An Interactive Environment for Dialogue Development: Its Design, Use, and Evaluation

-Or-

Is Aide Useful?

by

Deborah Hix Johnson
H. Rex Hartson

TR-85-35

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

September, 1985

Submitted to:
CHI '86 Conference on Human-Computer Interaction; Boston, April 1986

AN INTERACTIVE ENVIRONMENT FOR DIALOGUE DEVELOPMENT: ITS DESIGN, USE, AND EVALUATION *- OR -* *IS AIDE USEFUL?*

by

Deborah Hix Johnson
H. Rex Hartson

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
703/961-4857

Abstract: The *Author's Interactive Dialogue Environment (AIDE)* of the Dialogue Management System is an integrated set of direct manipulation tools used by a dialogue author to design and implement human-computer interfaces without writing source code. This paper presents the conceptual dialogue transaction model upon which AIDE is based, describes AIDE, and illustrates how a dialogue author develops an interface using AIDE. A preliminary empirical evaluation of the use of AIDE versus the use of a programming language to implement an interface shows very encouraging results.

Either author can be contacted for correspondence at the address above.

AN INTERACTIVE ENVIRONMENT FOR DIALOGUE DEVELOPMENT: ITS DESIGN, USE, AND EVALUATION
- OR -
IS AIDE USEFUL?

by

Deborah Hix Johnson
H. Rex Hartson

1. *INTRODUCTION AND BACKGROUND*

User interface management systems (UIMS) are attracting considerable attention from researchers and practitioners as a viable approach to producing quality human-computer interfaces (e.g., Buxton, 1983; Hayes, 1985; Kamran and Feldman, 1983; Kasik, 1982; Olsen and Dempsey, 1983; Wasserman, 1983). However, many UIMS emphasize the execution-time aspects of interface management. Only a few UIMS have interface design tools and even fewer have been empirically evaluated. The *Author's Interactive Dialogue Environment (AIDE)*, the dialogue development environment of the Dialogue Management System, is an exception. AIDE — its conceptual model, its use, and its empirical evaluation — is the subject of this paper.

The contextual setting for AIDE is the Dialogue Management System (DMS), being constructed at Virginia Tech. DMS is a comprehensive system for designing, implementing, testing, and maintaining software systems with quality human-computer interfaces (Hartson, Johnson, and Ehrich, 1984). Interface development in DMS is treated not as an "add-on" activity but rather as an integral part of the entire software engineering process. This is accomplished through a methodological approach which gives dialogue design emphasis equal to that of computational design. The only other approach known to the authors which relates interface development to software engineering is found in RAPID/USE (Wasserman and Shewmake, 1985).

In DMS at design-time, an interactive software system is separated into three components: the dialogue component, the computational component, and the global control component (which governs the sequencing among the dialogue events and computational events). This separation supports the concept of *dialogue independence* (Ehrich and Hartson, 1981), pioneered as a fundamental premise of DMS. Dialogue independence ensures the clear delineation of the dialogue component design from the design of the other two components of an application system. To emphasize this separation, individuals in different roles are responsible for each of the three components. DMS has a *Design-Time Facility* which provides different sets of interactive tools, one for each role to use in developing its appropriate component. In DMS, a *dialogue author* (Ehrich and Hartson, 1981) has sole responsibility for developing the dialogue component. The DMS Design-Time Facility provides AIDE for the dialogue author to use in designing and implementing the dialogue component *without writing source code*.

2. *A MODEL OF HUMAN-COMPUTER INTERACTION*

2.1 Formulation of the Model

The *dialogue transaction model* of DMS (Johnson, 1985) was developed to be used as the framework for the development of AIDE and to guide the dialogue author in using AIDE to produce interfaces. Few models, either theoretical or practical, exist to guide

human-computer interface design. Notable exceptions include dialogue cells (Borufka, Kuhlman, and ten Hagen, 1982) and interaction events (Benbasat and Wand, 1984). Few, if any, UIMS are based on such models, without which the components of an interface can be an overwhelming collection of unrelated prompts, displays, messages, expressions, inputs, and devices. The development of the dialogue transaction model was based on formalisation of a theory of human-computer interaction (Johnson, 1985). This theory grew out of extensive observations of humans using a wide variety of interface styles, techniques, and devices. The model is more comprehensive than what is presented below; only portions necessary for an understanding of this paper are given here.

