Common User Access—A consistent and usable human–computer interface for the SAA environments

by R. E. Berry

Systems Application Architecture (SAA) will allow customers to apply their investments in their computer operations across IBM's three major computing environments that exhibit unique characteristics in terms of architecture, workstations, operating systems, and system services. User experience is one of these investments. The Common User Access (CUA) establishes a degree of standardization that is compatible with the differences in the three environments and that supports transfer of users' experiences. CUA is based on a user-interface architecture that identifies fundamental elements of structure. The intent is to provide a transfer of users' conceptual-level learning across different and evolving technologies. CUA specifies user-interface components and guidelines to be used by application designers, and it provides a basis for programming development tool specifications.

The Common User Access (CUA), one of the elements of Systems Application Architecture (SAA), is a set of rules and guidelines for use during user-interface design. It specifies a standard set of user-interface components to be used by application designers. Each component is specified in terms of its appearance and how users interact with it. CUA also provides directions for using the components in human–computer dialogs for commonly occurring situations. Figure 1 shows an example of a programmable workstation displaying information as specified by CUA. Figure 2 shows the same information as it would appear on a nonprogrammable terminal. The term *programmable workstation* refers to keyboard-display devices in which all or most of the user-interface function is controlled by the workstation itself, such as in the IBM Personal Computer AT® (PC AT) and Personal System/2® (PS/2®) series of personal computers. The term *nonprogrammable terminal* refers to keyboard-display devices attached to a host processor in which all or most of the user-interface functions are controlled by the host, as for example in the IBM 3270 family of display devices.

The components and terms shown in the two figures are referred to throughout this paper.

The user interface is the boundary at which humans and computers communicate with each other. An interface supports a dialog, much like a dialog or conversation between two people. Just as societies and cultures need rules governing the way in which people interact with one another in a dialog, so does the human–computer dialog.

CUA is documented in a publication containing rules, guidelines, examples, and a glossary of terms.[1] Many of the user-interface components specified in CUA will be made available via toolkits such as the

Figure 1 A CUA panel in a programmable workstation window

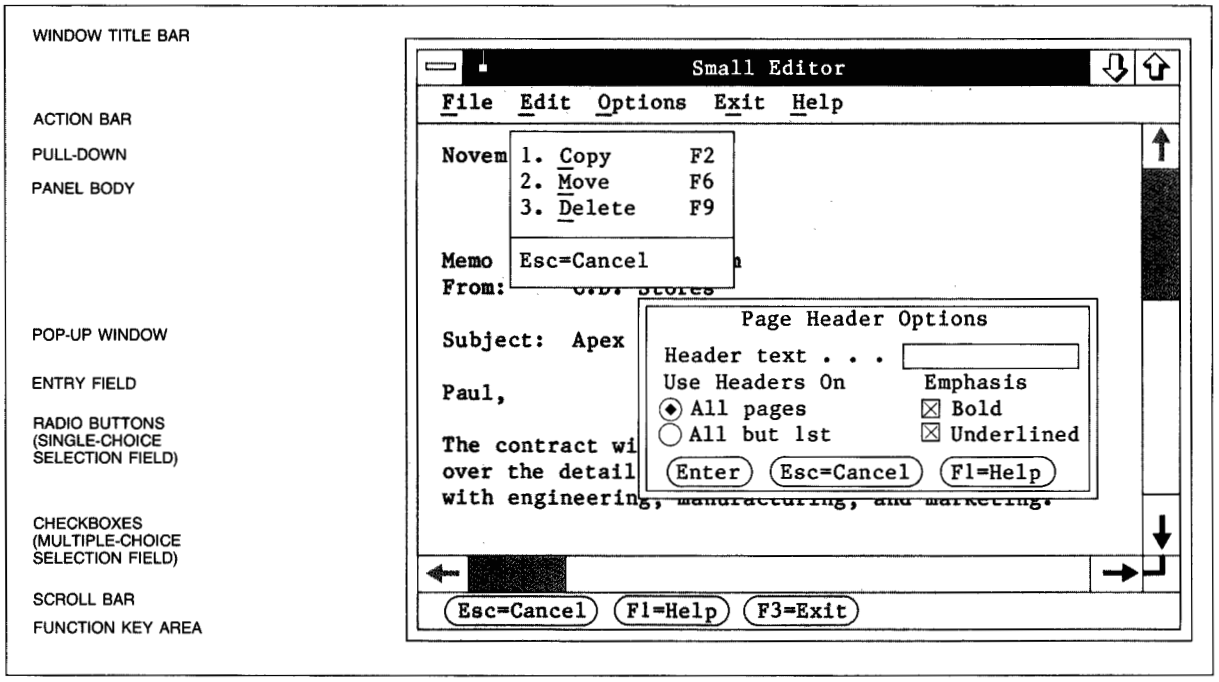
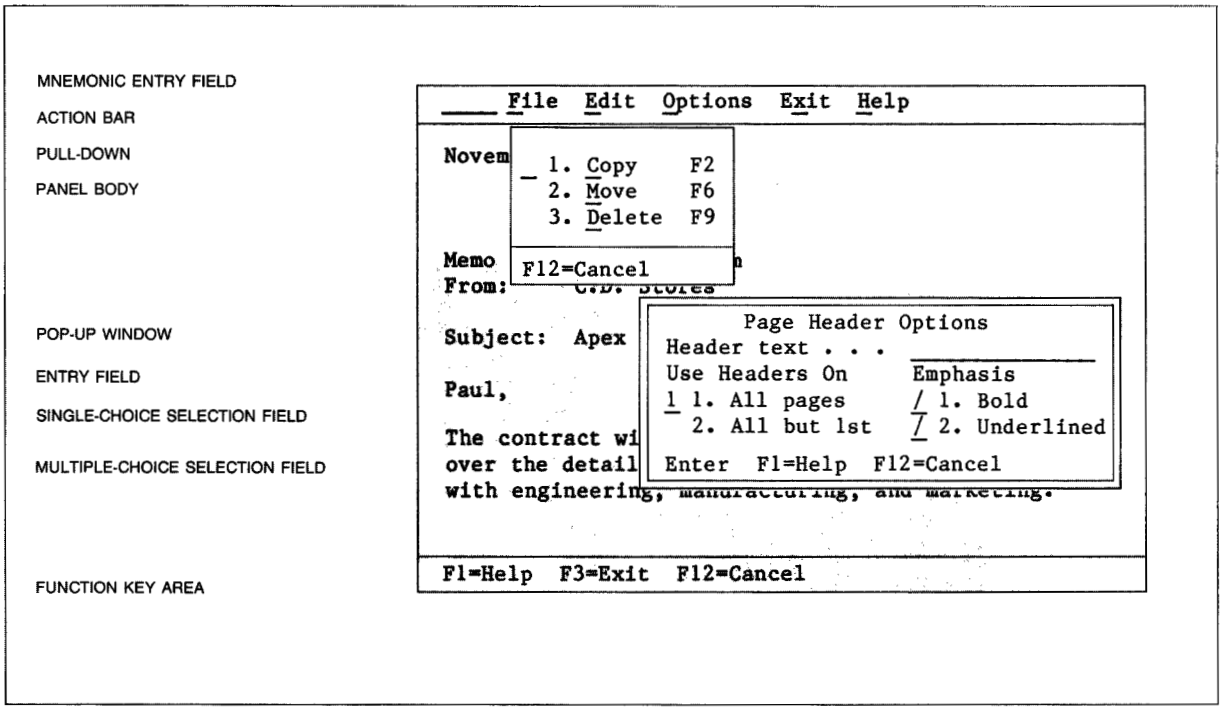


Figure 2 A CUA panel on a nonprogrammable terminal



Operating System/2™ (OS/2™) 1.1 Programmer's Toolkit. A toolkit provides application-development productivity as well as user-interface consistency.

Today's computer workstations provide users with processing power and storage capacity that several years ago was only available to data processing professionals. However, the users of these workstations no longer tend to be data processing professionals. They come from various fields and occupations, and include secretaries, accountants, lawyers, and bank employees. Users should be able to view computers simply as tools that help them accomplish their tasks. They should not have to understand how and why computers work. Computers should allow them to perform their tasks quickly and reliably, with minimal training.

