COMPONENT SOFTWARE

Phil has decided to leave ecotourism and enter the desktop software business. He has a few friends in venture capital, an idea he thinks will sell and a plan to get his product good exposure (he knows someone who writes a newsletter). He dusts off the business plan he wrote in 1988 when he last had this crazy impulse, looks it over and shakes his head. He invites some of his programmer friends over for sushi and sake, and together they consider which platforms to support and what the software should look like. This is definitely not 1988.

For example, Zoe points out, corporate software buyers don't often shop for individual software packages anymore. Now they make platform decisions based on vendor integration strategies that involve e-mail, suites and middleware such as Lotus Notes -- or at least the buyers try to. They are keenly aware of what happens when each department or branch office selects a different e-mail or office-automation package.

There has also been a price war. Suites, spreadsheets and databases are cheap, but they're no smaller or less complex than the old software. The combination of low prices and high complexity deters new entrants into horizontal tools or into suites/middleware: They would not only have to write, debug and field an entire application (often with its own text editor, communications module and macro language), but also make sure it works with all other packages and platforms, develop a distribution channel, design packaging, create a media plan and so on.

Later, when developers invented new features or data types, they would need to rework the software to take advantage. In fact, many features languish from a lack of supporting applications, and users are frustrated because their favorite applications don't support new features quickly enough. Developers constantly make tough decisions about which features to support first. Each new feature they do support increases the complexity of their code and lengthens the test and delivery cycles, canceling efficiencies they may be getting elsewhere. In the end, a company's great idea may have been a compelling feature, but it seldom makes a killer standalone application.

Phil is having second thoughts.

But at least, he thinks, choosing an OS won't be that difficult. After all, it looks as if the battle for

JOIN A SOFTWARE GUILD
the desktop is over. Microsoft Windows is everywhere, Unix hasn’t made sig-
ificant inroads (on the desktop), and the Mac is still a niche machine,
despite its relative simplicity, built-in multimedia capabilities and fall-
ing prices. The clincher, Phil figures, is that most desktop-software de-
velopers now create new applications for Windows first because it offers the
largest audience of potential buyers.

But wait. None of the major software platforms will be the same a year from
now. Microsoft wants to migrate its customers to Chicago (the code name for
what will likely be called Windows 4, which will at long last unify DOS and
Windows) and Cairo, its server counterpart scheduled for 1995. Apple is
committed to the RISC PowerPC with a new OS, as is IBM. To show real per-
formance improvements, Apple needs ISVs to develop or port applications so
they run natively on the PowerPC. Novell just bought WordPerfect and is
pursuing a multi-platform application strategy with AppWare.

Transitions open opportunities. The platform vendors know this and want to
make their changes as smooth as possible for developers as well as end-
users. Although the current model is defective, developers and users still
need an incentive to offset the pain of changing. The platform vendors’
 lure is component software, which stands to transform the industry and ad-
dress (if not resolve) the problems Phil encountered above — if it works as
promised, if it is widely used, if it provides a profitable economic model
for third parties....

Object infrastructures

Several object models (and the attendant tools and environments) are
wrestling for share of mind and market, including Sun’s Distributed
Objects Everywhere (DOE), HP’s Distributed Object Management Facility
(DOMF), NeXT’s Portable Distributed Objects (PDO), Novell’s AppWare
Bus, and two that we cover in this issue of Release 1.0: Microsoft’s
Compound Object Model (COM; page 15) and IBM’s System Object Model
and its Distributed cousin (SOM and DSOM, used in OpenDoc, page 19).
All but one of the vendors are moving toward portability and inter-
operability through the Object Management Group’s Object Request
Broker (ORB) specification, which requires that they each map to a
language-neutral Interface Definition Language.

Microsoft has taken its own path with COM, which does not comply with
the Common Object Request Broker Architecture (CORBA). It expects to
achieve some degree of interoperability with the others through an
arrangement with DEC, which has a compliant ORB called the Object-
Broker that it will link to COM and call the Common Object Model.

For some time, the future of object markets seemed to hinge on this
battle between Microsoft and the OMG. Taligent’s efforts encompass
both these architectures, but it is distant enough to be out of the
fray right now. Lately, however, OpenDoc’s potential to change the
desktop software market has put it squarely in contention with Micro-
soft. As the OMG’s president and CEO Chris Stone says, "From the
perspective of desktop politics, this takes the heat off me." Now
the heat is on OpenDoc.
The movement to deliver component software to desktop users has focused on two potential platforms: Microsoft’s OLE 2.0, and OpenDoc, which is supported by most of the other major software platform vendors. Although the two efforts compete for developer attention and are structurally different, they are not mutually exclusive. Some companies are working on both (despite Microsoft’s efforts to force exclusive loyalty to itself), and OpenDoc’s backers intend to provide seamless and complete OLE interoperability.

This issue of Release 1.0 analyzes the emergence of component software through the current tussle between OLE 2.0 and OpenDoc for share of developers’ minds and budgets. (For OLE and OpenDoc descriptions see page 11.)

In this cornah, in the green trunks...

Microsoft’s migration to Chicago and Cairo depends critically on the success of OLE 2.0, the first major revision of Microsoft’s Object Linking and Embedding architecture. OLE 2.0 goes far beyond 1.0 and offers a generalized way to develop and deliver component software. Much of it has been available (in a 16-bit implementation) for a year on Windows 3.1; a Macintosh version is promised, though with no delivery date. OLE 2.0 won’t be available for OS/2 or Unix very soon, if at all.

Roughly 50 Windows applications use OLE 2.0 today, although developers are struggling mightily with its complexity. It’s not uncommon to hear of a vendor that put programmers to work on it a year ago but has yet to deliver an application that uses it. Many are still in wait-and-see mode. But Microsoft has less need of a lure than all the OpenDoc vendors, since it can dictate its own software direction, and it is likely that software developers will continue supporting the latest version of Windows in any event.

...and in the tasseled rainbow trunks...

OpenDoc is a collaborative effort to create a new component-based application and document platform that uses several market-tested technologies developed at Apple and IBM. The principal players in OpenDoc right now are Apple, IBM and Novell/WordPerfect; other participants include Borland, Lotus, Oracle, Taligent and Xerox.

OpenDoc promises cross-platform support for software components, a compelling promise. For example, the business proposition to a Macintosh software

1 Object linking stores a presentation version of the object locally and points to the actual data and to the application that created it, either of which can be local or across the network. If you don’t have access to the application or the data, you can only view the object. Object embedding includes both the presentation version and the actual data, as well as a pointer to the proper application. That means the object can be shipped to others as a single unit, without regard to breaking network connections to remote objects. What’s lost with embedding is the updating of objects.

