A view on windows: 
Current approaches and neglected opportunities

by BENN R. KONSYNSKI, ARNOLD GREENFIELD, and WILLIAM E. BRACKER, JR.

University of Arizona
Tucson, Arizona

ABSTRACT

This paper offers a taxonomy of current approaches to the use of windows in screen dialogues of application systems. Windows are viewed as a mechanism for managing presentation space in the design of information systems. Several neglected application areas are explored, including the use of multiple windows in the design of single applications. The potential and limitations of using windows is also discussed.
INTRODUCTION

As the office becomes computerized, we are seeing increased use of sophisticated software packages in all areas of the business environment. For example, word processors help secretaries prepare letters, while spreadsheet programs help both accountants and strategic planners. However, even with all their power, many of these packages are failures because they are difficult to use and present their results poorly. As a result, research has proceeded in earnest towards improving both the elicitation and presentation of information between man and computer.

Information systems design has generally emphasized function while neglecting form, yet designing the format of the interface is just as important to system success, and may prove even more difficult than identifying and specifying functionality. Specifically, designers have had to work with a limited presentation space constituted by the viewport onto the system. That presentation space is the loci of the system’s presentation of information to the user, and the user’s clarification of communication with the system. As the central focus of the dialogue, it has often determined the limits of the user/system dialogue.Neglecting the management of this space results in system resistance, and encourages situations where dialogue is a constraint on effective use, rather than a facilitator.

Many breakthroughs in user interface design have resulted from studying the physical work environment, and determining whether management can make effective use of the growing computing capability. This has resulted in the adoption of certain motifs that form the basis for dialogue design in office systems. For example, when working at his/her desk, a manager generally has several items present; a pad of paper may be in front of him; a book may be open on his left; and a series of graphs indicating the previous year’s sales performance may be on his right. Such an arrangement can be translated to the computer in that the screen can be divided up into several areas or windows, with the user being able to work on different tasks in each of the windows rather than being limited to one task interaction at a time (as is currently the trend). As the computer environment more closely resembles the manager’s work environment, it should be easier for the manager to use the computer in a decision support capacity.

The notion of splitting the screen into multiple windows goes well beyond the desk paradigm. Windows help users to organize their work and integrate several subtasks. Current interfaces are generally set up for performing these subtasks sequentially. However, we know that users often work in parallel, switching their attention between several items. By allowing the user to interact with various subtasks on the same screen, windows help the user to integrate the results of the subtasks and thereby complete the overall task.

Office workers are frequently involved in the monitoring of change, or find themselves waiting for important messages. Managers are often interrupted or alerted to important events in the organization. The manager’s connection with the electronic network in the organization is a limited port or presentation space for such transactions, but windows and icons provide the manager with multiple viewports for situations that are normally handled by phone calls, distracting physical interruptions, untimely notes, and other forms of interruption. Windows can allow the user to accept, prioritize, or delay such interruptions, alerts, or reminders.

As might be expected, windows also have certain drawbacks. Many users have complained that windows can quickly become too small to adequately display information. Some complain about the slowness of the system. Others question the usefulness of working with more than one task at a time, citing limitations in short term memory. Market performance of currently available window packages seems to echo this sentiment; sales are low and vendors are drastically reducing their prices. (for example, one vendor package was reduced from $495 to $95).

As we shall see, the reasons given for user disappointment are only partially founded. In particular, the most likely reason for the dismal performance of window packages to date is not the infeasibility of windows per se, but how they have been applied and implemented. This paper reviews the windowing concept, details the current environment and discusses how appropriate window management can improve system utility and usability. We first give a brief history of windows, then review several of the current window support environments, discuss certain neglected opportunities for window use, and finally describe the benefits and limitations of windows.

A BRIEF HISTORY OF WINDOWS

One of the first applications to use windows was a text editor developed by Engelbart in 1973. The overall task context was well understood; the windows were useful in helping the user concentrate on the work at hand while keeping the overall problem context in the forefront. In 1976, windowing was extended to a multi-tasking environment by Buneman and colleagues in the implementation of the DAISY system. Several independent tasks could be displayed on one screen, and the user could direct input to and receive output from any one of them. However, transferring data between windows, an important part of task integration, was not well defined.

The first formal window environment available commer-
cially was the Xerox STAR. The STAR displayed a desktop composed of various icons, such as an in-box, out-box, and file cabinet. When "opened," the icons became windows on the screen, through which the user could interact with the system. Several of the techniques for manipulating windows were quite innovative, and were carried over into future window packages. Unfortunately, the STAR initially proved to be rather expensive for general users.

