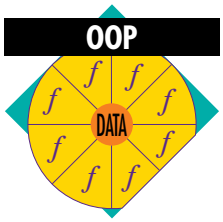
Creating Object-Oriented Solutions



By Peg MacPhail and George Noren

This article defines a road map that can guide you into the world of object-oriented application programming on AIX, OS/2, and OS/400. The road map uses non-proprietary, open technologies that exist today.

IBM and other companies, such as Hewlett-Packard, Sun, Apple, and Microsoft, are investing heavily in object technology. With such extensive development effort focused on object technology, we can expect new and innovative programs to solve previously complex problems, as well as new programs to replace existing procedure-oriented applications.

This transition to an object-oriented environment will take years, during which time a mixture of object-oriented and procedural programs will coexist. Object-oriented solutions that appear early in this transition period will have an advantage over later solutions by establishing the standards and shaping the object-oriented market of the future.

Background

Figure 1 shows the evolution from objects to object-oriented frameworks. Object programming uses components called *classes* as building blocks that can be used in programs—similar to integrated circuits in circuit boards. An object-oriented class provides canned code (called *methods* or *behaviors*) and attributes for a specific application component such as a device driver. An object class designer could create a general device driver class and then further refine it using *subclasses* to define any particular methods or attributes of a specific device driver. These components or classes can be used either directly by an application or customized by creating further subclasses.

The developer of a large object-oriented application is likely to deal with many different object classes. Since reusing code is a major objective, it is vital to keep track of these class definitions. Class libraries were developed to organize collections of class definitions for easy access. Usually, object-oriented class libraries provide methods and attributes for related sets of application components. For example, a set of device driver classes could be packaged as a class library.

Application frameworks provide sets of pre-integrated objects that perform specific tasks. They multiply the productivity gain of object classes by providing a higher level of prefabricated code. Application frameworks can be used in developing user interfaces or performing functions such as printing and communication. Typically, an application framework provides a basic set of functions for a task that the developer can customize and extend.

A Road Map for Cross-Platform Development

Figure 2 shows a common, cross-platform suite of high-level Application Program Interfaces (APIs) and services. These complementary APIs and services are open, non-proprietary, and intended for a wide range of industry platforms. For example, C++ and System Object Model (SOM) are rapidly becoming the object-oriented technology of choice for programming language, object distribution, and object packaging. We expect OpenDoc and the Taligent frameworks to set the standard for compound- document and framework technology.

OpenDoc technology comes from Component Integration Laboratories (CI Labs), a consortium being formed by Apple, Novell, WordPerfect, IBM, and others. OpenDoc provides the state-of-the-art compound-document support with visualization, Open Scripting Architecture (OSA), document storage (Bento), and SOM technologies. CI Labs will provide source code much like the X consortium provides code for X-Windows

George Noren

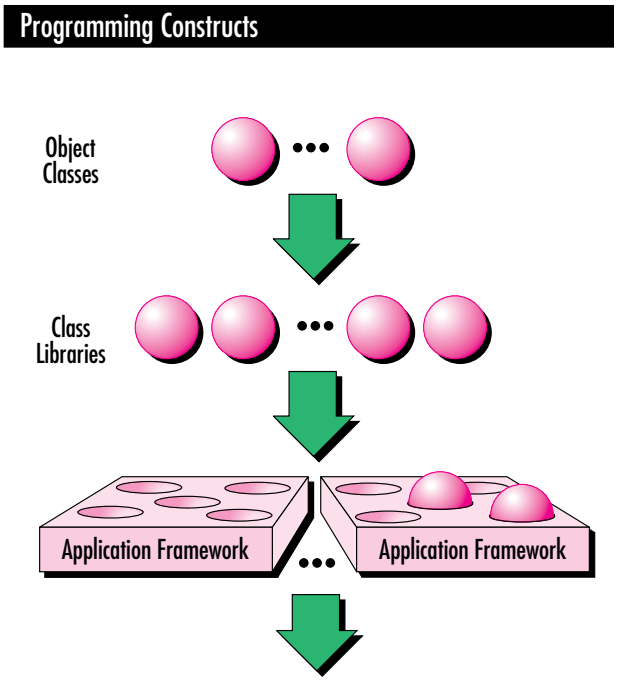


Figure 1. Object-oriented programming constructs

today. Developers can use the OpenDoc SOM object classes to create OpenDoc versions of their applications. The pervasiveness of OpenDoc and SOM means that OpenDoc applications are easy to port and can interoperate extensively, including two-way interoperation with Microsoft's OLE compound documents.

