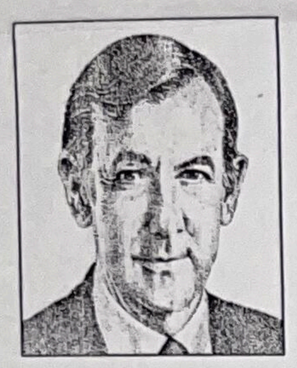
APPLIED INTELLIGENCE

OOP Holds Promise of Simplifying Computer Programming



JAMES MARTIN

This is the first in a series of articles on object-oriented techniques, a new technology that is changing the way programmers and users deal with computers.

The modifier "object-oriented" is more than just a buzzword. It describes a technology that's radically changing the user interfaces and the way computers are programmed. It requires a new way of thinking about the machine that is, in fact, totally intuitive. It is a technique that holds great promise for dealing with the ever-increasing complexity of computer applications.

Object-oriented techniques are having a significant effect on the way computers handle complexity. The techniques are used for encapsulating information, thereby shielding layers of complexity from view. Through the use of objects, a complex problem can be recomposed in simple components.

New operating environments are based heavily on object-oriented user interfaces.

Sophisticated graphics interfaces for CAD/CAM and computer-aided software engineering (CASE) applications are now possible because of these new techniques. Word processors and expert-system shells are using object-oriented techniques. Programming languages based on object-oriented techniques are showing productivity gains of 5-to-1 or better for complex programs. Object-oriented databases are being designed to handle complex data modeling and storage in applications such as CAD/CAM and CASE design repositories and large text databases. New machine architectures, such as the AS/400, are inherently object-oriented systems.

There are many uses of the word "object," but in most cases it encompasses both properties (data) and behaviors (procedures). It is used loosely to describe the icons in a user interface, the entities in databases and the elements manipulated by system software in newer machines. In programming languages, it refers to the encapsulated units of properties and behaviors.

This week, I'll focus on object-oriented interfaces and object-oriented programming.

Although "object-oriented" is used freely to modify both interfaces and programming languages, the two are not inseparable. It is possible to develop object-oriented user interfaces without using object-oriented programming. It is also possible—even easy—to develop bad user interfaces to an application written using an object-oriented programming language.

Object-oriented interfaces directly benefit users. They are much easier for users to learn and work with than conventional interfaces. One only has to compare the user interface of the IBM PC with that of the Apple Macintosh to see the radical difference between the two approaches.

The user interface to the PC is not intuitive, and users require training to deal with its alien command syntax. In contrast, the object-oriented user interface to the Macintosh can be learned quickly and is easy to use.

An object-oriented user interface is more appealing because it presents the user with a view of the application that is similar to the way humans deal with the rest of their environment. People handle physical objects at work, such as books, pens and coffee cups, or more business-related objects such as customer files, orders and invoices. In a non-computerized environment, a user finds an object visually, then manipulates it in some way.

For example, in a typical office, an employee might open a file cabinet of customer data, pull out a customer's folder and change the address. Using a computer with an object-oriented interface, the employee is presented with graphical images representing the objects. The employee would use a device, such as a mouse, to point to the customer data file and to the Select icon (or menu item), enter the customer's name in the window that pops up, and then edit the address in another window that pops up with the current customer information. Pointing to another icon or menu item saves the new information and closes the window.

This is a sharp contrast to character-based interfaces to conventional applications. In traditional systems, the user memorizes commands that must be issued at the right time to get the desired behavior from the system. The traditional focus is on initiating commands and operations first, and then selecting the objects to be manipulated. This approach is the reverse of the way most people manipulate objects in the real world.