The availability of increased power and capacity has led to an abundance of applications. Users can choose from a variety of offerings to optimize their function and cost requirements. User-interface designs have also proliferated. There has often been little or no commonality among applications in terms of appearance and how users interact with them. When users deal with several different applications, they spend additional time in training. When the applications require different interaction techniques, users are apt to make errors. These factors tend to increase learning time, reduce productivity, and cause user frustration. CUA establishes a framework for consistency across applications that can result in decreased learning time, improved productivity, and increased user satisfaction.

The three SAA hardware environments corresponding to IBM's major computer system families are provided as part of IBM's offerings.[2] Each environment exhibits unique characteristics such as processor architecture, workstations, operating systems, and system services. Across these environments SAA will provide portability of customers' investments, including user experience. The role of CUA is to establish a degree of standardization that is compatible with the differences in the three environments and that supports transfer of user experience.

In some cases users have developed requirements for user-interface standardization. In the report of the Interactive Systems Task Force (INTERSYS) for the period 1985–1990, Share, Inc., states, "Of even more importance than the functions that are needed is their ease of use." The degree of standardization achieved by CUA should prove useful in allowing users to apply what they have learned in one environment to the other environments. The intent is to provide a highly usable, easily learned user interface that supports significant transfer of user learning between applications within an environment and across the SAA environments.

CUA is based on a user-interface architecture.

The potential benefits of such an approach have been recognized by Meads:[3]

1. Encouraging the application of appropriate methods and principles should result in the design of better user interfaces.
2. Widespread adoption of guidelines and standards should result in increased consistency of user interfaces across various software systems and products.
3. By reducing time spent on discussing, debating, and evaluating various interface-design alternatives, guidelines and standards should reduce cost and improve the quality of software products.

CUA addresses user-interface components that are commonly used in business applications. Also included are components that, based on past experience, needed attention, such as lack of consistency in how users terminate applications.

Some components of the user interface may never become part of CUA. Application-specific components should be designed by application designers. For example, CUA does not specify appearance or interaction techniques for a text editor, a spreadsheet, or a painting canvas. Application designers are encouraged to use the CUA specifications as the basis for defining their own components. This approach allows for application creativity while still providing a basis for some transfer of users' learning. Application-specific components that find widespread acceptance may become candidates for inclusion in CUA as it evolves.

The current level of CUA establishes a base and sets the direction for user-interface design within the SAA environments. Evolutionary change is to be expected.

CUA is based on a user-interface architecture. Architectures specify *fundamental* elements of structure, and a user-interface architecture specifies fundamental elements that are significant to the way in which we want to *use* something. Architectures specify "what" happens independently of "how" it is made to happen. In other words, architectures specify conceptual elements independently of implementation detail. The user-interface architecture on which CUA is based identifies fundamental objects, such as selection fields (a group of choices), and fundamental actions, such as selection (the act of picking a choice). The intent is to provide a transfer of users' conceptual-level learning across various implementation environments.

The four SAA operating system environments provide a range of capabilities based on technology and system architecture differences. Included are a range of programmable workstations and nonprogrammable terminals, multitasking and single-tasking operating systems, and windowing and nonwindowing display managers. The initial level of CUA addresses these environmental differences and the multiple user groups that are moving from different past experiences in these environments.

Where the capabilities of the environments are the same, appearances and interaction techniques are the same. Where they differ, CUA specifies desirable appearances and interaction techniques for the environment having the greatest capability, and specifies adaptations as closely as possible in the other environments. The intent is, where possible, to provide a transfer of users' learning at the implementation level as well as at the conceptual level.

CUA raises the level of user interface commonly found in these environments by incorporating advances in the state of the art. Action bars, pull-downs, and pop-up windows are examples. Cross-systems consistency is an important factor in CUA, as is the evolution to advanced user-interface capabilities. This evolution should be easier within the common framework that CUA establishes.

The CUA publication is directed toward three classes of readers. It is primarily intended for use by application designers. It should also prove useful to managers and interface planners who need to understand the characteristics of the user interface. Finally, it has been the basis of specifications for toolkits, such as the OS/2 1.1 Programmer's Toolkit, that provide user-interface components for application development.

A rules format with specification-level detail is used to establish a clear and detailed direction prior to general availability of the toolkits. The CUA developers' previous experience indicated that guidelines alone were inadequate. This may be due to a lack of sufficient experience on the part of application designers in interpreting and applying guidelines in varying contexts. Further discussion on the role and effectiveness of guidelines is presented later in this paper.

The CUA publication sets a direction for both application and toolkit designers. Tools for building the user interface are essential and will evolve. As the tools evolve and CUA rules are implemented in toolkit components, the nature of the CUA publication will also evolve. Currently it is a detailed, specification-level, rule-oriented document, because it was a basis on which the specifications for the Presentation Manager and Dialog Manager of OS/2 were developed. In the future, the CUA publication will become more of a guide for using the user-interface components that are provided by the Presentation Manager and the Dialog Manager and less of one for implementing them. Continued achievement of consistency and usability should be possible through conscientious use of the components.

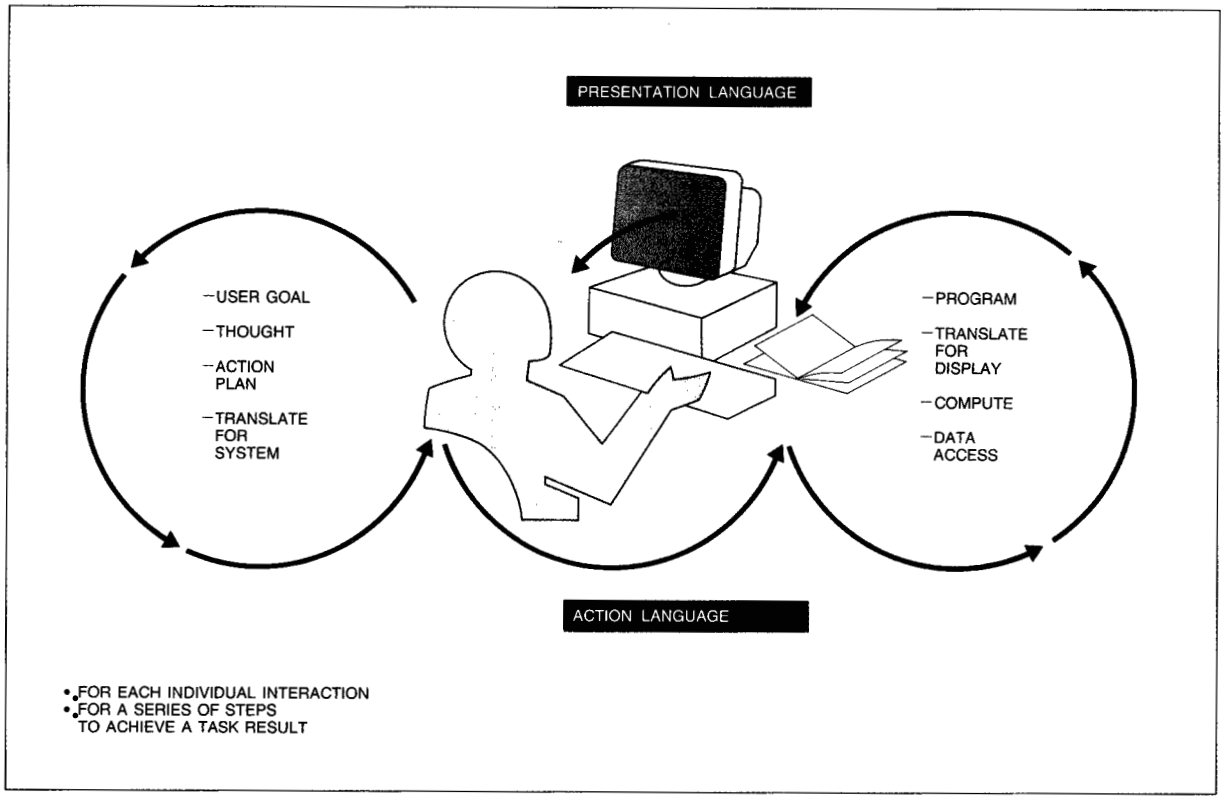
The remainder of this paper presents

- The foundations of CUA—An architectural approach.
- A description of CUA—What CUA is and what it is not. Examples of CUA-defined components are given.
- How developers should use CUA—Suggestions addressing how developers should apply CUA in their design activities, how to deal with what is not there, and how to deal with exceptions.
- Similar activity in the field—An overview of other activities dealing with user-interface guidelines, rules, and standards.

