OpenDoc

Shaping Tomorrow's Software

White Paper

Contents

The OpenDoc Architecture	3
Background	4
Benefits to the User	5
Benefits to the Developer	7
The Technology	8
The Competition	12
The Shape of the Future	14

The OpenDoc Architecture

IBM, Apple, Oracle, WordPerfect, Xerox, Novell, Borland, and Taligent share a common goal of making the user's computing experience as easy and productive as possible, and that goal has become increasingly elusive. People are using personal computers for more and more complex
tasks, often involving multiple programs and even a variety of media. In addition, they are increasingly working together on computer-based projects. This means that the earlier, individual desktop computer model is shifting to one of shared, collaborative computing resources—a shift that demands new capabilities. Finally, there is a growing demand for customization capabilities to meet the increasingly specialized needs of today's computer users.

Apple's Opendoc Architecture represents an approach toward reducing the complexity of computing today, while supporting the development of tomorrow's advanced, flexible applications. It is an open architecture, designed to integrate software and enable sharing across multiple computer platforms—providing users with a new level of computing power, flexibility, and ease of use.

This approach evolves desktop computing by providing an object-based framework for developing applications that are fully integrated and interoperable across platforms and distributed networks.

Background

Ten years ago, most of what people did with computers centered around text and numbers. The advent of graphical user interfaces brought a new emphasis on working with graphics on the computer, because the graphics-based user interface allowed easy manipulation, editing, and integration of words and images.

Today, however, many computer users engage in the creation of what can be called compound documents—documents with parts containing various media, such as text, tables, movies, sound and graphics in a variety of file formats. Currently, each medium requires users to work in different ways, and often in separate applications or editors, demanding a labor-intensive series of actions to move data from each creator application to the final document. This lengthy and cumbersome process tends to be error-prone and frustrating.

Today's computing world encourages an ever-increasing complexity in successive releases of most applications, because developers are under constant competitive pressure to add more features to their products. The result is paradoxical: as applications become more powerful in terms of features, they also become more difficult to use—and hence less useful to people. In addition, they require more time and effort to develop, enhance, and maintain.

Apple's Opendoc Architecture is a compound document architecture that addresses these issues by reducing the complexity and increasing the flexibility of software for both end users and developers. It offers an evolutionary approach to restructuring software into independent modules, or "parts," which can be flexibly combined in a variety of ways. The result is an entirely different way of both using and writing personal computer software—one that offers a number of significant benefits.

Benefits to the User

For end users, Apple's Opendoc Architecture will simplify and streamline the computing experience, while ensuring a much higher level of flexibility and customizability. Specifically, Apple's Opendoc Architecture offers the following user benefits:

Easy creation of compound documents. Apple's Opendoc Architecture is designed to handle current and future types of media. Users can place any kind of media into an OpenDoc document using the familiar cut-and-paste or drag-and-drop manipulation.

Editing "in place." With Apple's Opendoc Architecture, users can edit any type of content within a single document, without having to cut and paste between different application windows.

Powerful document management. Rather than manually assembling the various pieces of a document, users can let an OpenDoc document hold all of them. This reduces the task of managing files, and facilitates document exchange and updating. As documents are edited, changes are tracked through drafts, ensuring greater data integrity and allowing users to work on shared documents without content loss from version to version.

Cross-platform support. Because Apple's Opendoc Architecture is designed to offer full interoperability between platforms, OpenDoc users will be able to share and interact with complex documents, regardless of differences in software or hardware, or which platform the document resides on.

Consistency of operation. Because users can specify preferred part editors, they need learn only one way to edit each type of data—for example, using the same text editor for word processing, entering spreadsheet data, or labeling diagrams.

Uniformity of interface. Apple's Opendoc Architecture defines a consistent user interface for embedding and manipulating all kinds of media in documents.

Scalability. Apple's Opendoc Architecture human interface addresses a wide range of users, from novices to experts. No class of user has to understand the additional functionality typically used at the next level—novices can create compound documents easily, while experts can experience nearly unlimited potential.

