

The Benefits of Object-Based Authoring Systems for Small Businesses

Carl E. Keller, Jr.

*Assistant Professor of Accounting
Coastal Carolina University*

Benjoe A. Juliann

*Assistant Professor of Computer Science
Coastal Carolina University*

Abstract

Object-oriented authoring systems present many opportunities for small companies to develop their own customized software. This article describes object-oriented authoring systems, gives examples of how they simplify programming, and concludes with several illustrations of how small businesses may take advantage of these software development systems.

Introduction

Managers of small businesses may have the following wishes: "I wish that I didn't have to take time out to train a new employee again," "I wish that we could do a multimedia presentation to land that large contract next week," or "I wish that our company had a computer program designed to keep only the records that we need, in a format that we could understand." Such concerns are common to both large and small businesses, but in the past, only large companies had the capital and the manpower to develop software programs that would allow these wishes to become reality. However, object-based authoring systems are becoming a cost effective tool used by small companies to write their own software applications.

In the past, most small businesses purchased software "off the shelf," believing that customized software programs would be too expensive. The typical small business could not afford to hire a programmer, nor contract the project out to a programming company. Furthermore, most small businesses did not possess employees that had enough programming skill to write even the simplest application packages. Programming skill was required due to the complexity of the programming languages and their related applications, not to mention the difficulties of debugging a program. Even if a business had a skilled programmer, typically the company could not afford the capital investment (in terms of man-hours) to allow the employee to write the program.

The advent of object-based authoring systems allows small companies to overcome these obstacles to owning their own customized software. Many authoring systems use a fourth generation programming language which is much easier to use than earlier programming languages. These fourth generation languages are structured to be similar to people's speech patterns and are more accepting of different

terms to execute commands. The object/event orientation of these authoring systems allows some programming to be done with a mouse. Thus, programmers can visually see what they are creating as they are writing the program. This fact allows authoring systems to decrease the initial need for programming skill and to reduce the time to write an application. In addition, the more programming that you do with an authoring system, the faster you become at writing applications. Best of all, while there is the initial cost of buying an authoring system, (probably between \$300 and \$4,000), the rights to any programs that you write with the system are usually owned by you!

The first section of this paper explains the technical aspects of an object-based authoring system. The second section provides examples of how easy programming can be when a person is using an object-based authoring system. Finally, the paper discusses potential uses and benefits of authoring systems for small businesses.

Object-Based Authoring Systems

Several authoring systems are available to the public, although you will not usually find these packages on the shelves of your local computer software store. Exhibit 1 displays a partial list of authoring systems. Each authoring system operates uniquely and has different capabilities. The following paragraphs describe the basics of object-based authoring systems.

An authoring system is computer software that allows developers to create programs/applications with writing a minimal amount of programming code. *Object-based systems* and *object-based programming languages* usually refer to systems with *objects* with *inheritance* (Cardelli, 1985). The objects are the building blocks of any program created using an object-based system. An object can be any individual, identifi-

able item, unit, or entity that exists in the application. An object has a state, a behavior, and an identity. Examples of objects that typically exist in a program are navigation buttons, text fields, data or record fields, and graphic objects.

So how do these objects make life easier for unskilled programmers? Suppose you wanted to create a navigation button that would allow a user of your program to go from one screen (a.k.a. "page") to the next screen, in a sequential manner. Some authoring systems (such as Macromedia's Authorware 3.0) may have an icon that you select to insert the forward navigational button. Thus, you simply click the mouse where you want the button to appear, then click the mouse on the appropriate icon, and the button will appear ready for you to test. These are referred to as icon-driven authoring systems.

Other authoring systems require a little more work, but they may grant more flexibility. For example, to create a forward navigation button with a different authoring system (such as Asymetrix's Toolbook 3.0), the programmer would follow these steps:

1. Click the mouse to select a type of button from the tool palette (there are several types to choose from).

2. Use the mouse to point to where the button will be located, then "click and drag" to shape the button to the desired size.

3. Select "button properties" from the "Object" menu. Type in a name for the button (e.g., "Forward1") and a caption to appear on the button for users (e.g., "Forward") in the pop-up menu. Then select the "Script" button to type in the following code:

```
to handle buttonClick
    send next page
end buttonClick
```

4. Select Update and Save Script with the mouse. Then exit the button properties menu and you have created your forward navigation button.

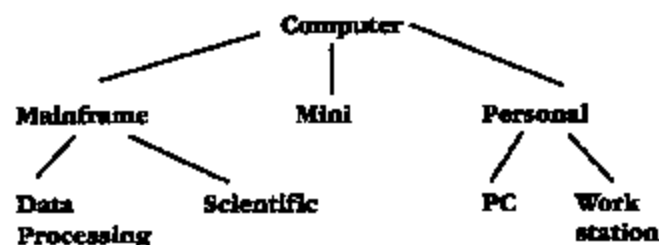
Note two things about programming with this second authoring system. First, one can easily see that the object (e.g., the button) has the three characteristics mentioned earlier. The button has a specific identity via the name of "Forward1." The button is either in an activated or an inactivated state, and performs the specific behavior of sending the user to the next screen (page) when activated. Second, simply by writing a little bit of code, a programmer can gain a lot more flexibility. To illustrate this point, suppose you would like a program that would either:

1. Automatically play music for the user as they view the next screen,
2. Record the amount of time the user spends reading the new screen, or
3. Automatically print a document as the user goes to the next page.