The foundations of CUA

A basis in architecture. CUA is based on a user-interface architecture that has evolved within IBM

Figure 3 A user interface model



over several years. Previous IBM efforts to focus on the growing complexity of the interface had emphasized individual product usability. These efforts were primarily based on existing practices and were documented as conventions for IBM developers. During this period of evolution, major innovations occurred within the industry. Some of the important concepts established during this period are the following:

- The desktop perspective—making the computerized presentation of information parallel that of users' work environments
- Object orientation—making users' data the focus of dialogs with available actions shown in context
- Point and select—recognition of choices is easier for users than recalling commands, as is the use of pointing devices such as a mouse versus typing commands
- Use of graphics and icons—making controls visible and concrete
- Windowing—providing concurrent display of related information
- Modeless design—putting the user in control and reducing the demands on human memory

Abstraction to describe the user's tasks in terms of *objects* and *actions* came into use during this period, as well as guidelines for user-interface design. Bennett[4] has contributed a model of the user interface shown in Figure 3. It includes a presentation language, an action language, and the user's conceptual model. This language analogy and the role of user experience provided a basis for development of the user-interface architecture and CUA.

During this period IBM established a corporate-level focus on the interface through the user-interface architecture. The architecture helps organize industry- and IBM-developed technology and experience in the area of user-interface design in a way that supports IBM customers. CUA represents the application of this architecture to the cross-systems environment defined in SAA.

We are familiar with the concept of architecture as applied in the construction of buildings. Architecture has to do with defining fundamental elements of style. Brooks[5] describes the architecture of a system as "the complete and detailed specification of the

user interface." The term "user interface" applies to each component of the system. For example, as Brooks notes, for a compiler it is the language manual.

A key distinction is made between architecture and implementation.

A key distinction is made between architecture and implementation. The user interface through which function is accessed can be relatively isolated from the internal mechanisms. Implementations of several models of the System/360 and System/370 architectures, based on different technologies, are examples.

Bennett[6] identifies a potential problem in developing a user-interface architecture. He suggests that standards tied too closely to current designs will fail to take advantage of new technology. Conversely, generality will not provide adequate guidance for designers, and users' learning will not be transferred. Iterative refinement is suggested as an approach to this challenge.

Guidelines and conventions based solely on common practices in particular environments tend to reflect the restrictions of those environments. They tend not to be "durable" in the sense of being extendable over time and applicable to changing implementation technology. The architectural basis of CUA supports concepts and relationships that might be held relatively constant as technology evolves.

Principles on which CUA is based. There have been so many suggestions for design principles that they are too numerous to list. The user-interface architecture and CUA reflect many of these principles. The following are examples relating to dialog design:

- Users' actions should be reversible. Within a dialog the Cancel action returns the user to the previous state. If, as the result of picking the wrong choice in a menu, a pop-up window appears, users can Cancel the pop-up window. This action returns the dialog to the menu and allows the user to pick the correct choice. Cancel is generally available at key transition points in the dialog and might be used to back up several steps. A Refresh action is also available. It restores the initial values of panel information altered by users.
- Context is provided to sustain user orientation. Panel titles and scrolling location information help maintain the context for displayed information. When a sequence of pop-up windows is used, each succeeding window is offset from the one underneath. This maintains the context in which the current pop-up window is displayed.
- The interface is highly visual, with minimal reliance on users' memories. Many memory aids and reminders are available. The action bar and pull-downs display available actions in a "menu" approach. Action-bar and pull-down choices can be selected via point-and-select or by typing single mnemonic characters. The user is not required to remember and type commands. The function-key area displays current key assignments, and additional keys are displayed beside respective actions in pull-downs.
- Feedback is provided for almost every user interaction. Color, emphasis, and other selection indications are used to indicate when the cursor is on a choice and when it has been selected. Audio beeps are used when users' action requests do not cause normally expected results.
- User actions that are potentially destructive require confirmation. Messages are displayed in pop-up windows to question intent and provide options.
- Consistency is supported by establishing common definitions in terms of concepts, appearance of displayed information, interaction techniques, and terminology. For example, the characteristics of selection fields are specified and used in essentially the same way in action bars, pull-downs, pop-ups, and panel-body areas. The theory is that if, for instance, a multiple-choice selection field has the same appearance, and selection is accomplished using the same techniques regardless of where the field is used, users' learning will be transferred within and across applications.

Principles are applied during a design decision-making process within a certain context. That is, contextual factors may tend to bias the decisions more in favor of one principle than another, and trade-offs must be made. For example, one principle calls for making objects visible and concrete and might be

taken to imply a visual representation on the display screen, such as an icon. Another principle suggests minimizing the use of screen space. Given that users' skills range from novice to expert, it becomes apparent that trade-offs must be made. It may even be appropriate to offer users alternatives in some cases, causing a conflict with principles recommending the fewest alternatives. The value of design principles may best be realized when applied by user-interface design specialists. Benefits can then be achieved by providing the insights of these specialists in carefully crafted toolkit components and through guidelines for use of the components in application development.

Following are some of the CUA design principles that are unique to the SAA environments:

- *Take advantage of programmable workstation capabilities where possible.* One of the dangers of designing for a cross-systems environment is a tendency toward a lowest-common-denominator solution. CUA attempts to avoid such solutions and specifies some capabilities that are unique to programmable workstations. For example, individual keystrokes can be sensed and acted upon in a programmable workstation. In nonprogrammable terminals, only certain keystrokes (for example, function keys and other interrupt-causing keys) are presented to the application.

  To select from a numbered list of choices on a programmable workstation, CUA specifies typing of the number. There is no typing cursor and no entry space on the display, and the number is not displayed. The selected choice is immediately highlighted via a color change or some other available technique to provide emphasis.

  On nonprogrammable terminals a typing cursor is positioned at an entry space near the list of choices. When the number is typed, it appears in the entry space, and the presence of the number serves as the selection feedback. This technique could have been employed on the programmable workstation, but the advantages provided by immediate keystroke recognition and feedback were deemed more important than consistency.

- *Make the keyboard interface efficient for most common office applications.* In other words, the user should not be required to use a mouse. In some applications, such as those allowing users to draw and paint, and in some scenarios in text-oriented applications, a pointing device such as a mouse is superior. Pointing devices are not in widespread use across the SAA environments. The intent is to provide users in these environments with consistent and practical keyboard-based solutions and not to require use of different input devices in each environment.

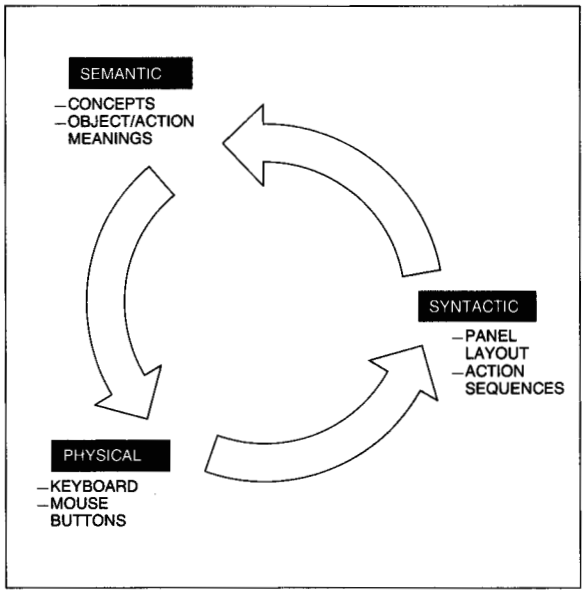
- *Provide concurrent mouse and keyboard support.* Users should be allowed to switch between a mouse and keyboard at almost any point in a dialog. Some tasks, such as drawing, are easier to do using the mouse. Others, like text entry, require a keyboard. Some tasks are situation-dependent. An occasional text-string move during document editing might be most efficient from the keyboard. Performing the move using a mouse would require hand movement between the keyboard and mouse. If a series of moves were to be made, the hand movement might become acceptable in view of the overall efficiency of using a mouse for the series of moves. The CUA approach supports both mouse and keyboard concurrently and allows the user to select the device that is best suited for the situation.

