U S L / D B M S N A S A / R E C O N

W O R K I N G P A P E R S E R I E S

Report Number

DBMS.NASA/RECON-24

The USL/DBMS NASA/RECON Working Paper Series contains a collection of reports representing results of activities being conducted by the Center for Advanced Computer Studies of the University of Southwestern Louisiana pursuant to the specifications of National Aeronautics and Space Administration Contract Number NASW-3846. The work on this contract is being performed jointly by the University of Southwestern Louisiana and Southern University.

For more information, contact:

Wayne D. Dominick

Editor
USL/DBMS NASA/RECON Working Paper Series
Center for Advanced Computer Studies
University of Southwestern Louisiana
P. O. Box 44330
Lafayette, Louisiana 70504
(318) 231-6308

A Personal Computer-based Protocols for Interface Prototyping and Evaluation (PC/PIPE) system is proposed. The system will be composed of two main components. The first component will be a set of tools to support the design and implementation of a user interface. The second component will be a set of run-time support tools which will handle interaction between the user and the system, and will provide facilities for monitoring user interactions for conducting serious evaluations of user interfaces.

This report represents one of the 72 attachment reports to the University of Southwestern Louisiana's Final Report on NASA Grant NGT-19-010-900. Accordingly, appropriate care should be taken in using this report out of the context of the full Final Report.

A METHODOLOGY FOR THE DESIGN AND EVALUATION OF USER INTERFACES FOR INTERACTIVE INFORMATION SYSTEMS

A Dissertation Prospectus

Presented to

The Graduate Faculty of

The University of Southwestern Louisiana

In Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Mohammad U. Farooq

Spring 1986

DISSERTATION PROSPECTUS

Major: Computer Science

Tentative Title: A Methodology for the Design and Evaluation of User Interfaces for Interactive Information Systems

Student: Mohammad U. Farooq

Approval Recommended:

Wayne D. Dominick, Chairperson
Associate Professor
Center for Advanced Computer Studies

William R. Edwards, Jr.
Associate Professor
Center for Advanced Computer Studies

Lois M. L. Delcambre
Assistant Professor
Center for Advanced Computer Studies

Joan T. Cain
Dean, Graduate School

ABSTRACT

Software development research is experiencing a significant shift in research emphasis from producing more elegant and faster algorithms toward producing more user-oriented systems. Researchers in psychology, human factors, computer science, and related disciplines have started serious analyses of human computer interactions. It has been recognized by the researchers that the design of good user interfaces is more of an art form than a science or engineering discipline. Also, methodologies and tools are lacking for designing, implementing, maintaining, and evaluating user interfaces for information systems.

The major objectives of this research are: the development of a comprehensive, objective, and generalizable methodology for the design and evaluation of user interfaces for information systems; the development of equations and/or analytical models to characterize user behavior and the performance of a designed interface; the design of a prototype system for the development and administration of user interfaces; and the design and use of controlled experiments to support the research and test/validate the proposed methodology.

The proposed design methodology views the user interface as a virtual machine composed of three layers: an interactive layer, a dialogue manager layer, and an application interface layer. A command language model of user system interactions is presented because of its inherent simplicity and structured approach based on interaction events. All interaction events have a common structure based on common generic elements necessary for a successful dialogue. It is shown that, using this model, various types of interfaces could be designed and implemented to accommodate various categories of users. The implementation methodology is discussed in terms of how to represent the various types of information pertaining to an interaction event, and how to store and organize the information.

A generalized evaluation methodology is also proposed for the evaluation of user interfaces. The methodology will allow interface developers to evaluate user interfaces from the viewpoint of the performance of their users. A Personal Computer-based Protocols for Interface Prototyping and Evaluation (PC/PIPE) system is proposed. The system will be composed of two main components. The first component will be a set of tools to support the design and implementation of a user interface. The second component will be a set of run-time support tools which will handle interaction between the user and the system, and will provide facilities for monitoring user interactions for conducting serious evaluations of user interfaces.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

"Man must become the prime focus of system design. The computer is there to serve him, to obtain information for him, and to help him do his job. The ease with which he communicates with it will determine the extent to which he uses it. Whether or not he uses it powerfully will depend upon the man-machine language available to him and how well he is able to understand it." [MART73]

1.1 The Problem

It is widely recognized by users of interactive information systems (a particular class of software systems characterized by conversational access to data [WASS84]) that the user interface is often designed without serious consideration for the user on the part of the designers. Implementation considerations such as program speed and size have always figured prominently in the design of most computer systems and these concerns often result in design decisions which are awkward for the user. The design of a user interface is often perceived as secondary to the system which it serves. This is a rather serious problem in its own right.

There are several reasons why software developers continue to produce software products with poor user interfaces [ATWO84, SCHV84, EHRI83, BOKE80, KRAU80]. For example:

1. Human engineering is expensive and there is not any real consensus on what good human engineering is.

2. Software designers are not always aware of the poor human engineering of their products.

3. The knowledge and background of the system designers and that of the users of the system are often radically different.

4. The definition of very high-level interfaces, including the support of a subset of natural language, and the development of strategies to answer questions require a deep understanding of general psychology, psychology of languages, and linguistics, which are not intuitively obvious to the designer.

5. Vendors assume that special training is essential for the use of their products and therefore often do not really care about the user interface.

6. Current methodology and software design tools do not adequately support the design, implementation, and evaluation of user interfaces.

The failure to take the design of user interfaces seriously can be remedied by a change in the attitudes of designers. There is an increasing awareness on the part of system designers that ad hoc design processes, based on intuition and limited experience, may have sufficed in the design of early programming languages and interactive languages, but are insufficient for designing user interfaces for information systems which are being used by an increasing number of diverse communities of users [SHNE79]. We also see a shift in the research emphasis from producing more elegant and faster algorithms towards producing user-oriented systems [BORG84]. Technological advances have made computers faster and more powerful so that the speed of algorithms is no longer the most important issue. Development of silicon chip microporocessors have made computers more and more inexpensive and accessible to a wider spectrum of potential users. Therefore, there have been many demands for more "user-friendly" systems, but we do not understand human-computer compatibility well enough even to agree on what "user-friendly" means. The U. S. National Research Council in an important policy report [USNR83] calls for a workable definition of "user-friendly"; a database of cognitive population characteristics related to human-computer performance; and, among other things, a call for some consensus on a classification of users.

Because of these needs, human-computer interaction has become an active area of research and has brought together a mixed group of researchers in psychology, human factors, computer science, and related disciplines [ATWO84, BORG84]. Research in human-computer interaction serves several goals [BORG84, REIS83, MORA81b]. Some of this research is directed toward formulating theory, as researchers attempt to understand the human processes involved in comprehending and manipulating a complex system. Some research uses human-computer interaction as a practical application for understanding broader issues of human behavior. Still other research is directed toward evaluating an existing design or developing guidelines or priniciples for future systems design. In a broad sense, all of this research eventually leads to design issues - the better we can understand the human processes involved, the better we can design systems to support these processes.

The majority of human-computer interaction studies are behavioral experiments. As Reisner [REIS83] points out, these experiments are usually difficult, costly, and time consuming to conduct. This problem has serious consequences. Because of being time consuming and costly, such experiments are frequently not conducted at all. In the system design phase, failure to uncover usability problems can be disastrous to the end user. A system which is poorly designed from the viewpoint of the user does nothing to improve his quality of life. Another consequence is

that experiments are run, but not run well. It might be possible that only the initial use of a system is tested, and not long term use. Only a few experimental subjects might be used, or worse yet, these users might not even be representative of the actual users of the system. A further difficulty is that an implemented system is usually required on which to perform the experiments. At the very least, a simulation or a prototype is frequently needed. By the time such a system is available, it might be too late for experiments to meaningfully aid in the system design process. The lack of a theoretical understanding of principles of human factors is another serious difficulty, which is also intellectually unsatisfying for the serious researcher. Most experiments indicate whether a system meets its usability goals, or which of two system is easier to use, or where users make mistakes. These experiments do not indicate why these results are obtained.

1.2 Formal Tools and Methodologies for User Interfaces

We understand human behavior much less than we understand computers; the designing of user interfaces is one of the hardest aspect of systems design. Clearly, there is a need to provide better methodologies and tools for designing, implementing, maintaining, and evaluating user interfaces for information systems.

The proposed research addresses this need by providing a methodology and tools for designing and evaluating user interfaces. The approach presented in the proposed research recognizes that the creation of a user interface requires special skills, special system capabilities, special tools, and special methodologies because it is an intrinsically different activity from the coding of computational algorithms.

There are several important reasons for introducing formal tools and methodologies into the task of producing well engineered user interfaces. One reason is that, in the current state of the field, there are too many unsupported, sometimes conflicting, design priniciples [GEBH78, MAGU82, SHNE80, WILS82]. Most of these guidelines are based on the intuition and experience of particular designers with particular systems. Few of these intuitions have been evaluated experimentally. Carefully designed tools may be able to enforce consistency and encourage the designer to use techniques selected for their effectiveness on the basis of behavioral evidence. Another reason is that current methodology requires interface design logic to be treated as though it were the tedious detail of the system it is designed to serve. As a consequence, a user interface is woven into the software fabric in such a way that software vendors and designers simply become committed to inferior interfaces because they are too complex and expensive to reprogram [EHRI83]. An example will illustrate this point.

Most programmers using a high-level language tend to specify input-output formatting at the point where the input-output statements reference those format specifications. These details are usually totally irrelevant to the computational task whose logic was interrupted by the occurrence of the input-output statements. Later, when the formatting needs to be altered, it may be almost impossible to locate the code that produced the erroneous format.

A more important reason for introducing formal tools and methodologies is that the design and programming of user interfaces for interactive systems is a high-cost activity. Industry surveys [ROWE83] indicate that around 50 percent of the coding effort in a typical data base application is usually spent on the implementation of user interface routines. There are commercially available software systems, called application generators, which are geared primarily to support data intensive application development [MART82]. These systems have their origin in the early report-generator systems, such as IBM'S RPG. Contemporary application generators typically consist of a database management system, report generator, database query language, graphics package, and special purpose software, such as financial modeling or statistical analysis packages. An investigation by Horowitz, Kemper, and Narasimhan [HORO85] indicated that there are very few application generators which provide any facility for the tailoring of user interfaces.

Easy to use tools which reduce the time and cost for tailoring user interfaces are necessary for the design and administration of user interfaces. The methodology presented in this research will force the designer of a system to think in a specific way about the user interface by providing a separation of interface and applications. The methodology will also assist a designer in developing a system using a rapid prototyping approach [BLUM82] to expedite the creation of user interfaces and make it possible to change them easily. More importantly, this research will contribute to changing the design of user interface from being an ad hoc process to being structured and planned.

A Personal Computer-based Protocols for Interface Prototyping and Evaluation (PC/PIPE) system is proposed. The view presented in this research is similar to that of a data base management system. A data base management system provides a service primarily for an application programmer and can be evaluated well in terms of computational efficiency. While an end user does benefit from multiple access paths, data security, recovery, and the like, the end user does not necessarily view these capabilities as principal goals of the data base management system [THOM83]. However, the end user is the principal audience of the PC/PIPE.

An automated design system is desirable for a variety of reasons in addition to the obvious potential for saving some of the expenses associated with interface design. For example, such

a system could produce more consistent user interfaces than those produced by one or more human designers, and, as new knowledge becomes available, it will be easier to update the PC/PIPE's data (or knowledge) base than to update the knowledge base of all individuals who design interfaces.

There are four research threads that the proposed research will bring together: the notion of user models (characterization of users by system designers, system image provided by the designers, and mental image of the system that shapes up in the user's mind), specification of user interfaces via formal methods, analytical tools for evaluating user interfaces, and the use of behavioral tests of models of user computer interaction. This proposal is centered around the research and development objectives to be identified in Chapter 2.

2. RESEARCH AND DEVELOPMENT OBJECTIVES

The intent of this chapter is to define the set of research and development objectives which will structure and direct all of the activities to be performed within the scope of this proposed research. Research objectives are first stated in general terms in Section 2.1 and then refined into specific research objectives in Section 2.2.

2.1 General Research Objectives

1. General Methodological Objective:

   The development of a comprehensive, objective and generalizable methodology for the design and evaluation of user interfaces for information systems.

2. General Theoretical Objective:

   The development of equations and/or analytic models to characterize user behavior and performance of the designed interface.

3. General System Design Objective:

   The design of a prototype system for the development and administration of user interfaces for interactive information systems.

4. General Experiment Design and Implementation Objective:

   The design and use of controlled experiments to support the research and test/validate the proposed methodology.

5. General Application Objective:

   The application of the methodology to the design, implementation, and management of a common user interface to selected existing information systems environments. The application environment for all of these activities will be a common desktop microcomputer such as an IBM PC.

6. General Evaluation Objective:

   The evaluation of the completeness, generalizability, and overall quality of the methodology and its supportive components.

2.2 Specific Research Objectives

The following subsections describe the specific objectives of this research. These are refinements of their respective general research objectives identified within Section 2.1.

2.2.1 Methodological Objectives

The following are the specific methodological objectives identified for this research:

1. Develop a model of user system interaction for evaluating alternative user interface designs. The model should provide support for multi-level user models, multi-level interface models, and performance criteria.

   Significance - The model will serve as a framework for integrating design and evaluation studies for alternative user interface designs and alternative user models. Generalizability across applications and user tasks is the main orientation of this model.

2. Develop algorithms to predict user performance within alternative user language designs for a given class of users.

   Significance - Analytic models based upon the user model and interface design model will predict the user performance before the interface is actually implemented.

3. Design and conduct experiments to compare predicted performance against actual performace (as measured by automated monitoring facilities) in order to fine-tune the prediction algorithms.

   Significance - The analytic models comprising the performance prediction algorithms will be calibrated according to empirical data for a given class of users.

4. Incorporate the performance prediction algorithms into PC/PIPE. This incorporation will aid in the detection of performance bottlenecks by exposing the major components of observed (or predicted) performance and the contributing factors in terms of underlying user interface design structures.

   Significance - The performance prediction algorithms will become the core of PC/PIPE whose shell analyzes the input and output parameters to determine performance bottlenecks and their major contributing factors.

5. Develop a user interface design aid which will generate interface designs to meet pre-specified user interface performance criteria for a given class of users.

   Significance - The PC/PIPE will choose, from available language structures, a set which will either meet pre-specified performance specifications or a set which will yield "optimal" performance for a given class of users.

6. Develop a user interface re-design tool which will determine an appropriate remedy, from a given set of alternatives, for a performance bottleneck detected within an existing user interface design.

   Significance - The re-design tool will be a part of the PC/PIPE and will identify any potential bottlenecks created

by the redesign.

7. Examine the feasibility of performing dynamic redesign of user interfaces based upon the chosen remedy for a bottleneck.

   Significance - Experiments will be designed and conducted to determine the degree of performance improvements attained by a suggested redesign of an interface. If significant performance improvements are detected, the interface redesign can possbily be performed automatically, without designer intervention.

8. Develop a mechanism to provide feedback into the redesign tool so that past decisions and their effects can become a part of the decision process.

   Significance - Truly adaptive interfaces can only be generated by making the system aware of its past performance. The PC/PIPE philosophy will be a step toward automated generation of adaptive interface designs.

2.2.2 Theoretical Objectives

The following are the specific theoretical objectives identified for for this research:

1. Identify the primary measurement parameters which characterize user interface performance.

Significance - A major component of the user interface evaluation model will be the identification of performance metrics which span input devices and user models.

2. Develop a formal grammar to characterize the various actions a user performs to interact with an information system.

Significance - The formal grammar will provide abstract representation of all actions performed by a user.

3. Develop formal models to predict user performance for alternative interface designs for a given class of users.

Significance - Analytical models will predict user performance before the interface is actually implemented.

