGY FOR ADAPTIVE USER INTERFACE DESIGN

Feng-Yang Kuo
Jahangir Karimi
Department of Information Systems
University of Colorado, Denver

ABSTRACT

A methodology, AUI (Adaptive User Interface), is presented in this paper for the design of user interfaces that can accommodate users of different skill levels. The conceptual user interface model, referred to as the dialogue schema in the AUI methodology, is derived by analyzing the control structure of task requirements from the real time system and formal language perspectives. The dialogue schema is used to generate three forms of dialogues: question/answer, menu/form, and formal language. Alternative software tools to facilitate the AUI methodology are also discussed.

1. INTRODUCTION

Recently, for user-computer interface design, formal specification techniques such as extended State Transition Diagrams (STD) and Backus-Naur Form (BNF) are used. The advantages of these techniques are that (1) they allow the designer to specify the user interface behavior independent of software implementation, (2) they can be stored in executable forms which are readily manipulated by software tools (Kuo and Konsynski 1987; Wasserman 1985; Wasserman et al. 1986) and (3) they can be used to incorporate human factor guidelines into the interface design (Reisner 1981, 1984).

The research so far, however, has not addressed how user interface specifications using STD and BNF can be derived conceptually. For the techniques to be useful to system designers, a top-down approach to deriving the user interface specifications from the original task requirements must be provided. This paper presents a methodology, AUI (Adaptive User Interface), to address this issue. Furthermore, the AUI methodology extends the techniques to achieve adaptive interface design (Dehning, Essig and Maass 1981; Edmonds 1978, 1981; Hall 1978) -- a design of interfaces that can accommodate needs of different users (e.g., skilled versus novice). Many software tools can also be used to facilitate implementation in various phases of the AUI methodology. Consequently, effort to design user interfaces can be reduced when the AUI methodology is used.

In Section 2 of this paper, an overview of the AUI methodology is presented. Sections 3 and 4 present the methodology and its software tools respectively. The final section discusses the conclusion and future research. Throughout the paper, a book-ordering system (Gane and Sarson 1979) is used as an example to illustrate the AUI methodology.

2. OVERVIEW

User interface design can be said to be a design of *user-system communication* and of *control* over this communication process by both the user and the computer system. In the AUI methodology, the user interface is treated as a *formal language* to facilitate the communication between the user and the system. The basic linguistic structure of the user interface is specified with modeling techniques extended from STD and BNF, both of which are context free grammars.

In addition, this basic linguistic structure is derived by analyzing the *task's control structure* (the timing and condition) in order for the language to adequately facilitate the user in completing the task (i.e., to invoke functions and to provide data). In the AUI methodology, the user-system interactions are regarded as *events* (actions initiated by the user or the system) happening in a *real time system environment*, where both the user and the system act concurrently and both demand the control of the system. To analyze the task requirements and

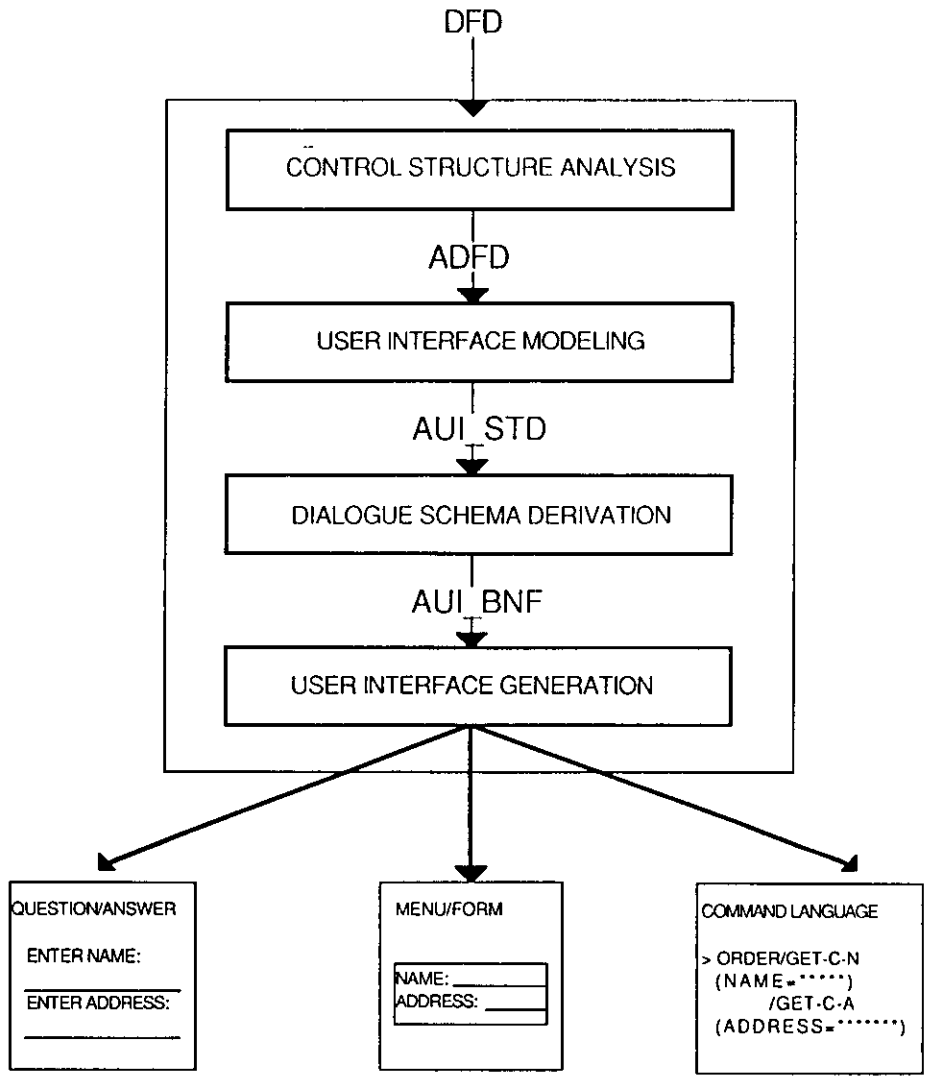

Figure 1. An Overview of the AUI Methodology

the control structure, the AUI methodology adopts two analysis tools: the Data Flow Diagram (DFD) (Gane and Sarson 1979) and the Augmented Data Flow Diagram (ADFD) (Ward 1986). The DFD, which represents the system's process and data requirements, is the input to the AUI methodology. From this DFD, the ADFD is derived to capture the control structure, which is an integral part of the requirements for any real time system.

The extended STD and BNF models can then be developed according to the ADFD. The extended STD and BNF models, referred to as the *dialogue schema* of an application system, are used to generate three forms of dialogues: formal language, menu/form, and question/answer. Also, software tools can be used to facilitate implementation; those tools will hide implementation details from both the user and the designer.

Figure 1 shows the steps of the AUI methodology. The AUI methodology takes the DFD as input from which the ADFD is derived. A technique extended from STD is used to model the behavior described in the ADFD. The extended STD representation is then mapped into the equivalent extended BNF representation.