- *Support graphical representations when graphical capabilities are available.* Do not attempt to imitate graphical techniques using alphanumeric displays. Graphical techniques can be extremely useful in presenting information to users. "Radio buttons" provide a visual cue for a single-choice selection field, and icons can be used to depict physical objects such as disk drives. When graphical techniques are not available, as for example when using the alphanumeric display of a nonprogrammable terminal, words typically provide the most effective communication. Use of character symbols in an attempt to imitate graphics typically results in a rendition that is not recognizable.

- *Apply consistency across the SAA environments, especially with respect to fundamental concepts and relationships.* These aspects of a cross-systems design are influenced least by the various technologies involved.

Aspects of consistency. Consistency is a quality exhibited in many systems. For example, a common text-editing paradigm involves using the shift plus arrow keys or holding the mouse button to "mark" portions of a text string. The marked portion can then be replaced by typing, or it can be moved or copied. Consistent implementation of this paradigm in text editors and in field-oriented data allows users to transfer existing knowledge to new situations.

Figure 4 Domains for analysis of consistency



Quality in consistency can be achieved by applying the principle of always being consistent. But saying something is "consistent" is like saying something is "bigger." Bigger than what? Consistency is relative and can only be evaluated within domains that provide alternatives for comparison. The user-interface architecture identifies three domains in which consistency can be evaluated: physical, syntactic, and semantic. These three domains are depicted in Figure 4.

Consistency in the physical domain is exemplified by cross-systems keyboards having the same keys in the same locations. The IBM Enhanced Keyboard is a first step in achieving physical consistency across the SAA environments. Consistent key placement has a direct bearing on learned actions, such as reaching to the upper left to press the F1 key.

Syntactic consistency has to do with order and position relationships. For example, it would be syntactically consistent to have panel titles always appear centered on the first line of a panel. It would also be syntactically consistent if the key sequence used to access an action-bar pull-down were to press F10, move the cursor to the desired action-bar choice, and press Enter.

Semantic consistency is based on objects and actions having the same meaning, that is, representing the same concepts within and across applications. For example, it would be semantically consistent if all applications implemented the Exit action as a way to leave the current application, prompting the user if significant data have not been saved, providing the option to save the data, and returning to an application starting point. This meaning of Exit is independent of the technique (syntax) used to request the action. The user might select Exit from a menu, press an Exit function key, or type a command.

Analysis of consistency within the three domains is useful because of the correlation between these domains and aspects of a design. The physical characteristics of an interface are established by the workstation hardware. These characteristics are the ones most likely to change as workstation technology evolves. Software designers have more control over the syntactic aspects, but even here capabilities such as graphics and input devices vary across environments and continue to evolve. The user-interface architecture concentrates on consistency of semantics—that is, carefully maintaining the same meanings of objects and actions throughout the interface. This domain of consistency appears to offer designers an opportunity for supporting the transfer of users' learning.

CUA design strategies. CUA was also influenced by the experiences of its developers in the three SAA environments. Differences existing in workstation technology, operating systems, file models, programming tool support, and user-interface styles had to be considered. Each environment has large user groups that have made significant investments in learning current concepts and techniques. The challenge was to offer a new user interface with significant advantages over the current ones while at the same time not requiring so much additional learning that users would have difficulty making the change.

The programmable workstation was acknowledged as providing the capabilities that best facilitate advances in user-interface technology. The most significant factor is the high degree to which a personal computer can interact with users. The strategy of designing for programmable workstations and adapting to nonprogrammable terminals was therefore chosen.

More emphasis was placed on usability *within* an environment than *across* environments. This emphasis is based on the expectation that the heaviest use of a workstation or terminal is within an envi-

ronment and that occasionally users will cross environments. In trade-off decisions, what was considered right for an environment took precedence over cross-systems consistency.

This approach appears to provide the best opportunity for moving users in each of the three environments toward a single design. It provides improvements over currently implemented capabilities in each environment, and it establishes a platform from which additional advances can evolve. Perhaps the greatest significance of CUA is in the fact that design work in each of the three environments is directed toward common goals, and evolution will take place in orderly, planned, and compatible steps.

CUA is a set of rules and guidelines.

Validation. The results of human factors testing are one measure of success. Because CUA is not itself a product, there is no single product design to test. Therefore, several testing approaches were used to test individual components of CUA as well as simulated products based on the CUA rules.

Actual testing of the CUA was conducted over a period of approximately seven months at six different test sites. The test participants were typically obtained from temporary employment agencies. This approach provided good proportions of users having previous computer experience and users with no prior computer experience.

In some tests a single design was evaluated on the basis of users' ability to complete the tasks and the amount of assistance required. Users' attitudes were often obtained by administering a questionnaire after the test.

In transfer testing, a programmable workstation design and a nonprogrammable terminal design were tested. Users were given tasks to perform on first one device and then the other. Some users were required to switch back and forth several times within a series of tasks. Measurements were again recorded and an assessment of transfer of learning and interference effects was made.

This testing has demonstrated that applications having ease-of-learning and ease-of-use characteristics, and at the same time supporting a transfer of users' knowledge across systems, can be developed using the CUA rules. The primary responsibility for application usability remains with the application developer. Application developers must apply CUA within a development methodology on the basis of knowledge of users' tasks and skills, the use of prototypes, testing, and iteration.

Description of CUA

CUA is a set of rules and guidelines governing many of the key aspects of user-interface design. (See Figure 5.)

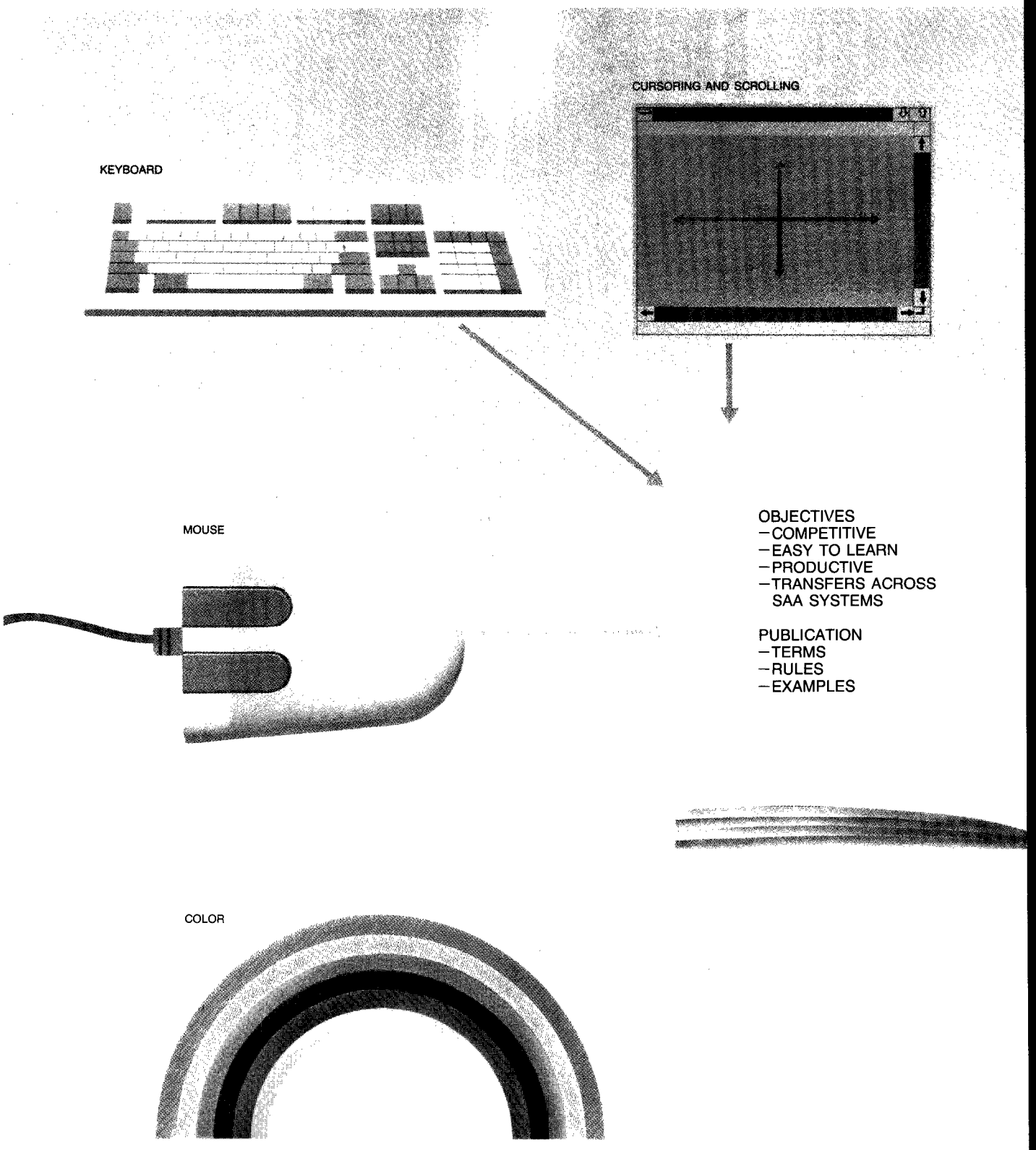
Where consistency is deemed more important, the rules leave less room for interpretation. In other areas the rules are only as tight as was felt necessary to achieve a desired effect, if not exactness. And in yet other areas, deemed to be of lesser importance to transfer of learning, guidelines are provided. Even within the spectrum of rules, designers are allowed some flexibility. In some cases "application options" are specified, meaning that the designer may choose from one of several specified approaches.