4. Develop formal models to characterize the user tasks which a user is trying to accomplish with the system.

Significance - The task model will provide a notation which the designer can use when performing task analysis and describing the problem a user is trying to solve.

2.2.3 System Design Objectives

The following are the specific system design objectives identified for this research:

1. Redesign and implement PC/MISI using conventional software development methodologies and programming tools.

Significance - PC/MISI is a Personal Computer-based Multiple Information Systems Interface which will provide access to remotely located information systems using one common language [HALL84, HALL85a, HALL85b]. This will serve as an experimental tool for benchmark purposes.

2. Design and implement a user interface development subsystem.

   Significance - The subsystem will provide facilities to an interface designer for the specification and modification of user interfaces.

3. Design and implement a user interface execution subsystem.

   Significance - The subsystem will provide runtime support for the testing of designed interfaces.

4. Design and implement storage structures for user interaction sequences.

   Significance - Separate structures for interface logic and dialogues will help in the rapid skeleton implementation of user interfaces.

5. Design and implement software monitor structures.

   Significance - A software monitor will be built into the developed user interface for evaluative purposes.

6. Design and implement a centralized help facility for PC/PIPE.

   Significance - An interface designer will not have to be an expert programmer.

7. Design and implement design validation procedures.

   Significance - Such procedures will allow the production of consistent user interfaces (similar user commands for similar functions, consistent conventions for commands and abbreviations, and consistent reactions to user errors).

2.2.4 Application Objectives

The following are the specific application objectives identified for this research:

1. Implementation of PC/MISI user interface (menu level, command level and direct access level of PC/MISI) using the PC/PIPE.

   Significance - This implementation will illustrate the applicability of the methodology to a common interface with different interaction modes.

2. Implementation of performance prediction algorithms for PC/MISI user interfaces.

   Significance - This implementation will illustrate the

applicability of the prediction algorithms in a microcomputer-based environment.

3. Implementation of the integration of all tools for both design and evaluation of the user interfaces into a design and evaluation system implemented on an IBM PC within the USL NASA PC R&D project [DOMI84].

   Significance - This implementation will illustrate the applicability of the methodology to a vast group of current and future users who are not professional programmers.

2.2.5 Experimental Design Objectives

The following are the specific experimental design objectives identified for this research:

1. Design of controlled usage experiments to gather data pertaining to the use of the PC/PIPE.

   Significance - Such data must be collected to quantitatively evaluate the usability of the tools.

2. Design of controlled usage experiments to gather data pertaining to the performance of individual user interfaces.

   Significance - The performance metrics defined by the user interface evaluation model will be captured via the automated monitoring facilities incorporated into the user interface execution environment.

3. Design of controlled usage experiments to determine the accuracy of the prediction algorithms.

   Significance - The prediction algorithms will be fine tuned according to empirical data collected.

4. Design of controlled usage experiments to determine performance improvements resulting from redesign of user interfaces.

   Significance - Monitor data regarding user performance will be associated with specific user interface designs so that significant changes in performance can be detected for alternative designs.

5. The use of formal hypothesis-testing and experimental design procedures supported by automated statistical analysis of empirical monitor data in accordance with established standards for conducting scientific research.

   Significance - User interface design research often suffers from a lack of discipline in terms of experimental design. All aspects of this research will be supported by careful experimentation and analysis of results.

2.2.6 Evaluation Objectives

The following are the specific evaluation objectives identified for this research:

1. Evaluation of the generalizability of the methodology across different applications.

   Significance - The methodology is designed to apply to all interactive applications.

2. Evaluation of the generalizability of the methodology across different user populations.

   Significance - The methodology is designed to apply to a variety of user types.

3. Evaluation of the completeness of the methodology.

   Significance - The methodology should be applicable to a broad range of user interfaces.

4. Evaluation of the objectivity of the various equations and/or models which are used to quantify specific aspects of user performance.

   Significance - The research should meet its goal of objectivity by providing automated tools. The primary focus will be on objective metrics. However, when necessary, subjective criteria will be identified as such along with the reasons why objective metrics could not be formulated in such cases.

5. Evaluation of the accuracy of the various equations/models with respect to the phenomena these equations/models are

intended to model.

Significance - The equations/models should accurately model the functions they are intended to model. Empirical monitor data is used where applicable to verify the accuracy of equations/models.

6. Evaluation of the overhead associated with the incorporation of software monitoring mechanisms into the execution environment.

Significance - The research will provide quantitative data on the execution overhead which is incurred by continuous monitoring of the user interface activity for specific interface/monitor environments.

7. Evaluation of the experiment conducted.

Significance - The evaluation process will verify that the experiments were conducted in accordance with established principles of experiment design.

8. Evaluation of the application of the methodology, utilyzing automated monitoring facilities, supportive equations and/or models and appropriate statistical and experimental design techniques to the objective evaluations of user interface performance as a whole and/or of the performance of a specific component of a user interface.

Significance - Because of monitor overhead, the amount of data collected, the complexity of the equations/models, and the complexity of the statistical analysis procedures required, the objective approach has often been rejected, and conventional, intuitive approaches to user interface performance evaluation has been taken. Few systems have been extensively monitored or modeled. The proposed research strives to illustrate the feasibility, effectiveness and, the practicality of an objective approach to user interface evaluation.

3. PROPOSED METHODOLOGY

In this chapter, several major concepts of design and evaluation of user interfaces are explored. These concepts are integrated into a preliminary methodology to achieve the desired objectives of this research. A user interface is treated as a virtual machine and various levels of this machine are described in Section 3.1. A model of user system interaction is described in Section 3.2. It will be shown that, using this model of interaction, various types of interfaces could be designed and implemented to accommodate various categories of users. The implementation methodology of the proposed model is discussed in Section 3.3 and a generalized evaluation methodology is discussed in Section 3.4. An introduction to an automated facility using the proposed methodology is given in Chapter 5 entitled "Protocols for Interface Protyping and Evaluation System."

3.1 User Interface as a Virtual Machine

It has been recommended that the user interface should be separated as clearly as possible from the rest of the system [BALL82, BRAN84, EDMO82, EDMO81]. A user interface can be thought of as consisting of an input language for the user, an output language for the machine, and a protocol for interaction [FOLE80, CHIU85]. Figure 3-1 illustrates this definition of the

user interface.

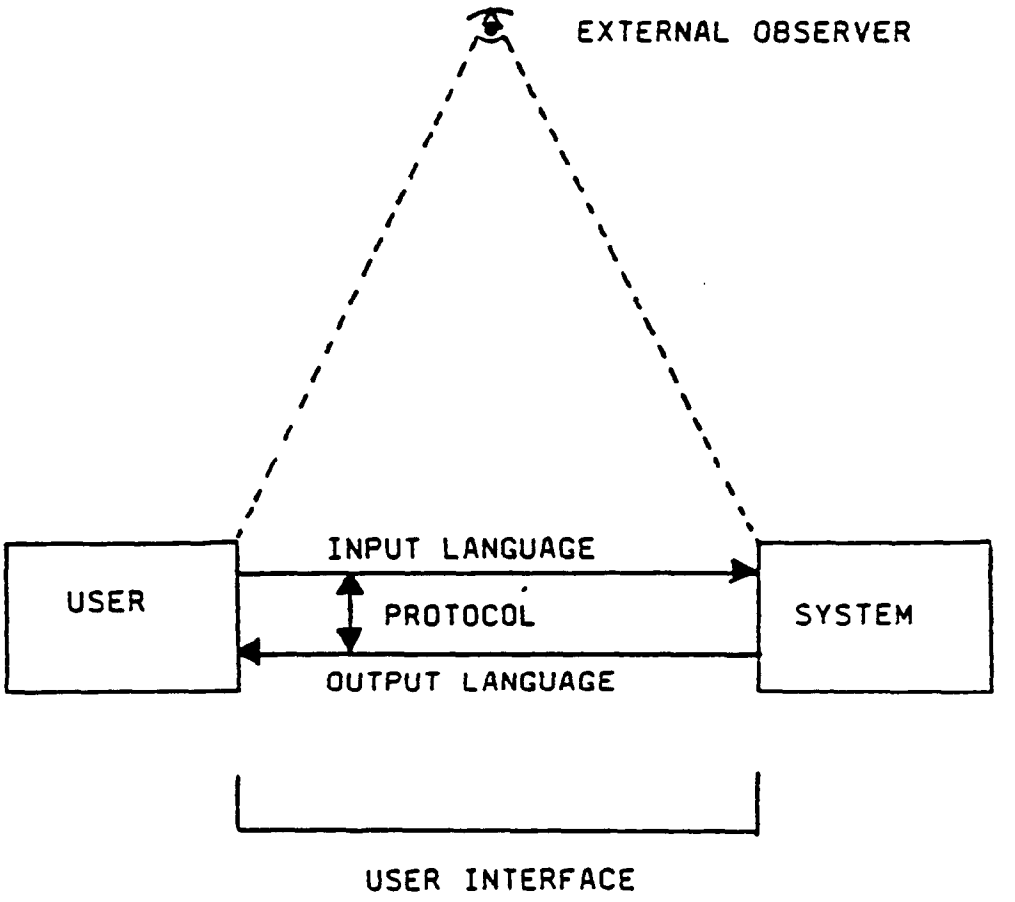
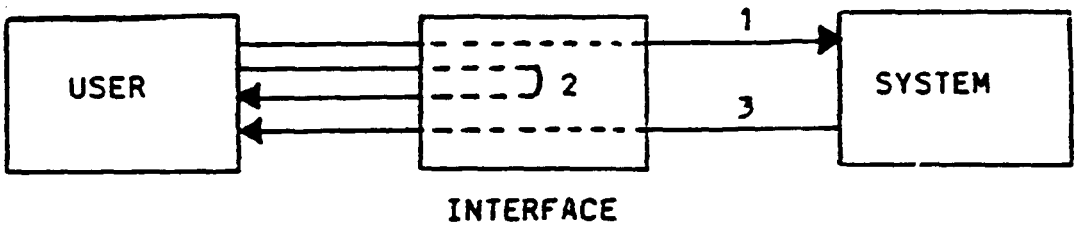


Figure 3-1 Definition of a User Interface

A user interface has three identifiable modes of operation which could equally well be described as the functions of the interface (see Figure 3-2).

1. It may accept an input from the user, transform it and cause the transformed messages to be given to the main system.
2. It may cause what is in effect a transformed message to be returned to the user.
3. It may take a message from the main system and transform it into a meaningful form for presentation to the user.



1. User input to system
2. User input error
3. System output to user

Figure 3-2 Functions of a User Interface

The second mode is quite similar to the first mode except that the input is tranformed into an error message and the recipient of the message is the user.

The identification of the operations of an interface as a separable process leads to the idea that a separate processor could be devoted to that process. Advancement in micro-electronics has made it possible that a microcomputer can be used to separate the interface from the main system. The interface might reside in a microprocessor, along with the tools for the design and administration of the interface. These tools could certainly reside in a mini or mainframe computer, however, the tools should be able to generate an interface which can reside in the microprocessor. Arrangements of this kind isolate the user from the mainframe operating system and from the resident operating system, thus providing the user with a virtual machine.

This virtual machine has three layers as shown in Figure 3-3. The interactive layer, the dialogue manager layer, and the application interface layer. The interactive layer is responsible for the physical appearance of the user interface including all the device interactions. The dialogue driver manages the dialogue between the user and the system. The application interface forms the interface between the user interface and the rest of the program. It provides the user interface's view of the application program.

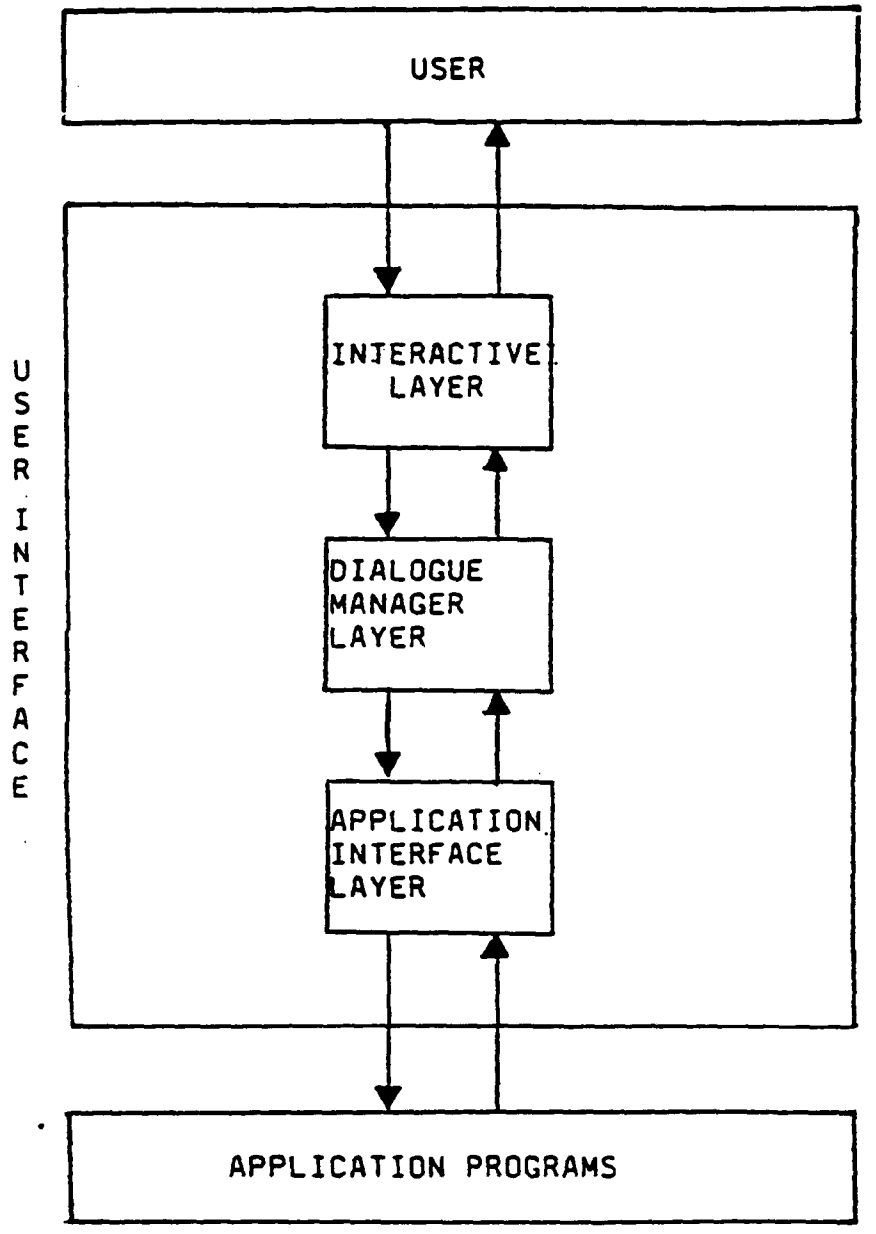


Figure 3-3 Layers of a User Interface

The system proposed as part of this research (PC/PIPE) will have interface specifications decomposed into these layers.

3.2 A Model for User System Interaction

A key requirement of an interface prototyping and evaluation system is to provide a technique for the interface designer to describe and organize the user interaction sequences. This problem is similar to that faced in the data base management area to allow the programmer to describe the data that is to be managed in an application independent manner. A DBMS provides this capability via a schema definition language which may be specific to a particular system and an underlying data model. The following subsections present criteria for evaluating the specification technique proposed for this research.

3.2.1 Criteria for Evaluating Specification Methods

A specification of user interfaces for interactive systems must satisfy a number of requirements if it is to be useful. The following criteria is established for evaluation purposes [WASS85, LISK75]:

1. Formality. A specification technique should be formal, that is, specification should be written in a notation which is mathematically sound.