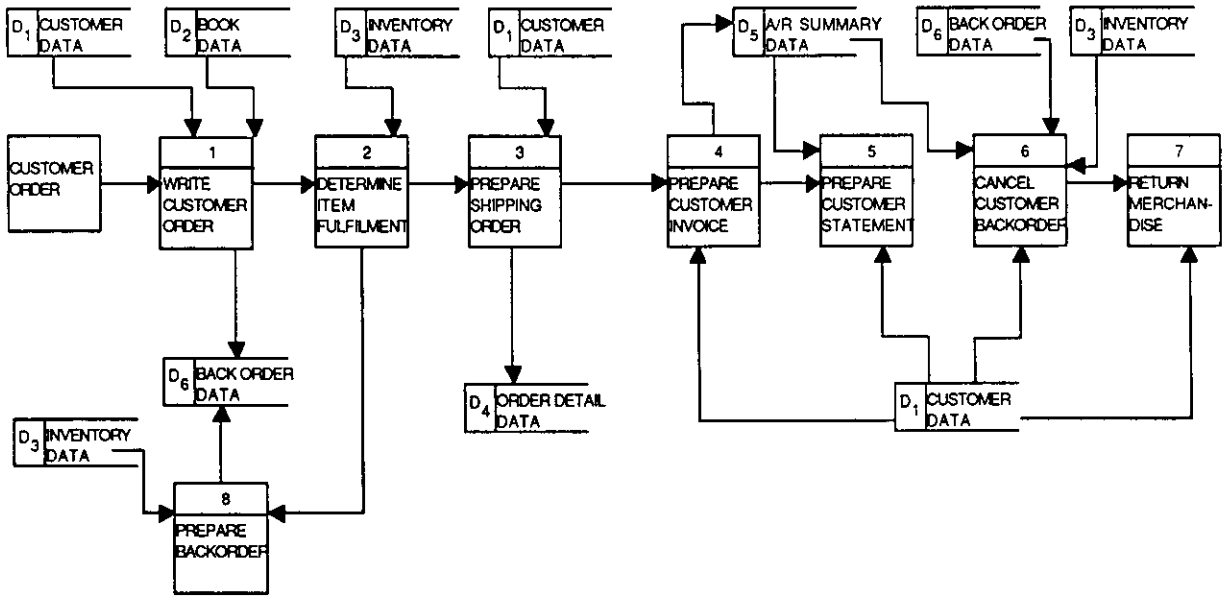

Figure 2. High-Level for Book-Ordering System

3.THE AUI METHODOLOGY

3.1 Deriving the Augmented Data Flow Diagram

The AUI methodology takes the DFD of the system to be developed as its input. Figure 2 shows the overall DFD and a section of the detailed DFD for the book-ordering system. Discussed in depth by Gane and Sarson (1979), details regarding using the DFD for requirements specification will not be discussed here.

To derive the Augmented Data Flow Diagram of the system, the following procedure (summarized in Figure 3) is applied. First, the designer identifies all data that must be received from and sent to the user and marks processes that support the user's access to those data. This step results in many marked processes in the DFD. Second, from this marked DFD, the designer provides descriptions of events affecting each marked process. An event, in the AUI methodology, is a user's action (e.g., entering data) to which the system must respond by invoking the marked process (e.g., validating data). Events are described without specifying how they take place procedurally. An event may affect several processes; in such a case, these processes are said to have interweaved events. Figures 4a and 4b illustrate the application of these two steps to the book-ordering system.

Third, for each group of processes that have interweaved events, an *agent* is assigned and its associated *event-flows* are specified. An *agent* is a named control process that monitors the events by receiving input event-flows from and sending output event-flows to the marked processes. An input-event flow, which must be received by an agent, is the signal representing the result of execution by the marked process. The output event-flow, which must be sent by an agent, is the signal to enable/disable (activate/deactivate) the marked process. In addition, each agent has a "starting" input event-flow and an "exiting" output event-flow to begin and end the agent's activities. When the "exiting" output event-flow is sent by the agent, all processes associated with the agent will be deactivated. Thus, the deactivation of processes in the AUI methodology is implicitly defined by the "exiting" output event-flow. Figure 4c shows the agent C-N-AGENT, marked processes GET-C-N and VALIDATE-C-N, and event-flows between the agent and the two processes. A number is assigned to the event-flows to show the order of event-flows.

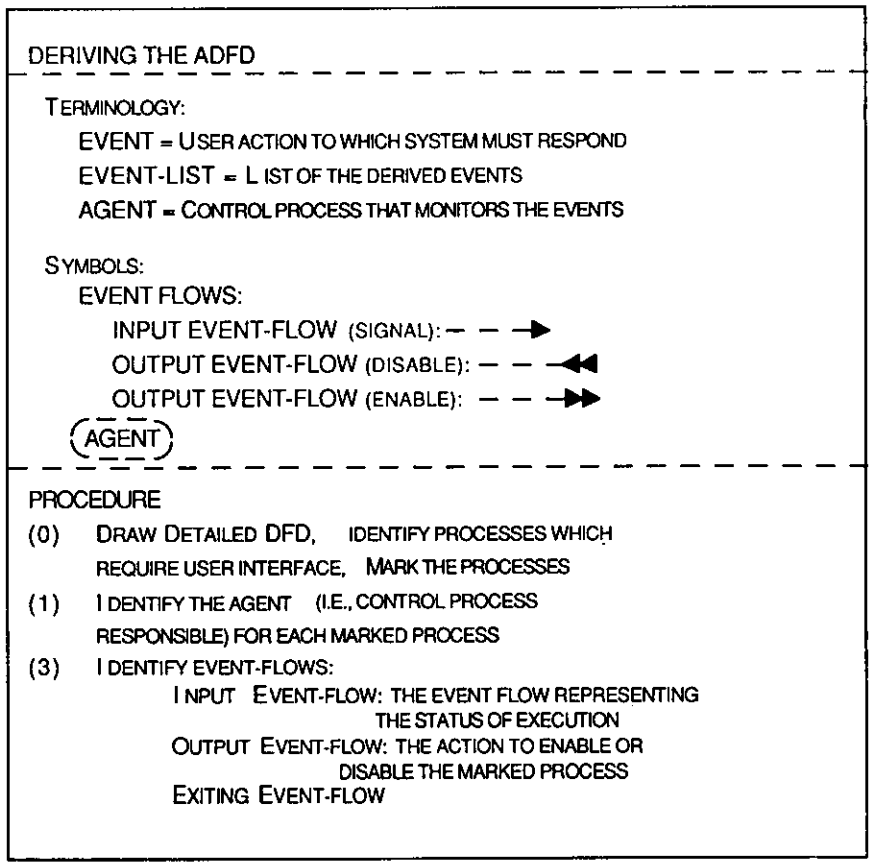

Figure 3. Procedure to Derive the ADFD

When this three step procedure is applied to all marked processes, the ADFD is generated. The ADFD includes (1) the data and process requirements specified in the original DFD and (2) a list of agents and their associated event-flows that describe the user-system interaction requirements from a real time system perspective.

3.2 Deriving the AUI-STD Model from the ADFD

In the AUI methodology, a technique extended from State Transition Diagrams (STD), called AUI-STD, is used to model an agent's behavior described in the ADFD (summarized in Figure 5). Figure 5 shows the AUI-STD for a section of the book-ordering system.

To derive the AUI-STD for an agent, the following procedure is applied. First, various states of the system are identified. A state is a result of an output event-flow sent by the agent. It represents that agent waiting for a certain user action or an internal process to be completed. In other words, a marked process is in an active state when "some activity (of the process) is taking place" and becomes inactive when its activity stops. A name is given to a state to describe what is happening outside the agent but within the boundary of the ADFD. Several states may be associated with any given agent. In addition, a START state is created to represent the state in which the system is ready and an EXIT state is created to represent the state in which the agent ceases all activities.

Second, transitions between states are specified. A transition has two parts: a condition and the action carried out when the condition is met. A transition occurs because the agent receives a certain input event-flow that requires the system to respond and therefore sends a follow-up output event-flow to enable or disable a certain marked process. Thus, an input event-flow becomes the condition of a transition and the follow-up output event flow becomes the action. In the AUI methodology, an action can only be (1) enable, (2) disable, or (3) leave the ADFD (corresponding to the "EXITING" output event-flow).

