Tools for building advanced user interfaces

by J. L. Bennett

System developers are noticing that their design decisions strongly affect computer usability. The design of the user interface has an important bearing on the knowledge users must have to accomplish work through the user-computer interface. Recognition of this fact is leading to the development of User Interface Management Systems (UIMSs). A UIMS is a design concept for separating the details of user interaction from the details of advanced applications. This paper shows how UIMS research and research into the representation of user process knowledge (i.e., user how-to-do-it skills) can help developers understand issues involving ease of learning and ease of use. This parallel progress in UIMS development and in user modeling makes it easier to build high-quality advanced user interfaces.

As a result of today's widespread use of the personal computer, developers are being asked to design user interfaces for advanced applications intended for users without data processing experience. These users are not prepared to learn the computer-oriented details typically required of experienced users, and often expect to walk up and use the promised power as easily as they might drive a rented car. But the operating systems supporting these applications were developed for users accustomed to carrying out complicated tasks. Meeting the needs of users who demand power without complication has made industry increasingly sensitive to the design of the user interface.

Developers are looking toward User Interface Management Systems (UIMSs) as a way to meet this demand. A UIMS is a set of services that supports the presentation of data on a workstation and accepts user actions taken in response to the displayed data. By serving as a bridge between a variety of applications and a variety of workstations, a UIMS serves as a tool for simultaneously reducing the cost of development, providing the technical flexibility needed in advanced applications, and meeting the usability requirements of users.

The design decisions made in a particular application affect the presentation to the user, the process of use, and therefore the knowledge needed by the user during interaction. Whereas each application has its own specific objects and actions, different applications have many objects and actions in common, such as text creation and editing of tables of values to be shown and modified. Users expect the processes to be supported across applications in a standard way, and developers want to be able to construct the needed support only once in a UIMS and then use that UIMS to supply interactive services to many different applications.

In this paper we explore examples of both research and applied work now in progress that are beginning to influence the design of user interfaces for advanced computer applications. The first section defines what we mean by the term *user interface*—its location, its properties, and a basis for evaluating it.

We then show how the concept of a UIMS is being developed in order to reduce the cost of constructing user interfaces and to improve their quality. Much of the internal logic of a UIMS (the necessary relationships among its internal parts) can be worked out

independently of any specific interaction style. Designers, however, recognize a need for guidance on how to shape a specific user interface when it is built with the tools provided by a UIMS. For this they need input on how users will judge user interface quality.

We then review models being developed to represent user process knowledge. Cognitive scientists (not the computer scientists developing UIMS concepts) are exploring formal methods for representing the thought and action processes required of the user by a particular interface design. Thus, if the relative effects of alternative design decisions are understood early in the development process, developers can make modifications that improve ease of learning and ease of use for the eventual users.

Cited references illustrate work in progress but are not intended to be comprehensive in coverage. While several IBM projects are mentioned, the reader will find many projects outside IBM. Additional user interface issues are outlined in conference proceedings.[1,2]

Defining the user interface

Placing and evaluating the interface. The user interface may be thought of as a surface through which data are passed back and forth between computer and user.[3] Physical aspects of the user interface (Figure 1) include the display devices, audio devices that may be used, and input devices such as tablet, joystick, mouse, microphone, or keyboard.

Data displayed on the workstation provide a context for interaction, giving cues for user action (we assume that the user knows how to interpret what is displayed). The user formulates a response and takes an action, and data then pass back to the computer through the interface. In this concept, all aspects of the system that are known to the user are defined at the interface. The quality of the interface, from the user perspective, depends on what the user sees (or senses), what the user must know in order to understand what is sensed, and what actions the user can (or must) take to obtain needed results.

From the user perspective, the implementation on the computer side can be considered as a whole. The user evaluation of what is observed at the interface gives us a focus for setting explicit requirements on the static and dynamic properties that are experienced from the user's side of the interface. In a similar way, the designer can consider the user as a "module" with prescribed processing capabilities that are also evaluated at the interface. This gives us a focus for understanding the interaction requirements placed by the computer system on the user who carries out tasks.

It is important to separate these interaction requirements from a particular implementation that builders supply to meet them. That is, ease-of-learning and ease-of-use requirements persist even if they are hard to meet. It is our purpose to design the interface to fit the user rather than to "design" the user (through training) to fit the interface. However, any implementation requires trade-offs and compromises that inevitably lead to some user training as an accommodation to the equipment. By distinguishing the requirements derived from the needs of

