M A R C H 1 9 9 4

Object Linking and Embedding 2.0 Questions and Answers

- **What about the IBM System Object Model (SOM) and the IBM Distributed System Object Model (DSOM)? Are they technically superior to the OLE 2.0 architecture?**

No. In fact, SOM and DSOM do not address many critical issues that are successfully addressed by OLE 2.0 and the Component Object Model. Some of these areas are listed in the accompanying comparison of OLE 2.0, OpenDoc, SOM and DSOM technologies. Unlike OLE, SOM and DSOM are an incomplete object solution. For instance, neither SOM or DSOM has built-in support for compound documents, a key customer requirement. To get these capabilities, IBM is relying on the OpenDoc consortia, through which multiple vendors are contributing source code and development time to produce a more complete specification. SOM also has some serious architectural limitations as a system object model, and does not compare to the more advanced architecture of OLE 2.0 and the underlying Component Object Model (COM). For instance, to achieve distributed object support, a different object model, the Distributed System Object Model (DSOM), is required. DSOM requires source code and binary modifications to SOM objects. Distributed object support for OLE will require no modifications to today's OLE 2.0-enabled applications.

SOM objects can only be combined within a single address space. Any object being used by an application can crash the entire application. Also, multiple applications cannot share the same SOM object. OLE gives objects the capability to be combined in the same or separate address spaces, as required. This protects applications from crashing when an individual object crashes; and it also allows multiple applications to share objects. DSOM does allow objects to be combined across different address spaces, and thus be shared between multiple applications. Unlike OLE, however, DSOM does not provide built-in software coordination for cross-application object interoperability. This coordination must be coded by individual programmers, effectively eliminating any possibility of integrating packaged software using DSOM, since different companies may implement different coordination models. This is a severe limitation. With OLE, objects in different address spaces are automatically coordinated by the OLE system software, and therefore OLE-enabled applications can be seamlessly integrated out-of-the-box.

Furthermore, SOM and DSOM are technologies that straddle object-oriented programming technology and true object-enabling system software technology. Because of this, neither meet the need for a robust, system software object model. For instance, the SOM/DSOM model allows objects to inherit source-code implementations from other objects through uncontrolled class hierarchies. While this arguably can make developing objects faster, it does so while sacrificing system robustness (see *Microsoft's Object Technology Strategy: Applications Without Limits*, Appendix B).

Moreover SOM and DSOM identify objects with simple names, not globally unique identifiers. This will lead to naming conflicts in systems with many objects, and any system with objects supplied by different vendors, because object names cannot be properly coordinated. OLE uses globally unique identifiers (GUIDs) to *guarantee* no naming conflicts. Furthermore, unlike OLE 2.0, SOM and DSOM lack a logical thread model to prevent object deadlocks, lack a security model, lack robust object versioning control, and suffer from other limitations outlined

in the accompanying technical comparison. All of the SOM limitations will also be limitations in the OpenDoc architecture, since this architecture will draw on SOM for its object model.

Today, SOM provides an object model that integrates with the IBM® OS/2® WorkPlace Shell™. However, the OS/2 WorkPlace Shell has only a few native applications, while OLE 2.0 provides object integration potential for the almost 10,000 native Windows applications. There are today many OLE 2.0-enabled applications available for the Windows platform, with hundreds more that are soon to be released. In addition, OLE 2.0 will soon be available for the Apple Macintosh platform, and support for multiple versions of UNIX and other platforms is planned. While IBM has announced plans to develop SOM support for the Windows platform, no Windows implementation has been made available.

As a technology integrated with the IBM OS/2 WorkPlace Shell, SOM will not provide applications seamless integration with the object-based innovations that Microsoft is building into the Microsoft Windows Family of operating systems. These innovations are based on OLE 2.0 and the more robust COM architecture, and are aimed at providing superior integration between applications and object-based system services, such as drag and drop to the desktop, integrated OLE Custom Controls, object interfaces to distributed system services, and many other important innovations.

• Isn't OpenDoc superior to OLE 2.0?