The OpenDoc environment includes both new object-oriented application programs as they are developed, and existing procedure-based application programs. A procedural application must be changed to be enabled on SOM so it can be easily moved to other platforms. It must also be changed to be enabled on OpenDoc, which allows it to interchange data with other programs that support OpenDoc.

The PowerOpen™ Environment (POE), defined by the PowerOpen Association, has been adopted by IBM and the other PowerOpen Association members (including Motorola™, Apple, Bull®, and Thomson-CSF™) as a strategic, cross-platform support environment for procedural applications. The POE consists of the base PowerOpen Application Binary Interface (ABI) and two graphical presentation engines: Macintosh Application Services (MAS) and OSF/Motif®. The PowerOpen ABI is a proper superset of the Spec 1170 work, a common UNIX specification that has been submitted to X/Open for approval. It provides a consistent binary interface across multivendor platforms for procedural applications. The PowerOpen ABI also provides a cross-industry operating system foundation for the emerging object-oriented application framework.

The SOM and OpenDoc components, shown in Figure 2, help to integrate applications across the different platforms. A SOM component provides a distributed object computing technology that is compliant with the Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA). This component allows objects from different types of class libraries—such as Smalltalk, C++, and C—to interoperate.

SOMObjects Developer Toolkit has been shipping on AIX and OS/2 since mid-1993. Because SOM is available on so many operating environments, interoperability among objects is a reality. Even if SOM is not on a platform, messaging can occur between SOM and the other Object Request Broker (ORB) if the platform has an OMG CORBA-compliant ORB.

A program can support compound documents by generating OpenDoc parts. Each part supports a specific set of methods, such as those used to open or edit its contents. Since each kind of part knows how to handle the data associated with it, an OpenDoc document can contain many different kinds of document parts without actually knowing how to handle the data from any of them. The OSA technology provides a higher-level messaging capability with predefined verbs, providing support for a defined interface to common services such as mail and print. OpenDoc parts are SOM-enabled.

Application Frameworks

Figure 2 shows several application frameworks that provide high-level APIs and services for programs running on many platforms. The following examples describe these SOM-enabled application frameworks:

- **Taligent Application Environment (TalAE)** includes object-oriented 2-D and 3-D graphics, international language support, a standard "look and feel," Taligent OS services, and multimedia support.
- **Taligent Development System (TDS)**, a set of object-oriented application development tools, allows incremental link and compile, automated build, multiple views, hyperlink navigation, integrated source-level debugging,

