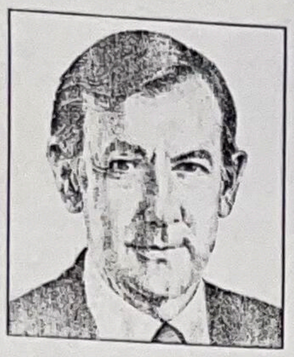
APPLIED INTELLIGENCE

OOP Just the Ticket for Complex Commercial Programs



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

Object-oriented programming techniques are becoming generally available to programmers and are finding use in such applications as user interfaces, word processing, graphics programs and databases.

These techniques haven't made a big impact on commercial applications, which usually are slow to embrace the available technology.

But it's inevitable that the tremendous productivity gains of object-oriented techniques soon will be applied in the commercial environment.

The commercial programming application backlog has been attacked with fourth-generation languages, application generators, relational databases and expert systems.

However, information-systems departments are still unable to keep up with the demand for newer, more powerful and more complex applications.

The object-oriented paradigm promises to be a major step toward reducing the application backlog, but professional programmers still need tools if they are to reap the benefits of object-oriented techniques.

The languages and applications discussed in previous articles use memory to store objects. When the program has stopped running, the objects disappear. The objects defined are specific to each program and cannot be used elsewhere in the system.

But what if the objects were stored on disks and could be accessed from run to run? This is exactly what's happening, and there are a number of such object-oriented databases just coming to market. They promise to vastly decrease the work needed to solve certain problems.

The early users of these object-based database systems are mostly those involved in computer-aided engineering (CAE), who face complex data-management problems.

The data objects they use are complex shapes that interact with one another. These objects can be viewed independently or as a group. The CAE tool must be able to store drawing information, retrieve it and manipulate it.

These are all database-management tasks, but the data does not fit easily into a relational format. Object-oriented databases provide a way to deal with this complexity, just as in the graphics programs I described last week.

Object-oriented databases are also being used by people building large document-handling systems. Again, the problem is the data model.

The document is mainly text, which does not lend itself to being stored in relational records. But in an object-oriented database, the text, sections and headings are all objects. The interrelationships are modeled in the objects.

While this isn't relevant for simple word-processing applications, it can be important for applications that store large amounts of text, such as those that keep track of all of IBM's manuals.

Computer-aided software engineering (CASE) is another area that can benefit from an object-oriented database. The issue again is complexity. An object-oriented database helps developers to model the complex objects and relationships needed in a CASE environment.

The early acceptance of object-oriented databases for applications with severe data-modeling problems indicates that such systems bring big benefits.

The same techniques can be brought to bear on commercial applications in the future. Consider the current way in which database applications are developed.

The user of a new system describes some data object, such as an order, to the systems analyst or programmer. The user already views the order as an object with associated behaviors. The programmer then goes through the painstaking process required to break down the user's concept of an order into the procedure. The code must take into account how updates and queries for orders access the various records involved in the order.

In other words, all of the knowledge on how to handle an order is stored in process code and not in the database.

An object-oriented database allows the complex order object to be built up from simple forms. The methods for handling an order are stored in the database in the form of properties (data) and behavior (procedures). The order responds to queries and update messages, and so it is an object that can be manipulated by the system. The user deals with a familiar object and manipulates normalized relational records that are stored in the database system.

Note, however, that nowhere in the database system is the concept of an order defined.

Instead, the individual relational records that make up an order are defined. This is very similar to the problem with the basic paint programs described last week, in which the program had no "knowledge" of the shapes, only of the individual bits on the screen.

In the conventional approach, after the programmer has defined the normalized database, it is then time to write the code that manipulates an order. This might include a query and an update it. The programmer deals with a programming model that is closer to the user's view of the world.

Other programmers only need to send messages to the order object and do not have to worry about the structure of the order or the database semantics involved in an update of the order.

New types of orders could easily be built from the existing order. For example, a purchase order is just like a regular order, except that it has outside vendor information in it as well.

A manufacturing order has details on how to satisfy the order on the factory floor, but the basic structure remains the same.

A Cure for Complexity

Modern applications are reaching the limits of complexity imposed by relational-database technology.

One mainframe manufacturing application's code handles orders by breaking them down into approximately 30 record types. The logic for manipulating the order requires maintaining the semantic integrity of all those records. The manufacturing order is just one small part of the system.

No wonder deadlines are missed, and the size of the application grows out of control.

An object-oriented database's approach to the same problem would bring this complexity into a more manageable form.