2.2 Description of the Dialogue Transaction Model

In DMS, the dialogue component of an interactive application system is comprised of dialogue units called *dialogue transactions*. There are two types of transactions: output and input. A *dialogue output transaction* is a non-interactive transaction which presents the results of computational processing (e.g., results of a database search) to the end-user; it will not be further discussed in this paper. A *dialogue input transaction* is a sequence of one or more interactions to extract from the end-user a set of linguistically (grammatically) related tokens — essentially a single complete command — as shown in Figure 1. An interaction, one for each token, is comprised of three ordered parts:

<computer prompt, human token input, computer confirmation>

A *prompt* can consist of any type of display, including such pieces as menus, labelled function key outlines, text, graphical objects, forms to be filled in, spoken words from a voice synthesizer, as well as combinations of these. The function of a prompt at run-time is to request an end-user input. Correspondingly, a token *input* definition can be comprised of pieces defined for menu selection, function key selection, mouse selection, command string input, forms completion, or voice input, any or all of which may be active simultaneously. The input part is the most complicated part of an interaction; its function at run-time is to accept, linguistically process, and (at least syntactically) validate all end-user inputs. A *confirmation* gives the end-user information about the syntactic validity of the input. Confirmations can be positive (reinforcing feedback) or negative (error messages), and can be comprised of such pieces as text, graphics, and voice. Positive confirmations are often implicit (null). To show how an interface is decomposed into elements of the model, a simple dialogue from a hypothetical information system for indexing documents is given in Figure 2. Development of a portion of this dialogue, using AIDE, is presented in the next section.

3. THE AUTHOR'S INTERACTIVE DIALOGUE ENVIRONMENT (AIDE)

3.1 Motivation for AIDE

AIDE is a set of direct manipulation tools in DMS, used by the dialogue author to design, implement, and modify dialogues without writing source code. Most interfaces are composed of objects that belong to identifiable classes of communication forms

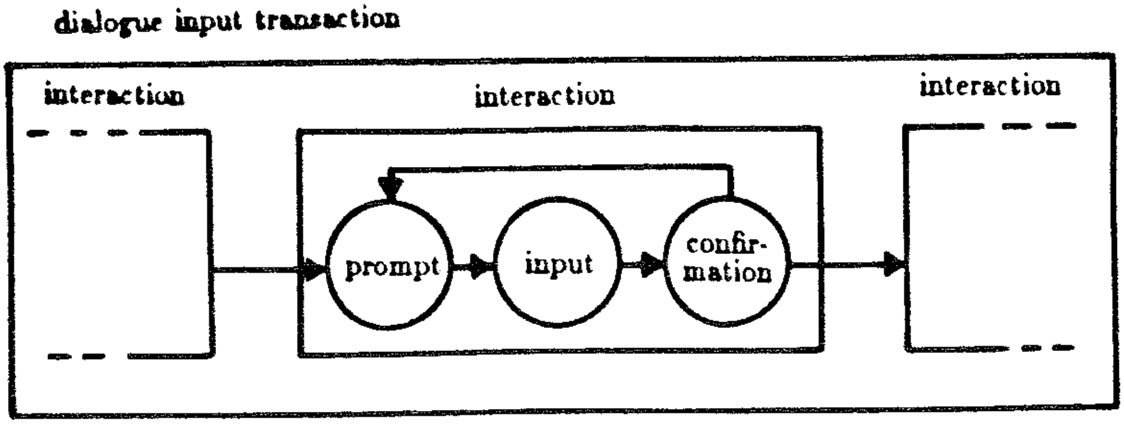
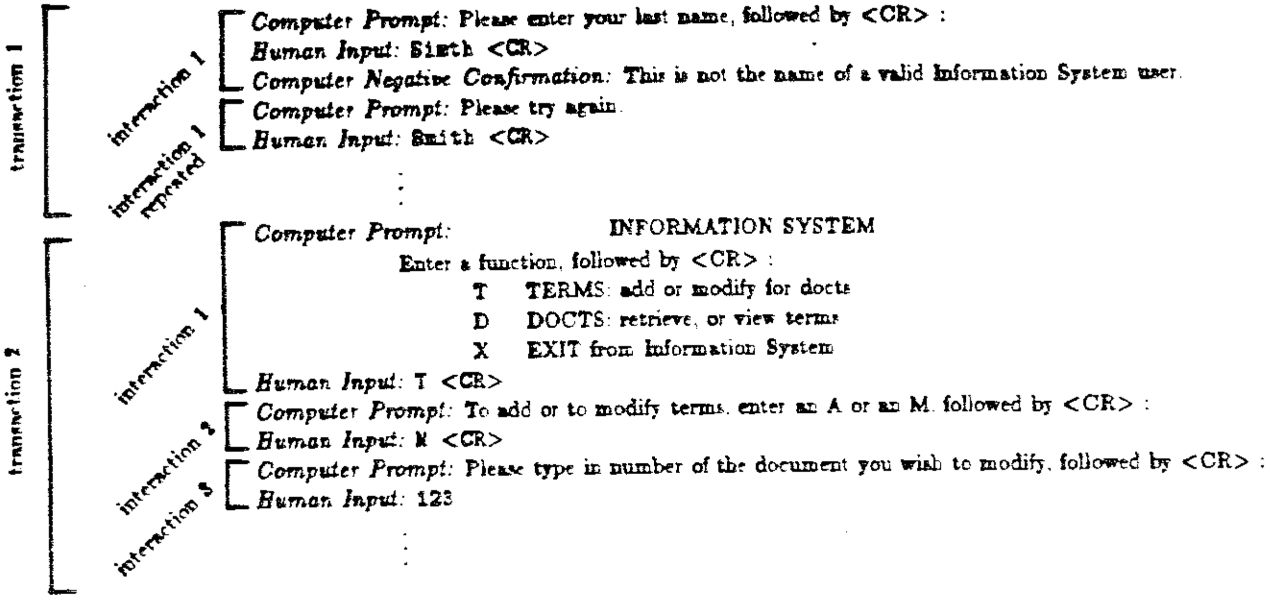