Figure 1 The two sides of the user interface

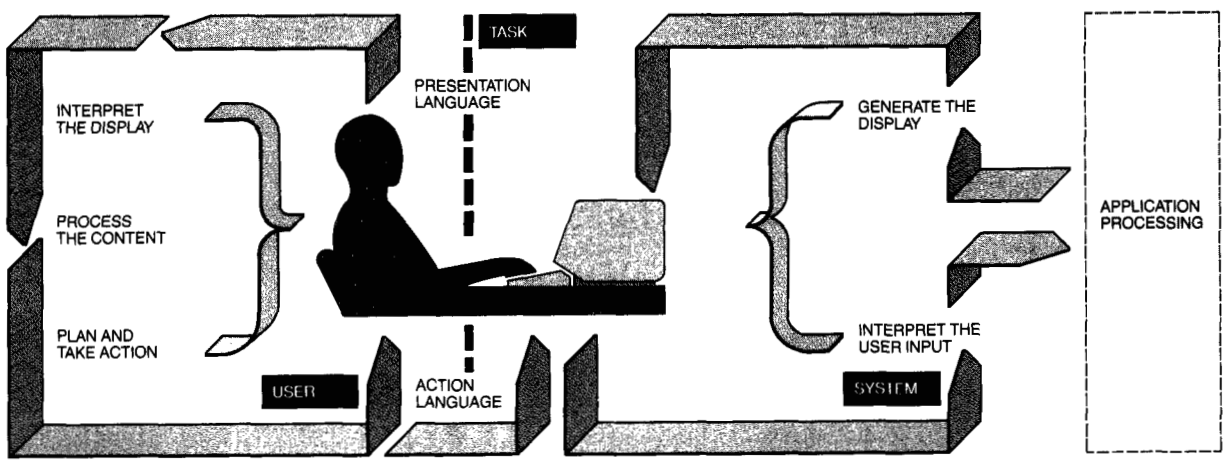
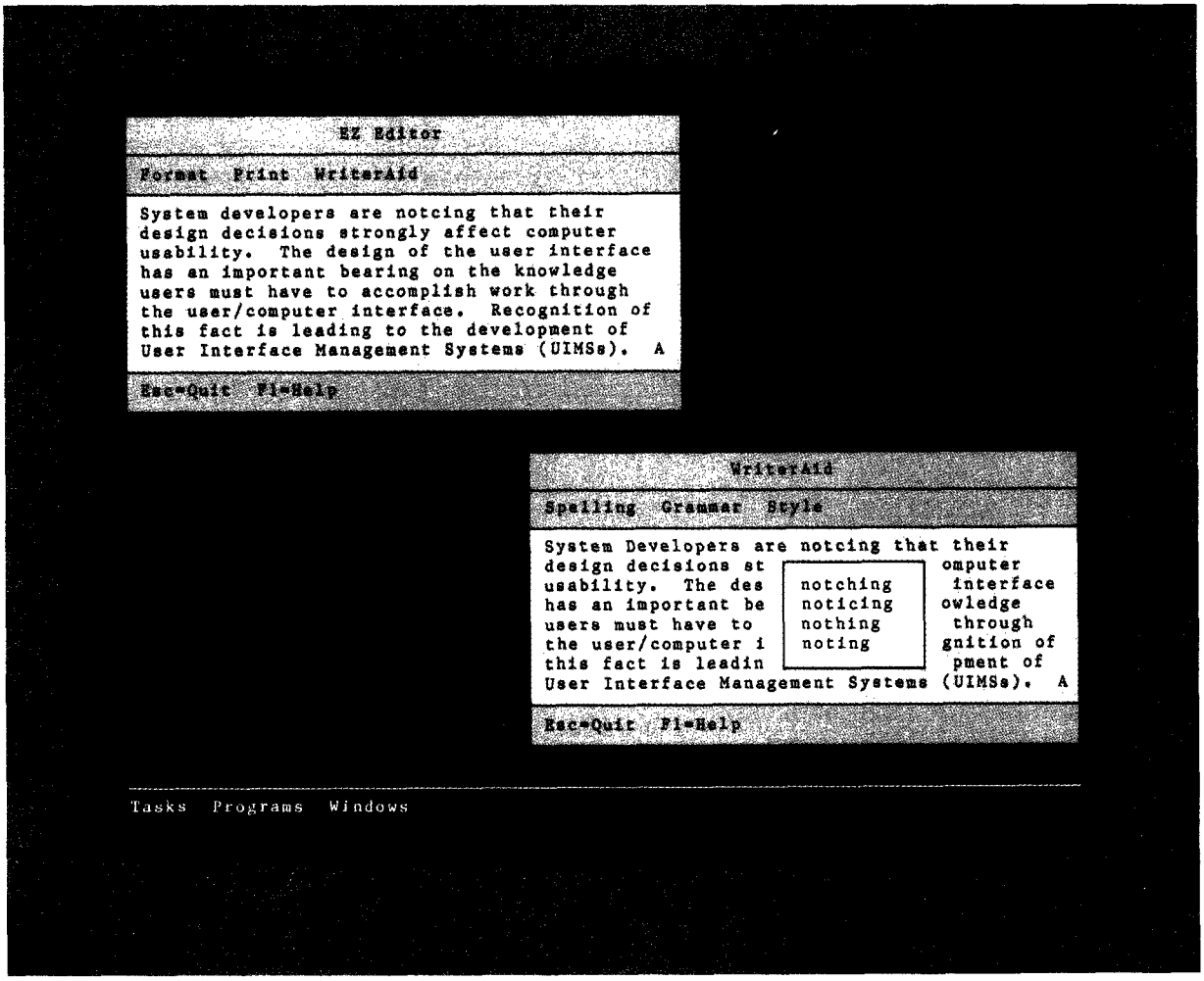


Figure 2 A specific example of screen content showing the dialog framework and content for an EZ Editor application and a WriterAid application



the end user from the implementation created by developers to meet the stated needs, we hope to recognize that the quality of any mechanism used to build the computer system, such as a UIMS, will be judged by how well it meets the requirements. This recognition encourages us to explore a variety of design approaches.

Languages used at the interface. We may analyze the data displayed by the computer at the interface as *sentences* in a *presentation language.* Just as with any other language, the computer as generator of the sentences must follow language rules; the user as interpreter of the language must know how to parse the sentences. Similarly, we may consider that the user responds to the computer through the interface by composing sentences from an *action language.* In this case, the user as generator must follow the rules so that the computer can parse the input sentences.

Figure 2 shows a representation of a specific user interface as it might appear on a display (Figure 1). The display provides access to the independent EZ Editor and WriterAid applications operating on the same data. The layout of the screen and the data displayed on the screen illustrate the presentation language. User action is carried out through the workstation devices. Actions may be invoked under

user control, under computer prompting, or through automatic computer processing. System actions are shown at the bottom of the screen. Application actions are shown in each application window. Thus the user interprets the context formed by the presentation language and reacts through actions allowed in the action language.

Goals evaluated at the user interface. We now consider typical goals of the people who build the interface and those of the end users who interact through the interface.

Goals of UIMS builders. The concept of a User Interface Management System[4] has been a recent focus for builders. The following are typical goals for improving the quality of the user interface, adapted from References 4 and 5:

- Place the user interface processing that is common across applications in a separate module so that many applications can use the same code.
- Use the common module to present a more consistent interface both within and across applications.
- Use the presence of a common module to encourage specialists to separate the design of presentation and action languages seen by the user from the design of specific application content.

Goals of end users. In considering the interface as a surface, we can outline dimensions that users evaluate in an "acceptance test" of the results of a design. Sample dimensions adapted from Shackel by Bennett[3] are as follows:

- Learnability. A specified level of user performance is obtained by a required percentage of a sample of intended users, within some specified time from beginning of user training.
- Throughput. The tasks required of users can be accomplished by a required percentage of a sample of intended users, with required speed and with fewer than a specified number of errors.
- Flexibility. For a range of environments, users can adapt the system to a new style of interaction as they change in skill or as the environment changes.
- Attitude. Once they have used the system, people want to continue to use it, and they find ways to expand their personal productivity through system use.

The importance of measures of success. Both the builder and the evaluator representing the user must establish specific measurable and testable goals for their work.[6] Ultimate success at the user interface can be evaluated only after the system is built and put into use. Because of the expense of building a system and the difficulty of making design changes at later stages of development, developers are looking for cost-effective ways to make the changes needed at the user interface, without affecting the relative stability of applications. Developers are also receptive to the potential success of early-warning methods such as modeling user process knowledge to identify before system completion those design decisions that may lead to user inefficiency. We explore what is being done to meet these goals in the next two sections.

Ultimate success at the user interface can be evaluated only after the system is built and put into use.

Building the user interface

The UIMS as a model for organizing computer resources. The concept of a UIMS is relatively new. As mentioned earlier, a UIMS is a set of services for presenting data on a workstation and interpreting user actions at the workstation. By serving as a bridge between a variety of applications and a variety of workstations, the UIMS serves as a tool for simultaneously reducing the cost of development, providing the technical flexibility needed in advanced applications, and meeting the usability requirements of lay users.