2 Apple contributed two things: the Open Scripting Architecture (its language-independent scripting system), and its document storage architecture, called Bento. IBM contributed its System Object Model (SOM) and the distributed version, DSOM. OpenDoc participants are now porting these elements across multiple platforms. Full descriptions begin on page 11.

Release 1.0

25 May 1994
developer is that OpenDoc components will have a potential market of very near 100 percent of installed PCs and workstations, as opposed to the 14 percent of desktop systems that Macintosh now represents. Interoperability with OLE is a key element of the OpenDoc strategy: Objects created in OpenDoc should also run in an OLE 2.0 environment, and vice versa, a capability that WordPerfect recently demonstrated (see page 21).

OpenDoc members coordinate their activities through Component Integration Labs, a small organization headed by Jed Harris, who created Bento, one of the OpenDoc technologies, while at Apple. CI Labs plays technical and process roles. CI Labs' first role is to act as a repository for OpenDoc's non-competitive technology elements. It is a vendor-neutral technology supplier to companies that want to develop frameworks and applications. It will also maintain the source code libraries for anyone's inspection. Want to know how some part of the environment works? CI Labs will show you the code. Finally, it will offer validation services.

CI Labs' second role is as a catalyst for consensus on new aspects of the technology. Over time, as software component interaction becomes more organic and less predictable, this role will become more important as a way of coordinating developments among component providers. Given this social, experience-transfer and coordination role, you could call it the Component Guild, rather than merely a laboratory.

Caveat: It's very tough to make all of this really work. With OLE 2.0, Microsoft has a Mac-like platform, while OpenDoc has problems like those of DOS on different platforms.

Interplay and coexistence

It's not clear how the Microsoft and OpenDoc approaches will co-exist. As mentioned above, some interoperability already exists. Nevertheless, developers may have the resources to pursue only one approach: Programming talent is scarce and budgets are thin. The difficulty of using OLE 2.0 cuts both ways. It may cause developers to postpone adopting it, or it could force them to stay once they've made the investment. That's why Microsoft wants to get developers into OLE 2.0 as soon as possible.

If OpenDoc gets off the ground commercially, many end-users may end up with two or more component infrastructures anyway. Say you buy a Power Macintosh and Microsoft Word a year from now: OpenDoc will probably be built into the Mac, and Microsoft expects to deliver an OLE-based version of Word on the Mac. Two years from now users may also run Taligent's frameworks.

Luckily for independent developers, there may be a path that includes both architectures. Vendors see an opportunity to become preferred development-platform suppliers by supporting both OLE and OpenDoc. Microsoft's aggressiveness so far has been a strong motivating factor. As Doug Donzelli, vp of Novell's AppWare group, says, "We can't afford to be six months behind Microsoft all the time."

Critical is timing, both for the base technology components and for the advanced development tools and frameworks that should make them possible to implement. OLE 2.0 has been available for a year, so it would seem that Microsoft has a headstart of a year and a half. However, that headstart
may be somewhat illusory, because the features that bring component capabilities to OLE 2.0, such as custom controls, are just emerging now.

Nonetheless, OpenDoc members are behind. Apple seeded an alpha OpenDoc SDK with one thousand developers only last month; it expects to go to beta in August and release a finished product this fall. Commercial parts from vendors should appear in early 1995. All of the OpenDoc technologies must be ported to each of the target platforms, and tool vendors must support OpenDoc and integrate their development environments.

THE VIRTUES OF COMPONENT SOFTWARE

If Microsoft and OpenDoc deliver, what will happen? Why should users jump in? How much will they be willing to change for what they get? How much will they pay? As we mentioned, customers already pay component prices for monolithic software. But that's the problem: It's monolithic. It may be cheap and vendors may improve the interaction of applications within a suite, but buyers still can't modify the applications to suit their needs (or if they can, it appears to be black magic; see box).

Price Waterhouse couldn't wait

When Price Waterhouse's audit department asked its technology group to help automate its working papers, it wanted the resulting system to replicate the traditional audit process as closely as possible. A system that replicated and automated the stylized way that auditors track and cross-reference information was originally estimated as a $5 million software-development effort and would not have offered state-of-the-art word-processing, spreadsheet and database functions.

The other option, which Price Waterhouse chose, was to work some magic on ordinary commercial applications. The final system, called Price Waterhouse TeamMate and developed over two years for about $1.75 million, incorporates WordPerfect, Borland's Quattro Pro and Paradox, and Watermark imaging software. It goes far beyond custom tick marks and notations, and adds sophisticated features such as spreadsheet hypertext (the ability to drill down through references that cross spreadsheet boundaries) and a function similar to Macintosh System 7's Publish and Subscribe -- all inside Windows.

In addition, all auditors get the full functionality of commercial productivity tools. As Sheldon Laube, national director of information and technology for Price Waterhouse, says, "I can't compete with Bill Gates, and I sure don't want my users to pay the price."

Component software architectures allow us to rethink the nature of applications and how we use them. Horizontal productivity tools become customizable platforms for other, smaller components. Below are some ways that components can change our computing and communicating environment.

__________

For previous analysis of related topics, see Release 1.0, 7-92 (Object Markets), 5-91 (Power Through Scripting) and 3-91 (Object Request Broker).

Release 1.0 25 May 1994
(Remember, this is still a fantasy!)

Use. Instead of running several applications and cut-and-pasting elements into compound documents, users work from active, customizable, compound documents with intuitive interfaces. They can add specific functions and features to their environments (not just to individual applications) quickly, without the pain of upgrades. Want better data-analysis capabilities? Get a multiple-regression component and drag its icon to your toolbar, add its name to your menu or add its stationery to your desktop. It will know when it encounters data it can analyze.

The better the component architectures work, the more fine-grained the component combination. It's one thing to create a spreadsheet window in a word processor; it's quite another to mix your favorite toolbar with another vendor's outliner and yet another's table creator, text engine and spelling checker.

Internal IS departments no longer need to write their own word processors or e-mail clients because the commercial ones aren't embeddable in their custom application suites. If they stay within the component architectures, they can mix commercial components with their own code.

Development. Large application vendors can modularize their applications and break the code monoliths into more manageable parts, for internal efficiencies and external sale. In either case, component re-use allows developers to share large segments of code across multiple applications on multiple platforms. This makes it easier for vendors or third parties to repackaging their software so that its focus shifts from generic to specific tasks (writing letters vs. contracts), and lets them target vertical markets in an economical way. There are benefits for small vendors, too. Companies that have been limited to selling add-in tools and utilities can participate more fully in the marketplace (see box, opposite).