2. Constructibility. It must be possible for an interface designer to construct specifications with less effort than writing a program to implement that user interface.

3. Comprehensibility. The system developer or user trained in the notation being used should be able to read a specification and then reconstruct the interaction which the specification is intended to describe. In other words, the interface designer must be able to maintain sequences described by others as a system matures.

4. Flexibility. The technique should provide the designer with the capability of specifying a wide variety of dialogue styles.

5. Portability. The technique should be device independent.

6. Executability. The specification should be directly executable to eliminate the need for writing programs to implement the specifications.

7. Completeness. A full range of primitive user actions must be supported as part of the interface specification. This range includes actions such as backing out of a sequence conveniently and accessing generally applicable functions such as 'help'.

The following section presents a model of user/system interaction which satisfies the above requirements.

3.2.2 The User System Interaction Model

The approach to user interface development presented here represents the first stage in an attempt to provide a comprehensive and generalized model for the design and implementation of user interfaces. The model is general enough to support most of the customary techniques of interaction provided for end users of information systems. The user interacts with an information system through a series of prompts to which the user responds with "commands" or "data". Although such interaction may seem to provide a restricted interface from a language point of view, it has the advantage that no programming knowledge is required by the information system user to use it. Moran [MORA80, MORA81a] calls this "command language" interaction because it is characterized by a command-execute cycle. This command language can be contrasted with a programming language, in which a set of commands is built before execution. Command language systems can also be contrasted with natural language systems. In the latter case, the variation is the complexity of the grammar and the subtlety of interpretation. The relatively simple nature of command language interaction leads to a structured approach to user interface definition.

The user system interactions are viewed as dialogues between two parties. The meaning of the term "dialogue" is intended to include a broad range of types of exchange between users and information systems. These exchanges may be in the form of

character strings (using a keyboard and visual display, for example), or they could equally well include the depression of function keys, the selection of graphical objects from a displayed image or the generation of shapes. In this model, the dialogues are represented as a sequence of basic interaction events. All interaction events have a common structure based on common generic elements necessary for a successful dialogue. The definition of a user interface is made up of the event definitions and transition controls which define all the possible sequences of events. An interaction event is defined as an occurrence in the dialogue where the system waits for an input from the user. The input may be a command or data. The event is finished when the system has finished processing the user input. The way the dialogue proceeds to the next event depends on the transition control actions which are considered part of the current event. The entire interaction event is viewed as a process which consists of four main phases (see Figure 3-4):

A. System Prompt - Indication is made by the system that an input is expected from the user.

B. User Input - An input is provided by the user.

C. System Action and Response - An action is taken by the system according to the user input and a response is returned by the system.

D. Transition Control - A decision is made by the system to determine the next interaction event.

Figure 3-4 Main Phases of an Interaction Event

The transition control simply directs control to the next event. In this approach, all processing which occurs between the current event and the next event is considered as the action of the current event. An event is represented by an event table and each element is input to a processor. The processing of the entire event is one event cycle. We will further elaborate on each of the above phases in the following paragraphs.

A. System Prompt

The system prompt may be in the form of a prompting character, a prompting message or display of a menu asking for the selection of an item in the menu. Prompts may be dynamically selected for presentation depending on the current style of the dialogue or the preferences of the end user.

B. User Input

The user input may be "command" or data. The "command" may be one of the available commands from the repertoire of system commands or selection of an item from the displayed menu. This input could equally well include the depression of function keys. A blank input would be valid where the system assumes a predefined default value as the response.

C. System Action and Response

The system action depends on the user input. Various possibilities exist; these are generically grouped as follows:

i. Retreat - If Input = "cancel" (meaning "do not proceed with the current interaction event") then
   a. "next event" indicator is set; and
   b. current event cycle is ended.

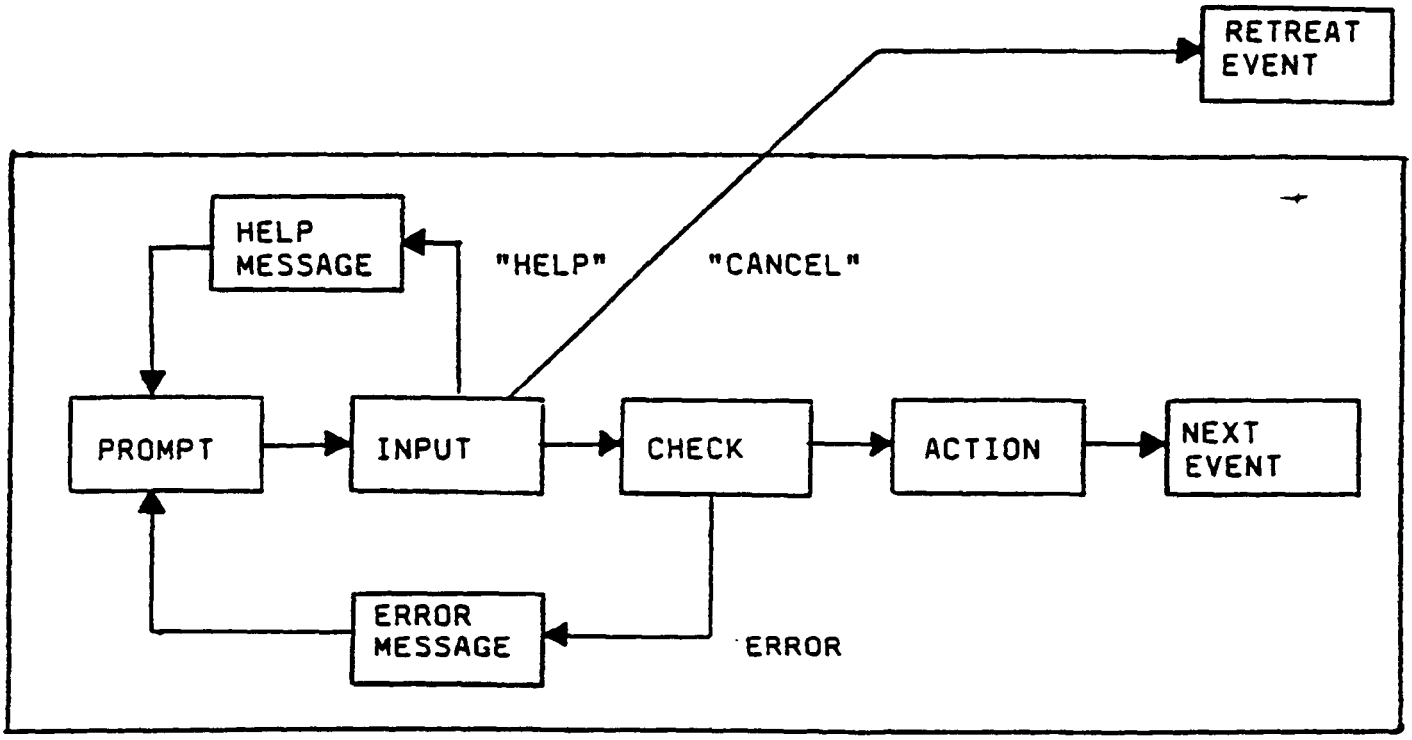
ii. Help - If input = "help" then
   a. additional assistance information is displayed; and
   b. current event cycle is ended.

iii. Check - Input is checked. If errors exist and no automatic error correction exists then
   a. errors are reported; and
   b. current event cycle is ended.

iv. Call application - Related application / database routine is called to process the users request.

The terms "cancel", "help" and "next event" are generic names. The above ordering is important for transition control actions. Any one of Retreat, Help, or Check may abort the normal cycle. Retreat precedes any input processing so this enables the user to interrupt an event which is not desired. It sets the "next event" index to one other than the current or the one which would have been normally set by Transition Control. Help does not change the "next event" index but it does cause a repeat of the "current event" cycle. To the user, Help will appear as a help message followed by a repeated prompt. In the case of errors which the system is not able to correct, Check would cause the display of an error message followed by a repetition of the "current event" cycle similar to Help. Note that Help and Check both allow the completion of the current event cycle, but do not prevent the user from exiting the current cycle.

Figure 3-5 illustrates the event cycle.



"HELP" and "CANCEL" are input tokens

ERROR is a condition

Figure 3-5 The Event Cycle

D. Transition Control

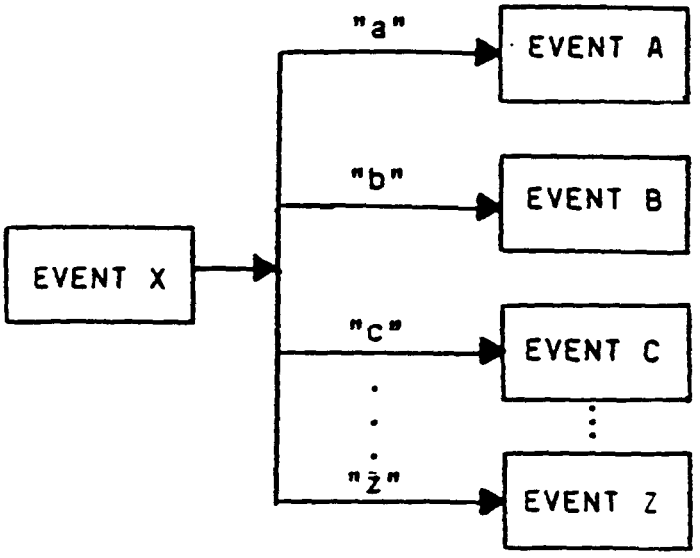
Transition Control directs the transfer to another or the same event. Strictly speaking, this information is not part of the event definition, but rather the arc of a conversation graph if we view the event as a conversation graph [HAGG83]. However, it is often convenient to handle information associated with an outgoing arc as an integral part of the predecessor node. The Transition Control allows the selection of the next event based on the structured programming control primitives (sequence, case, do-while). The next selection could be based on any one of the following primitives (see Figure 3-6):

i. "Sequence" - predefined order of the events.

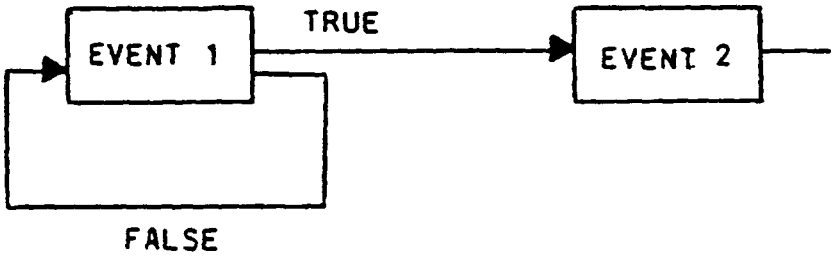
ii. "Select" - Any one of the predefined set of events based on a given control value.

iii. "Conditional-transfer" - pre-set to an event when a given control condition occurs.

(i) Sequence



(ii) Select



(iii) Conditional-Transfer

Figure 3-6 Transition Control Actions

The above control primitives also relate to different modes of interaction and different schemes for defining commands and related data or arguments which are common in user system interaction. The following three scenarios represent three common occurrences in command languages.

1. The system prompts for one data input, then for another. This case is implemented with the use of two events. The first event will prompt for the first data element and will have a pre-set "next" selection identifying an event prompting for the second data element. This way a "system guided" [GUED80, MILL77] dialogue could be effectively implemented.

2. The system asks the user for a command, then for its arguments. This case is implemented with the use of two events. The first event prompts the user for the name of a command. Then, through the use of the "select" primitive, control passes to the event which prompts the user for the arguments of the particular first event. This way a "user-guided" [GUED80, MILL77] dialogue is implemented.

3. The input required by the system is comprised of a list of arguments of indefinite length. This is implemented with one interaction event with a "conditional-transfer" primitive. The event has a pre-set next selection to itself. The end of the list is indicated by the user by input of a special value representing the termination command.

In the above examples, the first example illustrates the case where a user is led through a series of requests for data (arguments) which end with some processing. The second example illustrates the command driven mode, and the third is a combination of both where the system prompts for arguments, by repeating the same interaction event, until a particular command (i.e., the special termination value) causes the dialogue to move to a different event.

3.3 Implementation Methodology

This section discusses the implementation methodology for the proposed interaction model. Event descriptions are represented in a tabular form. An interaction event can be described as an ordered tuple:

<EVENT_ID; Prompt; Input (Default); Retreat; Help; Check; Action; Transition Control>

EVENT_ID is an identifier unique to each event. Each other element contains an index to information pertaining to the relevant phase in the event cycle. The information itself is organized in sets. Each set contains information of the same type or information which belongs to the same phase. Each member of a set may be referred to by entries in one or more event descriptions. Each set member is identified by a unique (within the set) identifier. Thus, the entries in the event description tuples become references to set members. Each reference is a

pair <Set_ID; Member_ID> and each member in a set is a pair of the form: <Member_ID; Information>.

The distinction between the event descriptions, which contain the dialogue "logic", and the actual information about text, processing definition, checks, etc., allows the two to be developed separately. This separation allows for a quick skeletal implementation which can be augmented later, for example by adding input checks and help messages. This feature provides the designer with support for prototyping. Also, the same event may be implemented with various versions of text (prompt, help and error messages) identifying levels of interface. A flag in each user's profile indicates which interface level is appropriate to that user. This flag can be changed either by user choice or by an algorithm in the system based on a pre-specified criteria, thus providing flexibility and customization of dialogue. The needs of different levels of the user population could be satisfied by the ability of the system to change easily with evolving levels of user knowledge.

3.4 Evaluation Methodology

In this section, a generalized evaluation methodology is proposed for evaluating user interfaces for information systems. A generalized evaluation may be contrasted with a specific evaluation which is tailored to a particular purpose or

situation, such as the evaluation of an information system to determine its utility in a particular working environment. A generalized evaluation focuses on the generic properties of a user system interface rather than on the idiosyncrasies of particular commands. A generalized evaluation attempts to address fundamental user interface issues and is thus applicable to a variety of user interfaces. These issues include questions such as:

(1) What range of tasks can a user perform with a system?

(2) How long does it take a user to learn how to use a system to perform a given set of tasks?

(3) What types of errors are made and what is their frequency?

(4) How long does it take a user to accomplish a given set of tasks using the user interface?

A benefit of using a generalized evaluation methodology is that a data base of consistent information about user interfaces could be collected over a period of time. This information would provide a standard for interpreting the results of any new investigation, a crtical factor missing from virtually all existing evaluation studies. The methodology proposed herein evaluates user interfaces from the viewpoint of the performance of their users and will provide for the generation of a valuable user interface performance data base of objective measures.

3.4.1 A Generalized Evaluation Methodology

A general methodology applicable to the monitoring and evaluation of any user interface for a computer based information system is composed of the following phases [DOMI78, BORM78]:

(1) Determine the monitoring/evaluation objectives.

(2) Determine the specific parameters to be monitored initially based upon the overall objectives.

(3) Design and implement the monitoring facility into the system.

(4) Design and implement the data validation procedures to validate the monitored data.

(5) Determine the data analysis tools to be used for analyzing the monitored data.

(6) Design and conduct the monitoring experiments to collect the data to be analyzed.

(7) Perform data validation on the monitored data.

(8) After the experiment has been completed, perform the data analysis making evaluations and drawing conclusions, as appropriate.

(9) Identify user interface improvements and enhancements as implied by the results of the analysis.

(10) Identify monitor improvements and enhancements as implied by the results of the analysis.

(11) Identify experimental design improvements and enhancements as implied by the results of the analysis.

(12) Incorporate all identified improvements and enhancements and repeat the cycle from step 6.

Figure 3-7 presents the above phases, illustrating parallelism where appropriate.