Implementers of these systems have synthesized ideas for them from a variety of sources. Many of the concepts, initially developed as fragments in graphics applications, are being brought together as a conceptual whole. Thus the UIMS serves as a focal point for understanding design options that are important to the developer (by making it easier to construct the interface) and indirectly important to the end user (by improving the quality of the user interface).

A UIMS is both a set of tools for building a user interface and a run-time processing system for supporting the interaction between an end user and the applications running in the computer. In principle, all user interactions with the computer go through the UIMS. The motivation for building a UIMS comes from the observation that much of the support for user interaction appears to be common across a wide variety of applications. When interaction code is lumped with application function, the work must be repeated for each application. In addition, end users are confused by the fact that designers of applications have varying ideas about what constitute good interaction methods. If these details of interaction (many of them device-specific) can be handled independently of the various applications in a way judged consistent by users, several potential benefits may be realized.

Olsen et al.[4] list these benefits as follows:

- A UIMS can provide a practical means to support design of an interface suitable for use across applications.
- User interface specifications (presentation language and action language), defined separately from the application specifications (i.e., physical input data required from the user), can be represented, validated, and evaluated more easily.
- User interface designs can be prototyped and implemented more rapidly using the tools provided as part of the UIMS.
- Construction and maintenance of interactive applications can be separated from construction and maintenance of user interfaces.
- Functions that support the user interface can be distributed among systems and processors in a way that increases responsiveness to user actions but does not require changes in user interaction methods.
- Members of the design and development team can apply specialized skills throughout the evolution of the software that shapes the user interface.

The tools provide these benefits by reducing the cost of development, offering technical flexibility, and assisting designers to meet the usability requirements of users. In addition to aiding the work of senior developers, perhaps an even greater benefit can come from improving the results obtained by relatively junior personnel. The concept of a UIMS serves as a focal point for the construction of such tools.

Structural relationships within a UIMS. A UIMS can present information in two categories. (Figure 1 gives an overview and Figure 2 gives a more detailed illustration of the specific content that could be produced by a UIMS.)

Presentation language. The first category (lighter area of Figure 2) relates to the presentation context, the dialog framework within which application information is shown to the user. This category is analogous to a desktop work area, with data giving support for writing a variety of documents. Examples of context structures are display windows and menu formats. The dialog framework separates views of objects, determines whether window overlapping is allowed, and supplies window border markings.

The second category (darker areas of Figure 2) is determined by the specific content of an application. To continue the desktop metaphor, one document on the desk is distinct from another. Examples of application content are lines of text for a document shown in a window or items in a menu of choices relating to a specific application.

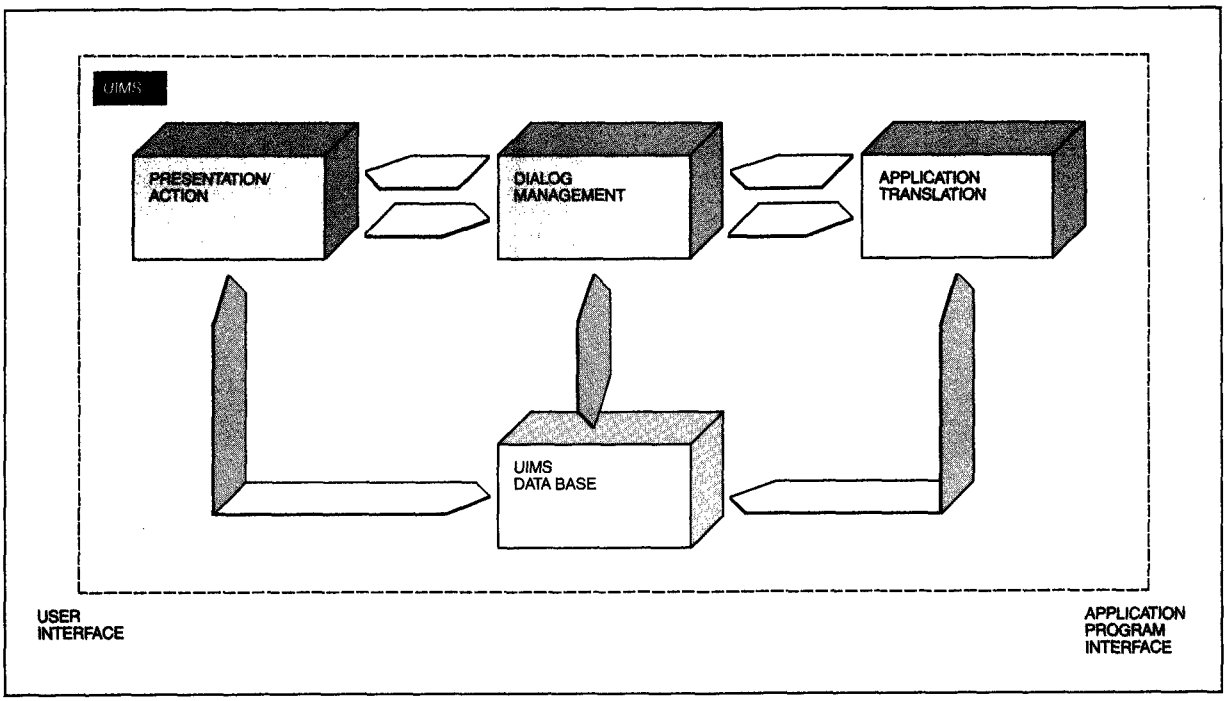
Action language. The UIMS supports parallel categories in the action language available to the user. The first category relates to actions on the dialog framework. Examples are the moving of a display window on the surface of the terminal or selecting a menu item. Rules are included for how the window and content can be moved with respect to the edge of the screen, whether one window can be moved to overlap another, and how to link an input device to the move action.

The second category relates to actions on the application-specific content. An example in this category is selecting a point in an image contained within the document. Rules determine what can be done with the image, the sequence of steps needed, and how the application shapes are linked to input devices.

A UIMS architecture. One way to understand the concept of a UIMS is suggested by Green[7] in his review of a workshop held by UIMS developers. Green discusses a model that divides the UIMS into several parts to emphasize a division of responsibility. We can understand the division represented in Figure 3 by tracing the flow of data from the application to the user. This produces, for example, the kind of data displayed on the screen in Figure 2. In the same way, we trace the flow of data from the user back to the application.