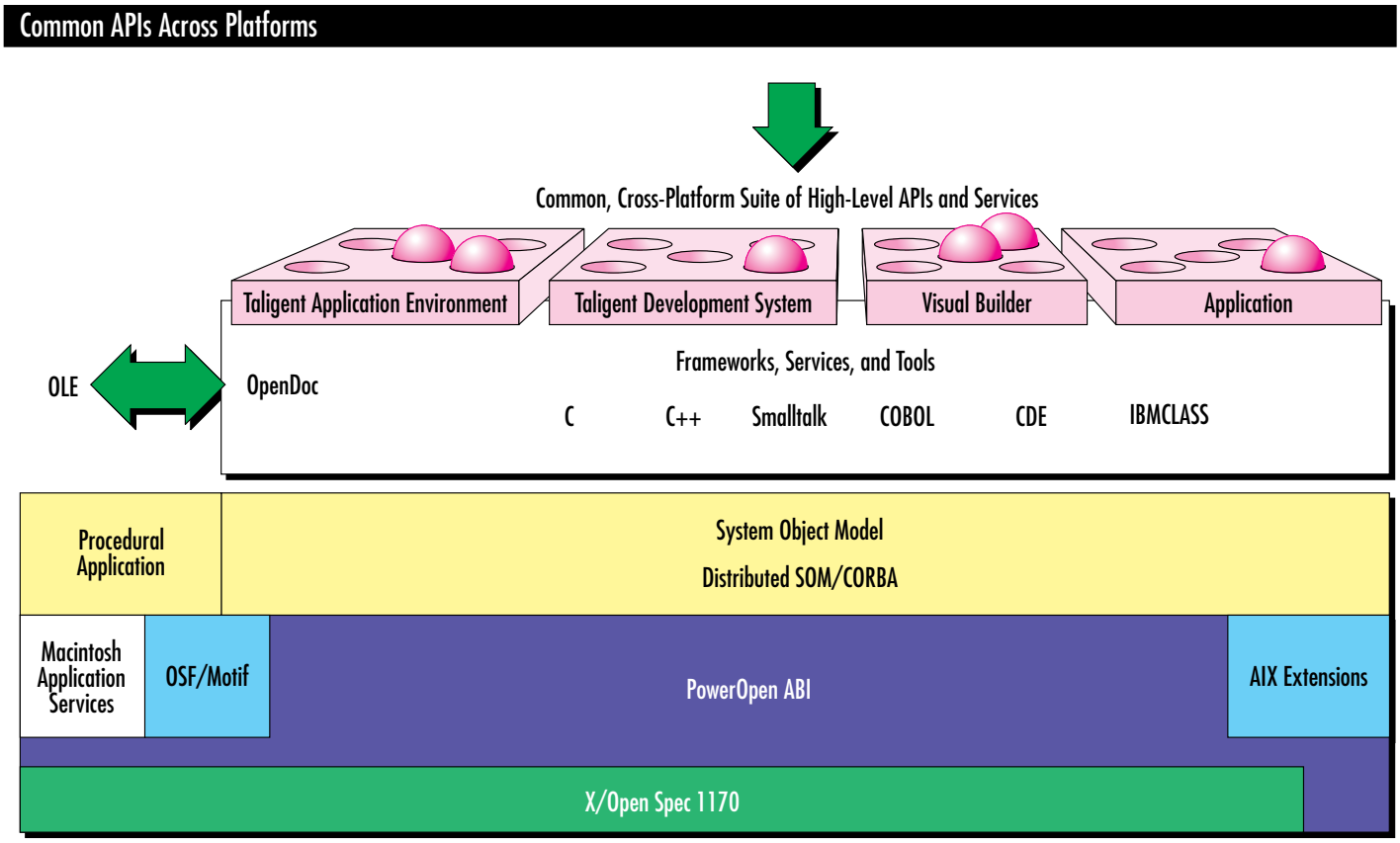


Figure 2. Object technology provides common APIs across diverse platformss

dynamic browsers, and a Graphical User Interface (GUI) builder.

- **A Visual User Interface (UI) builder** allows programmers to use graphical techniques to quickly build an application. These applications become OpenDoc parts for use within UI application frameworks.
- **VisualAge** is an integrated application development environment that provides visual programming and construction-from-components technologies. In addition, it provides support for team programming, library services, advanced GUI, communications and transaction processing, database components, access to other applications, visual SQL query builder capability, multimedia capability, and a complete application development environment including IBM Smalltalk.

C Set++ for AIX provides a robust compiler for C and C++ applications, as well as a C++ Class Browser, C++ Class Libraries, HeapView Debugger, and a test coverage tool.

Taking Advantage of OO Technology

Object-Oriented (OO) technology is a developing discipline. Although it is not yet fully available on all operating environments, you can still prepare today for its arrival. The first step is to develop object-oriented expertise. If you do not have experience with OO projects, take training classes, attend seminars, and read articles and books on the subject.

Once you have a good foundation, analyze your applications for potential improvements from OO technology. Consider not only the function your application offers to an external user, but also what your application might offer to another application. Consider subfunctions that might be useful to a user or another application. Then, model your application into one or more objects with the appropriate subfunctions that each would expose to another application or to a user.