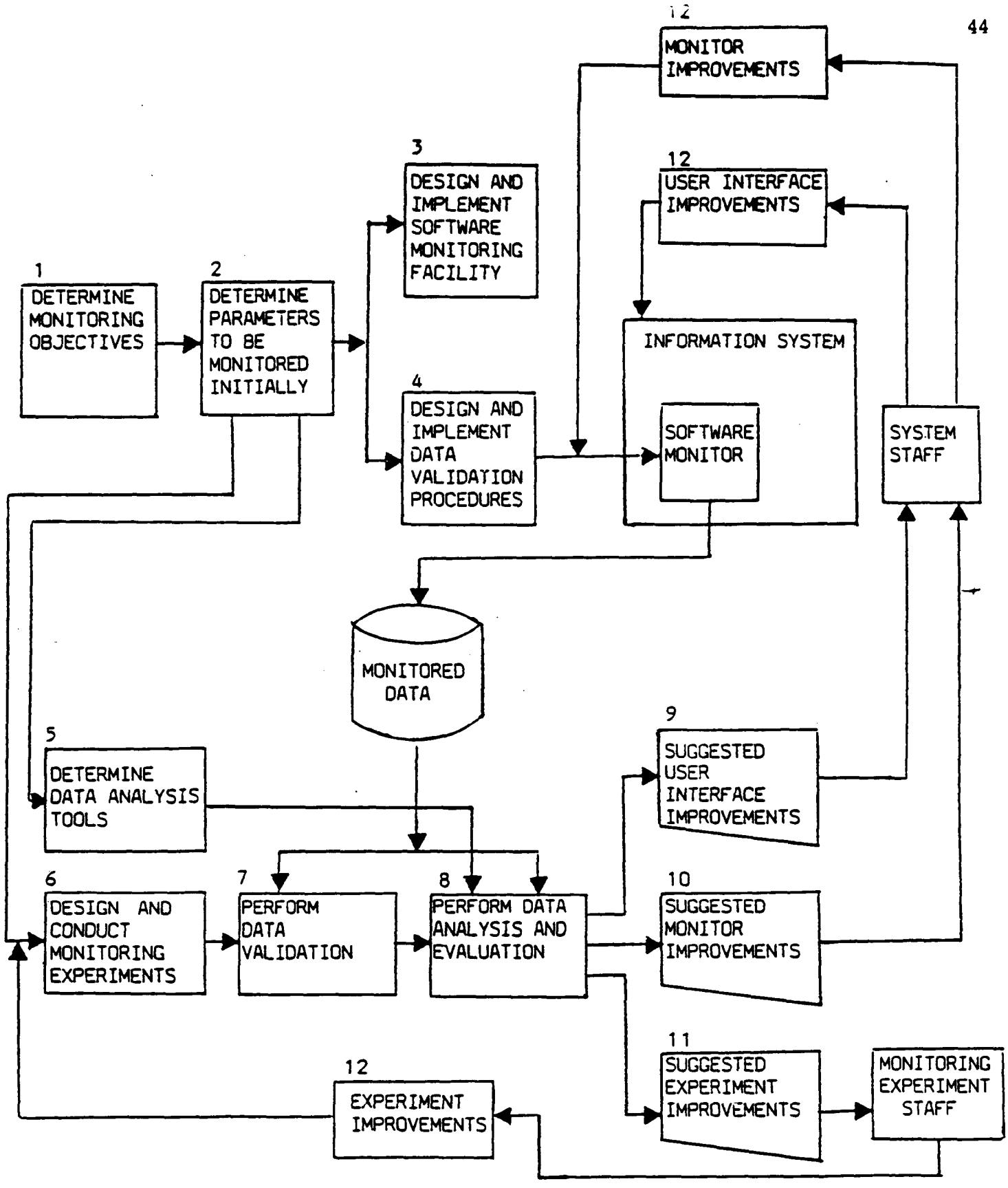


Figure 3-7 User Interface Monitoring and Evaluation Schematic

(Adapted from Dominick and Penniman [DOMI79])

The methodology proposed here will allow interface developers to evaluate user interfaces from the viewpoint of the performance of their users. In developing this methodology the following generic criteria will be used:

- Objectivity. The methodology must not be biased in favor of any particular user interface structure.

- Thoroughness. The methodology must consider the multiple aspects of an interface usage.

- Ease of Use. An interface developer/evaluator must be able to evaluate the performance of an interface, identify problem areas that exist, redesign the interface using the methodology, and cycle through the interface evaluation process, as necessary.

The methodology proposed for this research will allow the interface developer to select specific evaluative data measures from an available list of data measures. A software monitor will automatically be generated and incorporated into the interface designed. The monitor will collect the specified data measures at execution time on the following levels:

(1) Operation (or command) level

(2) Task level

(3) User session level

(4) Usage period level.

This capability will allow the interface developer to iteratively evaluate and improve a user interface. It will also allow the interface developer to compare alternative interface designs. To compare different types of user interfaces, there must be a common ground on which to base the comparison. For this purpose, tasks need to be identified. In the context of Information Storage and Retrieval (IS&R) systems, what is constant across all user interfaces is the information storage and retrieval tasks they permit their users to accomplish. There are two primary functions that an end user of an IS&R system performs, namely, search and output. Although many other capabilities may be provided, e.g., online help and tutorials, computations, manipulations, statistical analysis, graphical analysis, data base definition, data base maintenance (adding, deleting, updating), etc., the basic functions are those of data base search, retrieval, and output.

Various tasks may be defined by the evaluator for analysis, for example, various types of free text searching, selective field searching, and boolean searching. Task definitions will allow the characterization of sequences of searches and output operations into task-level units of analysis. The definitions of tasks may be changed, added to, or deleted from at any time as needs demand. A taxonomy of tasks must be identified. The functionality of a user interface is measured in terms of the set

of tasks in this taxonomy, by assessing how many of the tasks the user interfaces allow users to perform. Comparison between user interfaces concerning the time to perform a given set of tasks, types of errors and their frequencies, learning rates, and the like must be based on tasks that all compared interfaces allow users to perform.

The collection and analysis of data will be provided by the PC/PIPE evaluation component. At interface execution time, a time-stamped log of interaction events will be created. From this log, an interface developer/evaluator will be able to extract human performance data indicating "ease of learning" and "ease of use" of the interface. Viewing these criteria as contributing to the effectiveness of an interface is consistent with work performed by others in interface evaluation [LIND85, REIS84, GOOD82, ROBE83].

Within these contexts, "ease of learning" is defined as the amount of training time required of members of the user community in learning the user interface to reach an established performance criteria (to be defined in terms of speed of usage and number of user errors made), and "ease of use" is defined as the amount of interactive time needed with a user interface (once learned) in order to perform successfully a given set of tasks.

3.4.2 Factors Affecting Ease of Learning

We can classify factors contributing to the ease of learning a user interface into properties associated with the individual user and properties independent of the user. Properties associated with the individual user are:

(1) Similarity of the learned interface to other known interfaces; and

(2) Retention of similar interfaces.

For each interface component, there is a positive or negative learning influence determined by the similarity to or difference from interface components that the user has previously learned. The degree of retention measures the effect that past exposure to other interfaces has on learning a new interface.

Properties associated with the user interface are:

(1) Availability of a complete and accurate user's manual that aids in accessing the system;

(2) Existence of online assistance commands that increase the user's productivity;

(3) Existence of diagnostic messages that help in error recovery;

(4) Existence of prompting messages that aid in reducing the number of input errors;

(5) Syntactic homogeneity of the command language;

(6) Semantic homogeneity of the command language;

(7) Use of abbreviations allowed;

(8) Easy to remember command names and abbreviations; and

(9) Complexity of the interface.

Complexity of the interface is a major variable associated with learning of a new interface. To determine the learnability of an interface, we need to objectively measure its complexity. The complexity factor offers the greatest potential for developing predictive measures of usability [REIS83]. One measure of interface complexity is the number of distinct commands provided by an interface. An obvious hypothesis to consider is that the interface with fewer commands should be faster to learn [HALS77, ROBE83]. The point missed by this hypothesis is that commands are not useful in isolation; rather they are used in the context of methods or procedures to accomplish given tasks. The other hypothesis is that learning is related to the procedural complexity of a command language. One method to approximate the procedural complexity of an interface is to compute the average number of steps in the method for accomplishing a representative set of tasks, such as running a

benchmark [CARD80, CARD83, ROBE83]. The Keystroke-Level model [CARD80] provides a simple unamabiguous set of steps to count physical operations. However, the length of a method in physical operations can be a misleading indicator. For example, a method requiring a user to type "select" followed by return is not three times more complex than an abbreviation of the command requiring the user to type only "se". We can see that procedural complexity has more to do with mental "chunking" [DAVI84] of physical steps into coherent fragments than the physical steps themselves. This notion of procedural complexity as determined by mentally defined chunks is an instance of the "zeroth-order theory of learning" [CARD83] which states that learning time is proportional to the number of chunks of information that must be learned. To make this theory operational, the evaluator must be able to specify what the chunks are.

3.4.3 Factors Affecting Ease of Use

In a manner similar to the factors contributing to ease of learning, factors contributing to the ease of use of a user interface are classified into properties associated with the individual user and properties associated with the user interface. Properties associated with the user are:

(1) User's past experience with the interface;

(2) User's ability to recall how to use the interface;

(3) Frequency of use of the interface; and

(4) User's evaluation of his past experience with the interface.

Properties associated with the user interface are:

(1) Tolerance of user input errors such as minor spelling or typographical errors which can be recognized as such;

(2) Cancellation of previous input so as to restart from a designated point in dialogues;

(3) Existence of meaningful diagnostics to aid the user in error recovery;

(4) Existence of meaningful prompting messages to aid the user in reducing user input errors;

(5) Number of steps required to correct an error;

(6) Use of default options to reduce user response time;

(7) Use of function keys to reduce the number of keystrokes;

(8) Response time below a pre-specified or expected limit;

(9) Variations in system response time for equivalent tasks;

(10) Availability of abbreviations for command names;

(11) Number of steps required to accomplish a task using the interface; and

(12) Complexity of the user interface as discussed in Section 3.4.2.

3.4.4 Types of Data Measures

Two types of data measures are defined here. A raw data measure and a generated data measure [DOMI78, BORM78, MICH81]. A raw data measure is defined as a data item obtained directly from a monitor base. A generated measure is defined as a data measure which is generated in some manner, rather than being collected directly by a monitor. Generated data measures can be either first order generated measures or second order generated measures. A first order generated measure is a data item which is constructed via computations and/or manipulations performed on one or more raw monitor data items. A second order generated measure is a data item which is constructed via computations and/or manipulations performed on first order generated measures and/or second order generated measures.

The identification of variables applicable to user interface evaluation is of prime importance. The general categories of potential data measures which are relevant to this study are:

1. Interface usage profile variables;

2. User error and error recovery variables; and

3. User success and user satisfaction variables.

Within each category, generic variables will be identified that do not depend upon any particular user interface or information system, but rather are applicable across a wide range of user interfaces and information systems. The measures computed by the PC/PIPE will be flexible and extension facilities will be provided which will allow an interface experimenter/developer to specify procedures for computing measures which are not already provided by the system.

3.5 Summary of Proposed Methodology

This section summarizes the proposed methodology. The first major component of the methodology is that a user interface should be separated as much as possible from the rest of a system. Within this context, the user interface is treated as a virtual machine and its layers are described in Section 3.1.

The second major component is a technique that allows the interface designer to describe and organize the user interaction sequences. This technique should satisfy the criteria of formality, constructibility, comprehensibility, flexibilty, portability, executability, and completeness, as described in Section 3.2.1. Based on these criteria, a model of user system interaction is described in Section 3.2.2. The user system interactions are viewed as dialogues between two parties. These dialogues are represented as a sequence of basic interaction events. The definition of a user interface is made up of the

event definitions and transition controls. The transition controls allow the selection of the next event based on the structured programming control primitives (sequence, case, do-while). It is shown that, using this model, various types of interfaces could be designed and implemented to accomodate various categories of users.

The implementation methodology for the proposed interaction model is discussed in Section 3.3. The distinction between the event descriptions which contain the dialogue logic, and the actual information about text, processing definition, checks, etc., provides the designer with the rapid prototyping capability.

The third major component of the methodology is to provide the interface developers with a set of tools for monitoring and evaluation of their designed user interfaces. Section 3.4 addresses this component. A generalized evaluation methodology is presented in Section 3.4.1, and various evaluative factors and metrics are presented in Sections 3.4.2 through 3.4.4.

4. BACKGROUND AND STATE-OF-THE-ART

Over the past decade, we have heard much lore about what makes information systems easy to use, and about pros and cons of various interaction devices and techniques. There have been a few attempts to summarize, in a structured way, the design philosophy and accumulated knowledge and experience [MORA81b, RAMS83, REIS83, BORG84]. This chapter summarizes studies and research efforts related to user/system interface issues. Section 4.1 overviews research performed on users of information systems. Section 4.2 provides a survey of specification techniques used for the design of user interfaces. Section 4.3 provides an overview of analytical tools developed or being developed for the evaluation of user interfaces. Work addressing users' conceptual models and mental models is described in Section 4.4 and, finally, Section 4.5 provides an overview of research activities addressing the design and management of user interfaces.

4.1 User Research in Computer Science

User research in computer science is still in its infancy [BORG84]. Statements about users and their behavior have typically been by-products of research on library users. In general, ideas about the characteristics of computer users do not come from dedicated research but from designers' personal

experiences and beliefs [DAGW83].

Moran [MORA81b] describes four approaches to "an applied psychology" of the user, which lie along a continuum from empirical to theoretical. These are: 1) the experimental approach, 2) the features approach, 3) the factors approach, and 4) the calculational approach.

In the pure experimental approach, methods of experimental psychology are used to evaluate the specific system under development. No attempt is made to develop any theory or deep understanding that might help in the design of the next system. The typical approach is to construct a general interface simulator. However, this simulator not help with another expense of this approach, namely, that it requires multiple subjects and multiple trials to get reliable measures.

The features approach attempts to discover the general design features of systems that affect user behavior. It is expected that these general design features can then be used to formulate design guidelines. The factors that affect user behavior are quite complex and interact extensively with one another. This approach is clearly better than the experimental approach, but it still has major shortcomming. It, too, leads to a repeated focus on low-level issues, such as selection of an input device, but for different reasons. Further, by concentrating on features rather than principles, we often

perform research that has limited applicability to the next generation of technology.

In the factors approach, a researcher attempts to determine a pattern of psychological factors that are relevant to user behavior, perhaps through mutivariate statistical analysis techniques. Williges and Williges [WILS82] show application of this approach by using polynomial regression and response surface methods to study the patterns of effects of system response time, display rate, keyboard echo rate, and keyboard buffer length on user performance in a personnel records task.

The calculational approach involves the development of explicit information-processing models of user behavior in particular tasks. The Keystroke-Level Model [CARD80, CARD83] is an information-processing model that predicts the error-free performance times of expert users employing interactive command languages.

Penniman and Dominick [PENN80] reviewed the literature to identify data regarding user's characteristics and information use patterns. They found only a minimal number of studies containing hard data and even fewer containing any kind of behavioral measures.

Although one should generally be skeptical of questionnaires and interview studies, these are often the only direct measures available to reseachers. Dzida, et al. [DZID78] provide useful

insights into user perceived problems of usability. Based on a factor analysis of questionnaire responses from a reasonably large group of users, the authors found seven major usability factors: 1) self-descriptiveness of the system, 2) user control, 3) ease of learning, 4) problem-adequate functionality, 5) correspondence with user's expectations, 6) flexibility in task handling, and 7) tolerance for user errors.