The following interface topics are included in CUA:

- Use of windows
- Panel design, specifically panel layout, panel types, selection fields and entry fields, cursor movement and scrolling, and color and emphasis
- Dialog design in the form of dialog control actions and pop-up window dialogs
- User aids provided by messages and a Help facility
- Key assignments
- User options
- National-language support
- Terminology

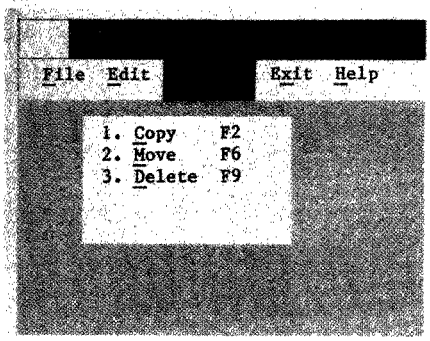
Use of windows. CUA does not currently specify rules for designing a window manager, such as how to move and size windows, but it does specify rules for applications to follow in a windowing environment. The CUA model for information display assumes a windowing environment. It treats nonwindowing environments as a subset in which the windows are

Figure 5 Topics in CUA

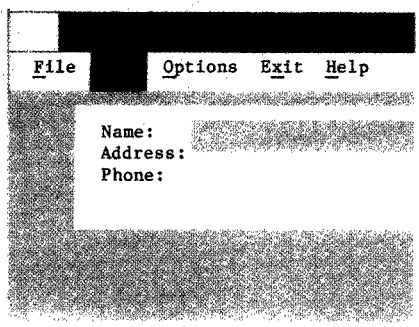


NATIONAL LANGUAGE

SELECTION



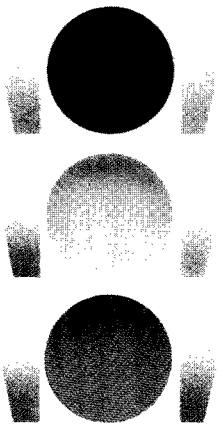
DATA ENTRY



IWS EXPLOITS
- WINDOWING
- COMPUTER POWER
- GRAPHICS
- KEYBOARD/MOUSE

MFI OPTIMIZES
- FULL SCREEN
- KEYBOARD
- ALPHA/NUMERIC

MESSAGES



HELP

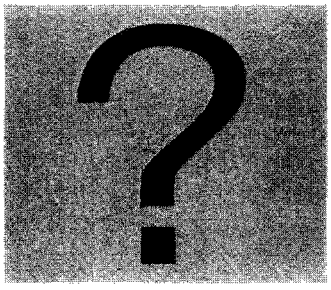
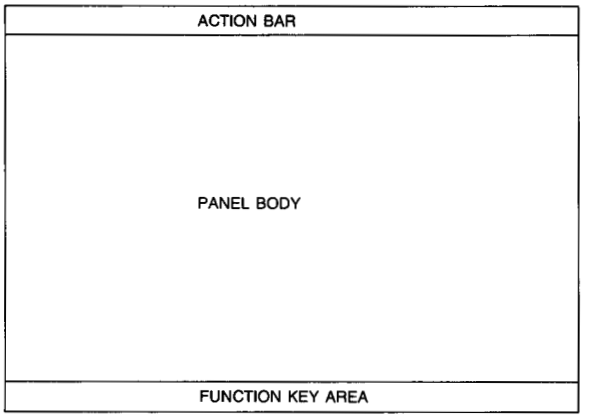


Figure 6 CUA panel format



full-screen in size and cannot be made smaller. This approach provides consistency between the two environments when windows having adjustable sizes are "maximized" (made to be full-screen in size). By the addition of window moving and sizing for users moving from a nonwindowing to a windowing environment, capability is extended in the interface without changing existing interfaces.

CUA has been developed in conjunction with the OS/2 Presentation Manager. The Presentation Manager provides windowing for the personal computer environment. CUA specifies how applications use windows by defining three types of windows that are related to three types of dialogs: primary, secondary, and pop-up. The rules specify how each window type appears, how it is used in the dialog, and how it is removed. Rules also specify how the contents of windows should be reformatted and clipped as users change the size of a window.

Panel design. Items to be considered in designing panels are now discussed.

Panel layout. The CUA model for information display specifies that information is presented to the user via panels displayed in windows. A panel is a specific combination and arrangement of information, and a window is a display mechanism for that information. Application designers create panels, such as menu panels, spreadsheet panels, and text-editing panels, which are then displayed to the user in one of the three types of windows, depending on the dialog context in which the panel is to be displayed.

CUA specifies a general panel format consisting of the three areas as shown in Figure 6.

The *action bar* is used to display choices users can select. Each choice on the action bar represents a group of choices available from the current panel. When users select an action-bar choice, a pull-down containing a list of choices dynamically appears. These pull-down choices typically represent actions that change the data in the panel body or allow users to access other application function.

The *panel body* is used to display information with which users interact and on which actions initiated via pull-downs may operate. It is the user's main focus for the panel. This relation between the information displayed in the panel body and the actions in the action bar and pull-downs is characterized as an object–action interface.

Users interact with information in the panel body by selecting objects, such as the name of a memo, and then selecting an action from the action bar, such as Print. This approach allows the list of available actions to be dynamically tailored to match the type of object selected. For example, in an editor, if users select a text string, a Change Font choice might be made available, but when they select a graphics rectangle, the Change Font choice would become unavailable. A Change Line Style choice might become available for graphics objects such as the rectangle. The action bar helps users by providing visual cues and access to the available actions. Users are not required to remember and type commands.

The *function key area* serves two purposes: It provides a visual reminder for key assignments, and it provides a touch area that allows mouse users to request functions assigned to keys. Users can choose between a long form or short form of this area, or they can choose not to display it at all, making more space available for the information in the panel body. The short form displays a minimal set of action choices typically required by mouse users, such as Enter, Cancel, and Help. The long form includes additional choices typically required by keyboard users, such as forward and backward scrolling actions, as well as any additional actions the application designer may wish to include.

Panel types. Some types of panels are common across many applications. Among these types are panels for selecting choices, such as typestyles in an editor, and panels for entering parameters, such as the format-

ting options for margins and line spacing. Since panels such as these occur so frequently across many applications, CUA defines them as standard panel types. It defines a menu panel, an entry panel, a list panel, and an information panel. For IBM logo products it defines a logo panel in which IBM's copyright information appears.