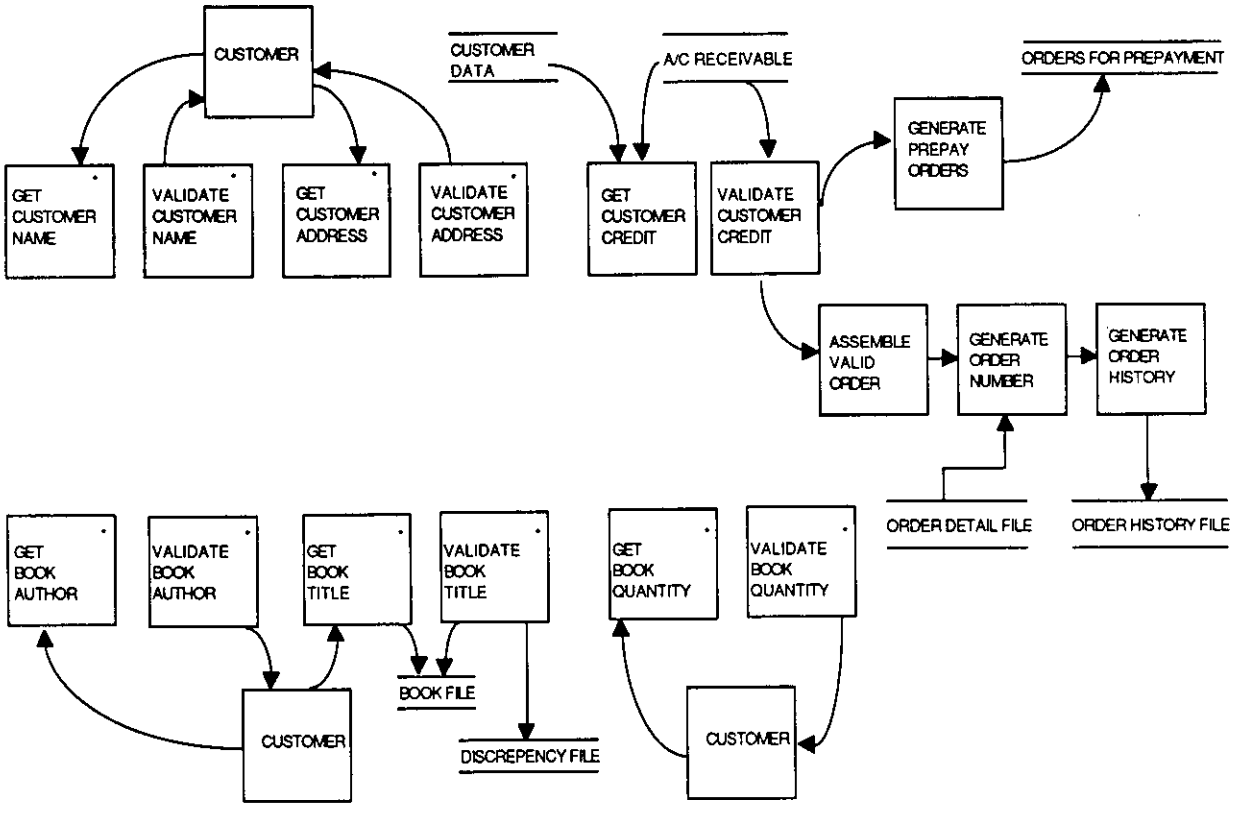

Figure 4a. A Section of Detailed DFD of the Book-Ordering System

- ARRIVING CUSTOMER ORDER
- FILING CUSTOMER NAME
- VALIDATING CUSTOMER NAME
- FILING CUSTOMER ADDRESS
- VALIDATING CUSTOMER ADDRESS
- FILING BOOK TITLE
- VALIDATING BOOK TITLE
- FILING BOOK AUTHOR
- VALIDATING BOOK AUTHOR
- FILING BOOK QUANTITY
- VALIDATING BOOK QUANTITY

0
0
0

Figure 4b. Event List

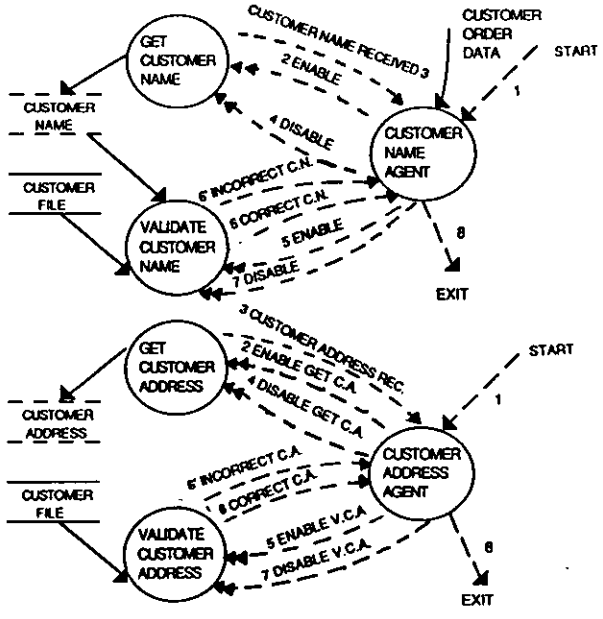

Figure 4c. The ADFD for a Section of the Book-Ordering System

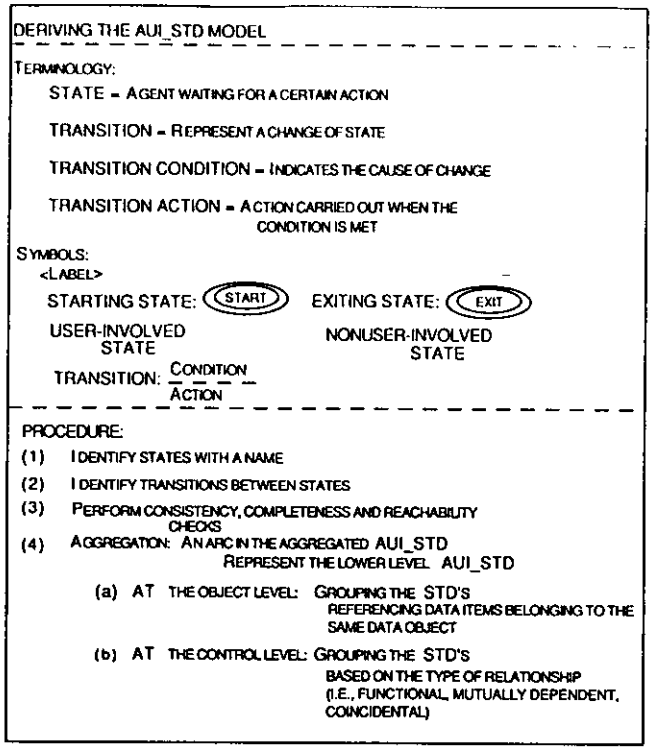

Figure 5. Procedure to Derive AUI-STD from ADFD

Third, the following checks for consistency, completeness, and reachability are performed to ensure that the AUI-STD is correct (Birrell and Ould 1985; Karimi 1986; Ward 1986). For the AUI-STD to be consistent with the behavior of its agent, two conditions must be satisfied: (1) each input event-flow of the ADFD must correspond to a transition condition and vice versa and (2) each output event-flow must correspond to an action and vice versa. The AUI-STD is complete if, for any given input event-flow, a transition and its action (output event-flow) are defined. The AUI-STD is reachable if at least one path (consisting of one or more transitions) is defined between the START state and the EXIT state. After these checks are performed, a name is assigned to the AUI-STD.

Fourth, an aggregation process is applied to integrate all AUI-STDs to show the overall user-system interaction requirements. To do so, a *recursive structure* is incorporated in the AUI-STD model (Jacob 1983; Woods 1970). In an aggregated AUI-STD, a labeled arc is used to represent a lower-level AUI-STD and a named node is also created as an intermediate state to connect the arcs. (Usually, the arc's label is the same as the name of the corresponding lower-level AUI-STD.) Aggregation will result in a hierarchy of AUI-STDs, in which a higher-level AUI-STD's arcs actually represent lower-level AUI-STDs and only the lowest level AUI-STD's arcs have conditions and actions.

Aggregation of the AUI-STD occurs at two levels: the object level and the control level. Aggregation at the object level requires the grouping of AUI-STDs that reference data items belonging to the same data object. From the standpoint of human cognitive factors, this grouping is necessary to ensure that, in the course of user-system interaction, user knowledge of "objects" used in the current task can be applied. Thus, the information processing capability (i.e., short- and long-term memory) required by the user (e.g., entering items belonging to the same object) can be reduced.

Aggregation at the control level is further divided into three types of grouping: functional, mutually dependent, and coincidental. Functional grouping is the grouping of AUI-STDs for ADFD agents that must take place serially (i.e., in a certain order) for a given task. Functional grouping must be performed first because, for the user to complete the task easily, the structure of user-system interfaces must closely resemble the user's knowledge of the task (which is supposedly embedded in the ADFD). Furthermore, functional grouping results in a reduction of information processing capability requirement because it releases the user from storing the task information in long-term memory. For the same reason, mutually dependent grouping is performed next. Mutually dependent grouping is the grouping of AUI-STDs for agents which (1) can take place simultaneously but the order of execution is immaterial and (2) refer to same data objects that are required to complete a given task. Finally, coincidental grouping is performed to group the STD of unrelated agents (no timing or data relationship).