Figure 1. *Model of a Dialogue Input Transaction.*



```
Computer Prompt: Please enter your last name, followed by <CR> :
Human Input: Simth <CR>
Computer Negative Confirmation: This is not the name of a valid Information System user.
Computer Prompt: Please try again.
Human Input: Smith <CR>
        :
Computer Prompt:                    INFORMATION SYSTEM
                 Enter a function, followed by <CR> :
                          T    TERMS: add or modify for docts
                          D    DOCTS: retrieve, or view terms
                          X    EXIT from Information System
Human Input: T <CR>
Computer Prompt: To add or to modify terms, enter an A or an M, followed by <CR> :
Human Input: M <CR>
Computer Prompt: Please type in number of the document you wish to modify, followed by <CR> :
Human Input: 123
        :
```

Figure 2. *A Simple Dialogue Decomposed into Parts of the Model.*

(e.g., typing command strings, selecting from menus, filling in forms, picking graphical objects). Rather than having an interface designer repeatedly code the same type of interface object (e.g., a menu), AIDE provides interactive tools for developing each of them. Thus, AIDE allows the dialogue author to concentrate on dialogue form and content, rather than on debugging source code.

3.2 Relationship of the Dialogue Transaction Model to AIDE

AIDE contains tools to develop each dialogue element (transaction, interaction, prompt, input, and confirmation) in the model. Navigation by the dialogue author within AIDE is guided by the relationships among model elements. For example, the author can easily move around within AIDE to develop the parts (prompt, input, and confirmation) of an interaction, or to work on another interaction within the current transaction, or to work on a different transaction.

3.3 Description of AIDE Version 1

AIDE Tools and Interface. **The first version of AIDE can produce prompts consisting of combinations of menus, keypads, forms, text, and graphics. Inputs can be defined for menus, keypads, and forms. Confirmations can be textual and graphical.**

Each element of the dialogue transaction model has a corresponding set of logical definitions for the programmed function keys. At any time during AIDE use, the currently active key functions are shown in a labelled keypad outline on the AIDE *command screen*. A function may be selected either by pressing a key on the auxiliary keypad or by touching its labelled image on the touch-sensitive command screen. All interface parts which the dialogue author creates at design-time appear on a separate color *dialogue screen* as they will be seen by the end-user at application system run-time.

The control structure among the AIDE keypads provides both vertical (e.g., from transaction to interaction to interaction part) and horizontal (e.g., among the parts of an interaction) navigation among dialogue elements. To illustrate, the keypad for developing the prompt part of an interaction has the following functions:

EXIT AIDE
DELETE CURRENT PROMPT
DEVELOP MENU
DEVELOP KEYPAD
DEVELOP TEXT
DEVELOP GRAPHICS
DEVELOP FORMS
DEVELOP ANOTHER INTERACTION PART
RETURN AND DEVELOP ANOTHER INTERACTION
HELP

At this point (i.e., when using AIDE to develop a prompt), the dialogue author may go vertically down the model hierarchy by selecting (for example) DEVELOP MENU (as a piece of a prompt), go vertically up by selecting RETURN AND DEVELOP ANOTHER INTERACTION, or move horizontally in the same level by selecting DEVELOP ANOTHER INTERACTION PART (i.e., an input or a confirmation). AIDE allows the dialogue author to move freely among the dialogue elements in almost any order as they are developed.