The real breakthrough for windowing occurred when windows became feasible for microcomputers. Due to the improvements in microprocessors and reduction in memory costs, software vendors rushed to develop window packages that would "revolutionize" the design of user interfaces. It is the availability of these new systems that has sparked both a renewed interest in windowing opportunities, and a glut of disappointing implementations.

We next discuss several of the initial directions that software vendors have taken in their attempts to provide window support for microprocessor-based applications and office environments.

A REVIEW OF CURRENT WINDOW ENVIRONMENTS

The windowing concept has been exploited by designers in various ways. Alternative designs reflect the various skills or technologies that were brought into play when developing the windowing capability. Three basic approaches for window use have surfaced.

1. Integration within the operating system
2. Multiple-application support
3. Business shell environments

Let us briefly examine each of these approaches.

Windows Central to the Operating System

Making windows a central mechanism to interface with the operating system is becoming increasingly popular. For example, Xerox, Apple, and Apollo have window-oriented operating systems. These systems make use of windows, icons, and often a mouse to facilitate the presentation, elicitation and identification of operating system command functions. These systems require the presence of high resolution displays to exploit window and icon concepts.

With the STAR, users can open one or more icons, resulting in several partially overlapping windows on the same screen. The STAR design goal was to make everything relevant to the current task visible on the screen; therefore, the desktop, while frequently obscured, always remains on the screen. This design goal is instrumental in helping the user organize his subtask while keeping the overall task in mind.

The Lisa and Macintosh from Apple Computer have improved upon some of the STAR concepts to aid task integration. In addition to using icons to indicate system functions, Lisa and Macintosh allow users to transfer information between windows. For example, the user can work on the text for a document in one window, and draw a figure for the document in another. The figure can then be transferred to the text window, appropriately positioned in the document. The mechanisms to indicate such transfers are more easily identified in the Apple implementation than were accommodated in earlier implementations.

Lisa and Macintosh (and several other systems) also make use of the pull-down menu, where lists of options appear in a window that partially obscures current windows. Once the user has made a choice, the menu disappears, again revealing the overlaid windows. This technique helps the user visualize current options in the context of the overall task.

The Apollo system’s windowing is somewhat different and less advantageous than that of Xerox or Apple in that its various windows do not have a unifying concept such as a desktop. Rather, each user-created window functions as a separate terminal. As a result, task integration is not as strong with Apollo as with Xerox and Apple. Apollo also uses windows for specific purposes. For example, one window is used to input system commands; another is used only for system error message display. Further, the Apollo system has not taken advantage of alternate input devices as Apple has with its mouse. With early versions of the Apollo system, many functions required keystrokes that could have been translated into motion, were other input devices used. We expect that this problem is being addressed in current installations.

In summary, different tasks may run in different windows, or as with the Apollo system, windows can serve to logically divide inputs and outputs. Such techniques are aimed at helping the decision-maker to better communicate with the computer and thereby organize his work; initial results to this point have been mixed.

The integration of windowing capability for dialogue interface in the operating system may prove to be a major boon that increases the general usability of various systems. For users of current operating systems, the functional activities are easily understood, yet the syntax and semantics of the interface terminology serve to inhibit and disinterest them. Building window functioning capability into sharable code in ROM for support of basic operating system interface requirements and application management support may, in the long run, surface as one of the most valuable contributions to making computers more accessible to the public.

Multiple-application Window Packages

While window packages that support multiple applications in different windows are often similar to their operating system counterparts, they have two important differences:

1. One of their main goals is to help integrate diverse software packages and provide for data-flow between packages.
2. Most such packages are built to run as shells on top of existing operating systems.

Two of the more popular multiple-application window packages are DesQ by Quarterdeck, and Windows by Micro-
soft. IBM recently announced Topview, a window package designed for the IBM PC.

DesQ and Windows are very similar in that they allow diverse software applications to be run in different windows. To aid task integration, DesQ has elaborate protocols for transferring information between windows. Quarterdeck depends on "local experts" at corporations where DesQ is installed to set up these transfer protocols, a strategy that may discourage quick acceptance of the product.

DesQ is claimed to be able to run a majority of IBM-compatible programs; Windows requires existing programs to be rewritten before they will run. Specifications for the recently released Topview are vague, and will remain so until the product is in distribution.