Eason [EASO80] argues that the flexibility of task handling is the most fundamental type of flexibility. He distinguishes "closed" tasks from "open" tasks. A closed task is "one in which the alternative states of input and output variables are well understood and will fall within a predictable range" [EASO80]. The properties of an open task may vary greatly, often as a result of influences outside the user's control. Eason contends that we have not been very successful in designing systems to support open tasks. He also notes that dialogue needs vary strongly with the degree of openness and the frequency of occurrence of the task. Open, infrequent tasks require a particularly flexible, adaptive user interface. With a different orientation, Nickerson [NICK81] presents a list of user frustrations based on informal interviews: wrong functionality, limited accessibility to the system, start-stop hassle (logon/off protocols etc.), system dynamics and response time, work-session interrupts (system crashes, etc.), training and user aids, documentation, command languages, consistency of system behavior,

and the user's conceptualization of the system.

The problems associated with poor user interface design are widely recognized and discussed in the literature. Hayes, et al. [HAYE81] present a fairly detailed, realistic example of dialogue with a mail system. In this example, many of the difficulties arise from the extreme literalness and lack of flexibility of the user interface. They suggest a particular type of dialogue front-end, oriented primarily toward command language dialogue, as a possible solution.

4.2 Survey of Specification Techniques

Much of the work in this area has been concerned with static programming languages rather than interactive languages [JACO83a]. In a static language, an entire text is presented as input before any processing begins or any output is produced; then, all the outputs are produced together. Processing of the text is affected little, if any, by previous inputs. While in the interactive language, the input can be described as a series of brief texts, where the processing of current input generally depends on previous inputs. Equivalently, one long text is input and the computer system takes actions and produces outputs at various points during the input, resulting in a dialogue. Most specifications for both static and interactive languages have been based on one of two formal models: Backus-Naur Form (BNF) or state transition diagrams. Each of these methods provides a

syntax for describing legal streams of user inputs. In order to specify user interfaces, the techniques must be modified to describe the system actions as well and their sequence with respect to the user input.

4.2.1 State Transition Diagrams

As early as 1969, Parnas [PARN69] suggested state transition diagrams to describe user interfaces for interactive systems. He differentiates "terminal state" from "complete state" in a way analogous to the separation of syntax from semantics in other specifications. His paper contains some simple examples and does not address how the scheme would work for more complex real world systems. A transition diagram has a labeled node which indicates an initial state, possibly multiple terminal states, and possibly multiple output states. The directed arcs are labeled with a possible input string followed by the system response to that string. Folley and Wallace [FOLE74] also advocate the use of a state diagram to represent the user interface. They too do not examine the problems of complex real-world systems. Feyock [FEYO77] described transition diagrams in the context of computer assisted instruction and help systems. Wasserman and Stinson [WASS79], like Feyock, emphasized that the system response on the arc may involve the invocation of another transition diagram and are more attentive to the details of interfacing with a procedural language to carry out computations.

The MUMPS interactive computer language specification uses nonterminal symbols extensively and gives a precise deterministic procedure for interpreting diagrams containing them [MUMP77]. The specification is noteworhty in that the actions associated with its transition comprise a complete specification of the semantics of the MUMPS language.

Singer [SING79] uses state diagrams in the context of an interactive Help system for Pascal. His notation is more complex and difficult to understand. It uses separate diagrams for nonterminal symbols and a global data structure which can be set by arbitrary semantic domain actions. By examining the values in this data structures, and not by directly looking at input tokens, transitions can be selected. Hence, a transition involving receipt of a particular token is described by two transitions in his notation - one to read it into the data structure and one to test the value just stored.

In the Taxis system [MYLO80, BARR81], the overall organization and structure of dialogue and process control for a particular interactive information system is achieved using "scripts". A script can be thought of as a known plan to accomplish some goal. Each script is represented using a transition net which is based on a simplified version of Zisman's augmented Petri nets [ZISM77]. Scripts provide facilities for modeling decision making, concurrency, and synchronization, rather than representing user interfaces.

Ling [LING82] describes designing data entry programs using state diagrams as a common model. Each data entry program supports one type of transaction. A compact model was obtained by limiting the set of states to those minimally required for representing the interface conditions. There are two types of conditions: those which occur after the user enters a field and those which occur when the user attempts to output a transaction. These observation lead to model of 20 states: one initial state, one final state, 8 states representing the logic for a generalized data field, and 10 states representing the end of transaction logic. The state diagrams were checked visually against the specified program behavior. A more rigorous approach to verify the logic was taken to construct a formal correctness proof for the state diagrams.

IBM's chief scientist, Dr. Branscomb [BRAN84], also advocates that one should define the interface as a set of states and transitions. To a great extent, the IBM Audio Distribution System [GOUL84] follows the priniciples advocated by Branscomb.

The use of state diagrams does not provide a totally formal description of the semantics of state transitions and usually does not include a specification of the screen layout. In addition, the specification is given by using a concrete syntax which requires the definition of many details not relevant in an early or prototyping design phase.

4.2.2 BNF and Formal Languages

Formal languages and automata have long been a part of user interface design. One of the earliest attempt was Newman's Reaction Handler [NEWM68]. More recently, a great deal of work has been performed in using formal languages to characterize and analyze user interfaces [BLES82, WASS81, REIS81, REIS84].

Resiner [REIS81] provides an example of how BNF can be used to describe a user interface. Her approach does leave out the semantics of the user interface - the system actions and responses. She describes the "action languages" for two versions of an interactive graphics system intended for use by nonprogrammers. She then shows how these languages can be described in terms of a production rule notation. Particular emphasis is given to actions the user has to learn and remember (i.e., cognitive factors). She then presents predictions about user performance based on the formal description and exploratory results of testing some of the predictions.

Shneiderman [SHNE82] proposes a modified form of BNF in which each nonterminal symbol may be associated with either the computer or the user. The human-related BNF grammar is used to parse the input while the computer-related BNF grammar is used to generate the output. These grammars contain labeled nonterminals to indicate the party that produces a terminal string.

4.3 Analytical Studies of User System Interaction

In academia and in industrial research, some researchers have begun to develop analytical tools to determine the ease of use of user interfaces. An analytical tool does not measure the user behavior directly, but it predicts what would happen if users were interacting with a system. This section describes some analytical tools that have been or are being developed. The studies to be disussed have a number of different goals. Most are attempts to aid system design. None of the tools have as yet been tested for ease of use.

Embley, Lan, Leinbaugh and Nagy [EMBL78b] propose a model to compare program editors from the end user's point of view. In their model, the total time to perform some "unit" task consists of two main factors: the time to key-in the commands and another factor. The other factor consists of "think time" and computer response time. The "think time" is the time for the user to decide what to do next. The time to key-in commands is taken as the number of keystrokes times the average time per keystroke. The time for a task is simply:

T.task = m*T.c + n*T.k

where m is the number of command response pairs

n is the number of keystrokes

T.c is the think time per command and associated computer response

T.k is the time per keystroke

Using parameters of T.c = 5 seconds and T.k = 1/2 second, the authors found statistically significant differences between two editors.

Card and Moran [CARD80] view their keystroke level model as a design tool. The model is intended to predict task time, for expert users, on routine tasks. The central idea behind the model is that the time for an expert to do a task on an interactive system is determined by the time it takes to perform the keystrokes. The precise method (sequence of commands) must be specified and no errors are expected. Like the Embley model, the keystroke-level model counts keystrokes. However, the keystroke model is very clearly and explicitly a model of the user. It contains four physical-motor operators: K (keystroking), P (pointing as with mouse), H (homing, moving the hands to the appropriate physical device), and D (drawing straight lines segments, using the mouse). In addition, there is one mental operator M for Mental Preparation (e.g., deciding which command to invoke). There is also an R operator for system response time. The time to execute a task is the sum of the times for the relevant parameters.

The Keystroke-Level model is severely restricted in the sense that the user must be an expert; the task must be a routine unit task; the method must be specified in detail; and the performance must be error free. Additionally, it predicts only one aspect of the total user-computer interaction, namely the

time to perform a task.

The work of Reisner [REIS81] is an attempt to provide a predictive tool to compare alternative designs for ease of use, and to identify design choices which would cause users to make mistakes. Her action language model views user actions at an input terminal (keying, moving a joystick, pressing a button) as a language. It uses a production rule notation. Two criteria were used to analyze designs in these grammars:

(1) length of sentences to be compared, and

(2) the number of extra rules in the grammar.

The latter was taken as an indicator of inconsistency in the language. In addition to the above two criteria, other possibilities were noted but were not explored. These included the number of different terminal symbols (words), the number of alterations in hand or eye movement, the total number of rules needed to describe some subset of the language, and other linguistic measures of sentence complexity.

4.4 Research on User Models

Application of cognitive psychology has been increasingly useful in the area of software development and especially in the design and evaluation of user interfaces. However, confusion in terminology runs rampant throughout the literature in this area. We provide here some definitions (due to [BORG84a], [NORM82], [KIER82]):

* Cognitive model - A model, typically built by a cognitive psychologist, that attempts to describe the mental processes by which humans perform some task. The usual purpose of such a model is to advance our understanding of human behavior.

* User conceptual model - A model, typically built by a designer of a system to provide the user an appropriate representation of the system (appropriate in the sense of being accurate, consistent and complete). This model is not necessarily the same as actual system behavior.

* Mental Model - A model, evolving in the mind of a user, representing the structure and internal relationships of a system, as the user is learning and interacting with the system. This is not a formal model and no one "builds" it. Mental model can be analogical, incomplete, and sometimes very fragmentary with respect to their understanding of how something works.

4.4.1 Cognitive Models

Notable contributions in this area have been made by Stu Card and Tom Moran in collaboration with Allen Newell [CARD83]. Their work involves several different models with different purposes and possible areas of application. The Model Human Processor depicts certain basic processes (e.g., perceptual,

cognitive, motor, and storage of information in perceptual and long term memory) as occurring in discrete cycles. These cycles take time and the time differs for the various processes. Three versions of the model are defined: one in which all the parameters listed are set to give the worst performance (Slowman), one in which they are set to give the best performance (Fastman), and one set for a nominal performance (Middleman). Examples are given of the model's use to describe several types of tasks: perception, motor skill, simple decisions, learning and retrieval, and problem solving. Clearly, this model is too simplified to do justice to the richness and subtlety of the human mind, but it does help to understand, predict, and even to calculate human performance relevant to human-computer interaction.

Another model, called the GOMS Model, attempts to model human problem solving behavior in terms of goals, operators, methods, and selection rules. It has been specifically applied to the modeling of the behavior of expert users of a text-editing system. The GOMS approach is highly task-specific and involves a considerable research investment. The model requires information about expert performance of a task. Goals must be specified which define states of affairs to be evaluated. Perceptual, motor, and cognitive acts are described as operators. Methods need to be established to accomplish a goal. Selection rules are then applied to select a method. The cost of obtaining the estimates

of all different operators and selection rules increases at a finer grain of analysis, because more data are required for a given level of robustness as the observation and measurement problems increase at the lower level. Kieras and Polson [KIER82] have proposed an extension in which the user's task representation is distinguished from the user's device representation. The mapping of user intention to specific actions in which the user manipulates a device is defined in terms of production rules. They propose that the complexity of a particular system, from the user's perspective, might be measurable by measuring the depth of the goal hierarchy and the number of production rules in the model.

4.2.2 User Conceptual Models

Recently, the idea of a "user conceptual model" of a system has begun to be viewed as a formal entity for designing user interfaces.

Several researchers [MORA81a, MAYE81, RUMM81, GENT82, FOSS82] have found that people can learn and apply conceptual models, though these conclusions are not without some constraints and limitations.

Mayer [MAYE81], in a series of studies, has shown that a concrete conceptual model aids in learning the BASIC programming language. The model appears to serve as an "advance organizer" for the material to be learned, but it works only if the model is

presented before the specifics of the material, not after.

4.2.3 Mental Models

Much of the work on mental models and interactive systems has stemmed from the premise that the user possesses a mental model of the system and has explored the characteristics of that mental model [BORG84]. The common approach is to begin without this assumption and attempt to show the existence of the model. Moran [MORA81a] defines the user interface as consisting of "those aspects of the system that the user comes in contact with - physically, perceptually and conceptually." He concludes that "to design the user interface of a system is to design the user model." It is a dominant opinion among many researchers that the system design imposes the user model ([GAIN81], [MORA81a], [NORM81], [YOUN81]).

The majority of the work in user models deals with the application of text editors [DOUG82, FOSS82, HALA82, LEWI82]. It was found that subjects induced a typewriter model for the text editor even though such a model had not been explicitly provided. The patterns of errors and types of misconceptions about system behavior were consonant with the typewriter model in both studies. Moran [MORA81a], Norman [NORM81], and Young [YOUN81] distinguish between giving the user a conceptual model and the system which can be assimilated and forcing the user to infer or induce the system model. In general, it is easier to assimilate a

model than to induce one. Moran [MORA81a], Norman [NORM81], and Rumelhart [RUME81] agree on the importance of providing an explicit and consistent model to the user.

Carroll and Thomas [CARR82] claimed that the activity of learning to use a computer system is structured by metaphoric comparisons. For example, the metaphore "a text editor is a typewriter" could be spontaneously referred to during the early learning period about text processors. Halasz and Moran [HALA82] contrast conceptual models with metaphoric, or analogical, models. By the latter, they mean suggestive but typically incomplete descriptions referring to near-neighbor domains, or to compositions of these. In contrast, their view of a conceptual model is intended to cover highly accurate and arbitrarily complete descriptions (the level of detail matches the needs of the target user) usually in some abstract format, like a flow-chart or a graph. They object to using a single analogical mapping to a computer system. They propose constructing an abstract model based on system behavior. Their views are supported by du Boulay, O'Shea and Monk [DUBO81], Moran [MORA81a], and Young [YOUN81]. They also endorse the use of metaphors to explain smaller units of the system's operation.

Gilfoil [GILF82] studied user's cognitive schema as they learned to use a text editing system. They were given a choice of a menu-driven or command driven interface and the option to switch between them. All users began with the menu-driven style

and gradually switched over as they became more skilled. As they began to switch over, the time per task dropped and the number of semantic errors and the frequency of asking for help dropped drammatically. Gilfoil concludes that user systematically develop a cognitive structure for the task environment. This finding follows other cognitive research (e. g., [MAYE81, SIMO80, CHAS73]).

4.5 Survey of User Interface Management Systems

The recent upsurge in human/computer interaction research has brought an interest in developing tools for the design and implementation of user interfaces. Most of these tools have been developed to support graphical interaction and are applicable in graphics environments. They could well be considered as graphics utilities. This section surveys some of the recent developments in the user interface management area.

Olson [OLSO83a] describes research into the automatic generation of interactive graphical systems to facilitate faster and cheaper generation of interactive user interfaces. This work has not progressed beyond the design stage. He observed that it is the design aspect of program creation which is suited to automatic program generation. This is because of the high cost in time and effort of hand-coding and the increased reliability of automatically generated software. He uses Pascal procedure definitions for the characterization of interactive commands in

the application program.

Kasik [KASI82] describes a system called TIGER which takes care of the bookkeeping associated with screen layout, interrupt handling and the definition of interactive dialogue sequences. The system has at its core the language TICCL, which permits an applications programmer to concentrate on the logical functions which the programmer wishes to perform rather than the low-level physical steps which must be taken to accomplish the task. TICCL can be used to describe algorithms which combine graphical primitives in response to user interactions as well as to define user interaction sequences.

To the extent TICCL is used for constructing graphical primitives for user interactions, it is more advanced than the table driven menu system of the User Interface Management System (UIMS) [BUXT83a]. UIMS has two menu components. The first component is a set of tools to support the design and implementation of interactive graphical programs. The second component is a runtime support package which handles interactions between the system and the user (things such as hit detection, event detection, screen update and procedure invocation). The design/implementation tool is a preprocessor, called MENULAY which permits the application programmer to use interactive graphical techniques and to design graphics menus. The output of this preprocessor is high level code which can be compiled with application specific routines. User interactions with the

resulting executable module are then handled by the runtime support package. Currently, no evaluation of the user interface is supported. The applicability of the current implementation of the preprocessor is restricted to menu based interaction.