Any of these events (or any combination of them) can be activated by the same forward navigation button with only a few extra lines of code using the second

authoring system. Icon-driven authoring systems would not allow the navigation button to control any of these events because the icon generates the code. Thus, most authoring systems are set up to require some programming code to provide the programmer the ability to customize each application that they develop.

Within an object-oriented environment, the structure and behavior of similar objects are defined in their common *class*. A class also specifies the *inheritance* of an object. Inheritance provides a natural classification for objects. The naturalness comes from the fact that we use concepts, classification, and generalization to understand and deal with the complexities of the real world. (See the example below on using computers.)



Inheritance is a relationship between classes where one class is the parent (a.k.a. base, superclass, or ancestor) class of another. In a classical object-oriented environment, inheritance is a relationship between classes only. Thus, most object-based authoring systems contain a hierarchical organization (often referred to as the *object hierarchy*). Exhibit 2 displays the object hierarchy for the Asymetrix's Toolbook authoring system. Object hierarchy is important because parent objects control child objects to some extent. For example, a page (parent object) controls buttons (child objects) that are located on it. If you cannot view the page, you cannot use the button. Furthermore, while all objects can send and receive event messages to each other, the messages have to pass through the object hierarchy. Therefore, the hierarchy becomes important when the order of events is significant. The authoring system will perform events based upon the order the objects receive their messages. Thus, in the Toolbook object hierarchy, an event that sends separate messages to a page and a button on that page will find that the page message is performed first.

Ease of Programming

The traditional programming environment creates applications by first defining data and then writing the related procedures to handle or manipulate the data. In the object-oriented environment, a program is developed by first creating objects, and then defining the object's behavior by writing program code for that specific object. Recalling the previous example of the forward navigation button, the reader should

Exhibits

EXHIBIT 1 Partial List of Authoring Systems*

Act III-Informatics Group, Incorporated	(203) 953-4040
Authorware-Macromedia, Incorporated	(800) 326-2128
Director-Macromedia, Incorporated	(800) 325-2128
Express Author 1.1-Research Triangle Media, Incorporated	(800) 282-2425
GUIDE Author-InfoAccess, Incorporated	(206) 747-3203
HyperCard-Apple Computers, Incorporated	(800) 776-2333
Icon Author-AimTech Corporation	(800) 289-2884
KSS: Author-Comware, Incorporated	(513) 791-4224
LinkWay Live!-EduQuest (An IBM Company)	(800) 426-4338
Media Master-Vision Imaging	(800) 292-4264
MediaMax-Videodiscovery	(800) 548-3472
Multimedia Toolbook-Asymetrix Corporation	(800) 448-6543
TBT Author-HyperGraphics Corporation	(800) 369-0002
Toolbook-Asymetrix Corporation	(800) 448-6543

*Adapted from *AuthorBase (NLM and ARL, 1995)*

EXHIBIT 2

Object Hierarchy*
(in descending order)

1. Toolbook development software
2. System programs
3. Program currently being developed
4. Backgrounds
5. Pages
6. Groups
7. Buttons, field, graphic objects

* Adapted from *Toolbook User Manual*
(Asymetrix, 1994)

EXHIBIT 3

Programming Code from a Recorder

```
to handle buttonClick
    set printerStyle to pages
    set printerMargins to 1440, 1440, 1440, 1440
    set printerGutters to 360, 360
    set printerScaling to actual
    set printerBorders to false
    set printerArrangement to 1, 1
    set printerPageBitmap to true
    start spooler
        go to page 14
        print 1
    end spooler
end buttonClick
```

References

Asymetrix Toolbook User Manual. Version 3.0. 1994. Asymetrix Corporation, Bellevue, Washington.

Booch, Grady. 1986. "Object-oriented development." *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, pp. 211-221.

Booch, Grady. 1994. *Object-Oriented Analysis and Design with Applications*, Second Edition. Benjamin-Cummings Publishing Company.

Cardelli, Luca. 1985. Amber. "In Cousineau, Curien and Robinet" (Ed.), *Combinators and Functional Programming Languages*, LNCS 242, Springer-Verlag, pp. 21-47.

Cardelli, Luca. 1988. "A semantics of multiple inheritance." *Information and Computation*, Vol. 76, pp. 138-164.

Jacobson, Ivar, Magnus Christerson, Patrik Jonasson, and Gunnar Overgaard. 1995. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Revised printing. Workingham, England: Addison-Wesley Publishing Company.

National Library of Medicine and the U.S. Army Research Institute. 1995. *The AuthorBase Alphabetical Listing*. (Downloaded from the Internet, <http://www.ncbi.nlm.nih.gov/authorb/idx/index.html>).

Stroustrup, Bjarne. 1987. "What is "Object-Oriented Programming?" In J. Bezivin, J-M Hullot, P. Cointe and H. Lieberman (Ed.), *Proceedings of ECOOP '87, LNCS 276*, Paris, France: Springer-Verlag, pp. 51-70.