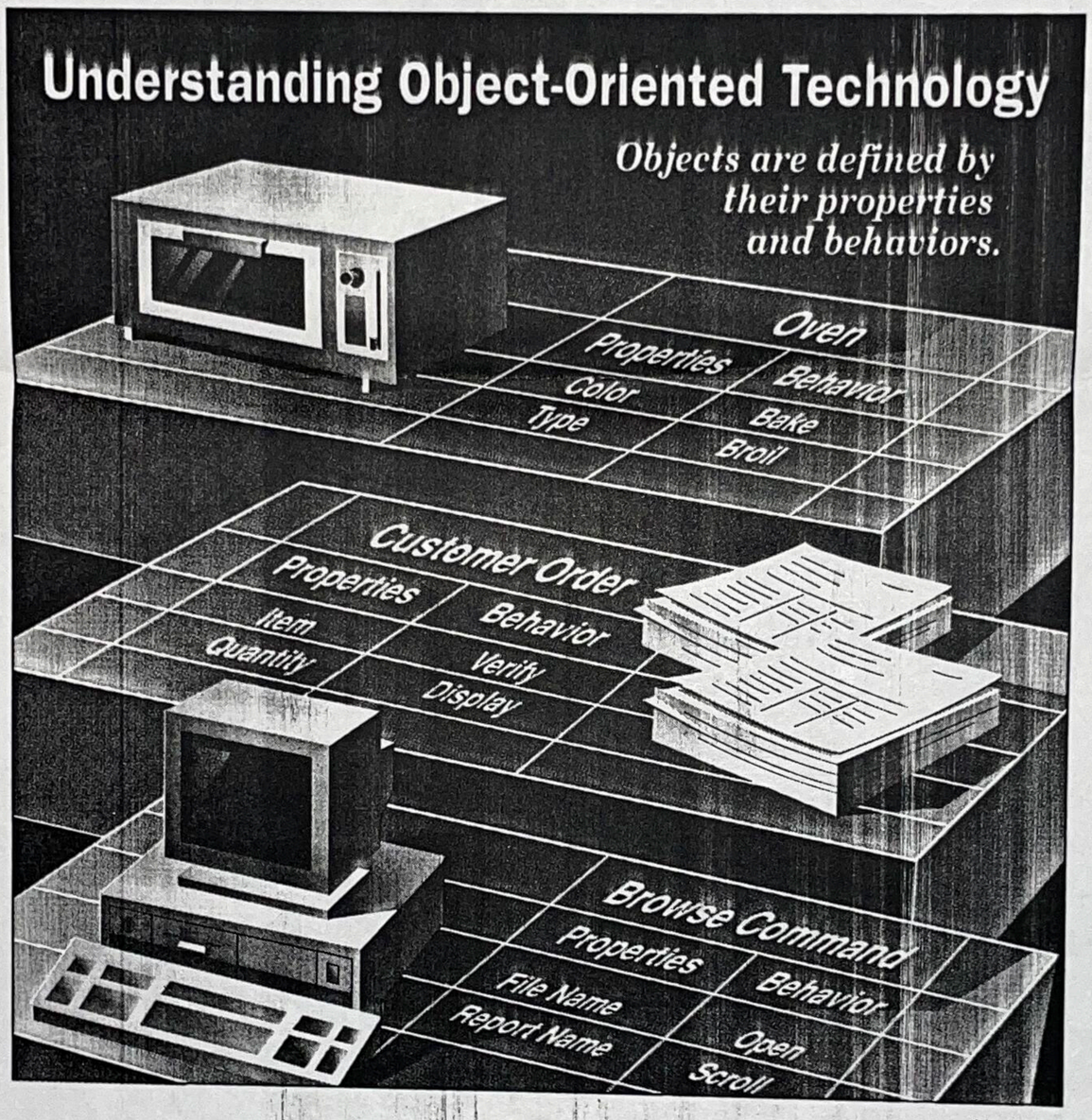
Programming environments that support an object-oriented user interface, such as the Macintosh, IBM's Presentation Manager and Microsoft Windows, provide the programmer with a rich collection of software tools to use in building the interface. The tools are designed to be called from conventional programming languages such as C and do not require a knowledge of object-oriented programming.

They do, however, require a different style of programming. The style is best called "event-driven," and it is similar to real-time programming. Unlike a program organized hierarchically by application functions, the highest level of organization in an event-driven program is a dispatch loop that responds to all user actions. The user is in complete control of the interface and might do anything. The program must be ready to respond—redrawing windows, displaying menus, calling application functions or whatever action is appropriate.

The tool boxes for these interfaces provide facilities for displaying the objects that make up the user interface, such as windows, icons, pointers and menus. They also provide the message passing and queues necessary to communicate between the interface and the application. Typically the interface will send messages to the application.

For example, the interface might send a message to the application that the mouse was clicked at a specific location. The application then needs to determine in what window, if any, the click occurred, and how to respond. If the click was in a non-active window, then the application must make the new window active and redraw the screen appropriately.

Higher-level interfaces to the tool boxes are being provided, but at this point it still takes an experienced programmer to work with them.



John Avakian

"Object-oriented" describes a technology that's changing the way computers are programmed. It requires a new way of thinking about the machine that is totally intuitive.

Managing Complex Code

Object-oriented programming provides major advantages to the programmer in managing the complexity of a code. Using object-oriented techniques, programmers can more quickly develop applications that are easier to maintain, have more reusable code and contain fewer bugs.

University students working with object-oriented languages require an initial learning period to understand the new technology and way of thinking about programming, but then report vastly improved productivity in writing programs. Vendors with large applications and severe maintenance requirements are beginning to use object-oriented languages for development.

Just as a user deals with familiar objects in an object-oriented interface, the developer writes code that defines and manipulates objects in the programming environment. The developer's objects can represent physical entities such as inventory items, but can also represent more abstract programming entities such as a stack, number, dispatcher or collection.

Object-oriented programming allows the programmer to encapsulate all of the code associated with an object, such that the details of the implementation of the object are completely hidden from the the rest of the system.

Next week I will describe object-oriented programming in more detail. ■

To learn more about the subject of these articles, please call The James Martin Report, an information service updated quarterly, at (800) 242-1240. For information on seminars, please contact (in the United States and Canada) Technology Transfer Institute, 741 10th St., Santa Monica, Calif. 90402 (213) 394-8305. In Europe, contact Savant, 2 New St., Carnforth, Lancs., LA5 9BX United Kingdom (0524) 734 505.