Each of these panel types is unique in terms of content and layout and is optimized for a particular role in the dialog. Many of the panels in a typical application will be instances of these common types. Only the application-specific panels need be different from one application to the next.

Each panel is made up of components such as panel titles, scrolling information, headings, field prompts, selection fields, and entry fields that are defined by CUA. Each component has a specified role in presenting information to the user, and each has specified appearance and interaction characteristics. CUA also specifies how users are to move the cursor between components, that is, how the tab and arrow keys work in each panel.

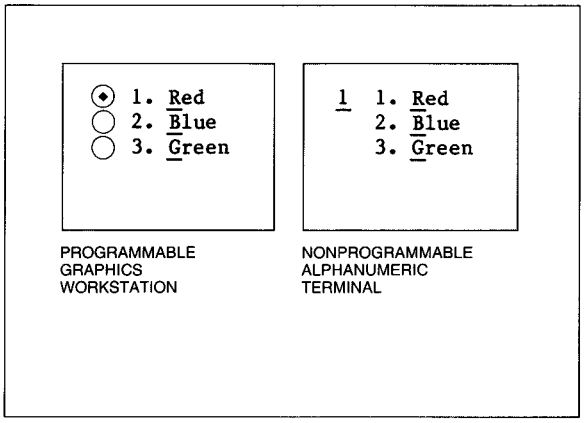
Selection fields and entry fields. Two of the most common interactions users have with computers are selecting from lists of choices and typing information into fields. CUA specifies two panel components, the selection field and entry field, to support these interactions.

In applications using a command language approach, users have to remember and type the names of commands, along with parameters such as file names. Users tend to learn more quickly when using applications that present choices and allow users to choose, rather than when they are required to remember and type. This tendency has been characterized as "recognition is easier than recall," and is a basis for "point-and-select" user interfaces.

CUA specifies a point-and-select interface in such a way that it is usable from both keyboard and pointing devices, such as a mouse, concurrently. In other words, users can switch back and forth between the keyboard and a mouse at any point in the dialog, using whichever device is most convenient for what they are doing at the time.

CUA specifies three types of selection fields: *single-choice*, *multiple-choice*, and *extended-choice*. It specifies rules of appearance and interaction for each type. For example, a single-choice selection field allowing users to select a color might appear as shown in Figure 7.

Figure 7 Single-choice selection fields



The "radio button" in front of each choice on graphics displays is a cue that this is a single-choice selection field. A black dot in one of the radio buttons indicates the choice that is currently selected. The single one-position entry field in front of the first choice on alphanumeric displays serves the same purpose.

Users can select a choice by typing the number or mnemonic (underscored letter) of the desired choice. In special selection fields called "auto-select" fields, users can select a choice by moving a selection cursor to the desired choice and requesting Enter.

In multiple-choice selection fields, users can choose more than one selection. For example, a multiple-choice selection field allowing users to choose formatting options for document headings might appear as shown in Figure 8.

The "checkbox" in front of each choice on graphics displays is a cue that this is a multiple-choice field. When users select a choice, an "x" appears in the box. The one-position entry fields in front of *each* alphanumeric choice serve the same purpose. To select a choice users can position the cursor at the desired choice and type a slash ("/"). On programmable workstations users can toggle the selection on and off via the space bar. On nonprogrammable terminals users can select by typing any character, including the slash. They can deselect by typing a blank (space bar) over the selection character. These

Figure 8 Multiple-choice selection fields

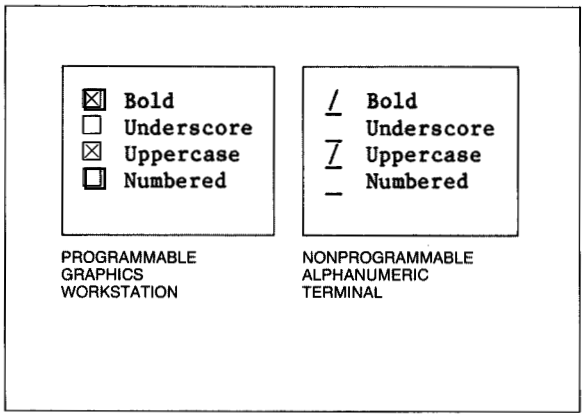
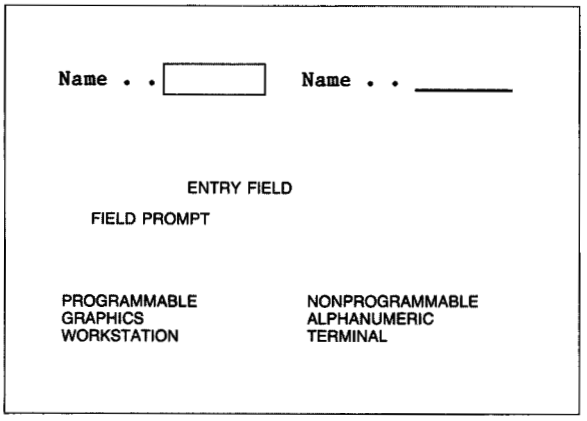


Figure 9 Entry fields



additional techniques are provided for convenience of use in each environment.

An extended-choice selection field is one that has two states, single-choice and multiple-choice. In its initial state it behaves as an auto-select single-choice field. Users move a selection cursor to the desired choice and request the Enter action. If users employ a multiple-choice selection technique prior to requesting Enter, for example by typing a slash, the field becomes a multiple-choice field. The extended-choice field is most valuable in situations where a single-choice selection is typical but where multiple choices can be allowed. Auto-select fields are common in typical applications. Novice and casual users can use extended-choice fields in exactly the same way. More experienced users can take advantage of the multiple-choice capability, for example, by selecting the names of several files to be archived at the same time, instead of archiving one file at a time.

An *entry field* is a space in a panel in which users type information such as names, dates, and account numbers. Entry fields are typically preceded by a field prompt indicating what kind of information users should type into the field. A typical entry field with a field prompt is shown in Figure 9.

CUA specifies appearance characteristics for entry fields and how users are to type information, move the cursor, and correct typing mistakes. Because the CUA-defined panels use entry fields in the same way, once users have encountered an entry field in one panel, they should recognize and know how to use an entry field in any of the other panel types.

Cursor movement and scrolling. CUA specifies use of the arrow keys (left, right, up, and down) and tab and backtab keys for moving the cursor between fields in a panel. The tab and backtab keys move the cursor from one field to the next following the order of the fields in the panel. The arrow keys move the cursor to the nearest cursorable item in the direction indicated, that is, to an entry field or a choice in the same or a different selection field. These cursoring capabilities allow users to choose the cursoring strategy that is most efficient for each situation.

Two types of scrolling are specified. Scrolling occurs when the cursor encounters a panel area boundary on a programmable workstation. Users can also request forward, backward, left, and right scrolling of panel areas independent of the location of the cursor on both programmable workstations and nonprogrammable terminals.

Color and emphasis. Color and techniques for emphasis are used to make distinctions, to provide feedback, and to indicate importance. These operational uses of color and emphasis are intended to help users interpret displayed information quickly and correctly.

One of the color specifications in CUA prescribes a distinction between items that are selectable (choices) and items that are not (protected information). For example, on a white background, the specifications call for titles, headings, field prompts, and other protected information to be blue and for choices to be black. Whenever users see something black, text or an icon for example, they should know

it is cursorable and selectable. This distinction should facilitate a learn-by-exploring approach.

Users' preferences for specific colors tend to vary. CUA specifies rules for allowing users to personalize colors in two ways: by changing the color of individual items, one at a time, and by changing the colors of all items simultaneously to one of several predefined sets of colors.

When users are allowed to change the color of individual items, they can become entangled in conflicts among different colors. For example, if users change the color of one item, it may no longer be distinctive with respect to another. If they change the color of the second item, it may conflict with yet a third, and so forth. After many such trials, users may simply give up and restore the original colors. Defining sets of colors for multiple items tends to be difficult because of color conflicts. Windowing environments cause added difficulty because the dynamics of window placement create a variety of positioning combinations.

Dialog control actions allow users to direct the conversation between themselves and the application they are using.

