User interface services in AD/Cycle

by J. M. Artim
J. M. Hary
F. J. Spickhoff

Significant progress has been made in the effort to separate programmers from the management of data storage. By comparison, the window of a workstation is still managed and controlled in great detail by the typical programmer. In AD/Cycle™ user interface services define a set of services that assist in the management of the displays on the workstation. These services also help increase the productivity of the tool builder by enforcing Common User Access rules and guidelines, and raise the level of consistency of user displays of the tools in AD/Cycle.

Increased display usability and consistency for tools and support functions is achieved with extensions to the SAA Common User Access (CUA)[1-3] rules and guidelines by AD/Cycle™,[4] a Systems Application Architecture™ (SAA™) application. Over time, AD/Cycle's user interface will evolve to the CUA workplace environment[2] as the preferred style for user interaction on the workstation. In this paper, we discuss the evolving AD/Cycle workplace and a set of services, called *user interface services*, that define the implementation.

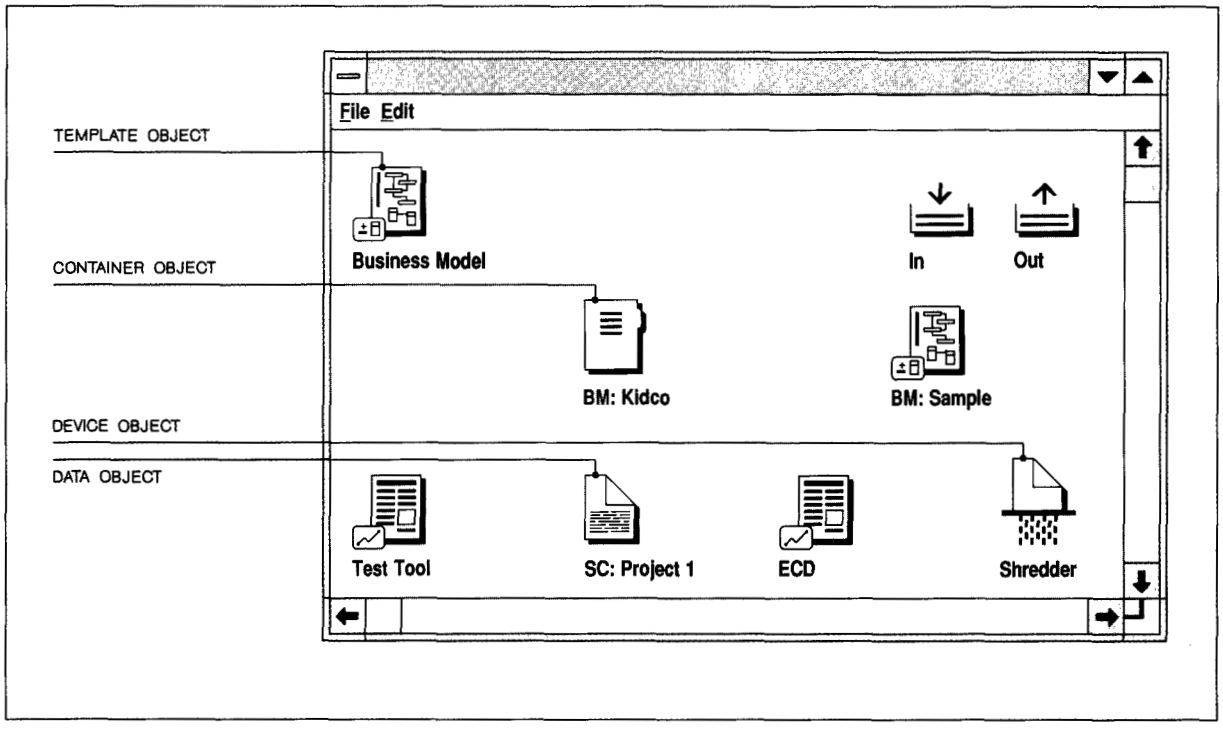
Objectives of the user interface services

The main objective of user interface services is to implement a workplace environment for AD/Cycle. This objective would be rather difficult to achieve if each individual tool *enabled*[5] in AD/Cycle managed its user interaction on the workstation without architectural control. In the absence of AD/Cycle's user interface services, AD/Cycle tools would independently use the standard set of user interface components as defined by CUA and implemented in OS/2® Extended Edition. Since these components are generic for current and future SAA applications, only a corresponding general degree of user interface consistency would be achieved. Also, if individual tools were to interpret and to implement the rules and guidelines for CUA's existing workplace model, the resulting workplace would more likely be segmented along tool boundaries that might not match the task boundaries of the application developer.

A secondary objective of the user interface services is to increase the separation of the management of user displays from the tool logic. AD/Cycle tools can then focus primarily on data transformations and data computations, decreasing the size of each tool and simplifying its structure. This should increase the productivity of the AD/Cycle tool builders.

User interface services will accelerate the realization of the AD/Cycle workplace by extending and refining the generic description of Common User Access definition of a workplace into programming services for AD/Cycle tools. The existence of this set of

Figure 1 An example of the main AD/Cycle workplace window



programming services will also accelerate the integration of the AD/Cycle workplace with other SAA applications, for example IBM's OfficeVision/2.[6,7] It is easier to change and to integrate a controlled set of services, such as user interface services, than to modify an open-ended set of workstation tools.

Figure 1 illustrates a model of the AD/Cycle workplace where some office applications and some AD/Cycle applications are integrated. For example, in Figure 1, office applications are the in and out basket; AD/Cycle applications are the business modeling applications (labeled BM:Kidco and BM:Sample) and the source code application (labeled SC:Project 1).

The implementation of user interface services will evolve over time. The specific services described are representative examples and do not necessarily describe the actual packaging and implementation in AD/Cycle. The AD/Cycle workplace will be complemented by enhancements that will increase the user interface consistency and provide incremental levels of tool integration on the workstation. Three functional enhancements have been identified to facilitate the evolution to the AD/Cycle workplace and are described later in this paper: a standardized window model, a generalized list handler, and a set of rules for AD/Cycle tools that provide a common "look and feel" for the user of AD/Cycle.

Positioning the user interface services

User interface services cannot be defined in isolation. Its relationship to the architected components of AD/Cycle and to the basic facilities of SAA bounds its content and its structure.[1] The user interface services are a workstation resident facility of the AD platform that can only be accessed from tools or distributed tool segments that reside on the workstation. They cannot be accessed directly from a host environment. Figure 2 shows that user interface services are positioned within the workstation services component of the AD/Cycle architecture. They are a set of services that support the implementation of the AD/Cycle user interface.