The idea is to separate the interface from the implementation. Think of it as a contract in which the application agrees to perform a task, and is empowered to do it any (legal) way that it wants.

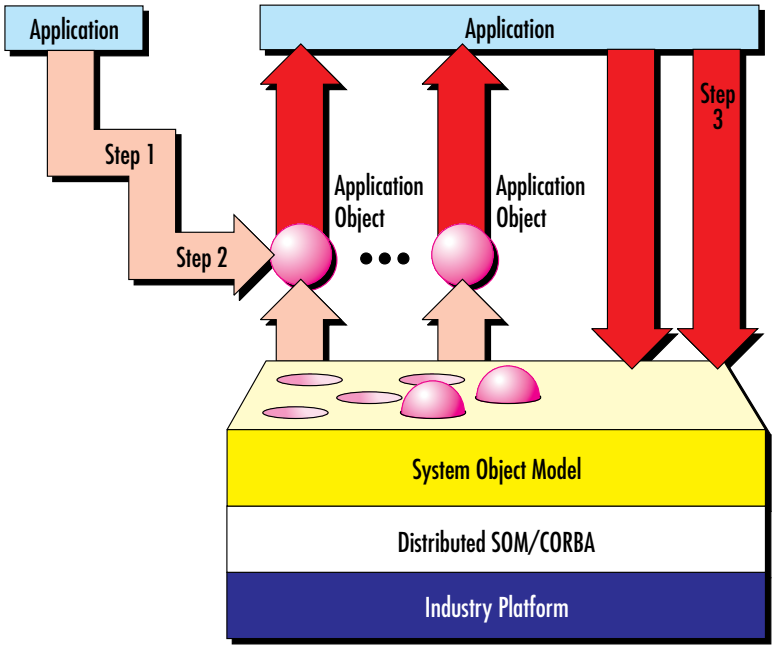


Figure 3. Migrating an existing application

It is important to set realistic expectations and goals, and make your first attempt with a non-critical application that can be easily separated and defined. The first step is to create a small, skilled OO team and develop a plan to move the application into the OO world. Once you have completed a successful project, you can then leverage that experience to other programming teams to create new applications using application frameworks. Consider using OpenDoc for applications that will run on OS/2 or Windows.

Migrating an Existing Application

Figure 3 shows a schematic of the process of moving an existing application to an OO environment. The numbers in the figure refer to the following steps involved in migrating the application:

1. Model your application design into theoretical objects.
2. Create SOM or C++ objects to make an object interface to the application.
3. Add SOM client calls to the application to use other object services.

Step 1 is a paper-and-pencil exercise. Divide your application into objects. If you have a highly modularized design, break your application into modules or groups of modules. You can then take advantage of the modular design when migrating your application. For example, if the function in `module_x` changes, you could rewrite `module_x` as an object. Calls to `module_x` in the procedural application would then be replaced by SOM client calls to the object `module_x`, bypassing the `module_x` procedural code in the application.

If you never want to change your procedural application, the objects you create to export the application's interface can be like a firewall between the object world and the application. New function can be written as objects. However, access to your program through a procedural call will not produce the new object function unless you change the procedural application to call the objects.

Steps 1 and 2 make your application available as a server to object clients. These steps do not require any changes to your procedural program. Although you may change your procedural program to make this process easier, it is not required. Steps 1 and 2 can be done independently of step 3. Step 3 changes your application to be an object client to an object server. This step can be done independently of steps 1 and 2.

Example: Migrating an Existing Application

The following example illustrates the concepts involved in migrating an application to object technology. All example code is in C; C++ would look slightly different because it supports object pointers, constructors, and destructors.

The example program presents exercises for a student on a computer. The student can stop during an exercise and finish it later. The set of exercises can be tailored to the student's performance. For example, if a student needs more emphasis on a certain topic, the processing state can be set to show that the student must do an extra exercise. The original exercise can be stopped until the additional exercise is completed, then the student can return to finish the original. Figure 4 shows pseudo-code for the structure of the program.

Step 1: Model your application design into theoretical objects. This application could be modeled into the following objects: Teacher, exercise_x, exercise_y, exercise_z, Student, and printer.