"Plug-and-play" solutions. With Apple's Opendoc Architecture, vendors will be able to assemble collections of parts into solution sets that target specific tasks or work styles, such as a legal publishing bundle, allowing users to choose the solutions most appropriate to their needs.

Access to object services. Apple's Opendoc Architecture will be based on the industry-standard CORBA (Common Object Request Broker Architecture) object technology. This will allow users to take advantage of a wide range of distributed services working consistently across many different CORBA-compliant desktop and server systems.

To more clearly understand the benefits Apple's Opendoc Architecture provides to the user, let's examine a usage scenario that most computer users can relate to—the use of text editors:

Most applications today—whether word processors, spreadsheets, databases, or more specialized tools—include text editing capabilities. Some rely heavily on text editing, such as WordPerfect or AmiPro. Some use text only incidentally, such as WordPerfect Office or Lotus 123G. Unfortunately for the user, all differ in their details. For example, some allow text styles, some don't. Some offer tab stops, some don't, and so on. In addition to such details, they also differ in functionality and mode of operation. The result? If you use six different programs, you need to learn six different ways to do the same task: edit text.

With Apple's Opendoc Architecture, a text paragraph becomes a software module, usable wherever text is needed. You could take your favorite text editor and use it whenever you needed to work with text. Applications built in this way would have an important characteristic: they would simultaneously be simpler and more powerful. They would be simpler because you would have to learn only one way to do something. And they would be more powerful because you always have the option of choosing a fully functional module, replacing less capable ones.

Benefits to the Developer

Apple's Opendoc Architecture also offers developers a number of important benefits:

Faster, more efficient development. Software developers can reuse already developed parts, eliminating the need to start from scratch with each development effort. This ability to reuse existing parts also means that developers need spend less time on parts that are peripheral to their main area of expertise.

Reduction of application complexity. Because Apple's Opendoc Architecture parts are independent of one another, a collection of parts that is less complex than the large, monolithic applications of today can offer equivalent functionality.

Diminished costs of software development. The fact that parts are smaller than applications makes them both quicker and cheaper to write, which reduces the penalties for failure.

Industry-standard object mechanism. Because parts can use SOM, a CORBA-compliant object mechanism, they can be written in a wide range of programming languages and development will be supported by many tool vendors. This mechanism gives developers high performance coupled with great flexibility in the use of "plug-and-play" objects.

A developer usage scenario:

Today, a developer who wants to create even a highly specialized tool such as a three-dimensional modeling and rendering program has to spend time developing a text editor, so that users can enter specifications and label their diagrams. But, as a developer of such a program, you would much rather concentrate on your specific area of expertise.

In the OpenDoc world, you would have two choices for providing functions that fall outside the primary focus of your application: bundle existing parts with your product or rely on the user to provide parts to support those functions. Either way, you are free to concentrate on the competitive value of your product, rather than on peripheral issues. The result is a much more useful, powerful program—developed in a fraction of the time it would have taken using the conventional, monolithic approach.

The Technology

Apple's Opendoc Architecture is designed to enable the construction of compound, collaborative, and customizable documents, which are interoperable across platforms and with other compound document architectures such as Microsoft's OLE 2.0. It will be an open architecture, with source code available to vendors who want to implement the architecture in their products. Apple's Opendoc Architecture is also flexible, providing replaceable facilities so platform vendors can implement their desired feature set.

Major concepts of the architecture include the following:

Documents

Apple's Opendoc Architecture fundamentally changes the meaning of the term document. In today's computing environment, a document has a type, which is tied to the application that will let the user view, edit, and print its content. With Apple's Opendoc Architecture, a document is no longer a single block of content bound to a single application, but is instead composed of smaller blocks of content, or parts.