Figure 2 Position of user interface services in the AD/Cycle architecture

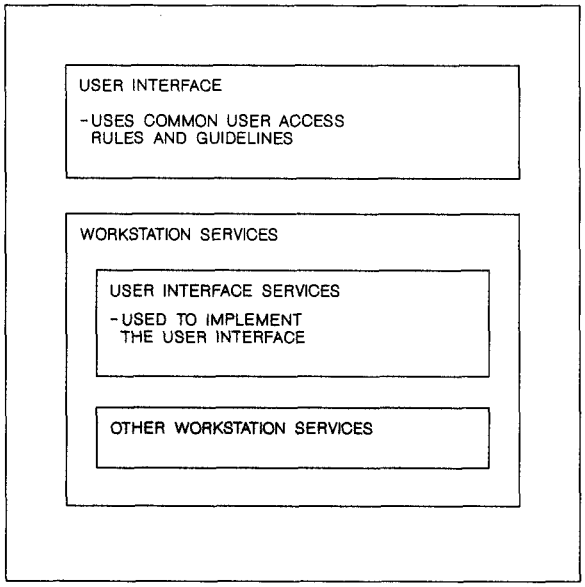
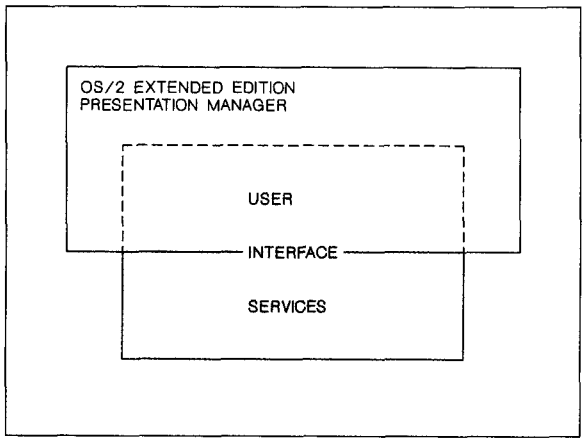


Figure 3 Relationship between user interface services and OS/2 Extended Edition



From the perspective of the designer of CUA, user interface services refine the physical, syntactic, and the semantic consistency rules and guidelines of CUA[3] into a specific set of available programming services that will form the AD/Cycle workplace. CUA is enabled on the workstation through the services of the Presentation Manager™ in OS/2 Extended Edition. Refinements of the kind defined by user interface services have been anticipated by the developers of CUA and they are important for the evolution of CUA and OS/2 Extended Edition.[8]

Figure 3 shows the relationship of the user interface services to the OS/2 Extended Edition Presentation Manager. The size and content of the part of the user interface services that extends into OS/2 Extended Edition is dependent on a wide acceptance of the functions provided by the user interface services. This means if user interface services are common and useful beyond the domain of application development, those services become candidates for extensions to the Presentation Manager services of OS/2 Extended Edition. A side effect of such an implementation would be that such services would not remain in the infrastructure of AD/Cycle but would become part of the more general SAA facilities.

For example, the object services and direct manipulation described in this paper are likely candidates for inclusion into OS/2 Extended Edition. On the other hand, the graphical network services appear unique for viewing and navigating the information models defined in AD/Cycle.[9] The remaining services fall between these two extremes.

OfficeVision/2 supplies a set of services that are useful in AD/Cycle as well. Examples of such services are the printer, the shredder, and the dragging of objects across windows. These will not be reinvented in user interface services but will be used within the generic framework provided by the workplace environment of CUA.

From the perspective of the AD/Cycle tool builder, there are two ways in which the user interface services can be used: (1) Tools (primarily new tools) can use the user interface services exclusively. (2) Tools can use both the user interface services and Presentation Manager. This is particularly the case for modal dialogs, since tools may need to implement such dialogs, through OS/2 Extended Edition services. It is expected that most tool builders will use this method. Tools that exist prior to the availability of user interface services that will not be converted will not use the user interface services at all. Careful design reviews need to minimize the need for tools that do not use the services exclusively.

The AD/Cycle workplace

In this section we discuss the user interface as defined by aspects of the AD/Cycle workplace that are ex-

posed (appear on the screen) to the user. The general principles behind the user's conceptual model are discussed. The look and feel of the workplace are outlined, and finally, the advantages of the workplace for AD/Cycle are presented.

The AD/Cycle user approaches a development task with some model of interaction in mind. This mental model must include a plan for executing the task at hand as well as a model of the system that supports the task. A useful computer interface will provide a strong, reliable set of visual and behavioral cues guiding the user to an effective mental model of the user interface. The interface also must provide visual information indicating the current state of the interface, minimizing the need for the user to remember these details. The purpose of the rules and guidelines of CUA is to present a common interaction paradigm guiding the user within this interface domain. Each new application provides new mechanisms to address specific requirements though in all cases the style of interaction remains consistent.

The user model inherent in CUA's description of the workplace environment[2] incorporates the concept of a customized user workspace organized around user-relevant objects. For the purposes of this paper, an *object* is the fundamental unit of manipulation on the screen. It supports the user's task at the user interface, allowing actions such as OPEN, MOVE, or COPY against an object. When the object is open, the object handler allows the user to carry out functions on the object. The objects are organized on the display screen by windows that can be sized and positioned in accordance with the user's task approach. This tailorable approach allows the user full control over the screen presentation space (see Figure 1).

Much of the power of the workplace environment derives from its object nature. The user's focus is on data objects that have unique actions associated with them, rather than on selecting an AD/Cycle tool and specifying the data to be processed by the tool. In a tool-oriented environment, the user must be concerned not only about which collection of data objects is relevant to the user task but also which tools are required to manipulate each object. In the object-oriented workplace, the environment integrates the tools that are relevant to a given object. At any point in the interaction, the user specifies an object or collection of objects, and the interface provides the user with information about the type of actions that can be invoked against that object. These two features—a user-customized presentation space and a focus on direct manipulation of objects—form an interaction style based on the user's approach to problem solving rather than an arbitrary interaction style that is predetermined by a set of tools.

In the AD/Cycle workplace the user perceives each window as a container holding one or more objects. By selecting one or more of these objects, the user limits the context to be explored. The actions that can be invoked against the objects can then be indicated both by selective bolding and graying of action bar pull-down items and by the contents of context pop-ups (later described in detail as context menus). In this fashion the user builds up a mental model of the object-viewing capability of the environment. Because the exploration is focused on task-relevant objects, it is easier for the user to map the interface's capabilities to the task domain. At the same time, the objects the user is manipulating and the actions being performed against those objects are all user-relevant. The AD/Cycle workplace supports the cognitive structure the expert user has developed prior to the use of AD/Cycle[2] by presenting familiar objects and actions, allowing the user to arrange these objects freely, and giving the user the freedom to dictate action sequences.