Use of AIDE to Develop Dialogue Transactions. When designing and implementing a dialogue transaction, the dialogue author begins by decomposing that transaction into its interactions. For transaction 2 of the information system of Figure 2, this decomposition is partially illustrated in Figure 3b. The dotted lines descending from the transaction, "get valid command", indicate that this transaction decomposes into the interactions (represented by concentric circles) shown below it. Figure 3a shows the corresponding grammar for this transaction. The first interaction, called "get valid command name", will be used here to illustrate use of AIDE. To develop the prompt, the dialogue author moves to the DEVELOP INTERACTION keypad and chooses the DEVELOP PROMPT function key. The keypad whose functions are shown above appears and the author can choose (among other functions) to DEVELOP MENU, DEVELOP KEYPAD, DEVELOP TEXT, DEVELOP GRAPHICS, or DEVELOP FORMS, any or all of which may be part of a prompt. Assuming the prompt for interaction "get valid command name" is to be a menu, the author selects DEVELOP MENU, and the keypad containing the functions for the menu tool appears. This tool allows the author to develop all the fields of the menu, including their content and visual attributes, on the dialogue screen as the end-user will see them at run-time. By choosing the DEVELOP GRAPHICS key, the author can add graphical objects to the same screen. The completed prompt might look as shown in Figure 4a, which is printed directly from the AIDE dialogue screen.