Parts

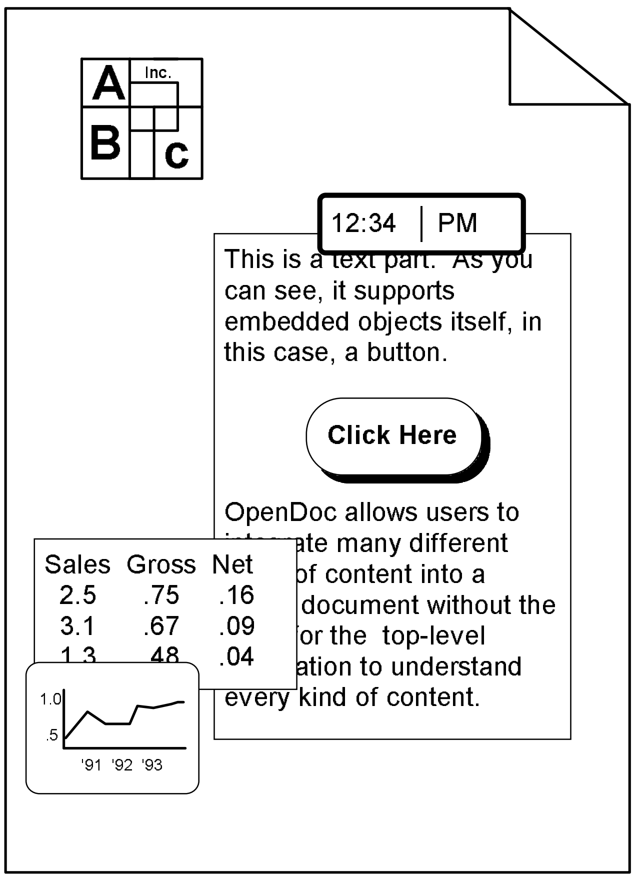
Parts are the fundamental building blocks in Apple's Opendoc Architecture, replacing today's monolithic applications with smaller units of content dynamically bound with related functionality. OpenDoc parts may be viewed in a number of ways:

o *Content containers.* Every part contains data—for example, text parts contain characters, graphics parts contain lines and shapes, spreadsheet parts contain spreadsheet cells with formulas, and video parts contain digitized video. The particular type of data that each part contains is defined by the developer and is known as the part's *intrinsic content.*

In addition to its intrinsic content, a part may be able to contain other parts. Every document has a single part at its top level, the root part, into which all other parts are embedded. Again, the part developer determines whether to support the capacity to contain other parts; however, a key characteristic of Apple's Opendoc Architecture is that if a part can contain one type of part, it can contain all types of parts. This is in stark contrast to the small number of standard data types supported today, such as text, JPEG and TIFF.

o *Part editors.* Part editors are independent programs that manipulate and display a particular kind of content. OpenDoc part editors can serve as components for solution building as well as document building. Plug-and-play solutions assembled from several parts will replace today's monolithic applications.

OpenDoc parts will allow developers to create new applications in a manner similar to that of constructing a document template in today's world.



o *Frames.* Parts can also be viewed as the boundaries at which one kind of content in a document ends and another begins. A key element of the concept of parts is that each part of a document has its own content model—the model of objects and operations that is presented to the user. The content model changes at the frame between parts.

For example, the compound document shown above has as its root part a PM-style graphics editor that provides a letterhead template. On the left side, a graph part from one vendor overlaps with a table part from another vendor to illustrate financial information.
A clock part has been embedded in the top right corner over a text part that contains a button part.

Notice that the two shapes in the upper left corner are not embedded parts at all, but were created by this specific PM-style graphics editor and are called content objects—data elements native to the part. An embedded part is fundamentally distinct from ordinary content elements such as simple shapes, characters, or cells.

o *Frames.* Frames within Apple's Opendoc Architecture are areas of the display that represents a part. Frames provide a handle onto parts, allowing them to be manipulated as a whole, as well as allowing the user to see and edit a part's contents.