Object handlers provide a view of an object and the means to manipulate that object.

One of the important concepts outlined in the CUA description of the workplace environment is that object handlers provide a view of an object and the means to manipulate that object. There may be many object handlers available to view any given object.[2]

In the absence of user interface services, individual AD/Cycle tools must manage the user interface directly. In this case the scope of the user task is determined by the tool and not by the task the user wishes to accomplish. For example, the incorpora-

tion of a structure diagram into a design document might involve the use of several tools. One tool might be used to enter the information associated with the structure diagram. Another tool might be needed to convert the structure diagram into a standard image format. A text editor would be used to compose the document. A page layout tool might be needed to view the combination of text and figure. In this case, the user has to manage the context switch between these tools. By contrast, in the AD/Cycle workplace with user interface services, a document would be *opened* by the user and the appropriate object handlers would be transparently invoked against that object. If a structure diagram is included as a figure in a design document and the document is opened, then the text would be viewed using a text editor and the diagram would be viewed using a graphic editor. The user would be unaware that two different editor tools were used, allowing the user to maintain attention on the development task.

In AD/Cycle, there may be many ways of using a specific screen object. For example, the source code might be manipulated by either a code-generating tool, or by a test tool. In every case the user's approach is the same: select the source code of interest and invoke one of the actions against it. A window containing the source code, would be opened and the action invoked against an object would determine the presentation of that object. A diagramming tool might support the action of presenting a structure chart of the source code, while a test tool might support the action of animating the execution of the code.

To re-enforce the object style, iconic representation is preferred throughout the interface, although text presentation will be available. Icons support direct manipulation by providing an easy cursor target. Icons depict basic classes of objects: data objects, templates to generate new data object instances, containers used for grouping, and various devices like printers, in and out baskets, and shredders. Although these basic class icons can be enhanced with specific details, the classes must be easily distinguished from each other. Through careful selection of classes and subclasses, icons can be used as a visual mnemonic of the basic behavior of any given screen object.

The AD/Cycle workplace accommodates complex objects that can be represented in multiple levels of abstraction. For example, a technology model may contain multiple submodels. Each submodel contains its own subparts. As the user traverses this refinement, a visual context is established that matches the user's need for more specific information about the development task. In this way the AD/Cycle workplace enables the user to maintain intellectual control over larger and more complex development domains than in traditional and more fragmented development environments. Increased intellectual control can lead to higher productivity and improved software quality.

User interface services

The following are examples of services needed to realize the workplace model in AD/Cycle. The AD/Cycle tool builder will use these services to create a user interface handler for a tool. These services include:

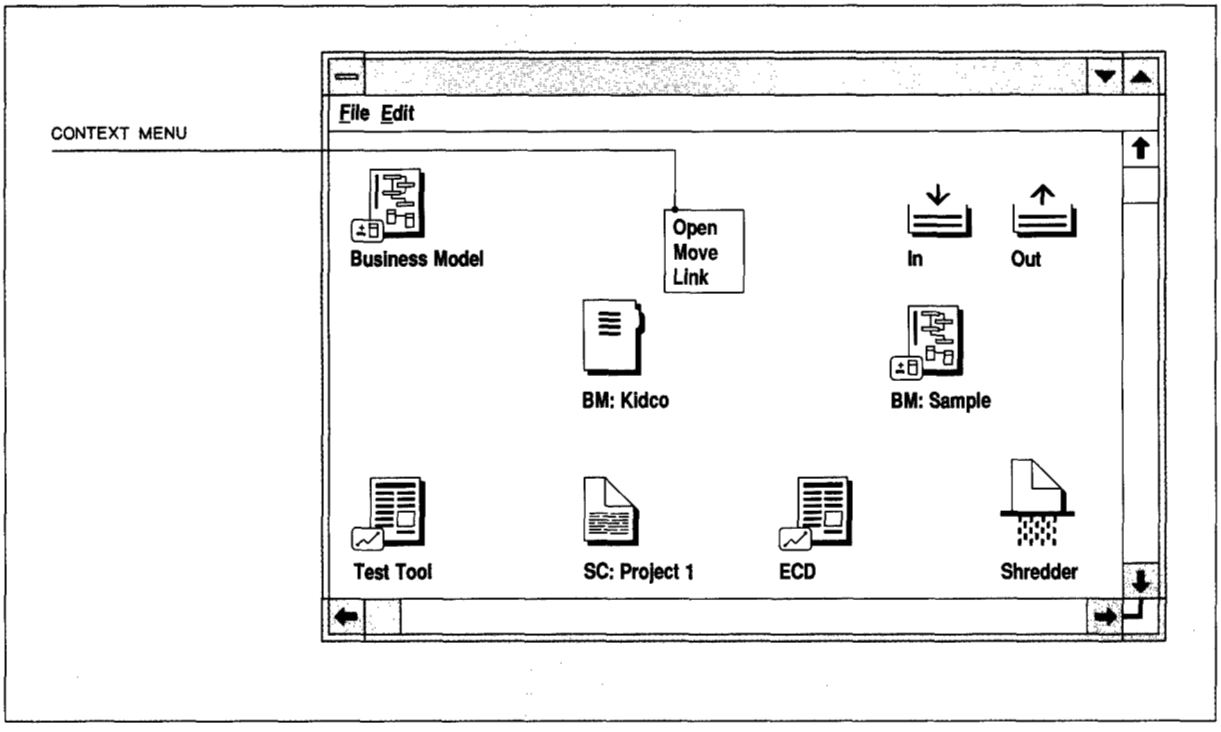
- A set of standard user interface controls
- A programming interface for object handling and direct manipulation
- Handlers to support displaying lists of objects

In addition to describing these services, we also use them to demonstrate how they refine and constrain the workplace environment as described by CUA rules and guidelines.

A major component of user interface services is a set of controls that support direct manipulation. Examples of controls include context menus, palettes, and network navigators. These controls provide the AD/Cycle tool builder with a toolkit of parts to assemble an interface. In the following discussion, we describe a set of user interface controls that are required and the user interface problems that they solve.

Context menus. The current workplace environment requires action bars and pull-down menu controls. These controls are efficient for dealing with simple, independent window objects; however, the action bar and the pull-down menus rapidly become cumbersome when an application developer must deal with several objects. For example, a developer might be working with objects that include a description of a program, the program's representation in a structure diagram, and a list of users who are dependent on this program. For these typical application development tasks, the developer will make frequent and widely-spaced mouse movements between the action bar and the object of interest. These mouse move-

Figure 4 Context menu



ments can confuse the user about the association between the pull-down menu and the selected window object. Since the AD/Cycle workplace will contain many objects, a more compact packaging of window objects with their menus is needed.