Many of these and related packages require and exploit RAM-disk capabilities to realize the redirection of input and output, and to facilitate the buffering required for viewport data management. Without such facilities, implementation would be intolerably slow (e.g., piping in MS-DOS). Still, few such implementations have met with much market success.

**Business Shell Environments**

Within the set of business shell environments, vendors integrate well-defined packages rather than diverse applications. Reflecting a user trade-off of flexibility for the power of integration, the applications run in different windows of these shells are well defined; as such, the transfer protocols between windows are easier to understand and use. However, the user is limited to the small set of applications that come with the business shell. Frequently, one or more of such applications are not as good as similar products on the market.

Three of the more popular business shell environments are Framework by Ashton-Tate, Symphony by Lotus Development Corporation, and VisiOn by VisiCorp. These shells provide similar capabilities, such as spreadsheets, word processing, database management systems, graphics, and communication software. Framework also provides an outline function (which makes Framework arguably superior to the other two packages) that allows users to create documents with outlines; entries in the outline, such as a graph or a spreadsheet item, can be from different component packages. The user can alter the document by changing the outline; the underlying parts of the outline are automatically moved.

All three of these business shells are useful, but it is not clear that the user needs all of the power provided by them. User work area size is often a problem, because most packages store all applications in main memory. The idea of close task integration is good, but lessening the number of packages while improving their quality might be a better option. A major problem associated with this approach is that each package has grown around the success of one subportion of the task activities (e.g., spreadsheet or word processing), and the others are only half-heartedly supported. This imbalance may be addressed in subsequent releases of these packages.

The OfficeWare system, available through NCR Corporation, offers a somewhat different approach to business shell design by redistributing the functions that are associated with dialogue and functional-processing between the workstation, or PC, and a host server. For example, in the MIS Department at the University of Arizona, an OfficeWare installation runs with NCR PC4s and IBM PCs as workstations, and an NCR Tower (UNIX) system as a server. The screen management functions that are best handled at the PC level are allocated to the PC environment, while the basic business functional applications are associated with the server function.

In examining the above environments, we notice an interesting dichotomy. Those environments that promise "mainframe-like" windowing capabilities (such as VisiOn) have generally not fared well, probably because the microcomputer is still not powerful enough to satisfactorily mimic the more expensive devices. However, those systems that understand the limitations of the current environment, and concentrate on exploiting the capabilities within these limitations have fared much better (e.g., Macintosh). An obvious comment on the MacIntosh is that the reasons it proves successful in the delivery of effective windowing precluded a degree of access to existing business software growing in the MS/PC-DOS environment.

We further note that the focus of current environments is on the use of windows to support multiple applications, either for operating system functions, or presentation of simultaneous displays. We believe that a neglected opportunity is the support for multiple windows through a single application.

**SUPPORT FOR SINGLE-APPLICATION, MULTIPLE WINDOWING**

It is the intent of most window support environments to provide for windowing across applications. In micro-environments, this usually takes the form of window support in RAM-disk packages that allow each application to maintain a single open window, and provides for piping between applications. While this form seems to be desirable for the integration of applications, it offers many problems with current and near-term technology environments.

Designers, who often fail to appreciate current microcomputer limitations, have concentrated on highly sophisticated window packages while overlooking the opportunity to use multiple windows in a single application setting. Such technological considerations are not the only reasons for advocating the single-application/multiple-windowing approach. With the advent of dialogue protocols that involve currency indicators, switches, clocking, status reporting, and other message transfers between the system and the user, there is a clear need for multiple windowing in order to make the dialogues more effective in terms of the use of a limited presentation space. The result is a system that is more maintainable, flexible, standardized in dialogue protocols (e.g., all system messages on bottom line, etc.), and easier to debug.

If we look back for a moment, we can analyze how the present state of affairs came into being. The traditional form of windowing involved the use of a single viewport for a single application. However, in some environments, one was allowed to have a single viewport with multiple applications operative, as in multi-tasking environments where only one
active process can be viewed at a time. From this situation, designers recognized the opportunity to exploit the viewing space for multiple simultaneous observations. However, the mistaken perception in multi-application multiple-windowing is one of numbers; a typical user will have no more than two or three applications active at any time. Screens get so cluttered that one loses track of ongoing activities when too many viewpoints are open or visible.

The key issue involves the recognition by developers of the relationship between the functional application needs of the user (i.e., a single- or multiple-application task active at any given time), and the technology, which is focused on a single- or multiple-windowing capability.