Distribution. Today, applications are too large and the transmission channels are too expensive for electronic distribution to make economic sense. Component software can economically be distributed electronically -- even over some wireless transports. The component architecture also offers speedier upgrades, with instant compatibility. Channels to market change as system integrators and VARs can easily add value to a multi-vendor component suite and target it to their particular market.

Economics. With more shared code, developers spread costs to maintain their own, smaller share. Companies can be small and still be viable in the software industry. Better yet, components create a mechanism to add value periodically and to extend functionality without rewriting all the previous code. This recurring revenue stream can be much more like a subscription. End-users are also more likely to make impulse purchases of very focused add-in functions, such as stock charting or visualization.

----------- End of dream sequence -----------
Beyond TSRs and INITs

DOS and the Macintosh spawned surprisingly fertile small markets around TSRs and INITs (DOS terminate-and-stay-resident programs, and extensions that loaded at initialization time on the Mac). But the TSR and INIT architectures were not designed for all the attention and use they attracted. Now there are better examples of what component software can be like. For example, image-manipulation programs such as Adobe Photoshop and Fractal Design’s Painter are designed to allow for third-party plug-in filters and extensions. One of the most popular plug-ins is HSC Software’s Kai’s Power Tools, which offers a dizzying array of visual effects. Another is Equilibrium’s DeBabelizer, which translates data file formats.

Although it’s possible to think of these as parts, available to any document, they’re really application-specific. Portland startup Extens is building utilities that extend mainstream productivity tools’ capabilities. Today they’re working with applications designed to take extensions, starting with PageMaker for Macintosh this month, and for Windows in the Fall.

The showcase third-party OLE 2.0 application -- or applet, rather -- is Shapeware’s Visio Express, a scaled-down, embeddable version of Visio, a template-based graphics package. Visio Express can’t be run as a standalone application. Instead, it adds an intelligent drawing module to the Microsoft Office suite and other applications that speak OLE Automation. Long run, there will be more peer-to-peer components, versus today’s model of host tools and snap-in modules.

DIFFERENT STRATEGIES

Microsoft and the OpenDoc companies have similar goals, but very different approaches. Most of what we’ve just described about component technology applies to both of them. Where it gets interesting is in how they diverge. (If you’re not at all familiar with OLE 2.0 or OpenDoc, turn to page 11.)

Microsoft’s cut-and-paste approach

Compared to OpenDoc, Microsoft’s OLE 2.0 is more incremental, pragmatic and direct (e.g., components negotiate directly with other components). OLE 2.0 is broader: It touches every part of the Microsoft environment. It’s less structured and explicit, which is consistent with Microsoft’s cultural values, but which makes it frustratingly difficult to decipher. Although Microsoft’s flexible approach leaves room for developers to innovate radically, it’s not clear how those innovations will become useful within the broader web of components. Nevertheless, Microsoft’s degree of control over its software environment improves its chances of achieving component compatibility, versus the inevitable platform problems that OpenDoc faces.

It’s hard to separate OLE 2.0 from the rest of Microsoft’s strategy in this area: to unify all its scripting and macro languages in Visual Basic and embed VB, where possible, in Microsoft applications; to promote and showcase the Office suite; to get developers in OLE 2.0 immediately; and to
listen carefully to stresses in the developer community, then fix high-priority problems as quickly as possible. Microsoft’s promise is that regardless of how complex the effort, it will pay off, because component software is where the future lies, and Microsoft can provide it with OLE 2.0 and Cairo.

Eat Here Now

Before tackling the specifics, allow us to dramatize the contrasts between OpenDoc and OLE 2.0. Think of them as restaurants. At OpenDoc there’s a maitre d’, who makes sure you don’t sit at an occupied table and resolves problems if you get the wrong dish or disagree with the amount of your check. There’s a menu where you can see what’s available, or (if you’re nosy) you can check the list of orders clipped over the range in the kitchen, where you can see everything that’s going on. You place an order with a waiter using commonly understood terms and are likely to leave satisfied. Better still, if you go to a member of the same chain in a different city, you’re likely to be able to order food and enjoy it, but there are limited menu choices (plus those off-the-menu, platform-specific items).

In contrast, the OLE 2.0 restaurant has no menus or maitres d’. At Cafe Ole, any guest could cook for you, or vice versa. There’s a cork board by the door where some occupants have posted things they could cook for you on separate slips of paper. You have to agree with each potential cook what they are to do for you and where to deliver it. If you like the results, you can have a standing order. There’s a chance you can get anything you want in this restaurant, but you may not be able to predict how it gets to you, and it will change as guests/cooks come and go.

Microsoft bets that nobody can agree on the menu -- or that any system sophisticated enough to deal with this problem will fold under its own complexity, especially across platforms, over long distances and with thousands of interacting components.

What was that again?

In case the metaphor just muddied things, here is something more concrete. Take explicitness and directness, for example. With the extensible Open Scripting Architecture (page 18), OpenDoc standardizes the core verbs (events) that applications of different kinds will support. Microsoft started down this path, but it couldn’t get the vendors to agree. In fact, Microsoft states that boundary cases, such as whether the action "select sentence" should include the spaces after the sentence or not, are significant application differentiators and should not be made uniform.

As a result, there are no predefined event semantics or event manager in OLE 2.0. It relies instead on objects that expose their functions and then communicate directly with each other. These objects communicate through rigorously defined interfaces that expose the properties, methods and events that an object can understand or perform. The interfaces must be negotiated between vendors, because objects from multiple vendors must act as "good citizens" in the OLE 2.0 environment. Microsoft believes that a consensus mechanism will emerge from the market; we remain unconvinced.

OpenDoc depends on collaboration and is organized for it. Though the different members have very different agendas, they have agreed to use a

Release 1.0 25 May 1994
framework that is already quite evolved (it began as the AppleEvents technology) and is extensible.

Beyond the desktop

Microsoft's WinPad is likely to be better integrated with OLE 2.0 than Newton will be with OpenDoc. Apple intends to wrap Newton software and data so that it can be an OpenDoc part. Although WinPad will not be a full OLE 2.0 platform, it is explicitly included in the interoperability strategy. For starters, Visual Basic will be an important programming language for WinPad. These integrative details could be significant.

Beyond the desktop in a different direction, both Microsoft and the OpenDoc alliance plan to submit their specifications to the Object Management Group in response to its Common Facilities Request for Information. A less planned but equally powerful angle is emerging as the Internet gains attention: Both OpenDoc and OLE 2.0 could complement some Internet tools in extremely powerful ways (see box, next page).