The result of aggregation is a complete AUI-STD model that describes the user interface structure. This model is conceptual because no implementation details have been described. A complete AUI-STD model for the book-ordering system is given in Figures 6 and 7.

3.3 Generating the AUI-BNF Model

The AUI-STD model, which uses a recursive structure, can be mapped into a context free

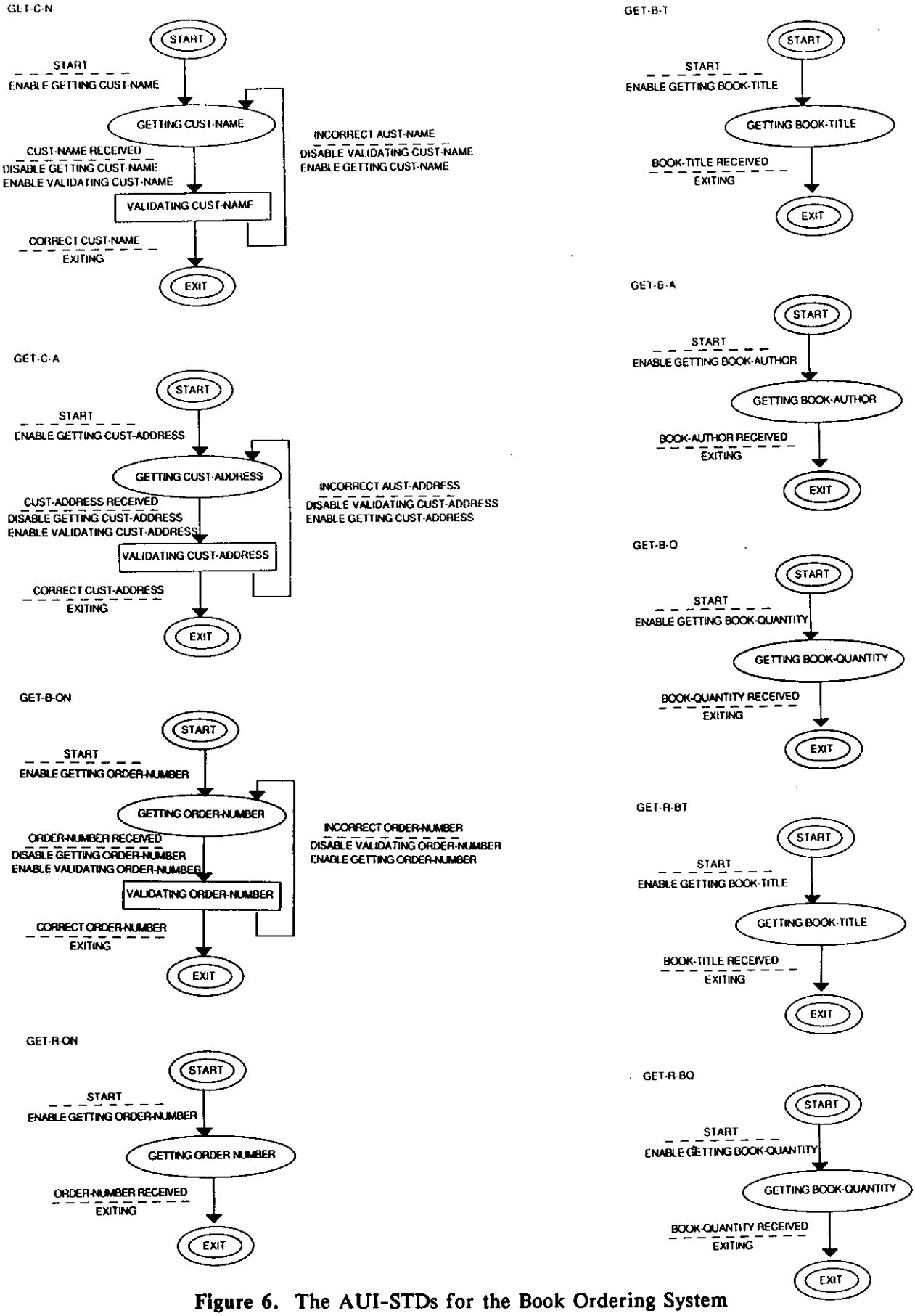

Figure 6. The AUI-STDs for the Book Ordering System

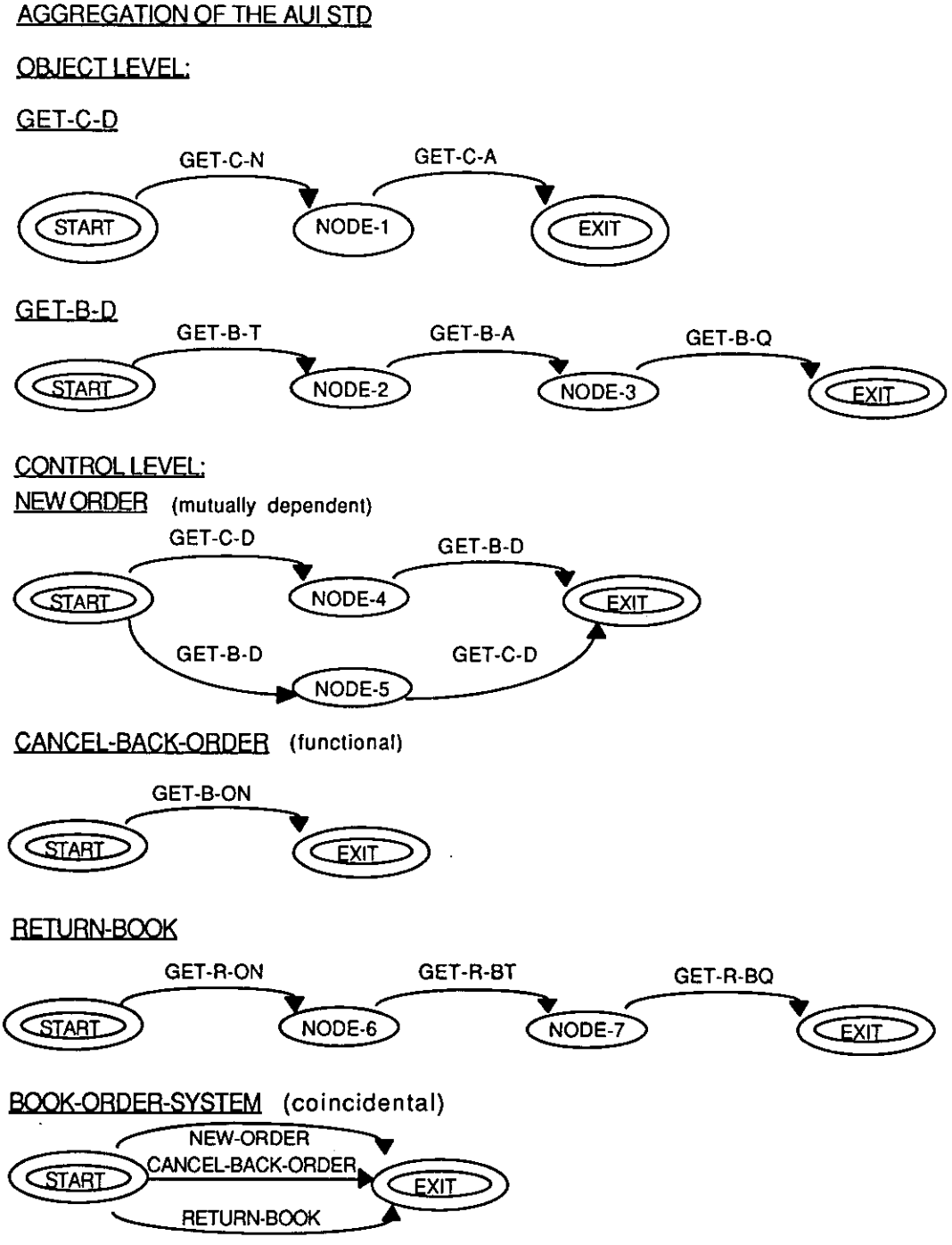

Figure 7. Aggregation of AUI-STD at the Object and Control Level

grammar using a BNF like representation. In the AUI methodology, a technique extended from BNF, called AUI-BNF, is used for this purpose (summarized in Figure 8). The AUI-BNF model consists of production rules that describe how the language (the user interface) can be generated through two types of symbols: terminal and non-terminal. A terminal symbol is a word in a lexical category