Although this description of a frame makes it sound much like a standard application window, it is not. Where windows are transitory views, only visible when the part is being edited or its content viewed, a frame is persistent. When a frame is opened into a window, the frame remains visible. When the window is closed, the part returns to the representation from which it was opened. In addition, a frame can often show only a portion of the entire content of a part. Opening a large part into a window allows its the entire part to be viewed and edited.

Part handlers. In Apple's Opendoc Architecture , part handlers are the rough equivalents of today's applications. When a part is being displayed or edited, a part handler is invoked to perform those tasks. A part handler is responsible for the following things:

oDisplaying the part both on the screen and for printing purposes.

oEditing the part. The part handler must accept events and change the state of the part so that the user can edit and script the part.

oStorage management (both persistent and runtime) for the part. The part handler must be able to read the part from persistent storage into main memory, manage the runtime storage associated with the part, and write the part back out to persistent storage.

Part handlers are dynamically linked into the runtime world of the document, based on the part types that appear in the document. Because any sort of part might appear in any document at any time, the part handlers must be dynamically linked to provide a smooth user experience.

Part handlers can be divided into two types, editors and viewers:

oAn editor displays a part's content and provides a user interface for modifying that content. This user interface may include menus, controls, tool palettes, rulers, and other modes of interaction.

oA viewer offers a subset of an editor's functionality; it allows users to display and print a part's content, but not to edit it. Viewers can be useful in two document sharing situations: when the recipient of a document does not hold a license to some of the kind of parts included in the document, or when the person sending the document does not want the recipient to alter it.

Both editors and viewers can interpret the contents of the part and display that content for the user. The idea is that, eventually, developers will create both kinds of handler for every part. The editor would be sold, but the viewer would be freely distributed, to enable and encourage document interchange.

Storage. Storage is a major feature of Apple's Opendoc Architecture. The existence of multipart documents necessitates a persistent storage mechanism that enables multiple part handlers to share a single document file effectively.

Apple's Opendoc Architecture storage model, based on Apple's Bento standard, assumes that the storage system can effectively give each part its own data stream—an individual area inside document files specific to part content—and reliable references can be made from one stream to another, enabling parts to be integrated into a single document. Because Apple's Opendoc Architecture is designed to support cross-platform capabilities, it must also support the ability to write multiple representations of a given part. Because many pieces of code may need to access a given part, the storage system must support a robust annotation mechanism—one that allows information to be associated with a part without disturbing its format.

The storage system also yields another advantage: collaborative access. Apple's Opendoc Architecture provides an architecture that allows developers to create part handlers that let users collaborate on document creation.

Object mechanism. Calling between objects is fundamental to Apple's OpenDoc Architecture . Apple's OpenDoc Architecture will use IBM's System Object Manager (SOM) as its object calling mechanism. SOM provides an efficient, flexible binary standard for object interfaces that conforms to the CORBA industry standard for distributed object messaging.

SOM lets developers create parts in a wide range of languages and lets these parts call each other with no additional effort. SOM also allows developers and users to take advantage of distributed services provided through CORBA-compliant application programming interfaces (APIs.) SOM and CORBA are widely accepted and are well supported by tool and system vendors on many platforms.

The Competition

In fact, we are also far from alone in our efforts to define and implement Apple's OpenDoc Architecture . Apple’s experience with human computer interaction, WordPerfect’s experience in document-centric computing and compound documents, and Novell’s expertise in distributed, collaborative systems are all playing a part in the definition and implementation of this technology. In addition, IBM's DSOM technology will make it possible to use parts in a networked environment following the CORBA standard. We expect support from many other companies for Apple's OpenDoc Architecture . You can expect to see more announcements in the future regarding companies that will be helping implement Apple's OpenDoc Architecture on additional platforms and from software vendors who will be supporting OpenDoc APIs in their applications.