The path from the application to the user. The options for presentation available to an application are

Figure 3 Structural relationships among the abstract components of a User Interface Management System



defined at the *application program interface*. The *application translation* component transforms an application request for display into a data structure designed for the UIMS. The *dialog management* component maps the application content onto the data structure representing the context to be displayed to the user (for example, places the representation of the document in a window). The *presentation/action* component is responsible for the physical appearance of displayed data in accord with the presentation language, as defined at the user interface. Requests for display from an application are matched to the particular capability of the terminal. For example, window borders on the screen may be in color (if the device allows it). If the document displayed in a window contains an image, and if the terminal cannot show images, the presentation/action component at least preserves spatial fidelity to show the relative position in which the picture would appear, within the resolution of the device. If cursor shape is a part of the presentation language, this component is responsible for implementing the design decisions for this terminal that are embodied in the UIMS.

The UIMS data base acts as a repository for data that must be known to the UIMS components. This includes terminal device characteristics, application structure, the state of the dialog interaction, and a means for preserving context when the user switches among applications.

The path from user action to the application. We now complete the data path by outlining the loop from the user back to the application. All interactions with physical devices are isolated in the presentation/action component, making it easier to add new devices without changing the application. Screen layout changes can accommodate user preferences, such as left-handedness. The modularity can encourage the development and the use of a standard library of interaction techniques, such as the protocol for changing a displayed value. The component interprets the actions by the user on the physical devices available at the interface. Examples of these actions are pushing a cursor movement key, moving a mouse, or moving a stylus on a tablet. The component forms messages to be interpreted by the dialog management component.

The data path through the dialog management component may take two branches. If the user action can be interpreted as a dialog control request, it may be processed through the first branch without application intervention. For example, a request to move a window on the screen (which by design does not change the content displayed within it) can be handled at the level of the dialog management component. Or another user action may signal the beginning of a sequence of actions to change the value of a variable. The dialog management component can interact with the user, thereby generating feedback signals (such as *change the shape of the cursor* or *highlight content*) for the presentation/action component to display.

Many questions remain in moving the work from a research laboratory to routine use.

The second branch is taken when the UIMS does not have the information needed to respond to an action. For example, such a branch might occur in the selection of an application object that requires application processing to place the cursor. This request must be transmitted back to the application. In this case the application translation component must translate from the UIMS data structure into the application data structure. This structure represents distinctions that are known to the application giving the request for display that started the loop.

Observations. It is interesting to note where development of the concepts represented in the UIMS originated in the evolution of user interface support tools. In the past, application designers put together whatever support was needed to access the application from a particular workstation. Then, as the field evolved, developers recognized common functions for each application and for each workstation. This work led to the synthesis of ideas represented in the UIMS.

ISPF. Several existing dialog managers and application development tools have some of the characteristics of a UIMS. For example, the Interactive System Productivity Facility (ISPF)[8] is a dialog manager that has been in use for several years to provide control and services to allow processing of dialogs in VM, MVS, and VSE environments. The display screens of IBM 3270-type terminals supported by centralized host services can be split to provide an interface to two independent applications, though only one can be active at a time. Panel skeletons and interface definitions for this class of terminal are stored in libraries separate from the applications, allowing programmers to tailor the display formats for different system environments. The panel definitions can be used to select application data for display to allow user modification of specified variables and to do verification checks of user input syntax. ISPF's current flexibility and potential for evolution are limited by required compatibility with prior VM and MVS practices.

GDDM. The Graphical Data Display Manager (GDDM)[9] is designed to handle the communication between application programs and terminals. In addition to the base tools that define an application programmer interface, features include an Image Symbol editor for working with raster images, a Vector Symbol editor for manipulating objects composed from vectors, and an Interactive Map Definition editor for implementing dialog frameworks to be shown through the dialog manager. These features were designed relatively independently over time in accord with the data structures then available. Given the framework for setting usability goals when functions are viewed from the perspective of the user interface surface, and given an evolution in the tools provided within GDDM (perhaps shaped by UIMS concepts), later releases in the product may be able to consolidate the different pattern of the user actions now needed to edit these objects.

TopView. TopView[10] has provided some initial windowing functions as a first effort to integrate the wide variety of applications found on the IBM Personal Computer. To do this required the setting up of an application programmer interface. However, many existing applications did not follow the required interface rules because TopView had not been defined at the time many of the early applications were written. Thus, some applications could not use the full features of TopView. Once the interface had been established, other applications could evolve to use it. In a similar way, the UIMS model for presentation/action independence from the applications may stimulate extensions of TopView and its successors.

The RT Personal Computer. The IBM RT PC[11] uses UIMS concepts to create a bridge between an operating system designed for a programmer and applications offered to occasional users as well as experts. A dialog manager is used to define, present, and control the user sequence of actions in order to shield the application from details of display frame design and user interaction. The dialog framework content, stored separately from the application, is preprocessed to make it efficient for use in run-time operation. Support utilities aid the developer in laying out dialog framework data on the screen. Other utilities store and access the terminal description used by the dialog manager. It will be interesting to see how these elements, similar to those in the UIMS concept, evolve in later releases as the initial ideas are tested in practice.

Other commercial and military applications. Significant progress in the use of UIMS concepts has been made in computer-aided design and military command and control. About ten large applications have been implemented at Boeing Computer Services using The Interactive Graphics Engineering Resource (TIGER)[12] system for decoupling physical interaction handling from logical function performance. Functional Language Articulated Interactive Resource (FLAIR),[13] using voice recognition to support the designer, has been in operation at TRW since 1981. It is in active use as a dialog design language for exploring space-age applications using advanced graphics.

Issues. Although the concepts used in developing a UIMS are influencing exploratory work, many questions remain in moving the work from a research laboratory to routine use in development projects. The following are examples of some of these questions:

- Does the memory space required for a UIMS on workstation processors make implementation feasible on today's equipment? In a system environment where many applications are served by one UIMS, we should eventually see a system advantage if all the applications can make use of the UIMS facilities.
- Is the time required for a UIMS workstation processor to handle user actions compatible with interactive response required at the user interface? This is a function of the available processing power and the way in which function is distributed between the UIMS and the application. Current products such as the PC RT[11] are serving as a testing ground for exploring this question.
- Can the workable division of function between UIMS and applications that is possible in principle be established in a wide range of practical situations? Realizing the full benefit of the UIMS requires an agreed-upon definition of the application programmer interface. Any such standardization requires engineering trade-offs. The division of function may require an evolution in application design as well as in UIMS design.
- Do the current tool-kit approaches to providing UIMS function give the designer valid defaults and establish enough discipline to meet the usability criteria as evaluated by the user at the user interface?
- Do the tools provided by the UIMS enable developers to construct applications faster than would otherwise be possible?
- Does a UIMS, once constructed, prove to be portable to other environments?

Although developers need tools to construct user interfaces with advanced features, the presence of a tool, though it may have been designed with internal elegance and creativity, does not itself automatically lead to user interfaces that meet user requirements. To identify operational characteristics of the UIMS that result in user satisfaction, we can build performance-monitoring tools into the UIMS run-time support. The issue, however, is knowing what needs to be measured in order to diagnose user problems.