Microsoft plays hardball

Microsoft is bringing its big guns to bear on OpenDoc. It has determined that OLE 2.0 is crucial technology, and is telling developers it is the only true path to Cairo. Until recently, Microsoft's contracts for developers working on advance releases of Chicago included restrictions that kept them from working on OpenDoc for three years.

Some of Microsoft's efforts are subtler. The company is making Visual Basic its ubiquitous scripting language. This includes VB for Applications (VBA), which it will embed in all its applications, especially the Office suite. Each application's current macro or scripting language is being migrated to VB. Microsoft has not licensed VBA to any other software developers, which means that only Microsoft applications will have it. Any other vendor could add OLE Automation capabilities to its scripting language, which will give it full power to run (or be run by) OLE 2.0 objects.

With Visual Basic pervasive in the Office suite and in other key Microsoft initiatives such as At Work and WinPad, any applications that use another scripting language will be second-class citizens. (All scripting languages in OpenDoc's Open Scripting Architecture are equal.) Also, corporate IT departments want to minimize the number of languages they support. The latest twist is the recent Microsoft Office Compatible campaign.

Who owns the frame?

Microsoft expects its VBA-enabled applications to be the principal container applications. Other companies' applications may run parallel to the Office suite, but with a different scripting language; plug in as components and OLE custom controls; or run in the background as servers. In the last two cases, end-users may never be aware that they are using a separate company's products. This may be a potent motivator for a third-party software developer to adopt OpenDoc or to make a visibility-enhancing deal with Microsoft: Why should they appear as potentially anonymous controls hosted within Microsoft's Office suite?
Piggyback on the Internet!

Here's an exciting (and potentially profitable) idea: Combine the functionality of OpenDoc with the Internet's World Wide Web.⁴ An OpenDoc browser could easily interpret HTML and display it, as Mosaic does now. Publishing an OpenDoc home page on a Web server is simple, too: Each component of an OpenDoc document is scriptable; any script could be a tag to another document. Documents could keep their internal structure, but present HTML to client applications. Non-OpenDoc clients wouldn't know the difference.

The real opportunity is deeper integration. Our unsolicited advice: Offer OpenDoc to the Internet community as an extension to the Web protocols. Since OpenDoc is published and openly available, it might fit well with the Internet culture and standards process. (We'll explore the directions HTML is taking and its relationship to SGML in a future issue.) This would add functionality to Web documents, catch the Internet's momentum and add a compelling sales point to OpenDoc.

Let's go a step further: OpenDoc documents could combine references to Web objects across the Internet with references to objects or processes inside an organization (visible and accessible only to authorized people, of course). This would offer an elegant way to move seamlessly inside and outside an organization, given appropriate security precautions. The research department could create a private report that includes pointers to resources or discussions on the Internet; marketing could create a public document that updates automatically and contains private references to internal documents.

We won't even mention what would happen if someone turbocharged this environment by linking Perl, Safe-Tcl or Telescript (see Release 1.0, 2-94) to OpenDoc's Open Scripting Architecture.

Now Phil's charged up again. He's decided to use OpenDoc to write an online conference browser component that works on the Internet and multiple online services.

The good news for Microsoft is that the OpenDoc group has not addressed the packaging issue any better. So far, in OpenDoc it's not clear which software houses' identities will be visible to end-users and which will be hidden. There are no splash screens, and there's nothing on the menu bar to say whose part you're using, except in the About item.

The rest of this issue looks at the two architectures in more detail.

---

⁴ The Web works because participants publish materials using HTML (the HyperText Markup Language, a subset of SGML, the Standard Generalized Markup Language) and embed pointers to other documents using URLs (Universal Resource Locators); for more details on the Web and Mosaic, its most popular browser, see Release 1.0, 1-94).
THE GORY DETAILS (RATED "O" FOR OBJECT)

Component software is quite different from conventional software. The move to object-oriented programming and component architectures essentially puts a communications system inside the programming environment. Not too long ago, applications consumed data and spat out results, which a human operator would then pipe into another process, say a sort routine or print job. Now applications negotiate such things themselves, in real time, while they execute. Increasingly, the code and data can be local or remote, written in the same language or in another one entirely.

Microsoft and the OpenDoc companies have placed different bets on the way to design a component environment. The most obvious bet is platform support. Although Microsoft will port OLE 2.0 to Macintosh and will achieve some object interoperability through DEC's ObjectBroker, it won't realistically have complete offerings for Unix and OS/2. OpenDoc is designed from the ground up for portability.

Explicitness and complexity

OpenDoc is a layer above multiple operating systems: It provides a shell that hosts a common object model and document storage services. In contrast, OLE 2.0 is pervasive in its environment, but inconsistently so. Although it operates everywhere from deep in the OS to the innards of each application, almost every element of OLE 2.0 is optional. This gives programmers a lot of leeway, but it also opens the door to great complexity (for example: Does this container support automation? Custom controls? Is it a server or a container-server?)

Explicitness also plays a role in raising complexity. Microsoft does not use explicit unifying schemes: It doesn't define event semantics, a document architecture or a governing metaphor. In contrast, OpenDoc has the Open Scripting Architecture, Bento and the document metaphor with parts and editors. Microsoft's omission is by design. Roger Heinen, the senior vp of Microsoft's developer division, says that Microsoft doesn't believe such constructs are needed and will offset OLE 2.0's added complexity with tools, training and support.

Execution

Finally, there are some subtle differences in the way the two systems work that may matter to users. OpenDoc permits multipage parts; OLE doesn't. In OLE 2.0, users double-click on embedded objects to activate them. In OpenDoc, activation mechanisms are transparent and depend on the platform. Click on a rectangle inside an embedded OpenDoc drawing and not only does the drawing activate, but the rectangle also becomes active.

Although the OpenDoc spec and its elements' implementations win frequent technical praise, and OLE 2.0's implementation elicits complaints, Microsoft has moved quickly to remedy the situation as best it can. One spectacular response is the way Microsoft's latest C++ compiler with Microsoft Foundation Classes can churn out OLE 2.0 component shells, automating much of the ugly work of creating the framing code.
MICROSOFT: THE ROAD TO OLE 2.0

By not defining event semantics, a document architecture or a governing metaphor in OLE 2.0, Microsoft allows programmers enormous flexibility. It also makes it possible to adopt different OLE 2.0 features without re-writing an entire application. Where possible, Microsoft's developers have made OLE 2.0 optional and evolutionary. As a result, it is complex and multi-faceted. Component compatibility depends more on object-to-object links than on the spec itself. This section offers a brief description.5

DDE, OLE 1.0, DLLs and VBXes

The way applications share information under Windows has changed considerably over the past few years. First there was cut and paste; then there was Dynamic Data Exchange (DDE), which offered limited but important application-to-application communications that required both applications to be active and to agree on how the interchange would work.