Figure 1 illustrates the various directions taken in the issue of multiple windows and applications. The neglected opportunity (III) in this classification is the use of the multiple-windowing capability to support the multiple information presentation needs in single applications. In fact, most effective interactive systems already use built-in screen management utilities to present multiple pieces of information during the execution of an application. Wordstar is one such example, with small status and currency windows displayed on the head of the screen followed by menu information which is in turn followed by the editing space that viewports on the editing buffer. In effect, there are several manageable windows that are displayed simultaneously and updated as if multi-tasked.

The use of multiple windows in a single application allows for better and more flexible design of the interface (e.g., windows can be moved independent of other application functions); better separation of presented information is available, reducing the cognitive demand on the user; further, the use of multiple windows offers certain clarity in screen management that may significantly simplify the design process and facilitate earlier implementation.

EXAMPLE USE OF MULTIPLE-WINDOWING IN MASH SYSTEM

An example of the use of window support in a single application environment is the electronic mail support facility called MASH (for Mail Access Supporting Heterogeneity). MASH was developed to allow users on PCs (IBM, NCR, etc.) to interact with multiple mail servers (VAX, Tower, ARPA, etc.) over multiple communications facilities (SYTEK, MODUS, voice-grade lines, etc.). Thus, with a single dialogue protocol, a user of a micro-based station is able to connect to multiple mail systems without having to deal with strange and volatile protocols.

MASH uses windows for screen and display control. Multiple windows in monochrome or color are used to control the user/system dialogue. For example, Figures 2 to 4 illustrate system interaction with the user. Figure 2 shows the initial screen at startup with a user profile window also displayed. In Figure 3, the user (John Smith) has created a mail message to be sent to Jones. Windows indicate header information, text of the message, and commands available at the present time. Figure 4 shows entries in the user's "in-box." Note that the commands have changed, and the header information has been replaced by the in-box directory.

The window management system that underlies the MASH package controls window presentation and placement. The user can change the location or presentation of windows without modifying the functional software of the application. Indeed, a callable window editor and field editor are used to standardize the interface between windows. Windows allow the user to focus on a windowed class of information in a read, as well as read-write organization. Scrolling and other non-destructive editing functions are also available in read-only windows.

MASH is an example of a package that takes advantage of the windowing function in the design of single applications. The application is more easily understood and quickly learned by the user. Modification and adjustment of the system are also facilitated through the use of the window management support that offers an independence for the dialogue similar to that offered for data management through the use of database management systems. The same window and callable editor functionality is available on a wide variety of micros and facilitates a significant degree of interoperability.

Thus with MASH, we demonstrate the ability to establish systems that provide effective window support across processor and operating system environments. Furthermore, the user can have a window picked up, resized, and placed elsewhere without affecting the application logic. In the absence of multi-tasking, as in the MS/PC-DOS environment, we are provided with a perception of multiple-task management through the currency indicators and alerting functions.

We have made other use of windowing in the current implementation of the PLEXSYS system (see Figure 5), and for an inference engine in an application of artificial intelligence in a clinical pharmacy. In these cases, we have found that windows were useful for:

1. organizing the presentation and elicitation of information;
2. offering a flexibility in coding; and
3. providing a framework for the architecture of the software system.
Windows: Current Approaches and Neglected Opportunities

Figure 2—MASH entry screen with user profile window visible

Figure 3—MASH display with header, currency and message windows

Thursday will be a better day for the scheduled walkthrough. Please advise as soon as you can.

Smitty

[EOB]
task_alias_priority................ canonical_form task_alias_form................

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>priority</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>aim</td>
<td>unspecified</td>
</tr>
</tbody>
</table>

**Description:** This form is used to relate tasks to organizational aims. The priority of a task is relative to the degree it realizes an aim.

Creating a new family, enter FAMILY description then (ENTER).

### Sample screens from PLEXSYS system illustrating multiple windows

**Peruse / James**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task priority</td>
<td>Matching criteria: ( (\text{Any Item}) ) Order ( \text{Relative} ) ( \text{Strict} ) A</td>
</tr>
<tr>
<td>Input statement_form_definition</td>
<td>Is used to assign priorities to tasks RELATIVE TO the aims the tasks accomplish.</td>
</tr>
<tr>
<td>Expr. task</td>
<td></td>
</tr>
<tr>
<td>Match aim</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4**—MASH display with a mailbox window visible

**Figure 5**—Sample screens from PLEXSYS system illustrating multiple windows
The third point suggests the specialization of language constructs for windowing as a means of guiding much of the structure of the logic in system specification.