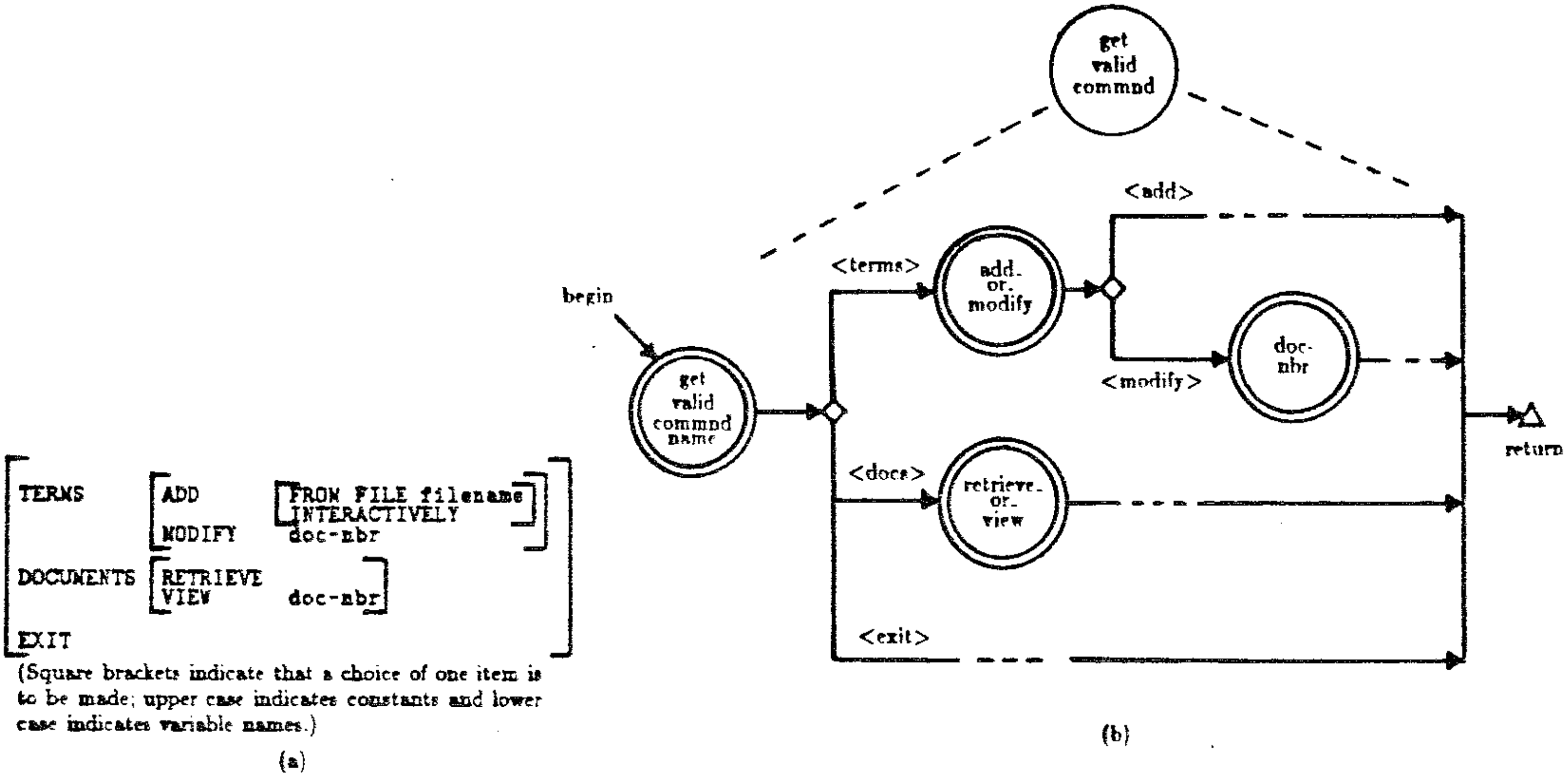
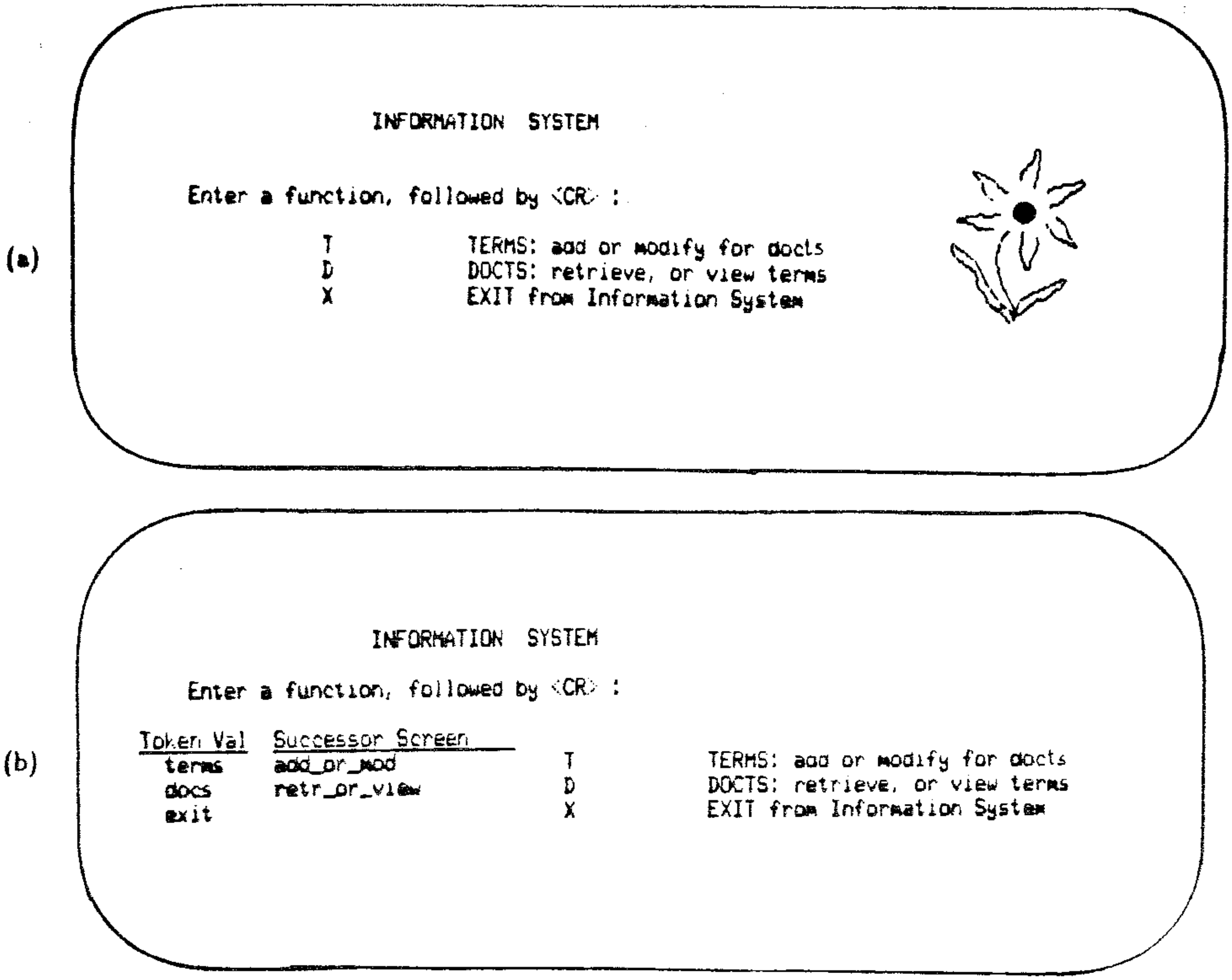