Solution. Context menus (see Figure 4) provide a compact grouping of an object and its actions by allowing the user to display actions in a menu next to the selected object to which the actions apply. The context menu is activated by placing the mouse pointer on an object and clicking a mouse button. If the mouse pointer is over an object, such as a file folder, clicking the mouse button displays a context menu appropriate to the folder object (for example, OPEN, MOVE). If the mouse is over the background of a window, the window itself is the selected object, and a list of general actions (for example, CUT, COPY, PASTE functions) is shown in the context menu.

Tear-off menus. AD/Cycle tools, such as debuggers, contain actions that must be executed frequently. These actions, such as STEP and RUN, become cumbersome to use if a user must select the action as a pull-down choice from the action bar. Dialog boxes and secondary windows are inappropriate to support action selection, since their purpose is to support extended user dialogs, not menu function. In addition, context menus are inappropriate for packaging these actions, since context menus disappear after a single action has been selected.

Solution. Tear-off menus (see Figure 5) provide immediate access to frequently used actions. These menus contain action choices in the form of text strings or push buttons. The user can select the desired actions from these windows without referring to the action bar pull-down. For tools that have frequently used global actions, such as debuggers, tear-off menus provide a useful extension to the action bar method of selecting actions.

Network creation and maintenance. In many information modeling tools, users construct and manipulate the content of information models by constructing graphical networks. In these diagrams, data are represented as nodes or graphic boxes and relationships among data are represented as graphic lines

Figure 5 Tear-off menu

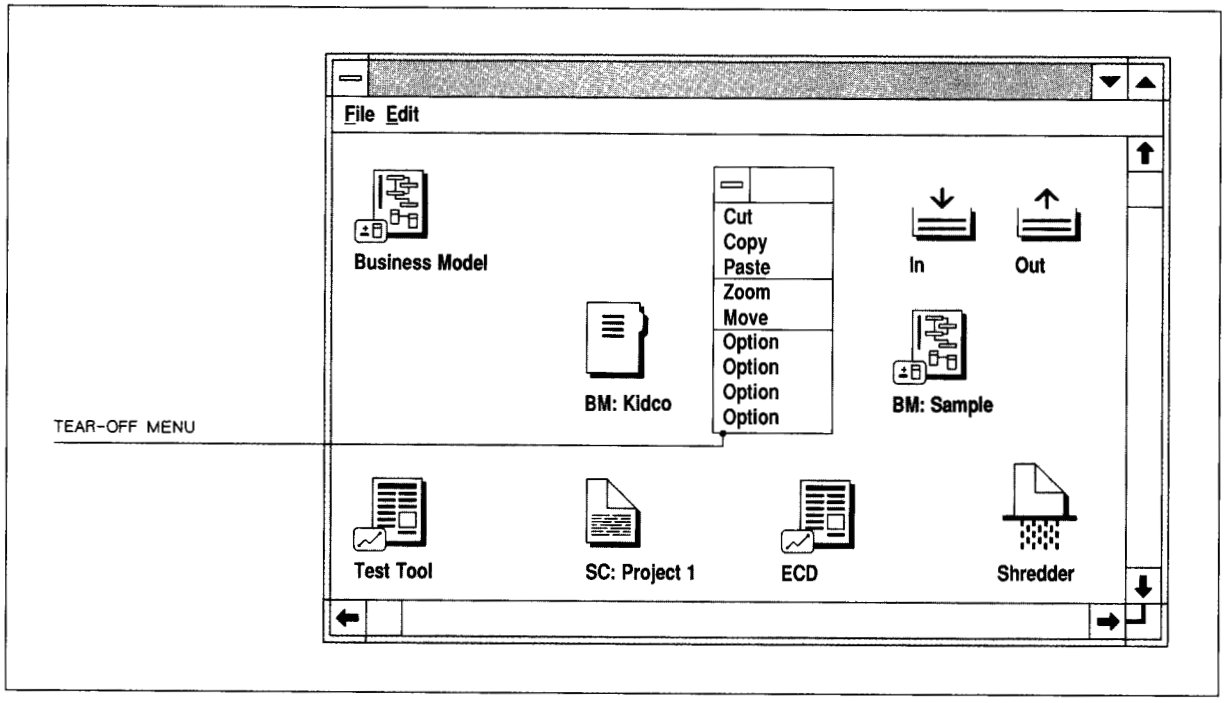
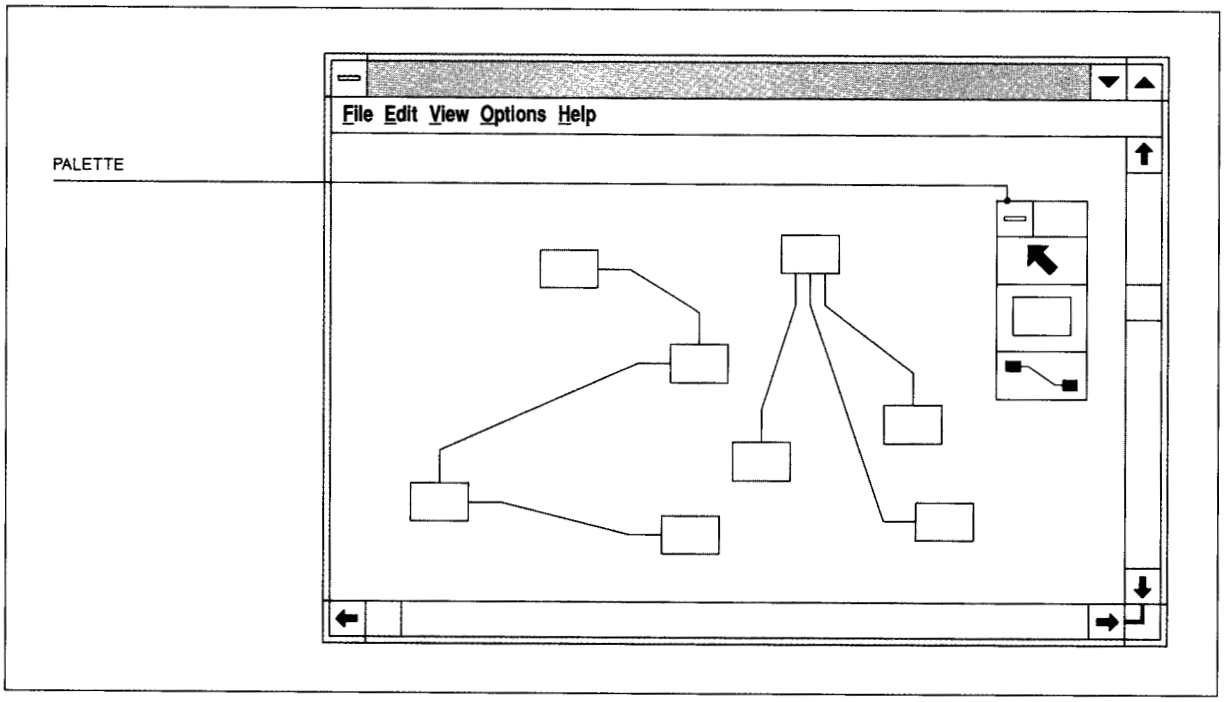


Figure 6 Palette



(see Figure 6). For example, in an entity-relationship diagram, entities appear as labeled boxes and relationships appear as lines drawn among the entity boxes. In these tools, the user constructs the network piece by piece by first creating graphic nodes and then drawing links among the nodes to define the relationships.