In 1991, in collaboration with other ISVs, Microsoft extended DDE with the Object Linking and Embedding architecture (OLE; see Release 1.0, 5-91). In theory, OLE 1.0 launches appropriate applications, encapsulates DDE transactions and offers compound-document support, but it is inefficient with resources and only marginally useful as implemented, so even though developers included it in their applications, it didn't get broad use.6

What did catch on were two things that allowed developers to extend their applications on the fly: Dynamic Link Libraries (DLLs) and Visual Basic custom controls, often referred to by their file extension, VBX. DLLs allow a program to load code selectively at runtime. If you add a new DLL, your program can support a new printer, communication driver, data type or API, among other things.

VBXes allow programmers to extend Visual Basic programs with functions written in more powerful languages. As VB improved and some developers delivered applications built entirely with it, VBXes took off, too. Now Microsoft is driving VB across its entire system software base (see box); it wants to tap the momentum behind DLLs and VBXes as it attempts to migrate developers to OLE 2.0. More on that in a moment.

Enter OLE 2.0

OLE 2.0 is very different from OLE 1.0. To review: OLE 1.0 is a (clumsy) way for one application to invoke another. Double-clicking on an embedded

---

5 The January 1994 issue of the Patricia Seybold Group's Distributed Computing Monitor does a great job of describing OLE 2.0, especially the Component Object Manager (Resources, page 23).

6 When you select an object in OLE 1.0 by double-clicking on it, it opens the corresponding application (in its entirety) in a separate window, which can lead to memory problems. You are responsible for saving its contents as if they were running in a separate application, even though they may be displayed together.
object (say, a chart embedded in a report) launches the creating application, which in turn loads the entire data file in question and displays it in a separate window, with its own menu bar. You could consider the called application to be a huge software component. (The VBXes described above exist in parallel, within the VB environment.)

Visual Basic at center stage

On the Windows platform, Visual Basic (VB) has become what HyperCard should have been on the Mac. Although HyperCard has a scripting language (HyperTalk), an object infrastructure and extensibility through external commands and functions (XCMDs and XFCNs), the extensibility is too limited and didn't catch on.

VB custom controls (VBXes), which allow programmers wide-ranging extensibility of VB programs with C code, took off. (VBXes extend DDE and are hosted by the VB runtime engine.) VB is now the centerpiece of Microsoft's scripting strategy. An embeddable, OLE 2.0-compliant version called Visual Basic for Applications (VBA) will replace application-specific scripting and macro languages -- while preserving compatibility with legacy scripts. VBA enables any software component to drive applications as if they were background services. Access and Excel can already be harnessed this way.

For example, Microsoft's Access Data Navigation controls allow users to browse or use Access data from other applications. The controls can easily add database access to any application. Once developers link the controls to a display field, they're almost done. Given a good object library and good object property inspectors, creating certain software tools can become a matter of editing properties and adding a few lines of code.

VB is the common language inside the Microsoft offerings. OLE Automation (page 14) allows applications to use any other Automation-aware language. Over time, Microsoft expects that people will use VB to create their own sharable objects, as well as custom components or applications for other elements of Microsoft's product array, such as Microsoft At Work and WinPad.

It's far easier to edit embedded objects with OLE 2.0, which adds visual (in-place) editing. There's no separate window, and the outermost container's menu bar changes to reflect the fact that a new component is in charge. OLE 2.0 also offers drag and drop; object nesting within containers (one level deep, then a separate window opens); storage-independent, adaptable linking; object conversion; and automation, which we describe in a moment. The entire system is built atop and deeply integrated with Microsoft's object-management system, the Component Object Model (COM, page 15). Cairo depends on OLE 2.0 and COM.

The initial OLE 2.0 system was released as a set of DLLs in April 1993, but its complexity daunted developers. Regardless of how difficult it is to master, there's likely to be a market for those who can get software out the door. OLE 2.0 works on Windows, NT and will soon work on Macintosh. The OLE 2.0 SDK is available for $50; no run-time royalties are needed.

Release 1.0 25 May 1994
OLE Automation

OLE Automation occurs when programmers expose OLE 2.0 components' properties, events or methods so that they can be driven or used by external scripts or programs. A component that uses Automation to drive other components is called an Automation Controller. A third-party developer could add OLE Automation capabilities to other programming languages, such as XLisp or Smalltalk V, giving programmers who use those languages access to OLE 2.0 objects.

Controls, containers and servers...and container-servers

Here's where it gets really thick. OLE 2.0 has two basic kinds of components: objects and containers. Objects can be text, graphics, sounds and other data types, or they can be gauges, palettes, buttons, database tools and other OLE custom controls, which we explain in a moment. Containers can hold objects or other containers. (By the way, a container that's embedded in another container is also an object.)

There are two special kinds of containers. A container that only offers services to other components is a server. If it can also hold objects or other containers, it is a container-server. The claim-processing example below is less abstract.

Here's an example to make OLE 2.0 more concrete. Container 3 is a server that does database access. It is put inside Container 2, a claims inquiry form. That container is, in turn, placed in Container 1, a claim-processing application. Media objects usually go in any document (a particular kind of container); custom controls usually go in any container.

Release 1.0

25 May 1994
All OLE 2.0 elements have access to compound document services, such as embedded or linked objects, drag and drop functions and visual editing. Beyond that, most of these elements are optional. OLE 2.0 developers have extraordinary leeway. They can enable their objects to invoke only specific OLE 2.0 services, such as event notification, structured storage services, uniform data transfer, automation, monikers (object link tracking) and controls. They can also determine low-level things such as how much of an application or data file is brought into memory.

All the details of component interaction have to be ironed out properly up front, because the first time a developer publishes the description of the functions that are exposed, it is committed to support them consistently in the future. To add functionality, it must define a separate, new interface, and it must still support early ones. The process that ISVs will use to agree on interfaces and the functions they deliver without conflicting with each other is unclear.

Finally, OLE 2.0 custom controls

OLE 2.0 has no event manager, so objects have to negotiate directly (as do CORBA objects on different platforms). In order for two components to pass events, they must first touch and establish a return path using OLE 2.0 custom controls (also known as OCXes, a term Microsoft prefers not to use). Custom controls add structured, two-way event management between containers to OLE Automation, the OLE mechanism that manages the stuff that objects expose and makes it available to other controls or applications. Unlike VBXes, which depend on Visual Basic, OLE 2.0 custom controls are language-independent.