Figure 3. *(a) Grammar and (b) Some Interactions for Sample Transaction "get valid command".*

To define the language input part of the "get valid command name" interaction, the dialogue author leaves the PROMPT keypad by selecting the DEVELOP ANOTHER INTERACTION PART key, and choosing, on the subsequent keypad, DEVELOP INPUT. For a menu, the input definition is relatively simple; it requires two pieces of information for each menu choice offered to the end-user: a *token value* to be returned to the computational component and the *name of the successor interaction* which is to follow the current interaction. The AIDE dialogue screen containing the completed language definition is shown in Figure 4b. (Token value and successor screen columns are not displayed at run-time; they are only for design-time definition of input.) At run-time, if the end-user enters the choice "T", the token value "terms" is returned to the computational component at the end of the transaction. In addition, the successor interaction will be "add_or_modify", as also seen in Figure 3b. For interfaces which use typed text strings, the language input definition is a bit more complicated; space



(a)

```
                    INFORMATION  SYSTEM

      Enter a function, followed by <CR> :

                 T          TERMS: add or modify for docts
                 D          DOCTS: retrieve, or view terms
                 X          EXIT from Information System
```

(b)

```
                    INFORMATION  SYSTEM

      Enter a function, followed by <CR> :

   Token Val  Successor Screen
    terms     add_or_mod            T         TERMS: add or modify for docts
    docs      retr_or_view          D         DOCTS: retrieve, or view terms
    exit                            X         EXIT from Information System
```

Figure 4. *(a) Prompt and (b) Input Definition for Sample Interaction "get valid command name".*

limitations do not allow further discussion. Developing confirmations is very similar to developing a textual or graphical prompt and also will not be detailed here.

4. *AN EMPIRICAL EVALUATION OF AIDE*

This first version of AIDE has undergone testing — not extensive testing, but rather a pilot empirical study to begin determining the usefulness of the dialogue transaction model and of AIDE for developing human-computer interfaces.

4.1 Experimental Hypothesis

The hypothesis of this evaluation was that creation and modification of human-computer interfaces is faster and easier by using AIDE than by using a conventional programming language. The independent variable was the mechanism used for performing the task of developing an interface (i.e., AIDE or a programming language); the dependent variable of primary concern was the length of time it took subjects to complete the task.