FLAIR (Functional Language Articulated Interactive Resources) [WONG82] is a dialogue design tool which enables a system designer to construct graphically a user dialogue for an application program. It is largely driven by voice input and incorporates text picture construction and editing (at the graphical primitive level) as well as dynamic frame layout. FLAIR is a language and package unto itself with no apparent "hooks" into other programming languages. As with UIMS, FLAIR does not have any validation facility or evaluation capacity built into the system.

The design of COUSIN (COoperative User INterface) at Carnegie Mellon [BALL82] is based on the notion of an environment. COUSIN acts as an interactive "environment modifier" through which the user can change the value of any slot in an environment. An environment is a set of named, typed slots which act as communication variables between an application system and its users; the environment is used to specify a command and provide the parameters which control the operation of the application system.

In conclusion, none of the above described systems provides support for monitoring facilities. The proposed system for this study differs from all other systems, whether at the design or production stage, because of its proposed built-in monitoring and evaluation facilities not only for the system itself but also for the user interface generated by the use of this system.

5. PROTOCOLS FOR INTERFACE PROTOTYPING AND EVALUATION SYSTEM

A range of tools are needed for the development and administration of user interfaces for interactive information systems. A collection of such tools is proposed here as a Protocols for Interface Prototyping and Evaluation (PIPE) system. The following sections describe the role of the PIPE, enumerate some of the benefits expected from this approach, provide an overview of the composition of the PIPE, and investigate some implications of the PIPE for information systems.

5.1 Role of the PIPE

The PIPE will mediate the interaction between a user and an application, satisfying user requests for application actions, and application requests for data or commands from the user. It will accept as input a dialogue specification, describing the detailed structure of the interaction. This specification is distinct from any application program, thus allowing for the application programmer's problem-specific skills to be concentrated on application issues and freed from any detailed concern with managing the flow of user action and responses. At the same time, a dialogue specification provides for the human factors skill of a user interface designer to be applied to improving the quality of the interaction, without detailed

concern for the techniques used to solve the application problem.

Here, an analogy is drawn between the PIPE and a Data Base Management System (DBMS). The DBMS frees the application programmer from detailed concern with the management of physical data storage and retrieval, and allows for the specific skills of the data base application programmer to be applied to the specifics of the application.

The building of the user interface of an information system using PIPE will provide the following advantages:

1. The knowledge required to construct a good interface is diffuse, uncertain, and hard to acquire [THOM83]. Only specialist user interface designers are likely to be able to devote enough effort to acquire such knowledge. The PIPE can be an essential tool for exploiting their skills, since it improves the efficiency with which these skills can be applied.

2. Without a sound methodology, each user interface design is likely to proceed in a time consuming and ad hoc fashion. The PIPE should accelerate the design process, permitting a much wider range of alternatives to be examined.

3. Prototyping via the PIPE should represent a valuable means of liaison with prospective users.

4. Experiments involving re-implementing entire applications are prohibitively expensive. The ease with which user interfaces can be revised using the PIPE should make realistic and cost-effective experimentation possible. It also should provide a basis for instrumenting user interfaces to gather information, for example on the evolution of patterns of use.

5. The PIPE should provide the capability to adapt to different user profiles.

6. The system should make it easier to integrate new application functions into the user interface, and assist in ensuring a uniform interface as new applications are developed.

7. The system should provide for applications to be portable, while allowing a resulting user interface to be tailored to a particular installation, while preserving user interface quality.

5.2 The Proposed System

This section is devoted to a brief overview of the proposed system, PIPE, to be developed on an IBM Personal Computer using the C language. The PIPE will be composed of two main components, the User Interface Design Subsystem (UIDS) and the User Interface Execution Subsystem (UIES). The UIDS supports the

user interface designer. It will provide tools for describing display layouts, dialogue structures, and interactions with application programs. The UIDS will allow the generation of a detailed specification of the user interface that can automatically be converted into the C language code required to implement the specification. The UIES will support the execution of the user interface generated by an interface designer using the UIDS. Following is a description of these components of PIPE.

5.2.1 User Interface Design Subsystem (UIDS)

This subsystem will provide support to the user interface designer at three different levels of the user interface: Interactive Level, Dialogue Manager Level, and Application Interface Level.

5.2.1.1 Interactive Level

The design of the Interactive Level will be supported by the following three modules:

1. Screen Layout Specification Module. The designer will be able to name a screen, specify background color, size of the windows, border color, foreground color and an Event-ID associated with the window to indicate when the window is to be displayed. A separate window could be defined for system prompts, user input, system responses including system error messages and system

state. A menu can be associated with each of the windows.

2. Interaction Technique Specification Module. The designer will be able to associate an interaction technique with a window. He will be able to select the interaction technique from a library or construct his own interaction technique.

3. Display Function Specification Module. The designer will be able to specify the name of a display function and a window where the information will be displayed. The major dialogue types supported would be: menu selection, user initiated command language, function keys with command language, form-filling, and question and answer.

5.2.1.2 Dialogue Manager Level

This level will allow the specification of event handlers. All event descriptions will be stored in a data base and these will be converted into an executable form.

5.2.1.3 Application Interface Level

The designer will specify the descriptions of all application data structures and routines that are accessible to the user interface. The description of application data structures will include the type of information stored and how it

is structured. The description of the application's routines will include the name of the routine and the number and type of its parameters. The description might also include the constraints on the use of the routines besides pre-conditions and post-conditions.

5.2.2 User Interface Execution Subsystem (UIES)

The UIES will be the central core of the PIPE. The application software will interact with the UIES when the information system is 'live'. The UIES supervises and implements the interface specified in the Event Description Table, essentially acting as an interpreter for the events.

The UIES will be transparent to the user and will be composed of the following components:

1. Feedback Generator. This component will generate appropriate user feedback. Initially, it will simply consult a standard set of messages which will be customized to suit the feedback required.

2. Adaptive Interface Handler. This component will control adaptation of the dialogue. It will simply check user input and will honor a change of interface request by informing other components of the UIES that the interface level has been changed. Effectively, this will act as a filter in the input stream.

3. Buffered, I/O Handler. If the data required has already been entered, the UIES will read this directly from an input buffer. If this buffer is empty, then the user will be prompted for the input.

6. SUMMARY OF PROPOSED RESEARCH

This chapter summarizes the proposed research discussed in previous chapters. The basic statement of the problem was discussed in Section 1.1, which stated that the user interface is often designed without serious considerations for the user on the part of the designers. Reasons were given for this problem. It was established that current computer science research is lacking tools and methodologies for the effective design and evaluation of user interfaces. There is a need to provide better methodologies and tools for designing, implementing, maintaining, and evaluating user interfaces for information systems. The proposed research addresses this need.

The research objectives were first stated in general terms in Section 2.1 and then refined into specific research objectives in Section 2.2. The major objectives of this research are: the development of a comprehensive, objective and generalizable methodology for the design and evaluation of user interfaces for interactive information systems; the development of equations and/or analytical models to characterize user behavior and the performance of a designed interface; the design of a prototype system for the development and administration of user interfaces; and the design and use of controlled experiments to support the research and test/validate the proposed methodology.

The proposed methodology was discussed in Chapter 3. Several major concepts of design and evaluation of user interfaces were explored. These concepts were integrated into a preliminary methodology to achieve the desired objectives of this research. A user interface is treated as a virtual machine and its layers were described in Section 3.1. Section 3.2.1 established the criteria for the evaluation of specification methods that describe and organize the user interaction sequences.

A model of user system interaction was proposed in Section 3.2.2. In this model, "command language" interactions are viewed as dialogues between two parties. The dialogues are represented as a sequence of basic interaction events. The entire event is viewed as a process made up of four main phases: system prompt, user input, system action and response, and transition control. The transition control allows the selection of the next event based on the familiar structured programming control primitives (sequence, case, do-while). It was shown that, using this model, various modes of interactions could be defined and implemented.

Section 3.3 discussed the implementation methodology for the proposed interaction model. The dialogue logic is separated from the actual text of messages between the user and the system. This separation allows for quick skeletal implementation which can be augmented later.

A general evaluation methodology was presented in Section 3.4. The proposed methodology will allow interface developers to evaluate user interfaces from the viewpoint of the performance of their users. A software monitor will automatically be generated and incorporated into a designed interface. From the data collected by the monitor, an interface developer/evaluator will be able to extract human performance data indicating "ease of learning" and "ease of use" of the interface.

Chapter 4 summarized current research studies relevant to user interface issues. A survey of user interface management systems was presented and it was pointed out that the evaluation component is virtually non-existent among these systems. The proposed system for this research differs from all other systems, because of its proposed built-in monitoring and evaluation facilities, not only for the system itself, but also for the user interface generated by the use of this system.

Finally, Chapter 5 provided a brief overview of the system, PIPE (Protocols for Interface Prototyping and Evaluation), being developed in support of this proposed research.

REFERENCES

APPE82 Apperley, M. D. and Spence, R., "Hierarchical Dialogue Structures in Interactive Computer Systems," Software-Practice and Experience, 13, 1983, pp. 777-790.

ATWO84 Atwood, M. E., "A Report on the Vail Workshop on Human Factors in Computer Systems," IEEE Computer Graphics and Applications, December 1984, pp. 48-66.

BAIL83 Baily, J. E. and Pearson, S. W., "Development of a Tool for Measuring and Analyzing Computer User Satisfaction," Management Science, 29 (6), 1983, pp. 519-529.

BALL82 Ball, E. and Hayes, P., "A Test-Bed for User Interface Designs," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 85-88.

BARB83 Barber, R. E. and Lucas, H. C., "System Response Time Operator Productivity and Job Satisfaction," Communications of the ACM, 26 (11), November 1983, pp. 972-986.

BARN81 Barnard, P. J., Hammond, N. V., Morton, J., Long, J., and Clark, I. A., "Consistency and Compatibility in Human Computer Dialog," International Journal of Man-Machine Studies, 15, 1981, pp. 87-137.

BARN82 Barnard, P., Hammond, N., MacLean, A., and Morton, J., "Learning and Remembering Interactive Commands," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 2-7.

BARR81 Barron, J., Dialogue and Process Design for Interactive Information Systems Using TAXIS, Tech. Rep. CSRG-128, Dept. of Computer Science, University of Toronto, Toronto, Canada, April 1981.

BASS85 Bass, L. J., "An Approach to User Specification of Interactive Display Interfaces," IEEE Transactions on Software Engineering, SE-11 (8), 1985, pp. 686-698.

BATC81 Batchelor, W. J. and Endicott, L. J., "An Experimental System to Support a Very High Level User Interface," Proceedings: AFIPS National Computer Conference, 1981, pp. 389-392.

BENB81 Benbasat, I., Dexter, A. S., and Masulis, P. S., "An Experimental Study of the Human/Computer Interface," Communications of the ACM, 24 (11), 1981, pp. 752-762.

BENB84 Benbasat, I. and Wand, Y., "A Structured Appraoch to Designing Human-Computer Dialogues," International Journal of Man-Machine Studies, 21, 1984, pp. 105-126.

BLAC81 Black, J. B. and Sebrechts, M. M., "Facilitating Human-Computer Communications," Applied Psycholinguistics, 2, 1981, pp. 149-177.

BLAC82 Black, J. B. and Moran, T. P., "Learning and Remembering Command Names," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 8-11.

BLEH80 Bleher, J. H., Caspers, P. G., Henn, H., and Maerker, K., "A Graphic Interactive Application Monitor," IBM System Journal, 19 (3), 1980, pp. 382-402.

BLES82 Bleser, T. and Foley. J. D., "Toward Specifying and Evaluating the Human Factors of User-Computer Interfaces," Proceedings: Human Factors in Computer Systems, Gaithersburg, Md., March 15-17, 1982, pp. 309-314.

BLUM82 Blum, B. I. and Houghton, R. C. Jr., "Rapid Prototyping of Information Management System," ACM SIGSOFT Software Engineering Notes, 7 (5), December 1982, pp. 35-38.

BOKE80 BO, Ketil, "Problems of the 80's in Man/Machine Communication," in Guedj, R. A., tenHagen, P. J., Hopgood, F. R., Tucker, H. A., and Duce, D. A., (Eds.), Methodology of Interaction, North-Holland, Amsterdam, 1980, pp. 149-158.

BORG82 Borgman, C. L., "Mental Models: Ways of Looking at a System," ASIS Bulletin, December 1982, pp. 38-39.

BORG84a Borgman, C. L., The User's Mental Model of an Information Retrieval System: Effects on Performance, Ph.D. Dissertation, Stanford University, California, 1984, 378p.

BORG84b Borgman, C. L., "Psychological Research in Human-Computer Interaction," Annual Review of Information Science and Technology, 19, 1984, pp. 33-64.

BORM78 Borman, L. and Dominick, W. D., Profile Evaluation, Research and Modelling for Science Information Systems: A Report on the Development of a Generalized Evaluation Methodology to Study User Interaction, Final Report, NSF DSI76-19481, Northwestern University, Evanston, Illinois, June 1978, 158p.

BORU82 Borufka, H. G., tenHagen, P. J., "Dialogue Cells: A Method for Defining Interactions," IEEE Computer Graphics and Applications, July 1982, pp. 25-32.

BRAN84 Branscomb, L. M. and Thomas, J. C., "Ease of Use: A System Design Challenge," IBM System Journal, 2 (3), 1984, pp. 224-235.

BROW83 Brown, P. J., "Error Messages. The Neglected Area of the Man-Machine Interface?," Communications of the ACM, 26 (4), 1983, pp. 246-249.

BUXT83a Buxton, W., Lamb, M. R., Sherman, D., and Smith, K. C., "Towards a Comprehensive User Interface Management System," Computer Graphics, 17 (3), July 1983, pp. 35-42.

BUXT83b Buxton, W., "Lexical and Pragmatic Considerations of Input Structure," Computer Graphics, January 1983, pp. 31-37.

CARD80 Card, S. K. and Moran, T. P., "The Keystroke-Level Model for User Performance Time with Interactive Systems," Communications of the ACM, 23, 1980, pp. 396-410.

CARD83 Card, S. K., Moran, T. P., and Newell, A., The Psychology of Human-Computer Interaction, Lawrence Erlbaum Associates, Hillsdale, N. J., 1983.

CARR82 Carroll, J. M. and Mack, R. L., "Metaphor and Cognitive Representation of Computing Systems," IEEE Transactions on Systems, Man and Cybernetics, 12, 1982, pp. 107-116.

CARR85 Carroll, J. M. and Mack, R. L., "Metaphor, Computing Systems and Active Learning," International Journal of Man-Machine Studies, 22 (1), January 1985, pp. 39-57.

CHAS73 Chase, W. G. and Simon, H. A., "Perception in Chess," Cognitive Psychology, 4, 1973, pp. 55-81.

CHIU85 Chi, U. I., "Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches," IEEE Transactions on Software Engineering, SE-11 (8), 1985, pp. 671-685.

COCH84 Cochran, D. R., Hobbs, R. W., and Meason, R. N., Survey and Analysis of User Computer Interface Rapid Prototyping Tools, Computer Technology Associates Technical Report, Landover, Md., February 1984.

CROF84 Croft, W. B., "The Role of Context and Adaptation in User Interfaces," International Journal of Man-Machine Studies, 21, 1984, pp. 283-292.

CUFF80 Cuff, R. N., "On Casual Users," International Journal of Man-Machine Studies, 12, 1980, pp. 163-187.