**THE POTENTIAL AND LIMITATIONS OF WINDOWING**

As the technology continues to improve and becomes more readily available, windowing will assume a broader role in the structure of user/system dialogues. We can expect to realize more of this potential, but to what limits? This section reviews issues related to the potential and limitations of windowing.

Windows offer a useful capability that certainly addresses many of the complaints that users have had regarding the nature of their user/system dialogues. At the same time, windows are merely one tool at the disposal of dialogue designers, not the answer to all concerns.

Some benefits of windows are well known; they indicate the presence of multiple-activity tasks; they reflect the preservation of activities, even when sequential-tasking takes place; furthermore, they serve to separate items that should be separated. It is no minor benefit that this separation offers a more aesthetically pleasing interface.

There are many other benefits to windowing, including the opportunity to gain multiple views of the same data in different perspectives (i.e., a graphic and tabular presentation). In addition, the “tidiness factor” encourages cognitive tidiness and screen management “cleanliness.” Piping from one window to another allows the direct perception of multiple-task management, and addresses the often-cited user concern for application interconnection. Some other benefits of windowing include, but are not limited to the following.

1. Improved perception of multi-tasking;
2. Separation in space and identity (border, color, etc.), thereby easing cognitive demand;
3. Ability to make direct comparisons;
4. Displays of currency indicators;
5. Displays of alerting or exception information;
6. Isolation and separation of functional displays (commands, messages, etc.); and

The limitations to current windowing approaches are partly due to increased user expectations and the over-sell of the ability to mimic expensive interfaces (e.g., the case of the Xerox Star) on inexpensive devices (e.g., slow processors like the 8088; limited display capabilities such as terminal I/O; and keyboard input). Thus, the market is over-sold on the capability and disappointed in its delivery. This dichotomy could have been avoided had proper care been taken in the introduction of the concept and capability.

Another major problem in the use of windowing is the nature of the display and I/O. Where bit-mapped graphics and direct memory addressing is available, the windowing operation is fast and efficient, but attempts to use windows in environments where BIOS operations and other software layers are used creates excessive overhead. Thus, in a majority of MS-DOS environments where such layers are involved, and the screen management is character-oriented, system performance as a multi-application, multi-windowing tool is dismal. Other limitations of currently installed windowing functions are as follows.

1. Interfaces are designed for lowest-common-input mechanisms, usually limited to the keyboard. Use of the mouse is just emerging in inexpensive environments;
2. Windowing is usually only offered in RAM-disk environments;
3. Window packages frequently use standard I/O facilities with significant overhead;
4. Current packages do not provide the hooks for single-application use of multiple windows;
5. Screen resolution is often limited, reducing potential window functionality;
6. Limited character graphics are often used for windowing support; and
7. Little thought has been given towards helping design window interfaces in specific application domains.

These and other problems need to be resolved before windowing benefits will accrue to the user community. We have seen the promise in the early workstation implementations, yet the inability to translate beneficial features in these environments to the small, inexpensive desk-top environment has resulted in a pervasive disappointment with the ability of windowing to make a difference in the functional utility of systems in the marketplace. The skeptical attitude on the part of potential users should lead developers to recognize that benefits of a windowing environment can be realized when certain minimal standards of functionality in the workstation environment are met.

**CONCLUSION**

Windowing offers significant potential for information organization and presentation control. Its functional capabilities suggest the means of integrating and defining applications in the context of window dialogues. We can expect that interest in windows will grow as support technologies become less expensive, and that window support will be a commonly exploited technology in operating systems, applications, and utilities.

This paper has reviewed several of the current windowing directions and briefly examined efforts in the commercial arena. We have also tried to bring to light the neglected opportunity presented by multi-windowing in a single application environment. Clearly, the potential benefits of windowing in user/system dialogue environments are many. The race by application designers to take advantage of emerging technology, and the costs of technology delivery will result in efficient, effective windowing environments in future information systems.

Windowing is one of many seldom-used technologies that dialogue designers can bring to play to promote effective dialogues that enhance the utility of information systems. Its
promise is not that of a guarantor of more useful and effective systems, but surely that of a major player in the process of making information systems more useful and effective.

REFERENCES
1. Miller, G. “The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information.” *Psychological Review*, 1956.