Absolutely not. OpenDoc is a proposal for a compound document architecture that will be based on separate technologies supplied by Apple and IBM. For instance, Apple will supply core OpenDoc compound document features; Bento, a file system; and its Open Scripting Architecture to give objects a cross-application scripting ability similar to OLE Automation. IBM at some time in the future will supply its System Object Model (SOM). However, key parts of the development effort have been split not only between IBM and Apple, but also among other partners in the OpenDoc consortia including WordPerfect and Borland. It remains to be seen if these different technologies can be gracefully combined and supported by many different vendors in a coherent, customer-focused manner. Unlike OLE 2.0, preliminary specifications were not distributed to all major software vendors for open industry review (preliminary OLE 2.0 specifications were reviewed by over 150 different ISVs).

Perhaps most importantly, OpenDoc requires that application vendors create, distribute and maintain two separate versions of their programs. One runs in standalone mode, while the other is a series of special "parts" that can be loaded into the address space of a container application to create a single, monolithic compound document application. This raises questions about the costs and eventual availability of OpenDoc applications. OLE, on the other hand, enables the seamless integration of shrink-wrapped software, and does not require software vendors to supply special "parts." Vendors simply write their applications to be OLE-enabled, and then they are ready to be integrated out-of-the-box with any other OLE-enabled application on the market. This is a much more efficient and practical model which has already been adopted by hundreds of major software vendors.

Furthermore, OpenDoc is at best a paper specification that has not been publicly released for any platform, and has only been demonstrated for the Macintosh® computer. Even so, a close look at the separate technologies on which it is based reveal some serious shortcomings in the OpenDoc architecture. Many of these are specifically identified in the accompanying technical comparison of OLE 2.0, OpenDoc and IBM SOM/DSOM. OpenDoc does not attempt to address several important issues, such as distributed object support. OLE with distributed object support has been distributed to over 5,000 developers in pre-release form. Many other areas of architectural weakness stem from weaknesses that will be inherited from the IBM SOM

architecture. For example, since all objects in a single OpenDoc document must execute in the same address space, any single object can crash/corrupt the entire document; and multiple applications cannot share the same object. OLE gives objects the capability to run in separate address spaces, protecting compound documents from crashing when an object crashes; and also allowing multiple applications to share objects.

To understand many of the other limitations of OpenDoc, read the accompanying comparison of these technologies, titled *Object Linking and Embedding 2.0, OpenDoc and SOM/DSOM: A Comparison of Technologies*. Also, read the previous question and answer that address IBM SOM. A recent PC Week article summed up the OpenDoc architecture as follows:

> **"Shortcuts taken by Apple will result in the re-design of aspects of OpenDoc. The documentation is unclear as to when these changes will occur, specifying only that 'later in the development cycle' the hierarchy will be replaced by one based on IBM's SOM."**
> ("First OpenDoc Spec Raises Many Questions" *PC Week*, January 10, 1994)

Microsoft is firmly committed to OLE and the Component Object Model. These technologies are more advanced than the technologies on which OpenDoc is based, and were refined in an Open Process in which major software vendors participated in open design reviews starting as early as January, 1992. These vendors included Apple Computer, Claris Corp., Lotus Development Corp., WordPerfect Corp. and Borland International, and many others. Preliminary specifications for OLE 2.0 were also distributed to over 150 other software vendors for further feedback. OLE is a proven technology that is available with many shipping applications, with hundreds more on the way. OLE is being implemented on the Apple Macintosh (it is in beta testing now), and the technology is being made available on many UNIX systems.

Finally, OLE 2.0 and the Component Object Model are the foundation for Microsoft's future releases of the Windows Family of operating systems. Applications built using OLE 2.0 technology today will be ready to seamlessly integrate with the system-level OLE support in these coming releases of Windows. This means existing OLE applications not only give benefits today-- they are "ready-made objects" that will provide seamless integration with the most powerful object-based system environments on the horizon: the next generation of Windows ("Chicago", targeted for late 1994); and the next generation of Windows NT ("Cairo", targeted for the middle of 1995).

• What about Taligent? Isn't it a powerful, object oriented operating system?

Taligent was started as a joint effort between Apple and IBM to produce a fully object-oriented operating system that would eventually replace IBM OS/2 and the Apple's System 7. Since it was formed, however, its mission has been redefined several times. While a complete OS is still planned, today Taligent is focused on creating application frameworks that will allow applications to tap into source-code object classes. Application frameworks exist for the purpose of simplifying software development. Instead of having to write source code that implements low-level APIs that are native to an operating system, frameworks "wrap" such lower-level APIs into higher levels of abstraction. Further, by encapsulating much of the primitive functionality into reusable classes, a developer is free to write less, more specialized code. Application frameworks, such as Taligent, are thus class libraries.