Step 2: Create SOM or C++ objects to make an object interface to the application. To do that, write the Interface Definition Language (IDL)

interface to support the teaching exercises application program. The code for the Teacher object might appear as follows:

```
#include <somobj.idl>
interface Teacher : SOMObject
{
   void teach_exercises;
   attribute string name;
};
```

The SOM compiler generates program stubs that will be filled in to call the procedural application. You would then write other IDL interfaces for the remaining objects. To make the objects externally available, register their IDL with the SOM Interface Repository and add the object methods to a dynamic link library.

Step 3: Add SOM client calls to the application to use other object services. Replace the calls to the database and print routine with SOM calls to the student and printer objects. For example, you could add code similar to Figure 5 to the application in place of the existing calls.

Example: Migrating an Existing Desktop Application

OpenDoc can be used to migrate a desktop application to object technology. The following three steps (illustrated in Figure 6) are similar to those in the previous example.

Step 1: Model your application design into compound document parts. In the previous teaching exercises example, make each exercise a document. The test questions in each section can be a compound-document part. Figure 7 shows the kind of functions that can be achieved using compound documents.

Step 2: Create OpenDoc parts for the application. The term *part* refers to an object class used by developers to derive any document part class. Developers must implement about 50 object calls for each part. The following are examples of these calls

```
void Draw(
   XMPFrame* frame,
   XMPShape* invalidShape)
XMPBoolean HandleEvent(
   XMPFrame* frame,
   XMPEventData event)
void AddDisplayFrame(
   XMPFrame* frame,
   XMPName* viewtype)
```

Step 3: Add OpenDoc calls to the application. To do this, call other OpenDoc script-enabled services such as mail, calendar, or print.

```
Teaching exercises application pseudo-code
     Global variables: processing_state structure, section_x,
     section_y structure,
          section_z structure, student_data structure, name

main routine (name)
     initialize
     get processing_state from database
     get student_data from database
     while (processing_state, exercise_status = not_done) do
          Case switch on processing_state
               Case processing_state = need_section_x: call give
               section_x()
               Case processing_state = need_section_y: call give
               section_y()
               Case processing_state = need_section_z: call give
               section_z()
          end_Case
     end_while
     if processing_state, print_status = print_section then print
     selected sections
     cleanup and return

give_section_x routine()
     get section_x from database
     fill in section x.structure with data already in student_data
     structure
     give exercise_x_to_student
     save section_x data and set new processing_state
     return;
```

Figure 4. Teaching exercises program pseudo-code

```
#include <printer_object.h>
#include <student_object.h>
printer_object objp
student_object objs
.
.
.
objp = printer_objectNew();
objs = student_objectNew();
.
.
.
get_student_data(objs,somGetGlobalEnvironment(),name);
.
.
.
_print_exercises(objp,somGetGlobalEnvironment());
.
.
.
_somFree(objp);
_somFree(objd);
```