Custom controls are the most component-like elements of OLE 2.0. Not all containers are "custom-control aware." So far, Access 2.0 is the only application that includes custom controls. Shapeware's Visio Express (page 7) implements only OLE Automation, not custom controls, which the company is not sure it will support.

As part of its move to advance and generalize its architecture, Microsoft wants to leverage the popularity of DLLs and VBXes and unify them with OLE in OLE 2.0. But ISVs should beware. VBXes are tied to Visual Basic and to Windows' current 16-bit architecture; they will disappear. Custom controls, which offer language independence, will be the standard.

The Component Object Model

Underneath OLE 2.0 is Microsoft's Component Object Model (COM), which provides the plumbing and wiring of OLE 2.0. Think of OLE 2.0 as a series of windows into COM, where the real work of naming, finding and assembling objects and managing their interactions happens.

COM manages all OLE 2.0 objects, which are guaranteed unique because they have universally unique 128-bit IDs. Once a developer defines and pub-

7 The OLE custom control development kit (CDK) is in beta now and will become available with Microsoft's C++ 2.0 compiler late this summer.
lishes an object specification, the interface definition is locked, and the vendor must support that interface from then on. COM doesn't support inheritance, so objects use a process called aggregation to collect attributes from referenced objects. When an object is upgraded or is used in a different way, the developer creates new interfaces to the object, rather than evolving the original one. That's how Microsoft hopes to guarantee that the objects will be there and will function properly in the future.

<table>
<thead>
<tr>
<th>Strap on the booster jets with MFC 1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>To make it easier to develop controls and applications that use OLE 2.0, Microsoft has written 20,000 lines of code in its Microsoft Foundation Classes (MFC), part of Microsoft's C++ developer's toolkit. A software wizard guides the programmer through a series of choices (want an application, a server, or a control?). At the end of the choices, MFC churns out the various skeleton modules needed to comply with OLE 2.0. The improvement over developing from generic skeletons or no code at all is significant. But this is a language tool, not system software -- and Microsoft's tool, at that. There's no reason to expect that MFC will support OpenDoc, which opens an opportunity for cross-platform development environments (see Novell/WordPerfect, page 21).</td>
</tr>
</tbody>
</table>
OPENDOC: THE COMPONENT GUILD

OpenDoc is a clearer, more explicit and more generalized component and document framework than OLE 2.0. It is also less incremental and will require more changes from participating developers and end-users, though not necessarily more effort: Both architectures require substantial work to adopt. Because it will run on many different operating systems with elements from different companies, it is also more likely to hit system problems.

Documents are the primary metaphor in OpenDoc. Applications all but disappear: They look like task-oriented stationery on the desktop, such as icons representing ruled paper, a spreadsheet, drawing paper or a database form. (The applications themselves may end up in a new part-handler directory.) To start a new document, you choose the type of stationery you want; that sets which application is your "root part."

OpenDoc documents are composed of objects called parts, which are viewed through part viewers and manipulated by part editors. In-place editing is easy. Object activation follows the model of the hosting platform: Objects on a Mac and Windows are activated when you click once on them; in Motif, when your cursor is over them. On all platforms, parts negotiate quickly for control over resources such as toolset and menu items, active selections and data streams, and reflect the results at the interface. Switching is quick and transparent.

Users can set preferences for which editor to invoke when they activate an editable part; OpenDoc defaults to the creator application.

You can have it all

OpenDoc gives each application developer the ability to incorporate all other types of OpenDoc-aware content, without having to modify its software. Customers won't have to wait for upgrades in order for an application to work with a new feature from some other OpenDoc vendor. For example, on the Mac, all OpenDoc parts are automatically mailer-aware. When someone creates a desktop video-conferencing part, it will be available to all OpenDoc documents.

OpenDoc's structure and the fact that it is designed with multiple platforms in mind should make it appealing to a broad range of developers. Large ones will be able to repackage their code and customize it for vertical markets; small ones will be able to sell software without having to create a whole channel themselves or build complete applications. They can develop parts that significantly augment larger applications.

The major elements of OpenDoc are the OpenDoc shell, the Bento compound-document storage specification, the Open Scripting Architecture, the System Object Model and (a recent and essential addition) OLE Interoperability.

-----------------

8 Consider the document to be a palpable, two-dimensional snapshot of underlying processes, some of which may not be visible to the end-user. Developers are exploring client/server and data-mining applications that use OpenDoc's multitasking and networking capabilities.

Release 1.0 25 May 1994
The shell, Bento and OSA define object layout negotiation, data transfer and scripting, respectively, more explicitly than OLE 2.0 does.

The OpenDoc document shell

An OpenDoc document runs in a shell that manages many aspects of the parts running inside it, including part placement, resource contention and message passing. The shell doesn’t dictate, but rather mediates between parts. For example, OpenDoc’s smart screen-brokering means that parts aren’t limited to rectangular areas: Irregularly shaped parts negotiate their placement on-screen. Thus every OpenDoc document has access to page-layout capabilities, which comes in handy when the document has to handle long blocks of text or odd-shaped graphics gracefully.

A user can have several OpenDoc documents or parts active at the same time, and can drag and drop content from one to the other. OpenDoc’s Arbitrator and Dispatcher make sure the right editors get user-interface and semantic events (system commands such as cut, paste, open and format, all reflected in the Open Scripting Architecture) to make sure they don’t conflict with each other. To prevent deadlocks, the Arbitrator tracks and reassigns resources with a two-phase commit process.

Bento

Bento, the Japanese word for a compartmented lunch box, is the storage and interchange format for OpenDoc documents -- on disk, in memory and in the clipboard. CI Labs’ Jed Harris created it when he was at Apple. Bento objects are not tied to the tools that created them. The format stores data schemas, so any application that can address that kind of part is welcome to read it. Unlike OLE, Bento supports versioning.

A Bento object has its own internal table of contents, which can describe sophisticated data structures. Or, if the developer prefers, Bento allows an application direct access to data streams.