CURT84 Curtis, B., Forman, I., Brooks, R., Soloway, E., and Ehrlich, K., "Psychological Perspectives for Software Science," Information Processing and Management, 20 (1-2), 1984, pp. 81-96.

CYER63 Cyert, R. M. and March, J. G., A Behavioral Theory of the Firm, Prentice-Hall, Englewood Cliffs, N. J., 1963.

DAGW83 Dagwell, R. and Weber, W., "System Designers' User Models: A Comparative Study and Methodological Critique," Communications of the ACM, 26 (11), November 1983, pp. 987-997.

DATE84 Date, C. J., "Some Principles of Good Language Design, ACM SIGMOD RECORD 14 (3), November 1984, pp. 1-7.

DAVI84 Davis, J. B., "Chunks: A Basis for Complexity Measurement," Information Processing and Management, 20 (1-2), 1984. pp. 119-127

DAVI83 Davis, R., "Task Analysis and User Errors: A Methodology for Assessing Interactions," International Journal of Man-Machine Studies, 19, 1983, pp. 561-574.

DEGR80 DeGreene, K. B., "Major Conceptual Problems in the System Management of Human Factor/Ergonomics Research," Ergonomics, 23, 1980, pp. 3-11.

DEME81 Demers, R. A., "System Design for Usability," Communications of the ACM, 24, 1981, pp. 494-501.

DOMI78 Dominick, W. D. and Urban J. E., Application of a Generalized Evaluation Methodology for Analyzing User Interaction with the MADAM System at the University of Southwestern Louisiana, Technical Report CMPS-78-6-3, Computer Science Department, University of Southwestern Louisiana, Lafayette, Louisiana, September 1978, 347p.

DOMI79 Dominick, W. D. and Penniman, W. D., "Automated Monitoring to Support the Analysis and Evaluation of Information Systems," Proceedings of the Second International Conference on Information Storage and Retrieval. ACM SIGIR FORUM, XIV (2), September 27-28, 1979, pp. 2-9.

DOMI84 Dominick, W. D., "The USL NASA PC R&D Project: Specifications of Objectives," USL/DBMS NASA/PC R&D Working Paper Series, Report Number DBMS.NASA/PC R&D-2, June 6, 1984, 6p.

DOUG82 Douglas, S. A. and Moran, T. P., "Learning Text Editing Semantics by Analogy," Proceedings of the SIGCHI and the Human Factors Society Conference on Human Factors in Computing Systems, Boston, MA, December 12-15, 1983, pp. 207-211.

DRAP84 Draper, S. W. and Norman, D. A., "Software Engineering for User Interfaces," Proceedings: 7th International Conference on Software Engineering, Orlando, Fla., March 26-29, 1984, pp. 214-220.

DRAY81 Dray, S. M., Ogdan, N. G., and Vestewig, R. E., "Measuring Performance with a Menu-Selection Human-Computer Interface," Proceedings of the 25th Annual Meetings of the Human Factor Society, Rochester, N. Y., 1981.

DUBO81 Du Boulay, B., O, Shea, T., and Monk, J., "The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices," International Journal of Man-Machine Studies, 14, 1981, pp. 237-250.

DUNS82a Dunsmore, H. E., Using Formal Grammer as a Design Tool to Predict the Most Useful Characteristics of Interactive Systems, Office Automation Conference Digest, San Francisco, AFIPS Press, 1982, pp. 53-56.

DUNS82b Dunsmore, H. E., and Reisner, P., "Some Further Evidence on the Formal Grammer Application to Human Factors Research," Technical Report 348, Department of Computer Sciences, Purdue University, 1982.

DURH83 Durham, I., Lamb, D., and Saxe, J., "Spelling Corrections in User Interfaces," Communications of the ACM, 26 (10), October 1983, pp. 764-773.

DURR82 Durrett, J. and Stimmel, T., "A Production System Model of Human-Computer Interaction," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 393-399.

DZID78 Dzida, W., Herada, S. L., and Itzfeldt, W. D., "User Perceived Quality of Interactive Systems," IEEE Transactions on Software Engineering, SE-4, 1978, pp. 270-276.

EASO80 Eason, K. D., "Dialogue Design Implications of Task Allocation between Man and Computer," Ergonomics, 23, 1980, pp. 881-891.

EDMO81 Edmonds, E. A., "Adaptive Man-Computer interfaces," in Coombs, M. J., and Alty, J. L., (Eds.), Computing Skills and the User Interface, Academic Press, London, 1981, pp. 389-426.

EDMO82 Edmonds, E., "The Man-Computer Interface: A Note on Concepts and Design," International Journal of Man-Machine Studies, 16, 1982, pp. 231-236.

EHRI83 Ehrich, R. W., "DMS - A System for Defining and Managing Human Computer Dialogues," Automatica, 9 (6), 1983, pp. 655-662.

EMBL78a Embley, D. W., "Empirical and Formal Language Design Applied to a Unified Control Construct for Interactive Computing," International Journal of Man-Machine Studies, 10 (2), March 1978, pp. 197-216.

EMBL78b Embley, D. W., Lan, M. T., Leinbaugh, D. W., and Nagy, G., "A Procedure for Predicting Program Editor Performance from the User's Point of View," International Journal of Man-Machine Studies, 10, 1978, pp. 639-650.

FELD82 Feldman, M. and Rogers, G., "Toward the Design and Development of Style-Independent Interactive Systems," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 89-100.

FENI81 Fenichel, C. H., "Online Searching: Measures that Discriminate among Users with Different Types of Experiences," Journal of the ASIS, January 1981, pp. 23-32.

FERR83 Ferrari, D., Serrazi, G., and Zeigner, A., Measurement and Tuning of Computer Systems, Prentice-Hall, 1983.

FEYO77 Feyock, S., "Transition Diagram-Based CAI/Help Systems," International Journal of Man-Machine Studies, 9, 1977, pp. 399-413.

FOLE74 Foley, J. D. and Wallace, V. L., "The Art of Natural Graphic Man-Machine Conversation," Proceedings: IEEE, 62, April 1974, pp. 462-471.

FOLE80 Foley, J. D., "The Structure of Interactive Command Languages," in Guedj, R. A., tenHagen, P. J., Hopgood, F. R., Tucker, H. A., and Duce, D. A. (Eds.), Methodology of Interaction, North-Holland, Amsterdam, 1980, pp. 227-234.

FOLE82 Foley, J. D. and Van Dam, A., Fundamentals of Computer Graphics, Addison-Wesley, Reading, MA., 1982.

FOLE84 Foley, J. D., Wllace, V. L., and Chan, P., "The Human Factors of Computer Graphics Interaction Techniques," IEEE Computer Graphics and Applications, Nov. 1984, pp. 13-48.

FOSS82 Foss, D. J., Rosson, M. D., and Smith, P. L., "Reducing Manual Labor: An Experimental Analysis of Learning Aids for a Text Editor," Proceedings: Human Factors in Computer Systems, Gaithersburg, M. D., March 15-17, 1982, pp. 332-336.

GAIN81 Gaines, B. R., "The Technology of Interaction-Dialogue Programming Rules," International Journal of Man-Machine Studies, 14 (1), 1981, pp. 133-150.

GEBH78 Gebhardt, F. and Stellmacher, I. "Opinion Papere: Design Criteria for Documentation Retrieval Languages," Journal of the ASIS, July 1978, pp. 191-199.

GELL83 Geller, V. J. and Lesk, M. E., "User Interfaces to Information Systems: Choices vs. Commands," Proceedings: 6th International ACM SIGIR Conference, Bethsada, MD., June 6-8, 1983, pp. 130-135.

GENT82 Gentner, D. and Gentner, D. R., "Flowing Waters or Teeming Crowds: Mental Models of Electricity," in Gentner, D., and Stevens, A. S., (Eds.), Mental Models, Lawrence Erlbaum Associates, Hillsdale, N. J., 1982, pp. 99-129.

GILF82 Gilfoil, D. M., "Warming Up to Computers: A Study of Cognitive and Affective Interaction over Time," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 245-250.

GOOD82 Good, M., "An Ease of Use Evaluation of an Integrated Document Processing System," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 142-147.

GOUL84 Gould, J. D. and Bois, S. J., "Speech Filing - An Office System for Principals," IBM System Journal, 23 (1), 1984, pp. 65-81.

GREE81 Green, M., "A Methodology for the Specification of Graphical User Interfaces," Computer Graphics, 15 (3), August 1981, pp. 99-108.

GREE83 Green, M., "A Catalogue of Graphical Interaction Techniques," Computer Graphics, January 1983, pp. 46-52.

GREE85 Green, M., "The University of Alberta User Interface Management System," ACM SIGGRAPH, 19 (3), 1985, pp. 205-213.

GUED80 Guedj, R. A., "Remarks on Some Aspects of Man-Machine Interaction," in Guedj, R. A., tenHagen, P. J., Hopgood, F. R., Tucker, H. A., and Duce, D. A., (Eds.), Methodology of Interaction, North-Holland, Amsterdam, 1980, pp. 235-238.

GUES82 Guest, S. P., "The Use of Software Tools for Dialogue Designs," International Journal of Man-Machine Studies, 16, 1982, pp. 263-285.

HAGG83 Hagglund, S. and Tibell, R., "Multi-Style Dialogues and Control Independence in Interactive Software," in Green, T. R., Payne, S. J., and van der Veer, G. C., (Eds.), The Psychology of Computer Use, Academic Press, London, 1984, pp. 171-189.

HALA82 Halasz, F. G. and Moran, T. P., "Analogy Considered Harmful, Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 383-386.

HALL84 Hall, P. P., "Design Criteria for a PC-Based Common User Interface to Remote Information Systems," USL/DBMS NASA PC/R&D Working Paper Series, Report Number DBMS.NASA/PC R&D-9, August 13, 1984, 21p.

HALL85a Hall, P. P., The Design of PC/MISI, A PC-Based Common User Interface to Remote Information Storage and Retrieval Systems, Master Thesis, University of Southwestern Louisiana, Lafayette, Louisiana, 1985, 83p.

HALL85b Hall, P. P., "PC-Based Multiple Information System Interface - Detailed Design and Implementation Plan," USL/DBMS NASA/PC R&D Working Paper Series, Report Number DBMS.NASA/PC R&D-16, April 10, 1985, 69p.

HALS77 Halstead, M. H., Elements of Software Science, Elsevier North-Holland, New York, 1977.

HANA80 Hanau, P. R. and Lenorovitz, D. R., "Prototyping and Simulation Tools for User/Computer Dialogue Design," Computer Graphics, 14 (3), July 1980, pp. 271-278.

HARD82 Hardy, I. T. Jr., "The Syntax of Interactive Command Languages: A Framework for Design," Software-Practice and Experience, 12, 1982, pp. 67-75.

HART79 Hartson, H. R., and Schnetzler, M. D., Generalized Interactive User Interface Dialog Designer, Computer Science Working Paper, Virginia Polytechnic Institute and State University, Blacksburg, VA., 1979.

HAUP83 Hauptmann, A. G. and Green, B. F., "A Comparison of Command, Menu-Selection and Natural-Language Computer Programs," Behavior and Information Technology, 2 (2), 1983, pp. 163-178.

HAYE81 Hayes, P., Ball, E., and Reddy, R., "Breaking the Man-Machine Communication Barrier," IEEE Computer, 14 (3), 1981, pp. 19-30.

HODG85 Hodgson, G. M. and Ruth, S. R., "The Use of Menus in the Design of On-Line Systems: A Retrospective View," SIGCHI Bulletin, 17 (1), 1985, pp. 16-22.

HOLC85 Holcomb, R. and Tharp, A. L., "The Effect of Windows on Man-Machine Interfaces (or opening doors with windows)," Proceedings of the 1985 ACM Computer Science Conference, New Orleans, LA, March 12-14, 1985, pp. 280-291.

HORO85 Horowitz, E., Kemper, A., and Narasimhan, B., "A Survey of Application Generators," IEEE Software, January 1985, pp. 40-54.

INNO82 Innocent, P. R., "Towards Self-Adaptive Interface Systems," International Journal of Man-Machine Studies, 16, 1982, pp. 287-299.

IVES83 Ives, B., Olsen, M. H., and Baroudi, J. J., "The Measurement of User Information Satisfaction," Communications of the ACM, 26 (10), October 1983, pp. 785-793.

JACO83a Jacob, R. K., "Using Formal Specifications in the Design of a Human-Computer Interface," Communications of the ACM, 26 (4), April 1983, pp. 259-264.

JACO83b Jacob, R. K., Survey and Examples of Specification Techniques for User Interfaces, NRL Report, Naval Research Laboratory, Washington, D. C., 1983.

KAMR83 Kamran, A. and Feldman, M. B., "Graphic Programming Independent of Interaction Techniques and Styles," Computer Graphics, January 1983, pp. 58-66.

KASI82 Kasik, D. J., "A User Interface Management System," Computer Graphics, 16 (3), July, 1982, pp. 99-106.

KIER82 Kieras, D. E. and Polson, P. G., An Approach to the Formal Analysis of User Complexity, Project on the User Complexity of Devices and Systems, Working Paper No. 2., University of Arizona and University of Colorado, 1982.

KRAU80 Krause, J., "Natural Language Access to Informtion Systems: An Evaluation Study of its Acceptance by End Users," Information Systems, 5, 1980, pp. 297-318.

LAND83 Landauer, T. K., Galotti, K. M., and Hartwell, S., "Natural Command Names and Initial Learning: A Study of Text Editing Terms," Communications of the ACM, 26 (7), July 1983, pp. 495-503.

LEDG80 Ledgard, H., Whiteside, J. A., Singer, A., and Seymour, W., "The Natural Language of Interactive Systems," Communications of the ACM, 23, 1980, pp. 556-563.

LEWI82 Lewis, C. A. and Mack, R., "Learning to Use a Text Processing System: Eveidence from 'Thinking Aloud' Protocols," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 387-392.

LIEB85 Lieberman, H., "There's More to Menu System Than Meets the Screen," ACM SIGGRAPH, 19 (3), 1985, pp. 181-189.

LIND85 Lindquist, T. E., "Assessing the Usability of Human-Computer Interfaces," IEEE Software, January 1985, pp. 74-82.

LING82 Ling, M. M., "Designing Data Entry Programs Using State Diagrams as a Common Model," Proceedings: 6th International Conference on Software Engineering, Tokyo, Japan, September 13-16, 1982, pp. 296-308.

LINS82 Lin Sin Cho, J. R., "Automatic Report Formatting from a Report Specification," The Computer Journal, 25 (2), 1982, pp. 242-247.

LISK75 Liskov, B. H. and Zilles, S., "Specification Techniques for Data Abstractions," Proceedings: International Conference on Reliable Software, ACM SIGPLAN, 10 (6), 1975, pp. 72-87.

MAED84 Maeda, K., Miyake, Y., Nivergelt, J., and Saito, Y., "A Comparative Study of the Man-Machine Interfaces in Interactive Systems," SIGCHI Bulletin, 16 (2), Oct. 1984, pp. 44-61.

MAGU82 Maguire, M., "An Evaluation of the Published Recommendations on the Design of Man-Computer Dialogues," International Journal of Man-Machine Studies, 16, 1982, pp. 237-261

MALL82 Mallgren, W. R., Formal Specification of Interactive Graphics Programming Languages, The MIT Press, Cambridge, MA., 1982.

MALO82 Malone, T. W., "Heuristics for Designing Enjoyable User Interfaces: Lessons from Computer Games," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 63-68.

MART73 Martin, J., Design of Man-Computer Dialogues, Prentice-Hall, Englewood Cliffs, N. J., 1973.

MART82 Martin, J., Application Development Without Programmers, Prentice-Hall, Englewood Cliffs, N. J., 1982.