The issue is knowing what needs to be measured to diagnose user problems.

Reference 4 suggests the integration of a spelling checker with a word processor as an example of a required technical capability for accessing functions from several applications. Such a capability can aid the user, but the timing of its invocation is critical. If, in the kind of work situation shown in Figure 2, a person is writing using EZ Editor and is barely able to type fast enough to capture transient thoughts in text, then the automatic invocation of the WriterAid

spelling checker might interrupt the flow of thought and therefore the user's progress in the task. A task analysis might suggest that the system should support text entry at typing speed, followed by a user-controlled pass through the text to correct spelling. The need for integration and rapid performance would still exist, but the invocation would be timed to avoid interference with the creative process of the user.

As we review recent literature on the developing concept of the UIMS[4,5,7,14–17] and the tools associated with it, we identify a need to provide a more "consistent," "uniform," "supportive" interface for the user. However, designers, both in the academic community and in industry, tacitly acknowledge that they do not have operational definitions for these terms.

Coutaz[5] calls for advice from psychologists, human factors specialists, and graphic artists who have acquired a better understanding of human behavior than have computer scientists when making specific design decisions affecting the user interface. Dialog design requires the cooperative efforts of technicians from a variety of disciplines.[5] By cooperating in this way, developers increase the chances of building in effective guidance mechanisms to help the user adapt to the design. Identified in Reference 5 is a need to present designers with a standard view of the potential users, so that designers can provide access to applications through the workstation in a consistent way.

Representing user process knowledge

As suggested earlier in this paper (see Figure 1), the user interface is evaluated by users according to what they see, what they have to know in order to interpret what they see, and what actions they can (or must) take to get useful results. An important current theme is consistency in presentation and action. We mentioned earlier that the developer using UIMS tools must make specific design decisions about the presentation and action language that affect the user interface. We are now ready to discuss how work in modeling the interactive processes needed by a user is intended to help the developer in making these decisions. The goal is to give some indication early in the design process as to how design decisions affect the ease of learning and the ease of use of an interface. Some insight may be gained even before an experimental prototype is built.

Much of the detailed UIMS work is in the area of tools for supporting user input. Also, much of the work on representing user knowledge is focused on studying user input to the computer. Output to the user tends to be application-dependent and of great variety. However, user input to the computer is much more constrained and limited by that which the computer can be programmed to deal with.

Addressing the issues of consistency and learning is related to the psychology of the user.

The designer of any interface assumes that the user knows from previous experience (or can be trained to know) the specific sequence of actions needed to accomplish a particular task required by the design. For example, the designers of an editor for office documents assume that the intended users will be able to acquire the knowledge needed to edit the documents found in their work. To estimate the possible gaps between what users already know and what they have to learn, it is helpful for designers to understand the knowledge already used by prospective customers as they carry out their current work. Designers can then arrange the design so that as much as possible of the current knowledge is transferred to the new work environment. Addressing the issues of consistency and learning is related to the psychology of the user—the see-know-do questions previously mentioned.

Current practice in product development groups is to establish usability plans as a basis for representing performance objectives and for specifying the way in which the interface will be tested. Excellent examples are given in a special issue of the *IBM Systems Journal*.[18] Botterill[19] reviews practices followed in the design of the System/38 user interfaces.

Phenomenological studies are carried out by human factors personnel to observe people as they work. An understanding of the need for change and sometimes of what the change should be can often come from these studies. A limitation of standard human factors

evaluations of interfaces is that the analysts need to observe the actual process of use in order to see whether the established objectives are met, to compare alternative designs with respect to performance goals, and to find sources of user errors. Such analysis requires a working model or prototype. Some human factors specialists have studied products currently available in the field, and such existing systems can serve as helpful prototypes for understanding proposed product features. However, given the current short development cycle (driven by competition and affected by new software techniques associated with new applications), analysis of the actual product under development often comes so late in the development process that significant changes to that particular interface are not economically feasible.

The formal modeling of user process knowledge focuses on early-warning analytical tools.

The formal modeling of user process knowledge focuses on early-warning analytical tools. One formal approach seeks to develop a representation, a method for constructing the representation, and a means for executing the representation to test its validity. This work is based on the assumption that the results can be made useful to developers for predicting, at least in some cases, how changes in interface design are likely to affect user learning and throughput.

Ultimately the approach is intended to be suitable for incorporation into the development cycle. This means that the user psychology must be introduced in a way that matches development practices and expectations. In the following section, we review a sample of this work and examine its potential to support predictive and diagnostic analysis of design decisions made when developers use a UIMS.

Criteria for analytic tools. Reisner[20] has reviewed analytic tools that may eventually provide an important supplement to behavioral tests currently performed by human factors professionals. Analytic methods are intended to provide a filter, suggesting what can safely be ignored, and to serve as a spotlight, illuminating which aspects of the interaction are important in the prediction of user performance. The experimenter builds a representation of the user interaction, performs systematic operations on the representation, and then studies the result to predict what would happen if actual user behavior were observed.