Today, companies like Microsoft, Borland, Symantec, and others provide such frameworks to ease software development. In fact, the Microsoft Foundation Classes (MFC) is the most popular object-oriented framework available, and can be used to more easily build component objects with full OLE 2.0 capabilities. Many of these frameworks, including MFC, have been refined over many years. How Taligent will add value as a new and competing framework is unclear. Also left unanswered is the question of how the Taligent Frameworks will support Apple Macintosh, Windows and OS/2-based applications, and at the same time be integrated into future IBM operating systems such as the WorkPlace OS. Today, even the Taligent Application Frameworks have not been commercially released, and according to Taligent, will likely not be released in their first shipping version until late 1994 or 1995.

- **What about CORBA and the Object Management Group (OMG)? Everyone but Microsoft seems to support this open industry standard. Why doesn't Microsoft support CORBA?**

The Common Object Request Broker Architecture (CORBA) is a specification, not a product. By OMG's own admission, the CORBA specification is very limited. In fact, no two CORBA compliant products interoperate with even basic CORBA services. The OMG continues to refine its specification, and it plans to release a more complete specification, but this will probably not be available until late 1995 or later. If this future version of CORBA eliminates the functional deficiencies in the current specification, and customers determine that the specification meets unfulfilled needs, then Microsoft will help build CORBA interoperability into OLE and COM. Today, however, Microsoft is not saying OLE and COM are CORBA-compliant, because this claim is misleading to customers. The CORBA specification is:

- **Functionally deficient** - CORBA defines so few critical object model functions that it cannot be used without significant proprietary extensions. In fact it defines only about 15% of a complete, workable system object model.

- **Unable to provide basic object interoperability**. No two shipping products that claim CORBA-compliance can interoperate, unless specific, proprietary and non-CORBA extensions are made to achieve interoperability. Because of these limitations, and the fact that there is no certification process, "CORBA-compliance" is undefined.

Some vendors have used the OMG CORBA label, however, to create the illusion of interoperability, and this has lead to significant confusion. Microsoft has taken a more pragmatic approach, and is focused on delivering value to customers *today,* through OLE 2.0 and the Component Object Model.

Microsoft is committed to delivering open systems, and has laid out a strategy to continue to do so (see 'Open Systems: Technology Leadership and Collaboration, part number 098-55058). OLE and the underlying Component Object Model (COM) are open, proven technologies. OLE 2.0 was refined through an Open Process. Almost every major software vendor participated in the open design reviews as early as January, 1992, including Apple Computer, Claris Corp., Borland International, Lotus Development Corp., WordPerfect Corp. and many others. Additionally, preliminary OLE 2.0 specifications were distributed for review to over 150 other ISVs. Today, the OLE specification is fully published, is being made available on all major platforms. There are no license fees for distributing OLE 2.0-enabled applications, and as an open specification, OLE provides a platform for broad industry participation and innovation.

OLE 2.0 for the Apple Macintosh platform is in beta testing, and will soon be available. OLE 2.0 applications will be able to interoperate between the Microsoft Windows and Apple System 7 platforms. Microsoft has also signed an agreement with Digital Equipment Corporation to build OLE/COM support into Digital's ObjectBroker™ product. This agreement will lead to OLE interoperability with multiple operating system platforms, including SunOS™, IBM AIX®, DEC® OpenVMS, DEC ULTRIX® and OSF/1. Support for even more platforms is planned for the future. Unlike CORBA, OLE-enabled component objects are *guaranteed* to interoperate with other OLE-enabled component objects, because OLE 2.0 is a fully functional, object-enabling system software based on the architecturally robust Component Object Model.

The Microsoft object strategy also allows for open, industry innovation and enhancements to the architecture. The OLE/COM strategy provides corporations, system integrators and software vendors a platform on which to build new and exciting applications which achieve a level of interoperability that no other object platform can match.