MASO83 Mason, R. E. A. and Carey, T. T., "Prototyping Interactive Information Systems," Communications of the ACM, 26 (5), May 1983, pp. 347-354.

MAYE79 Mayer, R. E., "A Psychology of Learning BASIC," Communications of the ACM, 22, 1979, pp. 589-593.

MAYE81 Mayer, R. E., "The Psychology of How Novices Learn Computer Programming," ACM Computing Surveys, 13, 1981, pp. 121-141.

MEAD78 Mead, R. L. and Schwetman, H. D., "Job Scripts - A Workload Description Based on System Event Data," Proceedings: AFIPS National Computer Conference, 1978, pp. 457-464.

MICH81 Michelsen, C. D., The Objective Evaluation of IS&R/DBMS Systems Utilizing Software Engineering Principles, Ph. D. Dissertation, University of Southwestern Louisiana, Lafayette, Louisiana, August 1981, 365p.

MILL56 Miller, G. A., "The Magical Number Seven Plus or Minus Two: Some Limits on our Capacity for Processing Information," Psychological Review, 63, 1956, pp. 81-97.

MILL77 Miller, L. A., and Thomas, J. C., "Behavioral Issues in the Use of Interactive Systems," International Journal of Man-Machine Studies, 9, 1977, pp. 509-536.

MILL71 Miller, R. B., Human Ease of Use Criteria and Their Tradeoffs Technical Report TR00.2185, IBM Poughkeepsie Laboratory, April 12, 1971, 16p.

MORA80 Moran, T. P., "A Framework for Studying Human-Computer Interaction," in Guedj, R. A., tenHagen, P. J., Hopgood, F. R., Tucker, H. A., and Duce, D. A., (Eds.), Methodology of Interaction, North-Holland, 1980, pp. 293-301.

MORA81a Moran, T. P., "The Command Language Grammer: A Representation for the User Interface of Interactive Computer Systems," International Journal of Man-Machine Studies, 15, 1981, pp. 3-50.

MORA81b Moran, T. P., "Guest Editors Introduction: An Applied Psychology of the User," ACM Computing Surveys, 13, 1981, pp. 1-12.

MORL83 Morland, D. V., "Human Factors Guidelines for Terminal Interface Design," Communications of the ACM, 26 (7), 1983.

MOZE82 Mozeico, H., "A Human/Computer Interface to Accomodate User Learning Stages," Communications of the ACM, 25 (2), 1982, pp. 100-104.

MUMP77 MUMPS Development Commitee, MUMPS Language Standard, American National Standards Institute, New York, 1977.

MYLO80 Mylopoulos, J., Bernstein, P., and Wong, H., "A Language Facility for the Design of Interactive Database-Intensive Applications," Transactions on Database Systems, 5 (2), June 1980, pp. 185-207.

NEWM68 Newman, W. M., "A System for Interactive Graphical Programming," Proceedings: AFIPS Spring Joint Computer Conference, 1968, pp. 47-54.

NICK81 Nickerson, R. S., "Why Interactive Computer Systems are Sometimes Not Used by People Who Might Benefit from Them," International Journal of Man-Machine Studies, 15, 1981, pp. 469-483.

NOET82 Noethe, V., "User Behavior at System Command Language Level," Computer Performance, 3 (1), 1982, pp. 5-9.

NOLA74 Nolan, L. E. and Strauss, J. C., "Workload Characterization for Time Sharing System Selection," Software-Practice and Experience, 4 (1), 1974, pp. 25-39.

NORM81 Norman, D. A., "The Trouble with UNIX," Datamation, 27, 1981, pp. 139-150.

NORM82 Norman, D., "Steps Toward a Cognitive Engineering: Design Rules Based on an Analysis of Human Error," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 378-382.

NORM84 Norman, D. A., "Stages and Levels in Human-Machine Interactions," International Journal of Man-Machine Studies, 21, 1984, pp. 365-373.

OBER84 Oberquelle, H., "On Models and Modelling in Human-Computer Co-operation," in van der Veer, G. C., Tauber, M. J., Green, T. R., and Gorny, P., (Eds.), Lecture Notes in Computer Science, 178, Springer-Verlag, 1984, pp. 23-43.

OLSO83a Olson, D. R., "Automatic Generation of Interactive Systems," Computer Graphics, 1983, pp. 53-57.

OLSO83b Olson, D. R., and Dempsey, E. P., "SYNGRAPH: A Graphical User Interface Generator," Computer Graphics, 17 (3), July 1983, pp. 43-50.

OLSO85 Olson, D. R., Dempsey, E. P., and Rogge, R., "Input/Output Linkage in a User Interface Management System," ACM SIGGRAPH, 19 (3), 1985, pp. 191-197.

PARN69 Parnas, D. L., "On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System," Proceedings: 24th National ACM Conference, 1969, pp. 379-385.

PAYN84 Payne, S. J., Sime, M. E., and Green, T. R., "Perceptual Structure Cueing in a Simple Command Language," International Journal of Man-Machine Studies, 21, 1984, pp. 19-29.

PENN80 Penniman, W. D. and Dominick, W. D., "Monitoring and Evaluation of On-Line Information System Usage," Information Processing and Management, 16, 1980, pp. 17-35.

PILO83a Pilote, M., "A Programming Language Framework for the Design of User Interfaces," Proceeding of the Conference on Principles of Programming Languages, ACM, June 1983, pp. 118-136.

PILO83b Pilote, M., A Framework for the Design of Linguistic User Interfaces, Ph. D. Thesis, Dept. of Computer Science, University of Toronto, Toronto, Canada, 1983.

RADH82 Radhakrishnan, T., Grosoner, C., and Benoliel, M., "Design of an Interactive Data Retrieval System for Casual Users," Information Processing and Management, 18 (1), 1982, pp. 23-32.

RAMS83 Ramsey, H. R. and Grimes, J. D., "Human Factors in Interactive Computer Dialog," Annual Review of Information Science and Technology, 18, 1983, pp. 29-59.

REES85 Reese, J., Twiddy, R., Buchannan, L., Tarka, M., and Leung, K. C., "GUIDES: A Tool for Rapid Prototyping of User-Computer Interfaces," Proceedings of the 1985 ACM Computer Science Conference, March 12-14, 1985, pp. 272-279.

REIS81 Reisner, P., "Formal Grammer and Human Factor Design of an Interactive Graphic System," IEEE Transactions on Software Engineering, SE-7 (2), 1981, pp. 229-240.

REIS83 Reisner, P., "Analytical Tools for Human Factors of Software," in Blaser, A., and Zoeppritz, M., (Eds.), Lecture Notes in Computer Science, Springer-Verlag, 150, 1983, pp. 94-121.

REIS84 Reisner, P., "Formal Grammer as a Tool for Analyzing Ease of Use: Some Fundamental Concepts," in Thomas, J. C., and Schneider, M. L., (Eds.), Human Factors in Computer Systems, Ablex Publishing Corp., Norwood, N. J., 1984, pp. 53-78.

RICH83 Rich, E., "Users are individuals; Individualizing User Models," International Journal of Man-Machine Studies, 18, 1983, pp. 199-214.

ROAC82 Roach, J., Hartson, H. R., Ehrich, R., Yunten, T., and Johnson, D., "DMS: A Comprehensive System for Managing Human-Computer Dialogue," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 102-105.

ROBE83 Roberts, T. L. and Moran, T. P., "The Evaluation of Text Editors: Methodology and Empirical Results," Communications of the ACM, 26 (4), April 1983, pp. 265-283.

ROBI85 Robinson, J. and Burns, A., "A Dialogue Development System for the Design and Implementation of User Interfaces in Ada," The Computer Journal, 28 (1), 1985, pp. 22-28.

ROHR84 Rohr, G. and Tauber, M., "Representational Frameworks and Models for Human-Computer Interfaces," in van der Veer, G. C., Tauber, M. J., Green, T. R., and Gorny, P., (Eds.), Lecture Notes in Computer Science, Springer-Verlag, 178, 1984, pp. 8-25.

ROSE82 Rosenberg, J., "Evaluating the Suggestiveness of Command Names," Proceedings: Human Factors in Computer Systems, Gaithersburg, MD., March 15-17, 1982, pp. 12-16.

ROSE83 Rosenberg, J., "Featural Approach to Command Names," Proceedings CHI '83, Human Factors in Computer Systems, ACM, 1983, pp. 116-119.

ROWE83 Rowe, L. A., and Shoens, K. A., "Programming Language Constructs for Screen Definition," IEEE Transactions on Software Engineering, 9 (1), January 1983, pp. 31-39.

RUME81 Rumelhart, D. E. and Norman, D. A., "Analogical Processes in Learning," in Anderson, J. R., (Ed.), Cognitive Skills and Their Acquisition, Lawrence Erlbaum Associates, Hillsdale, N. J., 1981, pp. 335-359.

SAJA85 Saja, A. D., "The Cognitive Model: An Approach to Designing the Human Computer Interface," SIGCHI Bulletin, 16 (3), 1985, pp. 36-40.

SAVA84 Savage, R. E. and Habineck, J. K., "A Multilevel Menu-Driven User Interface: Design and Evaluation through Simulation," in Thomas, J. C., and Schneider, M. L., (Eds.), Human Factors in Computer Systems, Ablex Publishing Corp., Norwood, N. J., 1984, pp. 165-186.

SEWA75 Seward, H. H., "Evaluating Information Systems," in McFarlan, F. W., and Nolan, R. L., (Eds.), The Information Systems Handbook, Dow Jones-Irwin, Inc., Homewood, Il., 1975, pp. 132-153.

SCHV84 Schvaneveldt, R., Cooke, N., Durso, F., Onorato, L., and Bailey, G., "A Taxanomy of Human-Computer Interactions: Toward a Modular User Interface," in Salvendy, G., (Ed.), Human-Computer Interaction, Elsevier Science Publishers, Amsterdam, 1984, pp. 121-124.

SHNE79 Shneiderman, B., "Human Factors Experiments in Designing Interactive Systems," IEEE Computer, 12 (12), December 1979, pp. 9-19.

SHNE80 Shneiderman, B., Software Psychology - Human Factors in Computer and Information Systems, Winthrop Publishers, Inc., Cambridge, MA, 1980.

SHNE82 Shneiderman, B., "Multiparty Grammars and Related Features for Defining Interactive Systems," IEEE Transactions on Systems, Man and Cybernetics, SMC-12 (2), March/April 1982, pp. 148-154.

SIMO80 Simon, H. A., "Problem Solving and Education," in Tuma, D. T., and Reif, F., (Eds.), Problem Solving and Education: Issues in Teaching and Research, Lawrence Erlbaum Associates, Hillsdale, N. J., 1980.

SING79 Singer, A., Formal Methods and Human Factors in the Design of Interactive Languages, Ph. D. Dissertation, Computer Information Science Dept., University of Massachussettes, 1979.

SMIT84 Smith, R. G., Lafue, G. M., Schoen, E., and Vestal, S. C., "Declarative Task Description as a User-Interface Structuring Mechanism," IEEE Computer, September 1984, pp. 29-37.

SPEI83 Speigler, I., "Modelling Man-Machine Interface in a Data Base Environment," International Journal of Man-Machine Studies, 18, 1983, pp. 55-70.

STUD84 Studer, R., "Abstract Models of Dialogue Concepts," Proceedings: 7th International Conference on Software Engineering, Orlando, Fla, March 26-29, 1984, pp. 420-429.

TAVE85 Tavendale, R. D., "A Technique for Prototyping Directly from a Specification," Proceedings: 8th International Conference on Software Engineering, London, U. K., August 28-30, 1985, pp. 224-229.

THIM80 Thimbley, H., "Dialogue Determination," International Journal of Man-Machine Studies, 13 (3), 1980, pp. 295-304.

THOM83 Thomas, J. J. and Hamlin, G., "Graphical Input Interaction Technique (GIIT) Workshop Summary," Computer Graphics, January 1983, pp. 5-30.

USNR83 U. S. National Research Council, Research Needs for Human Factors, Committee on Human Factors Commission on Behavioral and Social Sciences and Education, National Academy Press, 1983.

WASS79 Wasserman, A. I. and Stinson, S., "A Specification Method for Interactive Information Systems," Proceedings of the IEEE Computer Society Conference on Specifications of Reliable Software, Cambridge, MA., 1979, pp. 68-79.

WASS81 Wasserman, A. I., "User Software Engineering and the Design of Interactive System," Proceedings: 5th International Conference on Software Engineering, 1981, pp. 387-393.

WASS82 Wasserman, A. I., and Shewmake, D. A., "Rapid Prototyping of Interactive Information Systems," ACM SIGSOFT Software Engineering Notes, 7 (5), December 1982, pp. 171-180.

WASS84 Wasserman, A. I., "Specification and Implementation of Interactive Information Systems," Proceedings: AFIPS National Computer Conference, 53, 1984, pp. 261-266.

WASS85 Wasserman, A. I., "Extending State Transition Diagrams for the Specification of Human-Computer Interaction," IEEE Transactions on Software Engineering, SE-11 (8), August 1985, pp. 699-713.

WILL82 Williams, B., "The Human Side of Information's Converging Technology," ASIS Bulletin, 9 (2), 1982, pp. 24-26.

WILS82 Williges, R. C. and Williges, B. H., "Human-Computer Dialogue Design Considerations," Proceedings: IFAC Analysis, Design and Evaluation of Man-Machine Systems, Baden-Baden, F.R.G., 1982, pp. 239-246.

WONG82 Wong, P. C. and Reid, E. R., "FLAIR - User Interface Dialog Design Tool," Computer Graphics, 16 (3), July 1982, pp. 87-98.

YOUN81 Young, R. M., "The Machine Inside the Machine: Users' Models of Pocket Calculators," International Journal of Man-Machine Studies, 15, 1981, pp. 51-85.

4-25

1. Report No. IN-82	2. Government Accession No. 183570	3. Recipient's Catalog No.
4. Title and Subtitle USL/NGT-19-010-900: A METHODOLOGY FOR THE DESIGN AND EVALUATION OF USER INTERFACES FOR INTERACTIVE INFORMATION SYSTEMS 112 P.		5. Report Date January 22, 1986
		6. Performing Organization Code
7. Author(s) MOHAMMAD U. FAROOQ		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address University of Southwestern Louisiana The Center for Advanced Computer Studies P.O. Box 44330 Lafayette, LA 70504-4330		11. Contract or Grant No. NGT-19-010-900
		13. Type of Report and Period Covered FINAL; 07/01/85 - 12/31/87
12. Sponsoring Agency Name and Address		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This Working Paper Series entry represents the definition of proposed research addressing the development and validation of a methodology for the design and evaluation of user interfaces for interactive information systems. The major objectives of this research are: the development of a comprehensive, objective, and generalizable methodology for the design and evaluation of user interfaces for information systems; the development of equations and/or analytical models to characterize user behavior and the performance of a designed interface; the design of a prototype system for the development and administration of user interfaces; and the design and use of controlled experiments to support the research and test/validate the proposed methodology. The proposed design methodology views the user interface as a virtual machine composed of three layers: an interactive layer, a dialogue manager layer, and an application interface layer. A command language model of user system interactions is presented because of its inherent simplicity and structured approach based on interaction events. All interaction events have a common structure based on common generic elements necessary for a successful dialogue. It is shown that, using this model, various types of interfaces could be designed and implemented to accommodate various categories of users. The implementation methodology is discussed in terms of how to represent the various types of information pertaining to an interaction event, and how to store and organize the information. A generalized evaluation methodology is also proposed for the evaluation of user interfaces. The methodology will allow interface developers to evaluate user interfaces from the viewpoint of the performance of their users.

(Abstract continued on following page)

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Design and Evaluation of User Interfaces, Information Storage and Retrieval Systems	

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	103	

*For sale by the National Technical Information Service, Springfield, Virginia 22161