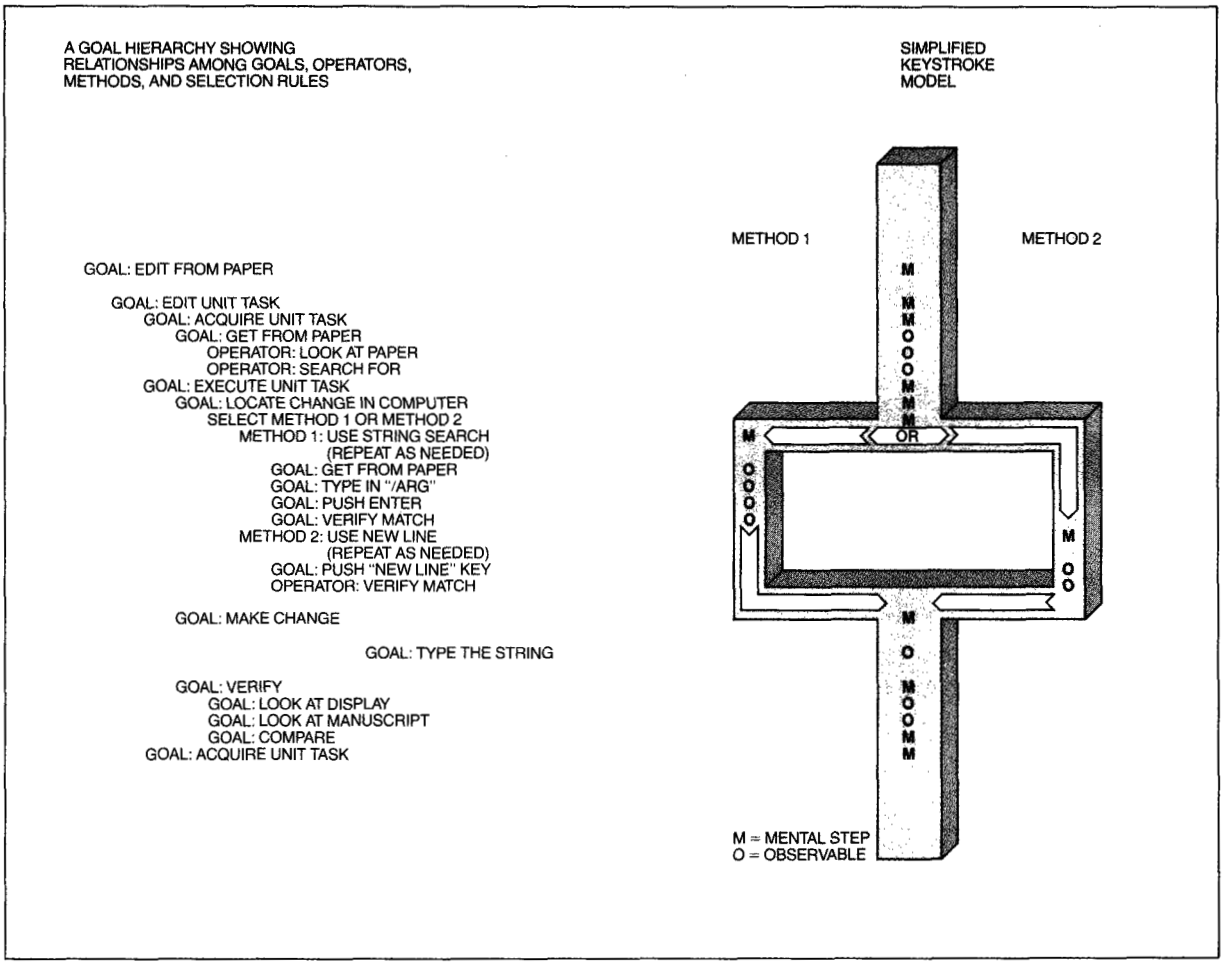
To be of practical use, such a method must itself be validated. The knowledge structure for a particular situation must be represented using the concepts defined within the theory. The representation of the structure must be analyzed to develop predictions for average user learning time and average productivity, and to identify likely user errors. People must be enlisted to use, in experimental situations, the processes that have been modeled, in order to obtain actual values for comparison with the predicted values (such as learning time for a task, time to complete a task once learned, or kinds of user errors observed).

In addition, Reisner points out that the modeling method must itself be easy to use if it is to be incorporated within the current system development process. That is, the knowledge and skills needed to build a model for a particular system under development must be available within the development process. Also, the method must be timely and economically feasible within typical development process constraints.

One approach to modeling the process of use. One approach to gaining a better understanding of user interfaces is to model the knowledge a user must have when that user carries out tasks through an interface. Card, Moran, and Newell[21] have developed a formal goal, operator, method, selection-rule (GOMS) model. This is an analytic approach intended to aid designers and to allow detailed descriptions of the mental and physical operations a user must execute to achieve task results.

Figure 4 (adapted from Reference 21) shows examples of goals, operators, methods, and selection rules for part of an editing process. In the GOMS model, a hierarchy of goals is used to specify the sequence of subtasks needed to complete a complex task. Shown on the right of Figure 4 is the Keystroke Model, a simplified version of the GOMS model. The times required for each observable physical motion (O) and the times needed for mental operations (M) can

Figure 4 An example of the GOMS Model and a simplified Keystroke Model for an editing task



be summed to give a Keystroke Model estimate for the total time needed to complete a task.

The terms used in the GOMS model can be understood as follows:

- *Unit task* is a way of expressing how people organize their behavior into 10- to 15-second, relatively independent tasks. This organization corresponds to human limitations such as short-term memory and also corresponds to the logical structure of a task.
- *Goal* is a symbolic structure representing a user's intention to perform a task, a mental operation as part of a task, or a task-required physical action. In Figure 4, Edit-Unit-Task represents a high-level goal, Locate-Change-in-Computer represents a middle-level goal with two possible methods, and Push ENTER represents a goal resulting in a physical action.
- *Operator* is an elementary perceptual, cognitive, or physical act the execution of which changes the user's mental state or affects the task environment. User behavior is recordable as a sequence of these operations that are assumed to be serial in execution in the GOMS model. In Figure 4, Look-at-Paper represents an operator high in the task decomposition that is relatively independent of interface devices; Type-In "/arg" represents an operator at the low level that may change, depending on the device used at the interface.
- *Method* is a sequence of operations. In the GOMS model a method is shown as a conditional se-

quence of goals and operators along with conditional tests on the content of user memory and on the state of the task environment. In Figure 4, the New Line method is one way for the user to move the cursor on the screen to the position in the computer representation of the text where a change must be made.
- *Selection rule* is a conditional test that models a user choice of the method to apply from among available methods. In Figure 4, Locate-Change-in-Computer represents the place where a selection rule would be invoked to choose a method, but the required condition testing is not shown.

The Keystroke Model shows the lumping of higher-level, not directly observable mental operations into one or more Ms. Physical actions are observable and are represented as Os.

Design of a goal structure representation is at the frontier of current research.

This example gives an overview of the concepts that appear in the theory and a view of the relationship between the GOMS representation and the Keystroke Model.

Building a GOMS model requires a special task-analysis skill. Design of a goal structure representation—particularly one tuned to the investigation of user-computer interaction—is at the frontier of current research. The complexity of the representation and the detail of the predictions that can be made through analysis of that representation can vary widely. For example, a high-level analysis might focus on device-independent decomposition, whereas at a very detailed level, times due to the particular physical details of a particular device are more important.

The scope of current research is limited to performance of routine cognitive skills. This research does not address the full range of human behavior observable at an interface. For example, error analysis is not included, though modelers can sometimes predict through an analysis of goal structures where user errors will be observed.

The potential value from use of the GOMS model is that formal, theory-based modeling of routine tasks carried out by a hypothetical well-trained user leads to an improved basis for interaction design. That is, we assume that a stable set of routine methods (such as those found in text editing) forms a foundation upon which users build their interactions with more advanced applications.