No description exists in CUA's workplace environment to create nodes and links in graphical networks or select construction tools that operate in the area of an application, such as pens in a drawing program.

Construction tools are different from actions. These tools provide the user with a method to perform multiple actions without constant selection of actions from a menu. However, in the current workplace environment, users must select construction tools, such as CREATE NODE, from the action bar, resulting in unnecessary movement of the mouse.

Solution. Palettes provide an efficient way for the user to build graphs by providing immediate access to network construction tools (see Figure 6). The palette is a movable, persistent child window, that presents a menu of iconic construction tools that can be used to customize the behavior of the mouse pointer. When the user selects a tool icon, the mouse pointer changes to indicate that the user has entered a mode for that tool. For example, if the user selects a CREATE NODE tool in a palette, the mouse pointer is transformed to the shape of the tool icon to indicate that the user is currently in a node to create nodes. The palette contains a system pointer icon to allow the user to exit a mode.

Palettes ease the user's task of constructing the network. Palettes allow the user to select construction tools to build their network without repetitive menu access. In addition, the iconic tools in a palette allow the user to perform repetitive actions without continuously selecting the action from a menu. As the AD/Cycle workplace evolves, palettes will prove useful for supporting direct manipulation methods of constructing information models.

Graphical network navigation and manipulation. Network representations of information models tend to be large. Navigators allow the developer to explore these large networks, but, in many cases, these networks are far bigger than the area of the window within which they are displayed. The current CUA workplace environment only defines scrolling with scroll bar controls for manipulating and navigating these networks when they exceed the boundaries of the window. Scrolling with scroll bars provides fine grain control in viewing or paging a network display; however, these techniques need to be supplemented with navigation methods that allow the user to see the current position with respect to the network as a whole.

Solution. The global overview window alleviates some of these problems (see Figure 7). The global overview window is a secondary window that provides a complete (zoomed-out) view of a graphical network. A selection rectangle in the global overview window indicates the segment of the network that is displayed in the parent window. The user can move this selection rectangle with a mouse. When the selection rectangle is moved, an updated network segment is displayed in the parent window. Conversely, if the parent window is scrolled, the global overview window is updated. This global overview window provides visual cues on the user's position in the overall network.

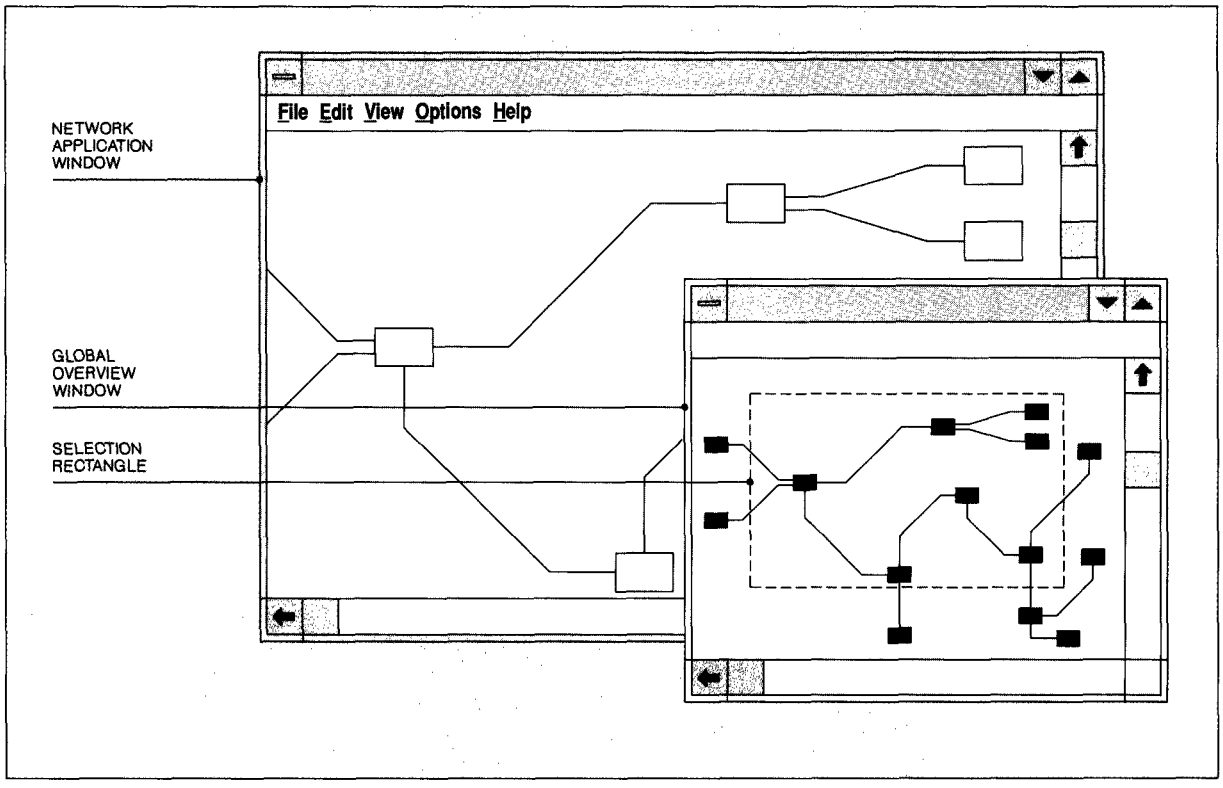
In many cases, the user may desire to zoom in on a particular graph segment. The global overview window provides a powerful method for zooming. The user can draw a new selection rectangle in this window with the mouse (see Figure 8). The area selected within the global overview window would then be zoomed and displayed in the parent area of the main window. The image in the parent window would be scaled to use the entire area of the window. Text and graphic attributes on nodes are displayed when the parent area is magnified enough by zooming to allow their display. Because of the importance of information models in AD/Cycle, zooming controls and global maps will be an important feature of the AD/Cycle workplace.