CUA specifies several color palettes (color-coordinated sets) from which users can choose. These palettes provide a useful degree of color tailoring while avoiding the pitfalls associated with tailoring individual items. Support for tailoring individual colors is also specified for users who desire to do so.

Dialog design. There are two major parts to dialog design—dialog control actions and pop-up window dialogs.

Dialog control actions. Dialog control actions allow users to direct the conversation between themselves and the application they are using. The term "dialog" is used here in the same sense as a dialog between two people, that is, a conversation. In human–computer dialogs, users should be in control. This situation should result in their feeling less intimidated by the computer.

Enter is a common dialog control action. It is one of the ways in which users turn control over to the computer. Enter typically causes some action to occur as determined by the application designer, not the user. To put the user in more direct control, CUA specifies two basic techniques: an object–action approach to the dialog, and providing ways for users to back up when they have made a mistake or changed their minds.

The basic characteristic of an object–action approach is presentation of the information with which the user is working concurrently with a list of actions that can be requested. For example, while a user is working with incoming mail, a panel might display a list of the mail received, along with actions such as view, send a reply, file, and forward to another user. At any point in time, users can decide which action should be requested on which mail items, and they can perform the actions in whatever order is most useful. This approach reduces menu hierarchies and action modes that tend to force users through paths prescribed by the application designer.

Another way CUA gives users control is by allowing them to back up in the dialog. Users should feel more comfortable with a computer if the results of their mistakes do not have serious or irreversible outcomes. "Learn-by-exploring" environments involve the trial-and-error process, which requires reversible actions.

CUA specifies standard actions to help the user in this area. The Refresh action displays the current panel as it was before any user changes were made, and a Cancel action takes the user back one step. For example, Cancel from the current panel returns the dialog to the prior panel. The Exit action allows the user to quickly leave the current dialog and return to the point where the dialog began.

Application designers are encouraged to provide "undo" capabilities. Such capabilities tend to be specific to particular applications and have not been addressed in the current CUA. Undo capabilities that appear to apply across a variety of applications may become candidates for inclusion as CUA evolves.

Pop-up window dialogs. Applications may request additional information from users via a pop-up window dialog. CUA specifies that the dialog with a pop-up window must be completed prior to resumption of the dialog with the underlying window. Window-positioning alternatives are also provided. When several pop-up windows are presented in sequence, they are overlapped to preserve context. The Cancel action removes the pop-ups one at a time and allows users to back up and change previously entered information.

User aids. Two aids are provided for users—one by way of messages and the other by way of a Help facility.

Messages. The topic of messages has often been dominated by the concept of "error messages." The intent of CUA is to support the interpretation that users often make proper and appropriate requests that are not understood by the computer. The CUA approach is to help users through the situation with nonintimidating messages and prompting. In a dialog between two people, one of them might ask, "What exactly do you mean? Can you clarify?" It is the responsibility of the application designer to take an active role in aiding such communication.

CUA specifies that messages are displayed in panels using one of the standard panel types: menu, entry, or information. The panels are displayed in pop-up windows like many other such panels normally encountered in a dialog. A message line within a panel is also supported as an alternative for some situations.

Messages tend to be spontaneous interruptions of the dialog by the application. This is a useful distinction that separates messages from the dynamic display of other information. The importance of the interruption is a key aspect and is the basis on which messages are categorized. The three types of messages are notification, warning, and critical. CUA specifies for each message type the presentation form, how users interact with the message, and how and when the message is removed. Application designers choose, on a situation-by-situation basis, which type of message to use on the basis of the importance of the interruption.

For example, if users request a Print action but fail to specify an available printer correctly, the message might consist of a menu panel in which the available printers were listed. This panel would allow users to select a printer and complete the print request. With this approach, users should view the application as being helpful and assisting them in accomplishing their tasks.

Help. An on-line contextual Help facility is important because it assists users in completing tasks with minimal interruption. CUA prescribes rules for two aspects of help interaction—the structure of the help information and the techniques for requesting it. The rules prescribe a multilevel help structure that provides information ranging from specific (field-level) to general (panel and application levels).

Consistent use of terms can help users form the proper conceptual model of the interface.

Help is contextual because the information presented is related to what the user is currently working on, as determined by the location of the cursor when users request help. Users can also request help for a panel, help on the keys used in the application, a help index that provides an alphabetic index to all help information, or an optional help contents organized by topic (types of tasks).

Help panels are standard CUA-defined panel types, and users request help via the same interaction techniques used in non-Help panels. For example, the help index and help contents are displayed in menu panels. Users can access more information on an index item or a contents topic by using standard selection techniques.

Key assignments. Key assignments are specified for CUA-defined actions. Application-specific actions may be assigned to keys for which there is no CUA-defined action, on a panel-by-panel basis. CUA encourages application designers to allow tailoring of key assignments by users.

User options. CUA specifies several user options; for example, users are allowed to choose from several

color palettes. In some environments (such as OS/2) a system service allows users to specify preferences. CUA requires that applications in these environments support the system-level user preferences. Users may also be allowed to establish application-unique preferences.

National-language support. Applications to be translated into other languages need to follow some rules to aid the translation process. The majority of IBM applications are translated into many languages, so CUA specifies rules for translation, non-English character sets, and double-byte character sets. It also specifies the terms to be used in each supported language.

Terminology. The terms used in the interface should convey intended distinctions to users. Consistent use of terms can help users form the proper conceptual model of the interface, that is, one that matches the designers' interface model. This consistency should allow users to accurately interpret what is seen and decide on an appropriate course of action to accomplish their goal.

Once a term has been chosen for a particular concept, it should be used throughout the design to convey the same meaning. The use of synonyms should be avoided. For example, the Exit action allows users to leave the current function in an application and return to a prior function. Two variables associated with this action are the point to which the dialog returns and the status of the information on which the user was working, whether it is saved or not. Whenever users see and use an Exit option, they should think in terms of these two variables. If the action is presented as Exit everywhere this meaning is intended, users' conceptual models should be reinforced, and they should learn this aspect of the interface quickly.

CUA specifies a set of terms for use in displayed information, help information, and publications addressed to users. The CUA publication includes a glossary of English terms and provides translations in fifteen languages.

How developers should use CUA

Application developers can best apply CUA by first becoming familiar with its intent and spirit. This paper and a review of the CUA publication, concentrating on themes as opposed to details, should prove useful in developing an application design.

Users probably spend more time in application-specific dialogs using application objects and actions than they do using CUA-defined elements. For example, in using a text editor, the major tasks are entering and revising text, setting document format parameters, adjusting paragraphs, underlining, copying blocks of text, and so forth. CUA-defined components might be used during these tasks, as in selecting from a menu of format options, but the concepts involved, the objects users work with, and the actions they request are all creations of the application design. These components will be a major factor in establishing the usability characteristics of the application. CUA-prescribed rules that may affect the overall application structure, such as the meaning of Exit in a multiple-level dialog, can be taken into consideration at this level of the design.

Once an application framework has been decided upon, the CUA rules can be applied during the detail design phase, including such rules as prescribing where the Exit option should appear and how users request it. As programming development tools evolve, many of the rules will be incorporated in the form of user-interface components. For instance, the OS/2 Presentation Manager enables action bars, pull-downs, pop-up windows, selection fields, entry fields, and several other CUA components. CUA rules for appearance and the interaction characteristics of these components are built into the Presentation Manager.

In those environments where such tools are not yet available, developers should start implementing as much of CUA as is practically possible, using their current development tools. Trade-offs will have to be made. Some CUA components may be more difficult to implement than others: For example, action bars and pull-downs may be more difficult than function-key areas. In some cases, CUA supports alternatives that can be used in the interim. For example, the list panel is similar to a menu panel with an action bar. It displays a list of items, such as file names, and a list of actions above the items. Users can select one or more items and then specify an action, similarly to the way in which a menu panel with an action bar is used.

For other rules current development tools are already adequate. For example, CUA specifies rules for panel titles, column headings, field prompts, and the display of key assignments (the function key area) that developers should be able to implement with existing tools.

Each rule implemented today gets the application closer to the goal and should result in less change for users as CUA tools become available.