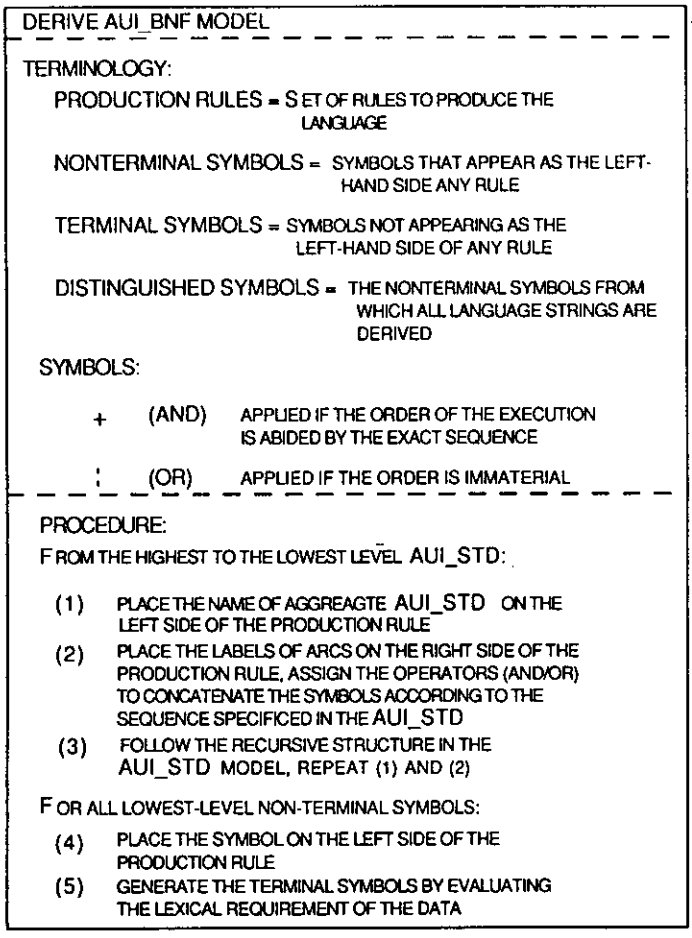

Figure 8. Procedure to Derive AUI-BNF Model

(e.g., alphanumeric) and can only appear on the right side of a production rule. A non-terminal symbol is a composite syntactic symbol to be described by other terminal or non-terminal symbols. A non-terminal symbol may appear on the right or left side of a production rule. Two operators are available to concatenate symbols on the right side of a production rule: the OR operator (represented by the "|") for symbols that can occur in any order and the AND operator (represented by the "+") for symbols that must occur in the specified order. A * sign is assigned to a symbol (on the right side of a production rule) when an arbitrary number of repetitions are allowed. The AUI-BNF model of the book-ordering system is described in Figure 9.

To translate the AUI-STD model into the AUI-BNF model, the following procedure is applied. First, from the highest to the lowest level AUI-STDs, generation of the language is obtained by placing the name of the aggregated AUI-STD on the left side of a production rule and, on the right side, the labels of arcs. To concatenate the symbols on the right side of a production rule, the operator OR (|) is applied if the order of execution in the AUI-STD is immaterial. Otherwise, the operator AND (+) is applied according to the exact sequence defined in the AUI-STD. Because of the recursive structure embedded the AUI-STD, this process can be repeatedly applied to define all non-terminal symbols used in the AUI-BNF model for a given system. To define the terminal symbols, the lexical requirements of the data are evaluated. Terminal symbols of a given lexical category (e.g., 1, 2, ..., 0 for DIGITS; A, B, ..., Z, a, b, ..., z for ALPHABETS) are then assigned to the right side of production rules to generate the lowest level non-terminal symbols (obtained from the lowest level AUI-STD).

Thus, for the highest level AUI-STD, a starting symbol (i.e., the name of the aggregated STD for the entire system) is created as the *distinguished non-terminal symbol* from which all strings of the language are derived. This distinguished non-terminal symbol is produced by placing the labels of arcs of the highest level AUI-STD on the right side of the rule. This procedure is repeated until reaching the lowest level AUI-STD, in which the name of the arc is obtained by using the name of data item referenced in the transition's action. (The data item name can be obtained from the DFD's data dictionary if available.)

The internal state (i.e., a rectangle) requires special treatment in the AUI methodology. The internal state and its outgoing transitions are ignored if no cyclic transition is caused by the activity of the internal process. (That is, the user need not have the language to interact with the system.) If, however, a cyclic transition is caused (e.g., GETTING-C-N triggers VALIDATING-C-N which in turn causes a transition to GETTING-C-N), a special token DISPLAY is used as a terminal symbol and a repeating definition is defined in the BNF. This token is used for the display of messages (such as "invalid customer name") regarding the semantics of the system (i.e., the name is syntactically correct but not valid in the database).

Finally, descriptors, specified within two double quotes ("), are assigned to all non-terminal symbols and the special terminal symbol DISPLAY. This step

```
BOOK-ORDER-SYSTEM :==
      NEW-ORDER" " | CANCEL-BOOKORDER" " | RETURN-BOOK" "
NEW-ORDER :== (GET-C-D" " + GET-B-D" ") | (GET-B-D" " + GET-C-D" ")
CANCEL-BACK-ORDER :== GET-B-ON" "
RETURN-BOOK :== GET-R-ON" " + GET-R-BT" " + GET-R-BQ" "
GET-C-D :== GET-C-N" " + GET-C-A" "
GET-B-D :== (GET-B-T" " + GET-B-A" " + GET-B-Q" ")*
GET-C-N :== CUST-NAME | (CUST-NAME + DISPLAY + CUST-NAME)*
GET-C-A :== CUST-ADDRESS | (CUST-ADDRESS + DISPLAY + CUST-ADDRESS)*
GET-B-T :== BOOK-TITLE
GET-B-A :== BOOK-AUTHOR
GET-B-Q :== BOOK-QUANTITY
GET-B-ON :== ORDER-NUMBER
GET-R-ON :== ORDER-NUMBER
GET-R-BT :== BOOK-TITLE
GET-R-BQ :== BOOK-QUANTITY
CUST-NAME :== (A|B| . . . |Z|a|b| . . . |z|.)*
CUST-ADDRESS :== (0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| .)*
BOOK-TITLE :== ((0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| . |%|$|&|/|)*
BOOK-AUTHOR :== (A|B| . . . |Z|a|b| . . . |z| .)*
BOOK-QUANTITY :== (0|1| . . . 9)*
ORDER-NUMBER :== (0|1| . . . 9)*
```

Figure 9. The AUI-BNF Model for the Book-Ordering System

enables the AUI-BNF model to support different forms of dialogues (discussed in the next section).

3.4 Transformation into Different Dialogue Forms

The AUI-BNF model, as the dialogue schema for an application's user interface, can be used to develop three types of user interfaces commonly used in business: formal language interface (for expert users), menu/form interface (for frequent users), and question/answer interface (for infrequent, novice users). The dialogue schema can also be used to develop online help and error messages as feedback to the user.

3.4.1 Formal language dialogues

The AUI-BNF model can be used to develop a formal language because it is a context free grammar. The parsing technique for a context free grammar has been studied in-depth in the area of formal languages and will not be further discussed here. A formal language such as a command language is considered a better tool for an expert user because of its efficiency in the user's entering language statements.