The Open Scripting Architecture (OSA)

During work at Apple on document architecture in 1989, developers realized they needed a way to add intelligence to documents. Two of the resulting technologies were AppleScript and AppleEvents, which were created by Kurt Piersol of Apple, OpenDoc’s chief architect. AppleScript is a Mac-specific scripting language; its foundation, the Open Scripting Architecture (OSA), is a language-neutral scripting interface that connects scripting engines and software components. OSA is based on semantics that collect application capabilities into domain-specific sets of commonly understood verbs and nouns that map to the AppleEvents model.9 OSA already defines event

9 OpenDoc participants agree upfront on the base vocabulary and on a way to extend it; in OLE 2.0, developers agree on a few words at a time. Microsoft believes that this method eliminates the subtleties that create the differences between applications. For example, should the command "select sentence" pick up spaces before or after a sentence, or ignore them? How should it treat punctuation?
suites for interacting with tables, rich text, telephone controls and other items. As novel applications describe and implement new actions or objects, developers can extend the model.

To use OSA, vendors need to make their applications scriptable. Applications can publish event suites at runtime, or, if appropriate, developers can work with CI Labs to standardize novel suites.

OSA applications can be scriptable, recordable or attachable. Almost all are scriptable, which means they support OSA events and objects. Recordable applications post their actions to OSA, where they can be recorded for playback -- which helps users record macros, for example. Attachable applications allow users to extend applications by attaching scripts to objects inside the applications.

There are already many AppleScript-aware applications on the Macintosh, from vendors such as Quark, PageMaker, DeltaPoint, Microsoft, Claris and Aladdin Software. There are also other scripting systems. On the Macintosh, UserLand's Frontier was available well before AppleScript (see Release 1.0, 5-91). IBM has settled on its trusty Rexx scripting language for system-wide use and OSA support.

Under OSA, developers can choose the scripting language they prefer and drive OSA-compliant applications on foreign platforms. The degree of control developers have increases as applications expose more discrete functions (as opposed to invoking an entire application and passing it parameters). At its current best, OSA allows for remote execution of a command such as, "set the style of caption of each figure to bold."

SOM and DSOM

IBM began development of its System Object Model (SOM) five years ago, as part of the OfficeVision project (codenamed Sonic) in IBM's Toronto lab. Eventually, this work also led to development of the Workplace Shell in OS/2, which was written using SOM.

As the developers used object-oriented technology to create OfficeVision, they realized that C++ alone fell short of the functionality they wanted. They needed a more robust, language-neutral system to package, store, track and assemble objects so that binaries developed in any language could be interchanged and used productively. They also wanted a system that didn't require the developer to recompile all modules when one changes, then redistribute new source code.

As a result, they created SOM, which separates an object's defined interface from its implementation, which can be in various computer languages. IBM's SOMObjects is available now on OS/2 and AIX. IBM is embedding SOM in its C++ compiler, which will allow developers to compile to C++ or to SOM. Apple's OpenDoc alpha uses the Apple Shared Library Manager (ASLM); it is porting SOM to Macintosh and expects to use it instead of ASLM in the beta release. WordPerfect has finished the port to Windows, but has not integrated SOM yet with its OpenDoc and OLE software.

SOM also has a distributed version, called DSOM, that takes over if a called object is not in SOM (it works similarly to the Lightweight RPC that OLE 2.0 uses). IBM's SOMObjects incorporates DSOM.

Release 1.0 25 May 1994
THE PLAYERS AND THEIR MOTIVATIONS

OLE 2.0 is far more ambitious than 1.0. It goes beyond interprocess communications or document architectures, and suffuses Microsoft’s entire software infrastructure from developers’ tools and libraries to the way customers use and buy software.

Apple, Borland, HP, IBM, Novell and WordPerfect had been discussing object architectures for several years. In early 1993, shortly after Microsoft seeded OLE 2.0 to developers, they coalesced around what became OpenDoc. Many of the companies believed that OLE 2.0 jeopardized their ability to compete on an equal footing in the software market. They also believed they had superior technology already in hand, so they pooled the technology that would help them work together but would still allow room for differentiation and competition.

The balance between competition and collaboration is tricky; many consortia have set out, but few have reached their goals. That makes it worth examining the OpenDoc participants’ motivations. The most important players at this moment are Apple, IBM and Novell/WordPerfect.

Apple’s one-two punch?

OpenDoc represents a major shift for Apple, which had never opened its technology to other companies. Now it is planning to make two important sets of intellectual property (OSA and Bento) available for broad licensing through CI Labs. This shift is crucial. Unmoved by Apple’s Open Collaboration Environment, developers have been defecting to Windows, where the market is. Apple needs to rekindle the enthusiastic following it once had -- the kind that exists around applications, VBXes and DLLs in Windows.

The new Power Macs clearly help, by putting Apple back in the hardware game at aggressive prices for high performance. The key for Apple is to get developers to adopt OpenDoc as they rewrite their applications for PowerPC.

Although Apple released a new version of HyperCard last December, think of OpenDoc as its successor and replacement. Apple will continue to support HyperCard as a prototyping tool, but OpenDoc can do most of what HyperCard does and offers much more power, not to mention potential cross-platform compatibility.

IBM’s object(ive)s

IBM is among OpenDoc’s strongest supporters, and is responsible for OpenDoc on OS/2 and AIX. The AIX version will likely be the launching pad for multiple versions for different Unix flavors. IBM’s major contribution to OpenDoc is the System Object Model, which IBM is also porting to all of its own platforms. Although they are not integrated, Taligent and OpenDoc are the cornerstones of IBM’s object strategy.
Novell/WordPerfect: a two-pronged strategy

With Windows NT, Microsoft is after Novell's core networking business. In response, Novell has begun to move down the food chain into applications and application development. Bob Frankenberg's recent move from HP to succeed Ray Noorda at Novell may change some of the priorities we mention here, but Frankenberg is a long-time object champion (he's a charter member of the OMG and a big booster of NewWave), so object-oriented systems are likely to continue to play a key role at Novell.

Novell and WordPerfect, its newly acquired division, have the most difficult job in the OpenDoc alliance. The combined entity has a major opportunity to make AppWare the best platform and toolset for companies to develop applications and components efficiently for OpenDoc and OLE 2.0. But that means finishing the integration of both models into Novell's still-evolving AppWare toolkit.

Novell and Borland are integrating Borland's ObjectWindows Libraries (OWL) with AppWare, as ObjectWindows for AppWare Foundation, which will allow developers to create OpenDoc parts.

WordPerfect recently demonstrated an essential capability: OpenDoc interoperability with OLE 2.0. WordPerfect's OpenDoc kits for Windows are scheduled for alpha test in June; the company expects to sell commercial OpenDoc components this year. WordPerfect turned to OpenDoc when its software architects saw too many ambiguities in OLE. In particular, they didn't see any guarantees that their components would work in other OLE containers. The company also saw that Microsoft's strategy with Visual Basic for Applications, the Office suite and other components would drive WordPerfect further outside.