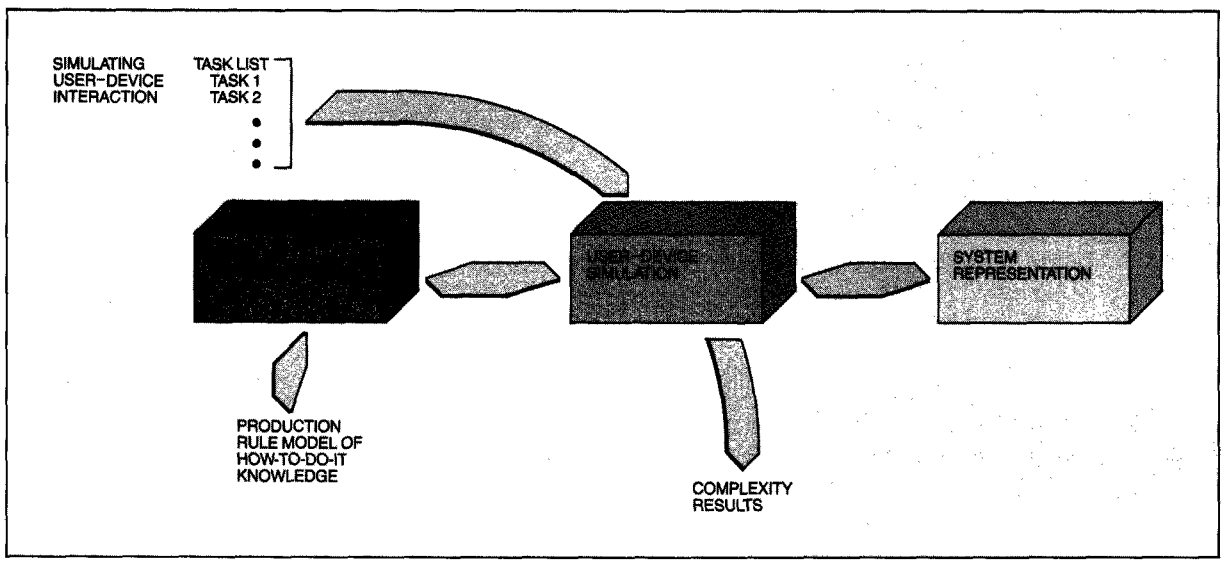
Studies built on the GOMS model. Kieras and Polson[22] are developing a cognitive-complexity theory that allows training time and throughput estimates to be made, based on analysis of the GOMS structures. Their additional step is to map GOMS constructs into executable representations of user how-to-do-it knowledge. Such representations are well known to persons in the fields of artificial intelligence, natural-language analysis, and expert systems as a way to model human information processing.

In Figure 1 we saw an overview of the two sides of the user interface. In Figure 3 we expanded the right side of the user interface to show how a UIMS could provide functions needed to support the presentation language and the user action language. In Figure 5 we illustrate the parts of the simulation model used by Kieras and Polson.

The knowledge required to operate the system is represented in production rules. In parallel with that representation, the system or device under study is represented in executable form. A user-device simulation interprets an external Task List. The simulator drives the simulated device to construct the task environment (for example, a page of text to be edited), passes control to an interpreter that processes the production-rule representation of the goal hierarchy, and generates a simulated user action. The simulator continues through the cycles until the Task List is completed. The statistics collected by the simulator are interpreted by the experimenters with respect to the theory to estimate ease of learning and throughput.

Polson and Kieras[23] discuss the relationship between the content, amount, and structure of user knowledge (as represented in production rules) and predictions of time to learn. As we now understand things, the time needed to learn a task is a function of the

Figure 5 The relationships among the parts of a simulator



number of new production rules in the task representation. The time required to carry out the task (productivity) is estimated by the unit time for human execution of each production rule summed over all production rules needed to emit the correct simulated stream of user actions.

Clearly, the model of hypothesized user knowledge must be independently validated through experimental observation of people carrying out the tasks represented in the model. The level of detail included in the model (such as time to push a key), the assumptions about the internal hierarchical structure of the rules, and the programming style of the person who creates the production rules can all affect simulator execution time.

In recent studies by Polson and Kieras on transfer of training, predictions are based on an assumption that no extra time is required for a simulated user to learn the parts of the production-rule structure in a new task when those parts are the same as those in a previously learned task. For example, if the MOVE and COPY commands for a text editor are designed so that the user knowledge required to learn them is represented by many common production-rule sequences, the theory predicts that user training on MOVE transfers to the COPY operation and the second command is learned faster than the first, independently of which one is learned first. This has been tested by Polson and Kieras using people in numerous laboratory experiments.[23] For this kind of experiment, the model results prove quite accurate.

Using these concepts, we can find in the Kieras and Polson formal approach a theoretical basis for talking about the simplicity, the consistency, and the transfer of user knowledge to new processes. A base of data obtained from experiments with actual users is now being built to validate and calibrate the model data. The user processes so far are relatively simple, but the researchers are encouraged by their current successes.

Limitations. A number of researchers are comparing results seen in their work with the kind of results predicted by the GOMS model, particularly at the keystroke level. Gould and Alfaro[24] report a comparison between their findings and the predictions of the Keystroke Model. The authors observed that 60 to 80 percent more time is required than the time predicted for comparable time categories by the Keystroke Model. This is true even for the skilled users assumed by the model.

Clearly, we must be cautious about the precision of estimates generated with the Keystroke Model. However, even these approximate models can be useful as an aid during initial system design.

Issues. With this brief overview, we are ready to look at some of the questions raised by the modeling work so far:

- Do present models capture enough of the variety of user behavior to be helpful? The rudimentary state of the existing theory limits study to routine cognitive processing. The problem-solving behavior known to be important at the user interface is not addressed. The working assumption is that human problem-solving behavior is built on a base of routine cognitive processing. If designers can support the routine work (by analogy, the dialog framework in the UIMS), the user can more easily do creative problem solving (by analogy, the applications).
- Is the particular kind of task analysis required by the modeling work itself well enough understood that it can be taught reliably to people working in the development process? Modeling work requires a task analysis before the goal structure can be created. The skill appears to be taught currently through examples provided by mentor to graduate student at university cognitive science departments. Progress in learning how to do task analysis appropriate for the method is valuable in itself, as the skills are needed for observational work in human factors experiments, quite aside from knowledge-representation models.
- Are the time and resources that it takes to model a proposed user interface design and to model the user processes that the design implies feasible within the product development cycle in industry? The UIMS approach is intended to make it easier to do top-down modeling of the devices and early prototyping of the user interface. That is compatible with the approach of Kieras and Polson. The art of creating the goal structure and writing the production rules is more problematic. It may be that current work will lead to standard subroutines or templates representing current designs that analysts may apply to new designs.

Answers to these questions are being explored in laboratories. The opportunity for analysis based on principles and the need for insight give us confidence that the current work will have practical impact.

Putting theory into practice

This review has examined areas of concern to user interface builders working within a product-development cycle. The concept of a UIMS shows great promise as a tool for simultaneously reducing the cost of development, providing the technical flexibility needed in advanced applications, and meeting the usability requirements of users. The analytic tools for modeling user process knowledge can fit well within an engineering approach to design and development.