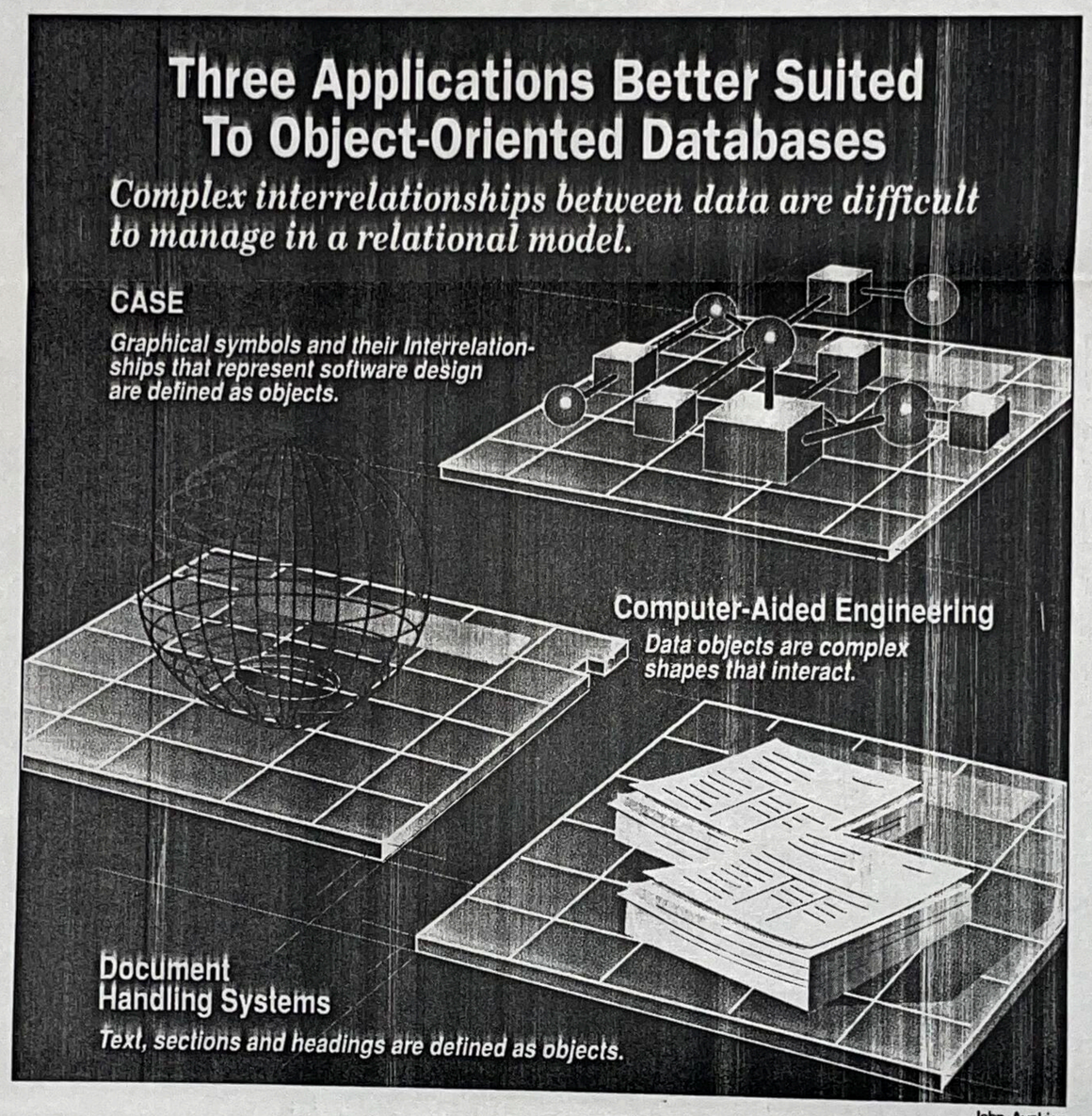
The manufacturing order would probably be a subclass three to four levels down in the class hierarchy. It would be abstracted to the point where it was easy to handle. If a user wanted to customize the software, the changes could be localized to the object definition.

Early commercial object-oriented database systems are available on workstation platforms. Servio-Logic (Portland, Ore.), Ontologic (Billerica, Mass.) and Graphael (Waltham, Mass.) are three vendors of object-oriented database systems.

The first system is based on Smalltalk, an application-development tool made by Digitalk Inc. The other two systems are based on object-oriented C extensions; in these, the user defines the database as objects in C. When the system runs, the objects are stored on disk.

These object-oriented extensions are called persistent object languages to distinguish them from the pure languages from which they were derived. They also handle such standard database activities as storing and accessing data.

Next week I'll begin a series of articles on the recently announced IBM repository, a significant component of IBM's application-development strategy for the 1990s. ■



John Avakian

The object-oriented paradigm promises to be a major step toward reducing the application backlog, but professional programmers still need tools if they are to reap its benefits.

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.