Now WordPerfect is using OpenDoc to deconstruct its monolithic software applications. That way, the company expects to compete more effectively with suites and re-enter the software war. Then it will tailor its components to vertical markets, to create, for example, a customized and customizable law-office system in conjunction with third-party developers or integrators.

GI Labs, the component guild

OpenDoc's long-term prospects are governed largely by how the participating companies work together. The coordinating entity, created for this purpose

---------

10 WordPerfect has extended OpenDoc so that it can act as an OLE 2.0 container and server. OLE 2.0 parts think they are in OLE, yet act as OpenDoc parts locally. WordPerfect can demonstrate three kinds of interoperability in Windows, in both directions (OLE 2.0 inside OpenDoc, and vice versa): Simple embedding, in-place activation and "inside-out." It also showed nestable OLE 2.0 container-server functionality. As a user selects different components from both environments, menus cascade properly to the outermost container. Drag and drop and the clipboard work as they should. Documents that these components may use are also cross-embedded: A Bento-format OpenDoc part can be stored in an OLE 2.0 file, under COM control, and vice versa.

Release 1.0 25 May 1994
in September 1993, is Component Integration Labs (described on page 4), which is run by Jed Harris. CI Labs was incorporated only this month.

Harris is a longtime student of the ecology of concepts. He started in AI and linguistics, then worked with Doug Englebart at SRI and Alan Kay at the Xerox PARC Learning Research Laboratory. Object-oriented systems have pervaded his work ever since, including stints on object-oriented operating system projects at Data General and Intel. In 1985, he decided that objects were only going to deliver real customer value on PCs, so he joined Apple, where he began working on software components. This work merged into a "future architecture" project with Larry Tesler, Kurt Piersol and others, called Family Farm. Family Farm spun out a series of developments including the Open Scripting Architecture and AppleScript, and also led to Bento, which Harris designed.

Harris expects OpenDoc development to evolve around multiple centers of interest, such as document management, client/server, telephony, multimedia, 3-D and virtual reality. For now, he is keenly aware of Microsoft, and will have to shepherd his allies better than Microsoft if OpenDoc is to thrive. He's clear about his purpose: "My goal is to have companies compete on delivering user value rather than compete on market control."

And many more...

Symantec recently announced that it will support OpenDoc with its Think C++ development system. Bedrock, the Apple/Symantec effort to create multi-platform object classes, is now part of the OpenDoc Parts Framework at Apple. Taligent is on the CI Labs board and is as active in OpenDoc's development as it can be while staying focused on the delivery of its own object frameworks. It expects to ship alpha code in June. Eventually, Taligent's frameworks will support both OLE and OpenDoc. Other companies are concerned about Microsoft, interested in component software but wary of overextending their development resources. As a result, their levels of commitment periodically rise and fall. Lotus, Oracle, Sun and Xerox showed early interest, but have yet to commit fully to OpenDoc.

A non-commercial break

After all these years of publishing Release 1.0 12 times each year, we have decided that we need to take vacations, too. Starting with the June '94 issue, Release 1.0 will be published monthly, except for a combined July/August issue. All current subscribers will receive 12 issues before renewal.
RESOURCES & PHONE NUMBERS

Kurt Piersol, Apple, (408) 974-1201; fax, (408) 974-8490; <piersol@apple.com>
Howard Rosenfield, Apple, (408) 862-7582; fax, (408) 974-9456; <rosenfield@applelink.apple.com>
Jed Harris, Component Integration Labs, (408) 974-6549; fax, (408) 974-9710; <jed@cil.org>
Cliff Reeves, IBM, (512) 838-2130; fax, (512) 838-1032; <cliffr@austin.ibm.com>
Roger Heinen, Microsoft, (206) 936-3357; fax, (206) 936-7329; <rogerh@microsoft.com>
Dave Seres, Microsoft, (206) 936-5275; fax, (206) 936-7329; <daveser@microsoft.com>
Doug Donzelli, Novell, (408) 973-8081 x304; fax, (408) 973-0989; <donzelli@sti.com>
Chris Stone, Object Management Group (OMG), (508) 820-4300; fax, (508) 820-4303; <stone@radiomail.net>
Mark Ericson, WordPerfect, (801) 222-7288; fax, (801) 222-7378; <marke@wordperfect.com>

For further reading:

"Microsoft OLE 2.0 and the Road to Cairo," Distributed Computing Monitor, January 1994, Patricia Seybold Group.

COMING SOON

- HTML and SGML futures.
- Multiplayer games.
- What's a zine?
- Software for education.
- And much more... (If you know of any good examples of the categories listed above, please let us know.)

Release 1.0 is published monthly, except for a combined July/August issue, by EDventure Holdings, 104 Fifth Ave., New York, NY 10011-6901; (212) 924-8800; fax, (212) 924-0240. It covers PCs, software, computer-telephone integration, groupware, text management, connectivity, messaging, wireless communications, artificial intelligence, intellectual property law and other unpredictable topics. A companion publication, Rel-EAST, is an information bulletin on emerging technology markets in Central Europe and the former Soviet units. Editor: Esther Dyson <edyson@eff.org>; publisher: Daphne Kis; managing editor: Jerry Michalski <spiff@panix.com>; circulation & fulfillment manager: Robyn Sturm; executive assistant: Loyce Baker; editorial & marketing communications consultant: William M. Kutik. Copyright 1994, EDventure Holdings Inc. All rights reserved. No material in this publication may be reproduced without written permission; however, we gladly arrange for reprints or bulk purchases. Subscriptions cost $495 per year, $575 overseas.

Release 1.0 25 May 1994
SUBSCRIPTION FORM

Please enter my subscription to Release 1.0 at the rate of $495 per year in the U.S. and Canada. Overseas subscriptions are $575, airmail postage included. Payment must be enclosed. Multiple-copy rates available on request. Satisfaction guaranteed or your money back.

Name __________________________________________
Title _________________________________
Company __________________________________________
Address __________________________________________
City __________________ State __________ Zip __________
Telephone __________________________________________

☐ Check enclosed.
☐ Charge my
  ☐ American Express  ☐ MasterCard  ☐ Visa
  Card Number ___________________________ Expiration Date __________
  Name on Card ___________________________ Signature __________________

☐ Please send me information on your multiple-copy rate.

Please fill in the information above and send to:

EDVENTURE HOLDINGS INC.
104 FIFTH AVENUE, 20TH FLOOR
NEW YORK, NY 10011

If you have any questions, please call us at 1 (212) 924-8800;
Fax 1 (212) 924-0240; e-mail MCI:511-3705;
Internet 0005113705@mcimail.com.

Daphne Kis
Publisher