Additional controls. The AD/Cycle workplace requires the definition and registration of additional user interface controls, such as palettes and context menus. Each control requires the addition of a set of messages to user interface services that define the control's behavior. In addition, a programming interface is required to:

- Create specific controls
- Register controls
- Delete controls

Presentation Manager provides a generic programming interface to register and delete user-defined controls. However, the programming interface nec-

Figure 7 Network navigator—illustrates the function of a global overview window



essary to create specific controls can be standardized for the AD/Cycle environment. For example, a function call can create a palette control window and install icons within the palette.

Object level control. In addition to user interface controls, user interface services of AD/Cycle define a set of general resources to support object services for the display and manipulation of tool data, list presentation, and direct manipulation functions such as DRAG and DROP.

Object services. Object services provide generic functions to allow AD/Cycle tools to use the display controls and direct manipulation functions of the AD/Cycle workplace. These object services affect how tools are presented to the user. Object services affect only the user interface and they do not provide system services, such as sorting lists or copying data between file systems or repositories.

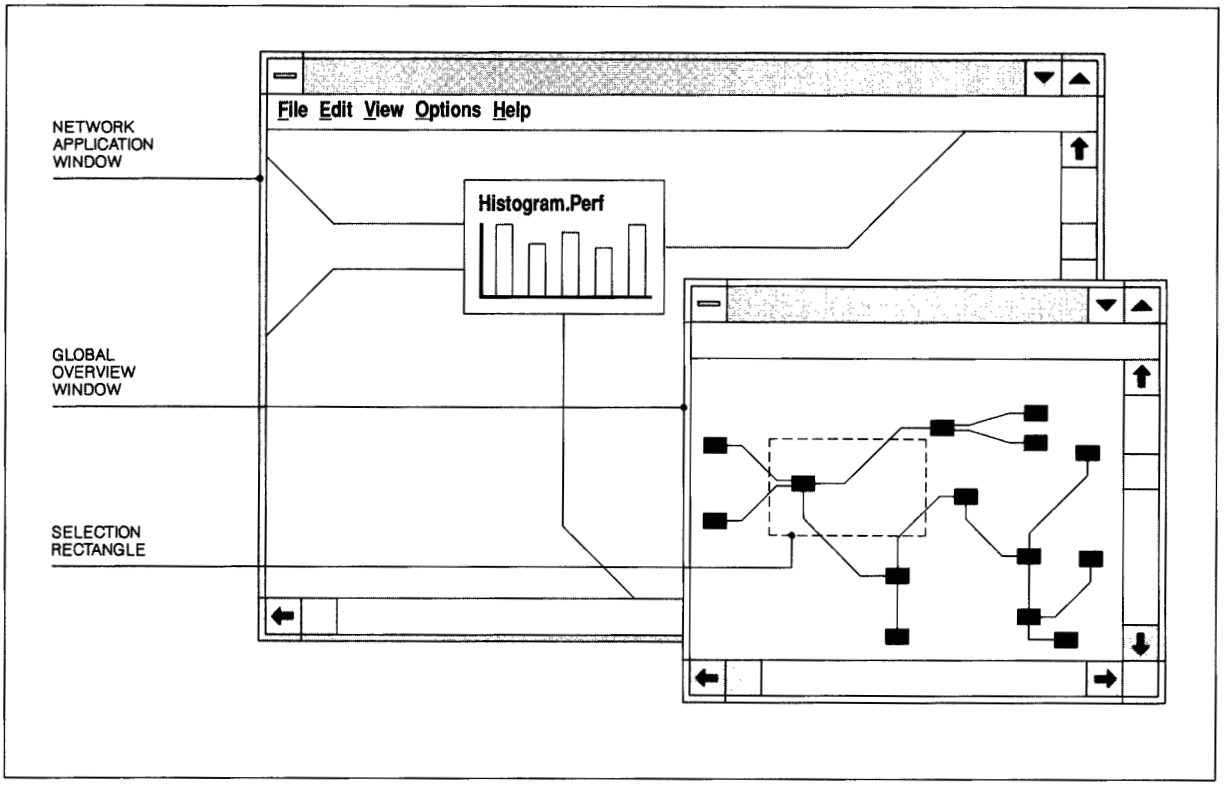
Object services include functions that:

- Support the visual presentation of an AD/Cycle tool as an object icon in the workplace
- Define the properties and behavior of workplace object views
- Define default views of workplace objects
- Support direct manipulation

Object services allow an AD/Cycle tool to: display an object icon to the user and allow the user to open default views on objects represented by icons, allow the user to change the properties of an object, such as its color and font, and send messages to other system services when a user interacts with an object icon or object view using a mouse or other input device. They also provide default views, or object windows, allowing the developer to easily produce CUA-conforming tools.

List services. List services provide functions that operate on collections of objects. These objects could be icons or text strings. List services provide a means of organizing objects within a workplace environment. Some list services include:

Figure 8 Network navigator—illustrates the effect of zooming in on one graphic element



- Displaying objects in a list
- Formatting lists
- Supporting of list element selection
- Filtering lists

The list services operate at the user interface level only. For example, a system service may query the names of the objects contained in the Repository Manager™ data store. The list services would be responsible for displaying the names of these objects in an organized fashion to the developer. Another example of a list service is a view of a container object such as that shown in Figure 1, where the presentation of a window can contain a collection of icons.

Selection services allow a user to select one or more elements within a list. If a developer clicks on a list element, the list services are responsible for sending a selection message to the list element.

Direct manipulation services. Certain direct manipulation services apply to the AD/Cycle workplace as a whole. These services include support for:

- Dragging objects among windows
- Dropping objects on other objects
- Pointer management

Dragging services are used when the user wishes to move objects across windows within the AD/Cycle workplace. The dragging services include a programming interface to record the initiation of a drag operation and drag messages that are sent while a user is dragging an object. Dragging services are also responsible for checking to see if an object is movable before initiating a drag operation. If the object is not movable, dragging services will display visual feedback to the user to indicate that the object cannot be moved by the mouse. Dropping services receive messages that indicate that one object has been dropped upon another.

Pointer management provides a programming interface to change the pointer image from the system pointer icon into the image of the icon to be dragged.

These services are also responsible for restoring the system pointer icon after the drag operation is terminated by a drop operation.

The effective use of AD/Cycle requires the definition of controls, which have been discussed, to refine and constrain the CUA workplace environment providing an operating environment for the workstation user. These controls, and the other services defined above, will facilitate the enabling of tools in the AD/Cycle workplace.

Interface interactions will conform to the object-action paradigm.

Evolution to the AD/Cycle workplace

We now describe some techniques that will allow existing AD/Cycle tools to evolve toward the AD/Cycle workplace. AD/Cycle tools that follow these techniques will achieve a desired degree of consistency and integration.