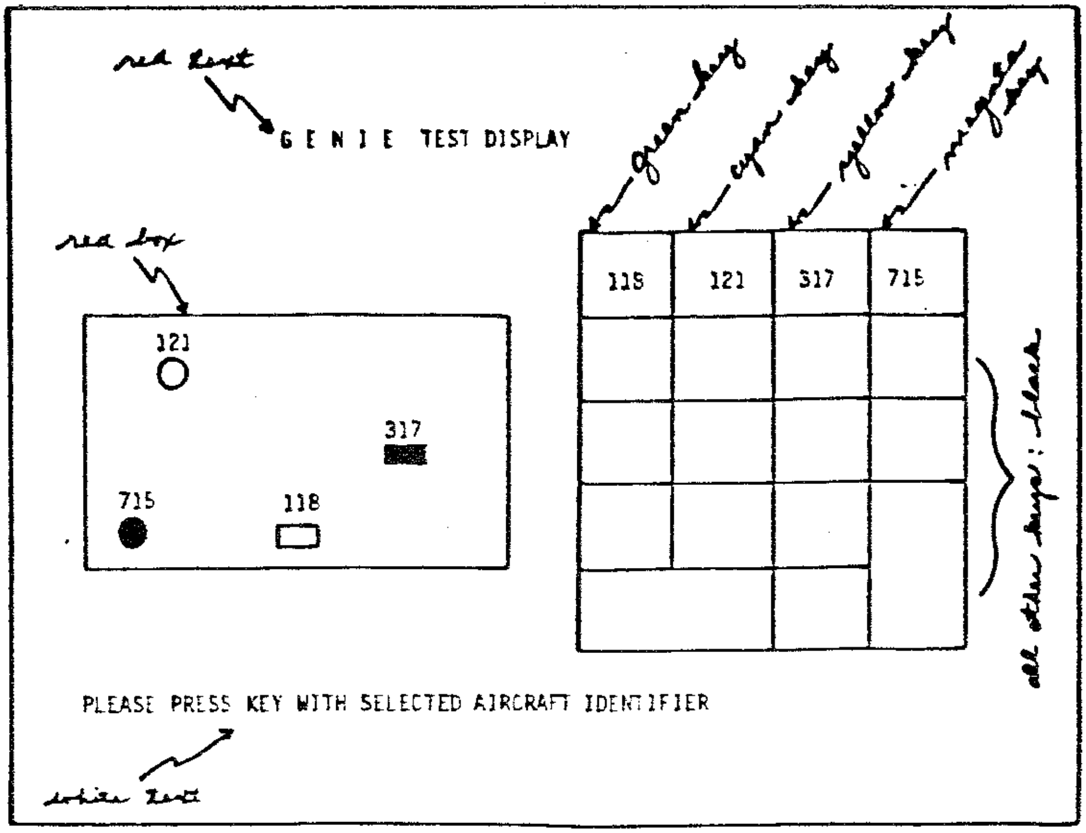


Figure 5. *Display with Keypad Prompt.*

4.2 Experimental Methods

Subjects. Subjects were programmers on the DMS project. Three people served as dialogue authors and three others as application programmers. The small sample size was due to the limited numbers of "expert" AIDE users and "expert" DMS-environment programmers. It was desired that all subjects initially be "experts" at their task so that the issue of training was not a confounding variable.

Experimental Design. The experiment was a two-level between-subjects design with one independent variable. The two levels were the use of AIDE and the use of a programming language. The task was to create a keypad-based interface and then modify that interface to be menu-based. The dialogue author subjects used AIDE. The application programmer subjects used their choice of either Fortran or C, and had the use of some powerful programming services for placement of text and graphics on the screen. The interface design, both for the creation task and the modification task, was completely defined for both groups of subjects. Subjects in both groups were given the same procedural instructions and written interface design task description.

Task Description. Each subject was first asked to create a transaction for a simulated carrier-based air traffic control environment, requiring creation of the graphical computer *prompt,* the human *input,* and the computer *response* parts. The prompt description given to each subject is shown in Figure 5. The system was to accept as input from the end-user only labelled keypad keys, and was to return to the computational component a token value which was the character string labelling the corresponding key (i.e., the aircraft identifier). The computer response (a dialogue output transaction) to the pushing of a keypad key labelled with an aircraft identifier was to make the corresponding aircraft symbol blink.

The modification task was given to each subject upon completion of the creation task. The prompt was the same as that for the creation task, except that the keypad was to be replaced with a menu for aircraft selection; again the subjects were given an exact description of how it was to look. The system was to accept as input only the typing of aircraft identifiers given in the menu, rejecting any other input as soon as it could be determined erroneous. This required character-at-a-time input validation. The computer response to the menu selection of a valid aircraft identifier was to make the corresponding aircraft symbol turn red.

4.3 Results

The mean times, in minutes, for each group of subjects to perform the creation, modification, and total tasks are shown in Table 1. The difference between the two groups was found to be significant for the creation task, $t(4) = 11.9$, $p < .005$; for the modification task, $t(4) = 5.2$, $p < .005$; and therefore for completion of the total task, $t(4) = 9.9$, $p < .005$.

TABLE 1: MEAN TIME TO PERFORM TASKS (IN MINUTES)

	Dialogue Authors	Application Programmers
Creation Task	43	168
Modification Task	29	63 *
TOTAL	72	231

* One application programmer subject became too tired to complete the modification task; that subject's time until quitting the modification task was used in the calculations, since a time to completion for the task was unavailable.

In addition to the quantitative results given above, subjective results were acquired by observing subjects and by post-task questions. Among the programmer subjects, the experimenter observed numerous signs of frustration and tiredness; when asked by the experimenter how it had been, all immediately responded "Tiring!". The author subjects, on the other hand, responded "That's all" and "Fine", and showed no signs of frustration or exhaustion. All programmer subjects said they had the most trouble with positioning objects on the screen and doing the character-at-a-time validation for the menu inputs. Author subjects named no problems other than response time of the database underlying AIDE. Not any of the subjects thought the task was too difficult.

4.4 Interpretation of Results

These results support the hypothesis that creation and modification of an interface is faster and easier by using AIDE than by using a programming language, at least for those types of interfaces AIDE was designed to develop. This is clearly demonstrated in the mean time to complete the tasks by each group of subjects. The dialogue author subjects performed the creation task 3.9 times faster than did the application

programmer subjects, the modification task 2.2 times faster, and the total task 3.2 times faster. The subjective observations made by the experimenter during the tasks and the questioning of subjects after task completion support the hypothesis as well.