In contrast, the other major effort along these lines, Microsoft’s OLE 2.0, is a proprietary approach. Currently, the OLE 2.0 source code is held by Microsoft, and provided only under Microsoft license. However, Apple's OpenDoc Architecture will be interoperable with OLE 2.0, so developers can take advantage of Apple's OpenDoc Architecture’s broader feature set and additional supported platforms without sacrificing OLE support.

Part of the work the WordPerfect and Novell OpenDoc teams will be doing is an implementation of Apple's OpenDoc Architecture on Windows in such a way that it will interoperate with OLE 2.0. Applications that also support the additional capabilities represented in Apple's OpenDoc Architecture will have the ability to interoperate at a higher level of functionality on Windows, OS/2 or Macintosh, and to interoperate across multiple platforms and distributed systems as well.

Apple, IBM, and Taligent plan to design complete interoperability between Apple's OpenDoc Architecture and Taligent, similar to the interoperability between OpenDoc and OLE 2.0. This is intended to enable developers and customers to migrate to Taligent without losing their investment in software or legacy information.

In addition to being much more open, Apple's OpenDoc Architecture offers a number of specific advantages to users and developers:

Human interface. The OpenDoc human interface reflects a user-centered approach to product design. User testing indicates that Apple's OpenDoc Architecture is a more comfortable, efficient model for users.

Cleanliness and simplicity of API. Apple's OpenDoc Architecture is designed from a minimalist viewpoint; it defines as few as possible, flexible features, with available source code and clean application programming interface.

Network readiness. OpenDoc handles networking support easily and flexibly. Through IBM's System Object Model and the CORBA industry standard, it will provide access to distributed object services. This aspect of its architecture has been thoroughly reviewed
by vendors concerned with distributed systems, such as Novell.

Range of platforms. Apple's OpenDoc Architecture will be initially released on OS/2, the Macintosh and Windows platforms. It is designed to be highly portable and will become available on other desktop systems, such as UNIX. Apple's OpenDoc Architecture is also designed with the future in mind: OpenDoc parts and documents are designed to be interoperable with other compound document systems, such as Taligent.

Consistent object model. Apple's OpenDoc Architecture's object model is compliant with the Common Object Request Broker (CORBA) specifications released by the Object Management Group (OMG). OMG is a standards body that has defined CORBA as a common, industry wide specification for access to object services. Because Apple's OpenDoc Architecture is CORBA-compliant, OpenDoc documents and parts will be able to use CORBA-compliant services and interoperate with other CORBA-compliant architectures.

The Shape of the Future

The benefits of widespread adoption of Apple's OpenDoc Architecture are revolutionary for both users and developers. Yet it will be a gentle revolution—one without dramatic upheavals. Because Apple's OpenDoc Architecture is designed to work with existing applications and documents, users will benefits from OpenDoc features without disruption. OpenDoc parts will behave much like current applications, enabling customers to upgrade without having to go through a new learning process. As customers become more familiar with the full scope of OpenDoc capabilities—in-place editing, plug-and-play customizing, and more—they will build new, convenient, efficient work environments while maintaining their current computer-software investment.

This is particularly true for users, who will at first experience little or no change, as developers begin to take advantage of the new architecture by binding together parts to form units that are virtually indistinguishable from the applications of today. It is only over time that the full benefits that OpenDoc brings—in-place editing within compound documents, the ability to customize applications by adding or substituting parts, and more—will become obvious.

For developers, the impact will be felt more rapidly. But here, too, the change will not be sudden or disruptive in nature. Developers will simply find that Apple's OpenDoc Architecture empowers them to work in new ways, enabling them both to concentrate on existing areas of expertise and to develop new ones. Apple's OpenDoc Architecture will also encourage the vertical market industry, with applications increasingly targeting specialized tasks.

The primary concern for each of these companies remains the experience of the user—not technology for the sake of technology. OpenDoc represents that continuing commitment: to empower people through the provision of technology that is both powerful and easy to use, helping them to take full advantage of the growing capabilities of computing.

Apple Computer, Inc. 20525 Mariani Avenue Cupertino, CA 95014