Consistent visual structure. Graphical interfaces present a multitude of visual cues ranging from type font choice to the use of color for various window components. These cues can be disorienting if they are arbitrarily applied. For example, if every AD/Cycle tool arbitrarily set the background color of a window rather than using the OS/2 default color, the user would likely become confused and frustrated by such a random cue of a context switch. In order to draw correct conclusions about the state of an application, AD/Cycle tools will start with a monochrome look as a base and add specific colors only if they convey additional information for the user of the tool. In addition, the system font should be employed for all text display, unless there is compelling need to do otherwise.

Consistent interaction style. A consistent interaction style, used to manipulate the graphical interface, is provided to the application developer for the operation of tools within AD/Cycle. Existing AD/Cycle tools can provide this consistent style by following the CUA design guidelines.[2] Action bars conform to the CUA standards and, as a rule, interface interactions will conform to the object-action paradigm where the object selection sets the context for the actions that are appropriate.

Reversibility. User actions should not cause accidental and irreversible side effects. Either an "undo" operation will be available or warnings will be issued to alert the application developer to side effects that are not necessarily obvious at the interface. For example, the shredding of an object may have hidden side effects on the Repository Manager level.

The window model. One basic feature of the AD/Cycle workplace is the window model. User interface services will support this window model as a key element for early tool integration and interface consistency based on the rules and guidelines describing the CUA workplace model.

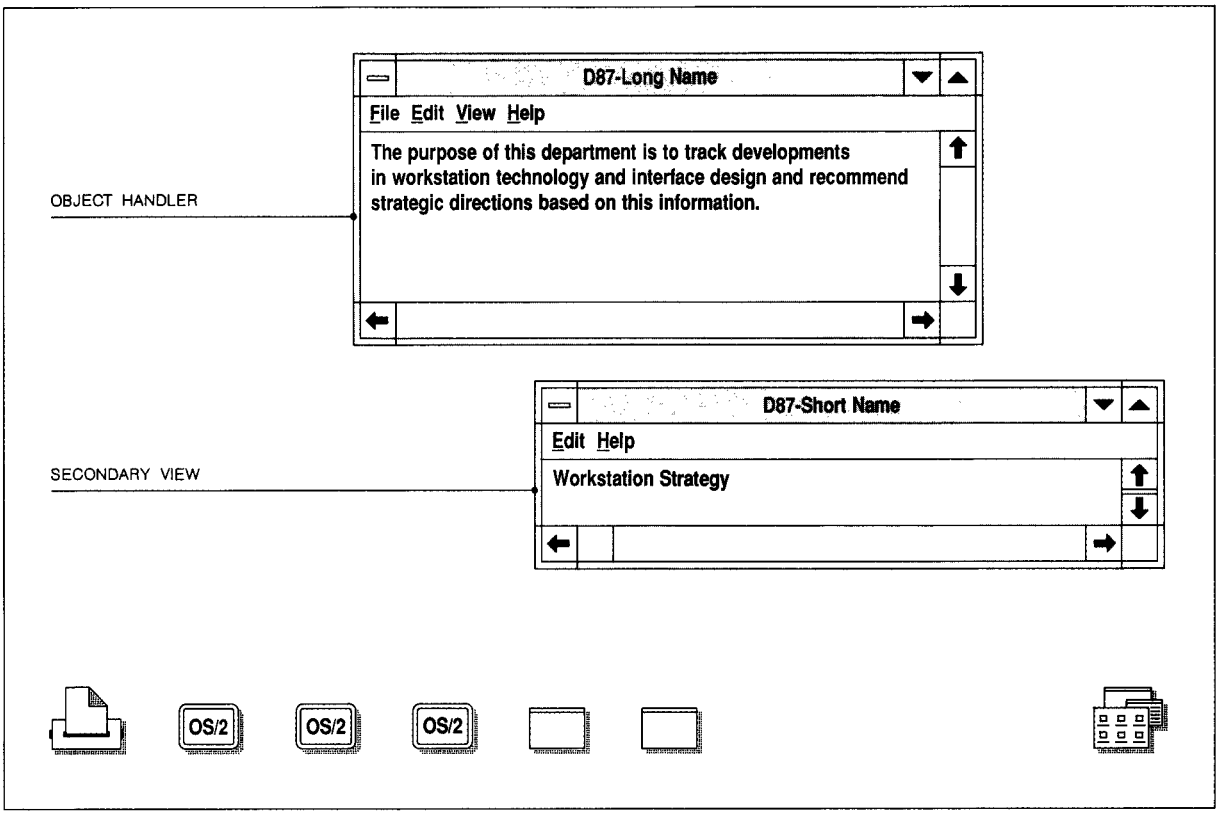
In AD/Cycle, a user may switch among multiple tasks or task threads, and thus integrated ways of grouping and switching windows according to these tasks would be essential. Windows in the AD/Cycle workplace are grouped into one of the following categories:

- A container window showing a view of a collection of objects
- An object handler window showing a view of one object
- A secondary window showing an alternate view of an object handler or container
- A dialog box

Container windows conform to CUA's definition of a workplace window. A container window contains a view of a collection of objects. That view can be either text based or a graphical display composed of individually selectable icons. Container, or workplace, windows can be nested to any degree. The most common presentation style will be the list handler.

An object handler window contains an object view that uses its present area to display graphics, text, data, lists, and other objects. Object handler window design is described by CUA's workplace environment.[2]

Figure 9 An example of an object handler window and an associated secondary view



Secondary windows are available on either container or object handler windows (see Figure 9). They allow the user to view the same information in more than one form. An example of a standard type of secondary view to be supported by AD/Cycle is a view of all objects currently in use. This is the equivalent of a list of all windows opened from the current window.

Dialog boxes are used to elicit information from a user. They can be either modal or modeless. A modeless dialog allows the user to switch between the dialog box and other windows before completing the interaction with the dialog box. Modeless dialogs allow the user a greater measure of control and should be used in preference to modal dialogs wherever possible.