The current level of CUA does not offer all of the components needed for many application designs. For example, there is currently no specification for a multiline entry field. Application designers should develop their own specifications, following as closely as possible precedents established in CUA. For example, the specifications for moving the cursor in a multiline entry field should parallel those for a single-line entry field. Application-defined components may become candidates for inclusion in CUA as it evolves.

The programming development tools will implement many of the CUA rules over time.

CUA is not a substitute for conscientious design to achieve usability, nor is it a substitute for usability testing and iterative design. During the development process, conflicts may arise between usability results and implementations based on CUA. The rules attempt to provide latitude for application designers in areas where such latitude is deemed desirable. For example, CUA does not specify detailed rules for the panel in which help for keys is given. The minimum requirement is for an information panel (i.e., read-only information) listing each key used in the application along with its meaning. Application designers are encouraged to use a menu panel that allows users to select each key for a more detailed description of its function. A graphical picture of the keyboard with selectable keys is also allowed. Many such aspects of the user interface are best left to the application designer.

Latitude is bad to the degree that it allows arbitrary differences to impact transfer of users' learning. In some cases, designers may be able to alleviate problems by exercising design alternatives within allowed latitudes. Designers should keep in mind the principles used in the development of CUA and make trade-off decisions based on balancing cross-systems consistency with optimization in a particular environment.

Within IBM a process exists for handling exceptions to the rules. When a product demonstrates justifiable reasons to be different and provides evidence that negative effects are minimal and acceptable, exceptions are allowed. Thus far, exceptions have been rare. One example is the requirement for Exit in the action bar. There is no exit capability from the operating system; therefore, the action bar in the OS/2 primary window does not contain an Exit choice.

Software vendors designing applications for the SAA environments should use a similar process, balancing the gains achieved against potential negative cross-applications and cross-systems effects.

The programming development tools will implement many of the CUA rules over time. The Presentation Manager and Dialog Manager together will implement more than half of the currently defined set of rules. Rule support can be provided at two levels, an enabling level and a conformance level. *Enabling* means that the tool allows the programmer to follow the rules by providing basic capabilities. *Conformance* means that, by default, the tool enforces the rule but may allow the application to override conformance to the rule. It is likely that some rules will be implemented at the conformance level and others at the enabling level, depending on the degree of control desired for each user-interface component. In the interim, conformance of IBM-developed applications is being enforced via a management process.

Approaches to assist in conformance assessment are being evaluated. The use of an expert system has been suggested. With the CUA rules as a knowledge base, an application developer might determine conformance by answering queries from the system. This approach appears to be problematic because it does not help provide designers with the insight needed to create usable applications. Implementation tools that provide automatic support of the CUA rules remain the best-known approach. Through this approach, developer productivity can be improved while a useful degree of consistency is ensured. This approach will allow CUA to evolve from detailed rules toward more general guidelines for using the tools.

Similar activity in the field

Smith and Mosier[7] have compiled 944 guidelines for design of user–systems interfaces. They address the areas of data entry, data display, sequence control, user guidance, data transmission, and data protection. These guidelines are being applied in the design of software for military command systems that parallel the range of systems included in the SAA environments. In a recent survey sent to 400 users of the guidelines, Mosier and Smith[8] concluded that guidelines were useful but that problems exist with generality and ease of use of the guideline document. Generality of the guidelines is defended as making them applicable to a variety of applications. The conclusions indicate that guidelines may be too general for application developers to apply directly. The suggested solutions involve providing tools to help software designers tailor the guidelines and providing an on-line access tool.

The CUA publication and guidelines such as those by Smith and Mosier are being examined by formal standards organizations. The International Organization for Standardization, Technical Committee 159, Sub-Committee 4, Working Group 5 (ISO TC159 SC4 WG5) on Software Ergonomics and Man–Machine Dialog is addressing dialog interface and software usability definition and measurement. The Human–Computer Interface Standards Committee of the Human Factors Society is the technical advisory group for the American National Standards Institute (ANSI) in this effort.

In Germany, DIN 66234 Part 8, Principles of Ergonomic Dialog Design, was adopted as a standard in 1987. It specifies dialog design principles such as making the dialog design suitable for the task. No means for assessing conformance are stated or proposed. It is expected that implementation and conformance will be problematic.

Other standards activities include International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) Joint Technical Committee 1, Sub-Committee 18, Working Group 9 on User–System Interface and Symbols (ANSI X3V1.9 provides American representation) and International Telegraph and Telephone Consultative Committee (CCITT) Study Group 10 on Language and Methods for Telecommunication Applications (with ANSI T1M1 representation). The CCITT group is studying user guidance, menus, dialog elements, and other user-interface topics.

In general, these standards groups are working toward a determination of what can and should be standardized. With the exception of DIN 66234 Part 8, there appear to be no imminent prospects for user-interface standards.

Smith[9] believes that present knowledge supports development of guidelines but does not justify imposition of standards. IBM participates in these standards activities and believes adoption of user-interface standards now would be premature.

Summary

CUA is based on an architecture that allows it to be applied across environments to support transfer of users' learning. This architectural basis should also serve to make it durable and extendable over time.

The initial level of CUA addresses many user-interface components that are common across applications. More need to be added. As design experience is gained, some rules may be relaxed; others may become more stringent. Additional appearance and interaction techniques will be added as new workstation technologies become available.

The rules approach was deemed appropriate for the current environment. The level of detail serves as a basis for toolkit specifications and may help developers avoid some of the problems with designer interpretation reported by Mosier and Smith. The overall goal continues to be one of providing support for the user-interface components via programming development tools. The OS/2 Presentation Manager is a significant advance for personal computer application design, partly because of its provision of user-interface components.[10]

CUA sets a direction for the development of IBM products and the products of software vendors in the SAA environments. But the benefits are not limited to just these environments. Many aspects of CUA remain applicable and will result in transfer of users' learning across applications regardless of the environment. Application designers in any environment can take advantage of the concepts and techniques found in CUA.

Ultimately the users of products conforming to CUA will receive the greatest benefit. They will be able to learn how to use these products rapidly, and much of what they learn using one product will be applicable to others. Just as the toolkit approach frees the

application developer to concentrate on application-specific problems, so CUA will free the application user to concentrate more on the task to be accomplished rather than on the computer-based methods required.

Cited references

1. *Systems Application Architecture: Common User Access, Panel Design and User Interaction*, SC26-4351-0, IBM Corporation (1987); available through IBM branch offices.
2. E. F. Wheeler and A. G. Ganek, "Introduction to Systems Application Architecture," *IBM Systems Journal* **27**, No. 3, 250–263 (1988, this issue).
3. Jon Meads, "The standards factor," *SIGCHI Bulletin* **18**, No. 3, 22–23 (January 1987).
4. J. L. Bennett, "Tools for building advanced user interfaces," *IBM Systems Journal* **25**, Nos. 3/4, 354–368 (1986).
5. Frederick P. Brooks, Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Co., Reading, MA (1982), pp. 41–50.
6. J. L. Bennett, "The concept of architecture applied to user interfaces in interactive computer systems," *Conference Proceedings, INTERACT 84*, London, September 4–7, 1984, B. Shackel, Ed., North-Holland Publishing Co., Amsterdam (1985), pp. 865–870.
7. Sidney L. Smith and Jane N. Mosier, *Guidelines for Designing User Interface Software*, ESD-TR-83-122, MTR 10090, MITRE Corporation, Bedford, MA (August 1986).
8. Jane N. Mosier and Sidney L. Smith, "Application of guidelines for designing user interface software," *Behavior and Information Technology* **5**, No. 1, 39–46 (1986).
9. Sidney L. Smith, "Standards versus guidelines for designing user interface software," *Behavior and Information Technology* **5**, No. 1, 47–61 (1986).
10. Ed McNierney, "The user at the controls," *PC Tech Journal* **6**, No. 3, 65–75 (March 1988).

Richard E. Berry *IBM Entry Systems Division, 11400 Burnet Road, Austin, Texas 78758.* Mr. Berry is a senior programmer and is lead architect of the Common User Access. His work experience since joining IBM in 1968 has included positions in field engineering, in systems engineering, and in the lead design of user programming, display formatting, file processing, and report generation products for industry and office systems. Mr. Berry has served in lead technical and management positions in both product programming and architecture development.

Reprint Order No. G321-5325.