Gould and Lewis[25] advocate early and continual focus on users, observation of usage on simulated, prototyped, and actual systems, and iterative modification as needed. They contrast this approach with principled design approaches, relying on design guidelines intended to get it right the first time. Their advice fits well with the UIMS development concepts advocated by the authors we have cited. The addition of models for user process knowledge is intended to give early-warning indicators of design approaches that may cause problems for users. These design implementation and evaluation aids may help creative designers focus their innovative skills on problems that need to be solved to support users even before the prototypes advocated by Gould and Lewis are ready.

We expect the research cycle of building systems, studying them to understand what has been built, and applying the knowledge to implement new systems to continue to augment our ability to build user interfaces that are truly advanced.

Acknowledgments

I found valuable the several hours that Jim Rhyne spent in discussing the experience he gained from his work with User Interface Management Systems. Phyllis Reisner has shared her insight over a period of time in discussions of analytic methods applied to interface design. The keen interest of Peter Polson in seeing analytic methods based on cognitive science theory put into development practice has been an inspiration for a number of the observations. John Richards contributed perceptive comments from both the viewpoint of system developer and cognitive psychologist. However, I am responsible for the interpretations made in the article.

Cited references

1. *Human Factors in Computing Systems—CHI'86 Proceedings*, April 1986, ACM Special Interest Group on Computer and Human Interaction, New York.
2. *Human-Computer Interaction—INTERACT'84 Proceedings*, B. Shackel, Editor, Elsevier North-Holland, Inc., New York (1985).

3. J. L. Bennett, "Managing to meet usability requirements," *Visual Display Terminals: Usability Issues and Health Concerns*, J. L. Bennett, D. Case, J. Sandelin, and M. Smith, Editors, Prentice-Hall, Inc., Englewood Cliffs, NJ (1984).
4. D. R. Olsen, Jr., W. Buxton, R. Ehrich, D. Kasik, J. Rhyne, and J. Sibert, "A context for user interface management," *IEEE Computer Graphics and Applications* **4,** No. 12, 33–42 (December 1984).
5. J. Coutaz, "Abstractions for user interface design," *IEEE Computer* **18,** No. 9, 21–34 (September 1985).
6. M. Good, T. Spine, J. Whiteside, and P. George, "Empirical impact analysis as a tool for usability engineering," *Human Factors in Computing Systems—CHI'86 Proceedings*, April 1986, ACM Special Interest Group on Computer and Human Interaction, New York, pp. 241–246.
7. M. Green, "The University of Alberta user interface management system" (*Proceedings of ACM SIGGRAPH'85*), *Computer Graphics* **19,** No. 3, 205–213 (July 1985).
8. *Interactive System Productivity Facility (SPF) and ISPF/Program Development Facility (PDF), General Information*, GC34-2078, IBM Corporation; available through IBM branch offices.
9. *Graphical Data Display Manager, General Information*, GC33-0100, IBM Corporation; available through IBM branch offices.
10. *TopView: Programmer's ToolKit*, IBM Personal Computer software, Program No. 1502483, Boca Raton, FL; available through IBM branch offices.
11. *IBM RT Personal Computer Technology, Product Design and Development*, SA23-1057, IBM Corporation (1986); available through IBM branch offices.
12. D. Kasik, "A user interface management system," *Computer Graphics* **16,** No. 3, 99–106 (July 1982).
13. P. Wong and E. Reid, "FLAIR—User interface dialog design tool," *Computer Graphics* **16,** No. 3, 87–98 (July 1982).
14. J. D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co., Reading, MA (1982).
15. W. Buxton, M. Lamb, D. Sherman, and K. Smith, "Towards a comprehensive user interface management system," *Computer Graphics* **17,** No. 3, 35–42 (July 1983).
16. P. Hayes, P. Szekely, and R. Lerner, "Design alternatives for user interface management systems based on experience with COUSIN," *Human Factors in Computing Systems—CHI'85 Proceedings*, April 1985, ACM Special Interest Group on Computer and Human Interaction, New York, pp. 169–175.
17. J. Foley, V. Wallace, and P. Chan, "The human factors of computer graphics interaction techniques," *IEEE Computer Graphics and Applications* **4,** No. 11, 13–48 (November 1984).
18. *IBM Systems Journal* **20,** No. 2 (1981).
19. J. H. Botterill, "The design rationale of the System/38 user interface," *IBM Systems Journal* **21,** No. 4, 384–423 (1982).
20. P. Reisner, "Analytic tools for human factors of software," *Proceedings: Enduser Systems and Their Human Factors*, A. Blaser and M. Zoeppritz, Editors, *Lecture Notes in Computer Science*, No. 150, Springer-Verlag, New York (1983), pp. 94–121.
21. S. Card, T. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1983).
22. D. Kieras and P. Polson, "An approach to the formal analysis of user complexity," *International Journal of Man-Machine Studies* **22,** No. 4, 3–50 (1985).
23. P. Polson and D. Kieras, "A quantitative model of the learning and performance of text editing knowledge," *Human Factors in Computing Systems—CHI'85 Proceedings*, April 1985, ACM Special Interest Group on Computer and Human Interaction, New York, pp. 207–212.
24. J. Gould and L. Alfaro, "Revising documents with text editors, handwriting-recognition systems, and speech-recognition systems," *Human Factors* **26,** No. 4, 391–406 (August 1984).
25. J. Gould and C. Lewis, "Designing for usability: Key principles and what designers think," *Communications of the ACM* **28,** No. 3, 300–311 (March 1985).

John L. Bennett *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120.* Mr. Bennett is a member of the Computer Science Department of the Almaden Research Center. Since joining IBM in 1961, he has developed his long-standing interest in all aspects of user interface design. His work on information retrieval projects and decision support systems was reported in the book *Building Decision Support Systems*, Addison-Wesley (1983), which he edited. During recent years he has worked on user interface design for office systems, and he has served as consultant for IBM product divisions on integration of measurable, testable usability objectives into the development cycle for interactive software products. He is an editor of the book *Visual Display Terminals: Usability Issues and Health Concerns*, Prentice-Hall (1984), and wrote the chapter "Managing to Meet Usability Requirements." At the Interact'84 Conference (London), he organized four theme sessions on "Behavioral Issues in the System Development Cycle" as a way of highlighting what it is like to do human factors work in software development projects. Mr. Bennett received a B.S. in engineering science from Stanford in 1959 and an M.S. in electrical engineering from the Massachusetts Institute of Technology in 1961. While at IBM Research he has been manager of Geographic Data Systems and of Interactive Problem-Solving Systems. He is currently an associate editor of the *Management Information System Quarterly* and is on the editorial board for the journal *Behaviour and Information Technology*.

Reprint Order No. G321-5280.