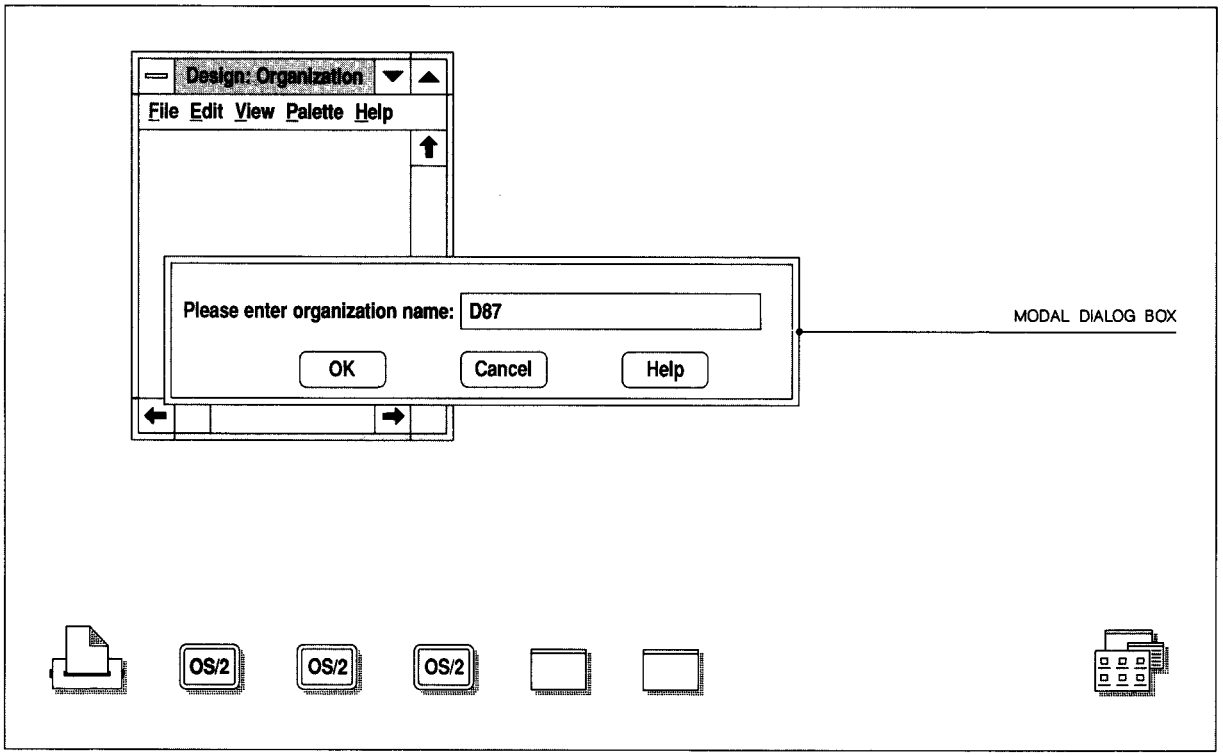
Modal dialog boxes (see Figure 10) are used to elicit responses from a user where the responses are required before the user-initiated action can be completed. Their purpose is to either collect additional information needed to complete a task the user has initiated or they are used to give the user the choice of proceeding with or canceling a task if that task results in change that cannot be undone.

Summary

User interface services play a dual role in AD/Cycle:

1. The services simplify the structure and content of the tools that are enabled on the workstation.
2. The services facilitate building the AD/Cycle workplace. They allow many tools to interact with an application developer, hiding disruptive context switching between tools, managing the control flow between tools on behalf of the application developer, and providing a consistent framework and style for the interaction with AD/Cycle.

Figure 10 An example of a modal dialog box



We have described how user interface services extends the rules and guidelines of the CUA workplace environment. The examples for some of the key services provide a basis to assess the impact user interface services will have on the productivity of tool development in AD/Cycle and the degree of consistency achievable in AD/Cycle.

Acknowledgments

User interface services summarizes ideas of a team of people. In particular the following members of the AD/Cycle architecture department in the IBM Santa Teresa Laboratory contributed to this paper: B. Costain, D. Ecimovic, N. Eisenberg, and B. Meyers. Members of the human factors department, also in the Santa Teresa Laboratory, helped in the definition of the specific pieces of the user interface services. They are K. Bury and G. Moore. Our thanks go to members of the CUA team in Cary, D. Roberts and R. Smith, and to D. Collins from Thornwood who contributed to this paper from its initial conception. We also wish to acknowledge the helpful comments by the referees of this paper.

Cited references and note

1. *Systems Application Architecture, Common User Access—Basic Interface Design Guide*, SC26-4583, IBM Corporation (December 1989); available through IBM branch offices.
2. *Systems Application Architecture, Common User Access—Advanced Interface Design Guide*, SC26-4582, IBM Corporation (June 1989); available through IBM branch offices.
3. R. E. Berry, "Common User Access—A Consistent and Usable Human–Computer Interface for the SAA Environments," *IBM Systems Journal* **27**, No. 3, 281–300 (1988).
4. V. J. Mercurio, B. F. Meyers, A. M. Nisbet, and G. Radin, "AD/Cycle Strategy and Architecture," *IBM Systems Journal* **29**, No. 2, 170–188 (1990, this issue).
5. In this paper, the term *enabled* refers to a tool or application that is ready for execution, being requested directly by a user or indirectly by the controlling system at a determined point in the development process.
6. *OfficeVision/2, Using OS/2 Office*, SH21-0421, IBM Corporation; available through IBM branch offices.
7. *OfficeVision/2, Managing OS/2 Office*, SH21-0422, IBM Corporation; available through IBM branch offices.
8. S. Uhlir, "Enabling the User Interface," *IBM Systems Journal* **27**, No. 3, 306–314 (1988).

9. R. W. Matthews and W. C. McGee, "Data Modeling for Software Development," *IBM Systems Journal* **29,** No. 2, 228–235 (1990, this issue).

John Artim *IBM Programming Systems, Santa Teresa Laboratory, P.O. Box 49023, San Jose, California 95161-9023.* Mr. Artim is currently a staff human factors scientist in the Application Development User Interface department at IBM's Santa Teresa Laboratory in California. Mr. Artim received an M.A. degree in experimental psychology from the University of California, Santa Cruz, in 1988. His work is involved with the application of cognitive and perceptual psychology to the design and evaluation of software user interfaces.

Joseph Hary *IBM Programming Systems, Santa Teresa Laboratory, P.O. Box 49023, San Jose, California 95161-9023.* Dr. Hary is an advisory human factors scientist in the Application Development User Interface department at IBM's Santa Teresa Laboratory. Previous to joining IBM, he worked as a research scientist at the Institute for Perception Research, in the Netherlands. His current work includes interface design for object-oriented systems and multimedia applications of workstations. He received his Ph.D. in experimental psychology at the University of California, Santa Cruz, in 1984.

Franz Spickhoff *IBM Programming Systems, Santa Teresa Laboratory, P.O. Box 49023, San Jose, California 95161-9023.* Mr. Spickhoff is a senior programmer in the Application Development Systems department at IBM's Santa Teresa Laboratory. Prior to taking his current position in June 1988, he worked for three years in Poughkeepsie, New York, with a team exploring the architectural solutions for an integrated facility for software development, precursor work leading to AD/Cycle. He worked as a systems engineer for IBM in New York and Asia. Mr. Spickhoff received a Masters degree in micro economics (Diplom Kaufmann) from the University of Cologne, Germany, in 1962.

Reprint Order No. G321-5396.