It is important to understand that a key part of the Microsoft open systems strategy is to incorporate market-driven standards in a customer-focused manner. A good example is Microsoft's adoption of TCP/IP, a proven, market-accepted (*de facto*) networking standard. CORBA is not a market-accepted standard-- rather it is a committee-based (*de jure)* specification that is far from complete. John H. Rymer, editor of the Distributed Computing Monitor, states:

> "**Users have comparatively little say when technology is conceived and molded by a standards body. Our fundamental belief is that markets are the best way for a group of vendors and customers to determine the direction of a technology.**"
> (*Distributed Computing Monitor*, January, 1994, Patricia Seybold Group)

OLE is a technology that was refined by input from hundreds of software vendors, and was fundamentally architected to meet customer requirements *today*, as well as tomorrow. The success of OLE can be concretely measured. OLE has broad industry acceptance: hundreds of vendors are incorporating OLE 2.0 support into their applications and development tools. OLE has won numerous industry awards including the **Technology Award for Excellence**, *BYTE Magazine*; the **Technical Excellence Award**, *PC Magazine*; and the **MVP: Software Innovation Award**, *PC-Computing.* And OLE has been endorsed by the most important player of all: customers, through the widespread purchase of software that is today fully OLE 2.0-enabled. Today, over 1.5 million component objects with OLE 2.0 capabilities are being used by customers across all major industries.

- **What about Novell's Appware product, how does OLE fit in?**

The Novell® Appware product, which is still very much under development, is a mix of several technologies that were developed by different companies, and then acquired by Novell. Appware is an attempt to combine these technologies in order to produce a "virtual API". Other than the fact that the Appware provides a set of C++ classes, it does not provide any object-enabling system software. In fact, Novell has publicly stated that Appware will support OLE 2.0, as well as OpenDoc.

As a virtual API, Appware's goal is to provide a layer of software that will make applications portable between Windows, Macintosh, OS/2 and UNIX. It is important to understand, however, that the Windows API (Win32) is becoming a portable API. It will soon be available for UNIX through Insignia Solutions, and it can be freely licensed for other platforms as well. Microsoft is also making the Win32 API available for the Macintosh, providing source-code application portability between Windows and Macintosh implementations. And by writing to the Win32 API, independent software vendors are guaranteed that applications will run with full native performance on the highest-volume desktop platform - Windows.

Using a virtual API supplied by a third-party such as Novell, on the other hand, has some limitations. Such APIs have been available for many years (such as XVT and Galaxy), but they have met with limited success. There are several reasons. First, any virtual layer of software must make sacrifices in the features it supports for each different operating system in order to provide complete portability. Any feature that is not fully supported across all operating systems (and there are many), are usually dropped. The "lowest common denominator" approach of third-party virtual APIs lead to disappointing applications that offer no competitive advantage. In addition, virtual APIs must translate calls to the native operating system, and are typically much slower than native implementations.

But by using the Win32 API and the hundreds of tools and applications available that use this API, applications will run with full, native performance on the most popular, highest volume desktop platform -- the Windows Family. Furthermore, since Windows provides so many advanced enabling technologies such as extensive multimedia support, TrueType fonts, OLE 2.0 support and many more, no sacrifices in application features need be made when using the Win32 API as a cross-platform technology.

Many corporations are reluctant to deploy applications based on a third-party API such as Appware, because this makes the corporation completely dependent on the provider of the virtual API to provide technological innovation. For example, as Microsoft and other industry leaders introduce software innovations to the Windows Family, these innovations must be incorporated into the virtual API by the API provider. This occurs (at best) with significant lag time, while the innovations will be immediately available to applications written to the cross-platform Win32 API. Or worse yet, the innovation might not be incorporated into the virtual API at all, because it might make the virtual API non-portable. Thus, while adding some value for less critical applications, third-party virtual APIs can present real dangers to corporations which want to avoid getting trapped into the API, and locked out of the cycle of rapid industry innovation. In today's competitive, global economy, it is not surprising that third-party virtual APIs have not been widely accepted. The good news is, the Win32 API will provide the highest level of cross-platform support, without making the sacrifices of a technology such as Appware.

- **What about other object models, such as Hewlett-Packard® DOMF (Distributed Object Management Facility), Sun DOE (Distributed Object Everywhere), and NeXT™ NextStep?**

These technologies may someday offer value for corporations adopting a UNIX-only strategy. However, only NeXTStep is available today, and none of these systems successfully address the need for an open, robust object model for the mainstream corporate desktop, nor do they interoperate with each other. These models, unlike OLE 2.0, do not provide a standard for object interoperability in the high-volume, packaged software market. They do not enable the critical integration between desktop applications and enterprise back-end services. To better understand how these technologies compare with OLE, see the accompanying table "Object Technology Strategies: How They Compare."