In the AUI-BNF model, online help messages are created for all non-terminal symbols by identifying symbols, terminal or non-terminal, available on the right side of the production rule. Similarly, error messages are generated for errors committed by the user during the course of interaction. Messages related to semantics are generated by displaying the descriptor of the terminal symbol, DISPLAY. Figure 10 shows a possible command language and the handling of online help and error messages for the book-ordering system.

```
Command:
   ORDER-SYSTEM/NEW-ORDER/
      GET-C-D=(NAME=. . . ,ADDRESS=. . . )+
      GET-B-D=(BOOK-TITLE=. . . , BOOK-AUTHOR=. . . , BOOK-QUANTITY=. . . )

Help:                          Response:
   HELP ORDER-SYSTEM              Commands in ORDER-SYSTEM are
                                  NEW-ORDER, CANCEL-ORDER, RETURN-BOOK
   HELP BOOK-QUANTITY             QUANTITY must be digits (1, 2, . . ., 9, 0)

Error:                         Response:
   ORDER-SYTEN/. . .              ORDER-SYSTEN/. . .
                                          ^
                                    Incorrect syntax
```

Figure 10. A Possible Command Language for the Book-Ordering System

3.4.2 Menu/form dialogues

To use the AUI-BNF model to generate menu/form dialogues, the following two heuristics are applied:

Heuristic 1: for non-terminal symbols that are produced purely by the OR operator, a *menu* can be created. The name of the menu is the name of the non-terminal symbol on the left side of the production rule. The options of the menu are the symbols on the right side of the production rule. A number is assigned to the options in ascending order. The descriptors can be used in the menu to explain the meaning of the option. Two modifications regarding the semantics of this menu are made here. First, a QUIT option, although not defined in the production rule, is assumed. Second, a repeating structure (with a * symbol) is also assumed so that the user can stay in this menu unless the QUIT option is selected. Figure 11 illustrates the use of Heuristic 1.

```
Production Rule:
BOOK-ORDER-SYSTEM :==
   NEW-ORDER" " | CANCEL-BACKORDER" " | RETURN-BOOK" "

Possible menu structure:
                                   BOOK-ORDER-SYSTEM
                                1. NEW-ORDER" "
                                2. CANCEL-BACKORDER" "
                                3. RETURN-BOOK" "
                                4. QUIT
The production rule assumed implicity:
BOOK-ORDER-SYSTEM :==
   (NEW-ORDER" " | CANCEL-BACK-ORDER" "| RETURN-BOOK" ")*
```

Figure 11. Using Heuristic 1 to Generate Menu Dialogues

Heuristic 2: for (1) non-terminal symbols t
hat are produced by the AND operator and (2) on the right side of the rule, non-terminal symbols can be further produced by terminal symbols that represent data items of the same object, a *form* consisting of fields can be created. The name of the form is the name of the non-terminal symbol on the left side of the production rule. The text labels of fields are the non-terminal symbols which are produced directly from the terminal symbols. The descriptors can be used to explain the meaning of the labels. The length of the field is extracted from the definition stored in the data dictionary (if available) or given by the designer. Figure 12 illustrates the use of Heuristic 2.

```
Production Rules:
   GET-C-D :== GET-C-N" " + GET-C-A" "
   GET-C-N :== CUST-NAME | (CUST-NAME + DISPLAY + CUST-NAME)*
   GET-C-A :== CUST-ADDRESS | (CUST-ADDRESS + DISPLAY + CUST-ADDRESS)*
   CUST-NAME :== (A|B| . . . |Z|a|b| . . . |z| . )*
   CUST-ADDRESS :== (0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| . )*
Form:
                                   GET-C-D
   CUST-NAME "" : _____
   CUST-ADDRESS "":_____
```

Figure 12. Using Heuristic 2 to Generate Form Dialogues

It is also possible to generate a hierarchy of forms based on this heuristic. Using the following production rule:

NEW-ORDER "" :== (GET-C-D"" + GET-B-D"") |
(GET-B-D"" + GET-C-D"")

the OR operator in this rule is used to show that GET-C-D and GET-B-D need to be performed although the order is immaterial. Because both GET-C-D and GET-B-D can become forms (using heuristic 2), the above NEW-ORDER production rule can then be used to generate a form which consists of two other forms (Figure 13). Note that because GET-B-D has a repeating definition in the production rule, the user can stay in this form until a signal (e.g., a function key) is issued by the user to stop this form.

From the implementation point of view, the windowing technique can be used to implement the menus and forms (see Section 4). A window is assigned to a menu or a form, and a window consisting of sub-windows can be assigned to a hierarchy of forms.

Finally, online help regarding the syntax can be handled by typing the token "?" and, in response, the system will generate help messages regarding the syntax requirements. This is a standard feature assumed for all menus and forms in the AUI-BNF model although not specified explicitly. If additional messages are to be provided to the user, an indicator to an external file can be assigned to the production rule. Online error messages regarding

Production Rules:

```
NEW-ORDER" " :== (GET-C-D" " + GET-B-D" ") | (GET-B-D" " + GET-C-D" ")
GET-C-D :== GET-C-N" " + GET-C-A" "
GET-C-N :== CUST-NAME | (CUST-NAME + DISPLAY + CUST-NAME)*
GET-C-A :== CUST-ADDRESS | (CUST-ADDRESS + DISPLAY + CUST-ADDRESS)*
GET-B-D :== (GET-B-T" " + GET-B-A" " + GET-B-Q" ")*
GET-B-T :== BOOK-TITLE
GET-B-A :== BOOK-AUTHOR
GET-B-Q :== BOOK-QUANTITY
CUST-NAME :== (A|B| . . . |Z|a|b| . . . |z| . )*
CUST-ADDRESS :== (0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| . )*
BOOK-TITLE :== ((0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| . |%|$|&|/|)*
BOOK-AUTHOR :== (A|B| . . . |Z|a|b| . . . |z| . )*
BOOK-QUANTITY :== (0|1| . . . 9)*
```

Forms:

```
                    GET-C-D
NAME "" : _____________________
ADDRESS "": _____________________________

                    GET-B-D
BOOK-TITLE "": _____________________
BOOK-AUTHOR "": _____________________________
BOOK-QUANTITY "": _____________
```

Figure 13. A Hierarchy of Forms Generated by a Nested Production Rule

the syntax can be handled in the same way as a formal language. Online help and error messages regarding use of the menu/form technique should be a part of the software tool's functionality and therefore are not a concern here.

3.4.3 Question/answer dialogues

To generate question/answer dialogues from the AUI-BNF model, a top-down, sequential execution of the model is performed. A question can be (1) one that requires the user to select from available options (e.g., options of a menu, referred as type 1 question) or (2) one that requires the user to enter the value for a data item (e.g., fields of a form, referred to as type 2 question). The above two heuristics are modified for generation of question/answer dialogues.

Heuristic 3: for non-terminal symbols that are produced purely by the OR operator, a *type 1 question* can be created. The question is derived by using symbols on the right side of the rule as the available options. Also, the two modifications made in Heuristic 1 are made here. Figure 14 illustrates the use of Heuristic 3.

```
Production rule:
BOOK-ORDER-SYSTEM :==
    NEW-ORDER" " | CANCEL-BACK-ORDER" "| RETURN-BOOK" "

Possible question format:
    ORDER-SYSTEM
    Choose from   (1)  NEW-ORDER" "
                  (2)  CANCEL-ORDER" "
                  (3)  RETURN-BOOK" "
                  (4)  QUIT

    Answer: __
```

Figure 14. Using Heuristic 3 to Generate Questions for Options

Heuristic 4: for (1) non-terminal symbols that are produced by the AND operator and (2) on the right side of the rule, non-terminal symbols can be further produced by terminal symbols that represent data items of the same object, a series of *type 2 questions* are created for each of those non-terminal symbols produced directly by terminal symbols. The generation of a question occurs by simply placing the non-terminal symbol and its descriptor following the word "enter." Figure 15 illustrates the use of Heuristic 4.