This study does not completely address the question of just how these results reflect on the dialogue transaction model, instantiated in the AIDE interface. Presumably a bad model would not have a good instantiation. Thus, since both the design and the use of AIDE is directed by the model and since this study indicates that AIDE performs well, it is can be inferred that the model is a reasonable representation of human-computer interaction. It is also difficult to isolate other significant factors that might affect results of this study, such as segregation of the effects of the model from the obvious advantages of direct manipulation incorporated into the AIDE interface. At the least, this study is an evaluation of an approach to interface design, specifically one using the DMS dialogue transaction model and the tools of AIDE. The results of this study definitely indicate that interactive tools for interface development are worth more research. Future research will further address the validation of this conjecture.

5. CONCLUSIONS AND FUTURE WORK

While the general goal of AIDE is to provide an interactive tool for a dialogue author to use in developing human-computer interfaces, AIDE Version 1 had a specific goal. Its purpose was to prove that the theory and concepts of DMS — dialogue independence, the role of a dialogue author, and interactive tools for the dialogue author — could by implemented. Version 1 was limited in the types of interfaces it could produce. Nonetheless, it has been used to produce several demonstration application systems having two different interfaces (e.g., one which is keypad-based, the other menu-based) with identical functionality. At application system run-time, both these interfaces execute with the same computational component, thus demonstrating that dialogue independence is a viable concept. The evaluation of AIDE discussed above shows very promising results, indicating that a highly diverse group of tools can be integrated into a single, cohesive, usable interactive system for developing interfaces.

AIDE 2.0, with significantly more features, is currently being implemented on a Silicon Graphics IRIS 2400 Workstation. The appearance of the AIDE 2.0 interface is quite different from that described in this paper, making use of the IRIS windowing capabilities and mouse. One goal for development of the second version is to produce an AIDE which can be taken to an industrial test-bed for use in a real world development situation. Many other exciting possibilities for further development and empirical evaluation of DMS, and especially of AIDE, lie ahead.

REFERENCES

Benbasat, I., and Wand, Y. (1984). A structured approach to designing human-computer dialogues. *International Journal of Man-Machine Studies, 21,* 105–126.

Borufka, H. G., Kuhlman, H. W. and ten Hagen, P. J. W. (1982). Dialogue cells: A method for defining interactions. *Computer Graphics and Applications,* 7, 25-33.

Buxton, W., Lamb, M. R., Sherman, D., and Smith, K. C. (1983). Towards a comprehensive user interface management system. *Computer Graphics, 17, 3* 35–42.

Ehrich, R. W., and Hartson, H. R. (1981). DMS – An environment for dialogue management. In *Proceedings of COMPCON81* (p. 121). Washington, DC.

Hartson, H.R., Johnson, D.H., and Ehrich, R.W. (1984). A human-computer Dialogue Management System. In *Proceedings of INTERACT '84: First IFIP Conference on Human-Computer Interaction* (pp. 57-61). London, England.

Hayes, P. (1985). Executable interface definitions using form-based interface abstractions. In *Advances in Human-Computer Interaction.* H. Rex Hartson (Ed.), Ablex Publishing Corporation, 161–190.

Johnson, D.H. (1985). *The structure and development of human-computer interfaces.* Doctoral dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Kamran, A., and Feldman, M. B. (1983). Graphics programming independent of interaction techniques and styles. *Computer Graphics.* 58–64.

Kasik, D. J. (1982). A user interface management system. *Computer Graphics, 16, 3* 99–106.

Olsen, D. R., and Dempsey, E. P. (1983). SYNGRAPH: A graphical user interface generator. *Computer Graphics, 17, 3* 43–50.

Wasserman, A. I. (1983). The unified support environment: Support for the User Software Engineering methodology. In *Information system design methodologies.* A. Verrijn-Stuart (Ed.), North Holland.

Wasserman, A. I. and Shewmake, D. T. (1985). The role of prototypes in the User Software Engineering (USE) methodology. In *Advances in Human-Computer Interaction.* H. Rex Hartson (Ed.), Ablex Publishing Corporation, 191–210.

ACKNOWLEDGEMENTS

The authors wish to thank the members of the DMS team who have so competently and cheerfully participated in the research and implementation efforts presented in this paper, especially Dr. Roger W. Ehrich, Dr. Robert C. Williges, Beverly A. Williges, Jeff Brandenburg, Antonio Siochi, and Dr. Verna Schuetz. This work has been funded by the National Science Foundation, Information Technology Program Grant No. IST-8310414, Dr. H.E. Bamford, Jr., Program Director.