Figure 5. Adding SOM calls to a procedural application

Using OpenDoc for Migrating Application

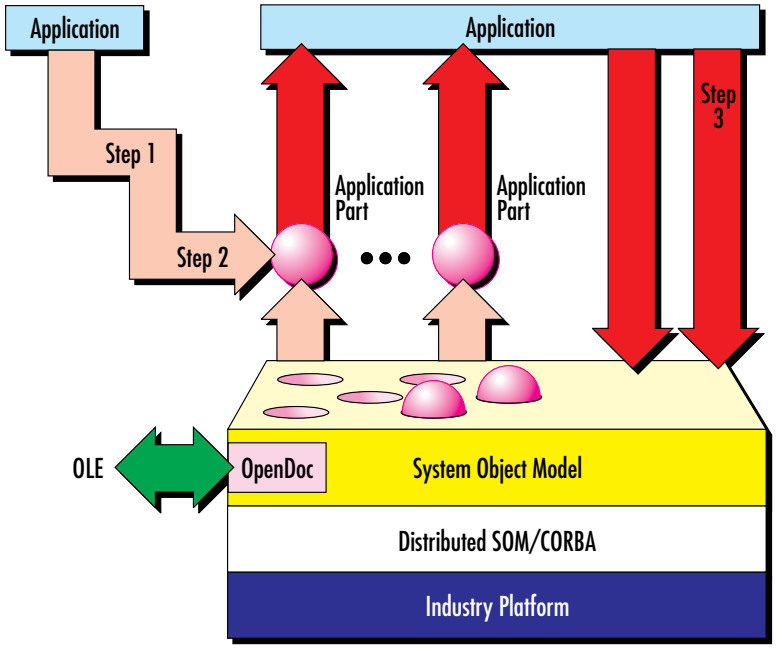


Figure 6. Migrating an application using OpenDoc

Compound-Document Screen

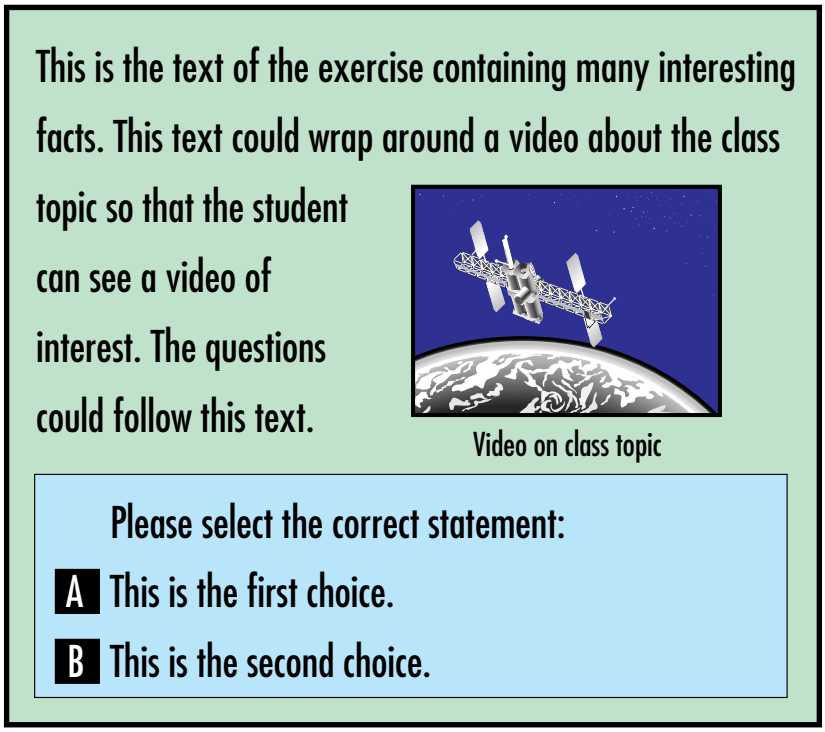


Figure 7. Example compound-document screen

Summary

IBM has already started to roll out an object-oriented, cross-platform development and run-time environment. This environment will provide interoperability between platforms and application portability across platforms (not limited to IBM platforms) that support the new environment. In the new environment, older procedure-based applications can still interoperate with objects and frameworks using the common base of SOM. Procedural applications will not, however, be as integrated into a framework as an object that is written to that framework.

Because various operating systems will support the new environment, porting an application from one operating system to another becomes very easy, usually involving only a recompile of the source. Using frameworks as building blocks in the code further reduces development and code-maintenance time and costs. Applications can be available more quickly on a larger number of platforms.

Peg MacPhail, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: macphail@austin.ibm.com. Ms. MacPhail is a technical consultant for RS/6000 solution providers. She has spent nearly 20 years with IBM in a variety of technical positions. She holds a BA from the University of Connecticut at Storrs and a MCS from Texas A&M University in College Station.

George Noren, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: geo@austin.ibm.com. Since joining IBM in 1979, Mr. Noren has written manuals for System/34, System/36, and AIX on both the RT and RISC System/6000 platforms, and was a member of the InfoExplorer design team. He has also worked as system administrator for several AIX server machines and their clients, and is currently responsible for the Prototype Evaluation Labs in Austin. Mr. Noren studied engineering at Illinois Institute of Technology. He has a BA in English from the University of Minnesota and an MBA from St. Edwards University in Austin.