Production Rules:

```
GET-C-D :== GET-C-N" " + GET-C-A" "
GET-C-N :== CUST-NAME | (CUST-NAME + DISPLAY + CUST-NAME)*
GET-C-A :== CUST-ADDRESS | (CUST-ADDRESS + DISPLAY + CUST-ADDRESS)*
CUST-NAME :== (A|B| . . . |Z|a|b| . . . |z| . )*
CUST-ADDRESS :== (0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| . )*
```

Possible question format:

```
GET-C-D:
Enter NAME ""?
Answer: _____________
Enter Address ""?
Answer: _____________
```

Figure 15. Using Heuristic 4 to Generate Questions for Data Entry

When a repeating definition is used in the production rule, a question such as "more (yes/no)?" is automatically placed at the end of the series of

questions (Figure 16). Finally, online help and error messages can be handled as they are for menu-form dialogues.

```
Production rules:

NEW-ORDER* " :== (GET-C-D* " + GET-B-D* ") | (GET-B-D* " + GET-C-D* ")
GET-C-D :== GET-C-N* " + GET-C-A* "
GET-C-N :== CUST-NAME | (CUST-NAME + DISPLAY + CUST-NAME)*
GET-C-A :== CUST-ADDRESS | (CUST-ADDRESS + DISPLAY + CUST-ADDRESS)*
GET-B-D :== (GET-B-T* " + GET-B-A* " + GET-B-Q* ")*
GET-B-T :== BOOK-TITLE
GET-B-A :== BOOK-AUTHOR
GET-B-Q :== BOOK-QUANTITY
CUST-NAME :== (A|B| . . . |Z|a|b| . . . |z|.)*
CUST-ADDRESS :== (0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| .)*
BOOK-TITLE :== ((0|1| . . |9|A|B| . . . |Z|a|b| . . . |z| . |%|$|&|/|)*
BOOK-AUTHOR :== (A|B| . . . |Z|a|b| . . . |z| .)*
BOOK-QUANTITY :== (0|1| . . . 9)*

Possible questions format:
   GET-C-D:
   Enter NAME* "?
   Answer: _____
   Enter ADDRESS* "?
   Answer? _____

   GET-B-D:
   Enter BOOK-TITLE* "?
   Answer: _____
   Enter BOOK-AUTHOR* "?
   Answer: _____
   Enter BOOK-QUANTITY* "?
   Answer: _____
   More (Yes or NO): __
```

Figure 16. Using Heuristic 4 to Generate Questions for Repeated Data Entry

4. AUI SOFTWARE TOOLS

The present research investigates an integrated software tool environment, referred to as the User Interface Management System (Figure 17), to facilitate implementation. The input to the UIMS is the BNF model of an application system. The UIMS has several software tool components for handling different dialogue forms. Prototypes of several UIMS software tools have been implemented in a related research project (Kuo and Konsynski 1987).

For formal language dialogues, the UIMS component is a BNF parser (like YACC in the Unix environment). This parser can provide buffers for all strings entered by the user so that, in case of errors, the user can correct the input string without reentering the entire string.

For menu-form dialogues, the UIMS component is more complex. In the AUI methodology, the windowing technique is chosen for handling menus and forms. Two parts of this component are required: the window specification tool and the window manipulation tool.

The windowing specification tool can automatically create the window for a menu or a form (e.g., assigning the location and its layout) according to the syntactic and lexical definitions specified in the BNF model (Gait 1985). This tool also allows the designer to modify the window if desired.

The window manipulation tool is invoked, during the run time, to generate windows and facilitate user-system interaction. This tool accepts user input and provides temporary buffers for the input. The buffers are passed to the application system when the interaction is successfully completed. A log file can be created to record the user input so that he/she can later be shown what occurred during the session. This log file may also be used for the user to "undo" certain dialogues. Errors, when made by the user, can be displayed using a separate window. To increase keystroke efficiency, function keys can be used. A HELP key is used for reviewing help messages, including those for the syntax and for how to use the window manipulation tool. A DO key allows the user to send a signal to the system for successful task completion. Other commonly used function keys such as the EXIT and ABORT keys can also be provided. In the AUI methodology, the use of function keys is considered a part of the software tool's functionality and should be standardized and made available to all users of all applications. Thus, the designer need not be concerned with the use of function keys in the model.

For question/answer dialogues, the UIMS component is a software tool that sequentially delivers the questions and answers according to the dialogue schema. This tool also provides a buffering facility like the window manipulation tool for the user to "undo" dialogues.

4.1 Adapting to Different Users

An important feature of the UIMS identified at present is that the three forms of dialogues must be interchangeable upon the user's request. To do so, it is planned to use a MODE key to allow the user to change mode of interaction at any time. To do so, the user's input, regardless of the form of dialogues, is converted internally to the BNF

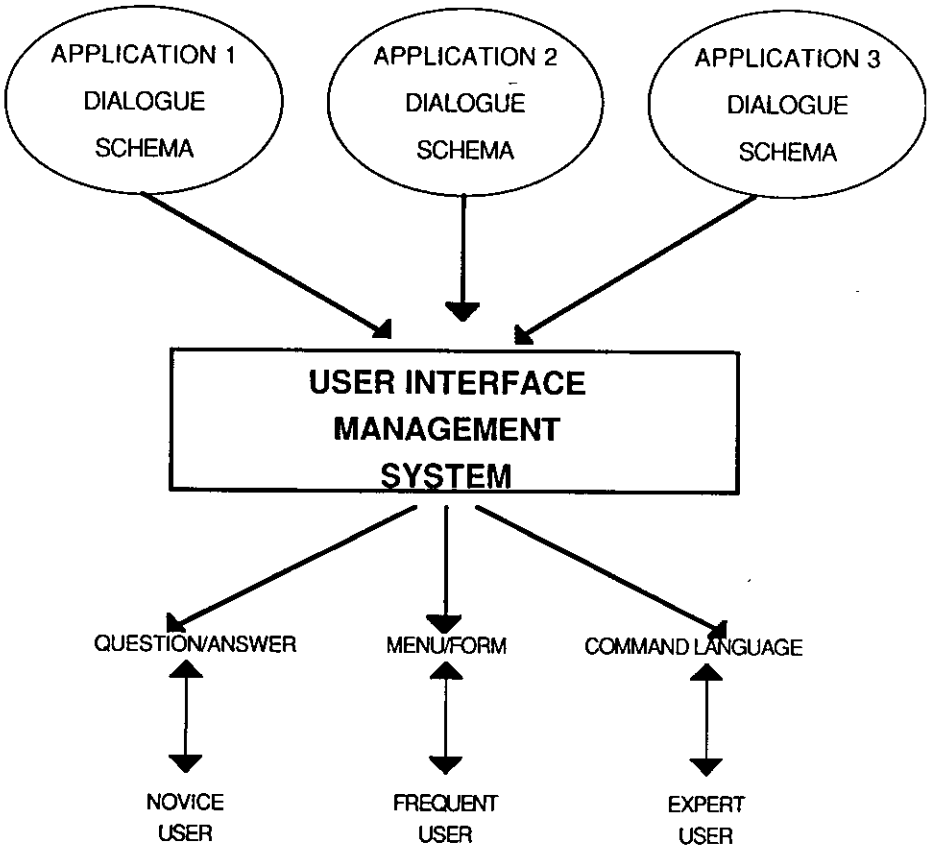

Figure 17. The AUI Software Tool Environment

representation. Thus, the user can switch to the menu/form mode for data entry after entering a command string such as ORDER-SYSTEM/NEW-ORDER. By the same token, a novice user can change from question/answer mode to menu/form mode when he/she becomes more experienced. The user need not know how the three modes are handled internally.

4.2 Dialogue Independence and Virtual Interface Handler

The UIMS, like the DBMS which hides the physical storage structure from the designer in data management, must hide from the designer in dialogue management the complexity encountered in the implementation of interface design (such as parsing, displaying screen, and handling user keystroke). This objective can be achieved by allowing the UIMS to support various programming languages and simple, high-level function calls to the UIMS to facilitate development of application software.

It is important to point out that dialogue independence, which refers to the separation of definition (i.e., the BNF model) from the manipulation of dialogues, can therefore be achieved. The UIMS tool can be shared by various applications and the design and implementation of the user-system interface can be simplified due to dialogue independence.

Furthermore, the UIMS can be designed to support implementation on many incompatible hardware environments (e.g., workstation, terminal). The architecture of the UIMS is divided into many layers while only the lowest-layer software is hardware dependent (Kuo and Konsynski 1987). The UIMS can

invoke appropriate lowest-level software for each different hardware environment where the interface must be supported. Thus, the UIMS can virtually support user interface manipulation for all application systems on all available hardware environments. This is referred to as the virtual interface handler concept in the AUI methodology.

4.3 The Model Generation Tool and Iterative Design

For a methodology to support adaptive user interface design, the methodology itself must be adaptive (i.e., respond to changes of requirements as soon as possible). In this regard, the AUI methodology is adaptive because changes of requirements (in the DFD, events, or event-flows) can be immediately reflected by the user-system interface model. Furthermore, many modeling steps in the AUI methodology can be automated (e.g., deriving the STD from the ADFD and performing a consistency check). The present research is studying this software tool (Figure 18). The idea is to have this tool accept the DFD and descriptions of events and event-flows to help the designer developing the user interface model. The model, when developed, can be directly used by the UIMS for implementation. Consequently, the AUI methodology can more effectively facilitate iterative design of user interfaces.

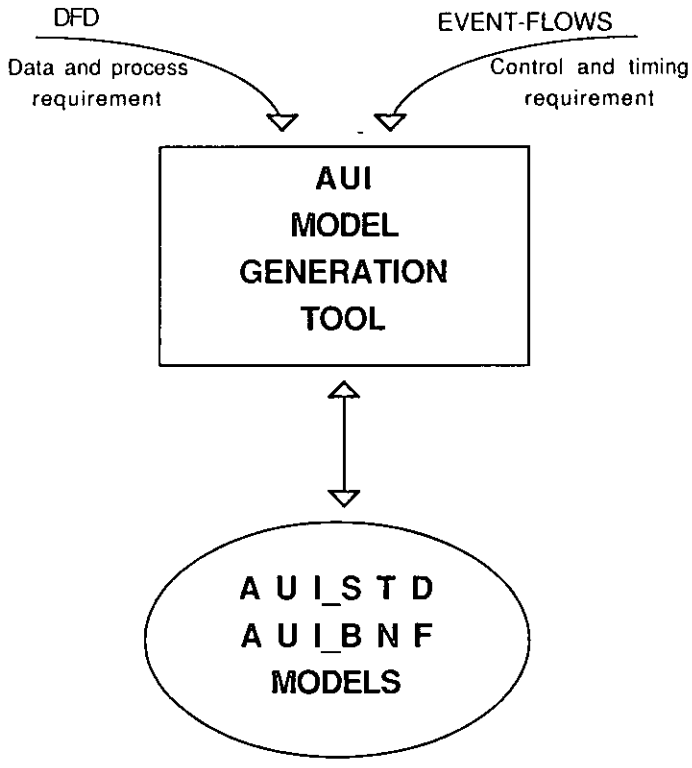

Figure 18. The AUI Model Generation Tool Environment

4.4 Using the AUI Software Tools without DFD

The AUI methodology, although assuming the DFD of a system as its input, can be used independently. The modeling tool described above can be modified to accept data and process requirements, events, and event-flows from the designer. Thus, the AUI methodology can be used with many automated structured design methodologies such as CAPO (Karimi 1986).

5. CONCLUSIONS AND FUTURE RESEARCH

In conclusion, this paper presents a methodology for designing user interface based on a system's requirements embedded in the Data Flow Diagram. Three styles of user interface can be generated. The roles of modeling the user-system interfaces in this methodology are useful to dialogue management, like Abstraction (Smith and Smith 1977) and Entity-Relationship (Chen 1976) approaches to the conceptual modeling in data management. Many advantages can be attained by using this methodology:

1. It can be easily integrated with and take advantage of many existing structured design methodologies. The user-system interface design therefore becomes an integral part of the system development life cycle. The overall effort to develop the interface can also be reduced.

2. The methodology offers clearly defined steps for conceptual design of the user interface (i.e., modeling the user-system interface from both the real-time system and the formal language perspective). Other approaches such as USE (Wasserman et al. 1986), FLAIR (Wong and Reid 1982), and SCREEN RIGEL (Rowe and Shoens 1983) have not addressed this issue.

3. Using a technique such as BNF enables the incorporation of human factors into the design evaluation. More important is that the dialogue schema of various applications can be evaluated

to further ensure consistent user interface across application systems. This evaluation can be performed on the use of terminal and non-terminal symbols and the production rules for each dialogue schema. If, for instance, two applications refer to the same data item using different terminologies (i.e., inconsistent non-terminal symbols), the evaluation will reveal this inconsistency. Inconsistent production rules regarding the same symbol can also be detected by comparing the symbols and the operators used on the right side of the rule. Consistent user-system interfaces across applications can reduce the user's learning effort for those applications.

Two directions are planned for future research. The first direction is to investigate the integration and sharing of dialogue schema for various applications. "System-initiated" adaptation, which refers to the system's ability to dynamically identify and adjust the user interface to the user's skill level, will also be studied.

The second direction is to study which various human factors, cognitive or behavioral, are relevant and how they can be incorporated into various stages of the methodology. The AUI methodology in its present form does not address issues such as the user's mental model construction for task analysis, memory capacity implication of alternative syntax structure, and ergonomics in the human-computer interaction when different interface styles are selected. Those issues will be carefully evaluated in order to improve the AUI methodology.

REFERENCES

Birrell, N. D., and Ould, M. A. *A Practical Handbook for Software Development.* Cambridge University Press, London, 1985.

Chen, P. "The Entity-Relationship Model-Toward a Unified View of Data." *ACM Transactions on Database Systems,* Vol. 1, No. 1, 1976, pp. 9-36.

Dehning, W.; Essig, H.; and Maass, S. *The Adaptation of Virtual Man-Computer Interfaces to User Requirements in Dialogs.* Springer-Verlag, Berlin Heidelberg, New York, 1981.

Edmonds, E. "Adaptable Man/Machine Interfaces for Complex Dialogues." *Proceedings of the European Computing Congress,* London, 1978, pp. 639-646.

Edmonds, E. A. "Adaptive Man-Computer Interfaces." In M. J. Coombs and J. L. Alty (eds.), *Computing Skills and The User Interface,* Academic Press, London, 1981, pp. 389-426.

Hall, P. V. "Man-Computer Dialogues for Many Levels of Competence." *Proceedings of the European Computing Congress,* London, 1978.

Gane, C., and Sarson, T. *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, Prentice-Hall, New Jersey, 1979.

Gait, J. "An Aspect of Aesthetics in Human-Computer Communications: Pretty Windows." *IEEE Transactions on Software Engineering,* Vol. SE-11, No. 8, 1985, pp. 714-717.

Jacob, R. "Using Formal Specifications in the Design of a Human-Computer Interface." *Communications of the ACM,* Vol 26, No. 4, April 1983, pp. 259-264.

Karimi, J. "An Automated Software Design Methodology Using CAPO." *Journal of MIS,* Vol. 3, No. 3, 1986, pp. 71-99.

Kuo, F. Y., and Konsynski, B. "Dialogue Management: Support for Dialogue Independence." *MIS Quarterly,* Vol. 11, No. 4, December 1987.

Reisner, P. "Formal Grammar and Human Factors Design of an Interactive Graphics System." *IEEE Transactions on Software Engineering,* Vol. SE-7, No. 2, 1981, pp. 229-240.

Reisner, P. "Formal Grammar as a Tool for Analyzing Ease of Use: Some Fundamental Concepts." In J. C. Thomas and M. Schneider (eds.), *Human Factors in Computing Systems,* Ablex, Norwood, NJ, 1984, pp. 53-78.

Rowe, L., and Shoens K. "Programming Language Constructs for Screen Definition." *IEEE Transactions on Software Engineering,* Vol. SE-9, No. 1, 1983, pp. 31-39.

Smith, J., and Smith, D. "Database Abstractions: Aggregation and Generalization." *ACM Transactions on Database Systems,* Vol. 2, No. 2, 1977, pp. 105-133.

Ward, P. "The Transformation Schema: An Extension of the Data Flow Diagram to Represent

Control and Timing." *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, 1986, pp. 198-210.

Wasserman, A. "Extending State Transition Diagrams for the Specification of Human-Computer Interaction." *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 8., 1985, pp. 699-713.

Wasserman, A.; Shewmake, D.; Pircher, P.; and Kersten, M. "Developing Interactive Information Systems with the User Software Engineering Methodology." *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, 1986, pp. 326-345.

Wong, P., and Reid E. "FLAIR -- User Interface Dialogue Design Tool." *Computer Graphics*, Vol. 16, No. 3, July 1982, pp. 87-98.

Woods, W. A. "Transition Network Grammars for Natural Language Analysis." *Communications of the ACM*, Vol. 13, No. 10, October 1970, pp. 591-606.