DOMAIN OF OBJECTS: THE OBJECT REQUEST BROKER REQUEST

The Object Management Group, the leading commercial association of vendors and users of object-oriented tools and systems, is about to select a specification for distributed object management technology, known as the Object Request Broker (ORB). The ORB spec provides a framework for -- and its implementations will manage -- cross-system communication among objects. The selection, which should be completed this summer, is the first step in making possible such communication across remote systems, and in making diversity coherent. (We are among a small number of consultants selected to evaluate (pro bono) the seven responses to the OMG’s request for proposal.)

Unlike the Open Software Foundation, OMG does not plan to build and market products that implement the standards it blesses; it will merely define specifications for members and others to follow. The ORB fits in well with the object-oriented vision of open interfaces covering opaque, potentially diverse interiors -- so far, but no farther. Objects should be able to communicate but need not reveal their insides.

The Object Management Group’s move to adopt an ORB standard is important as the first technical step towards interoperability of object systems, and the first effort by a recognized commercially oriented standards group in the area. The particular choice of ORB technology probably doesn’t matter as much as it does just for there to be a single choice that vendors and users can rally around. The ORB is a discrete item that should be relatively easy for most vendors to incorporate into or encapsulate onto their systems with little disruption (depending on the degree of integration). Thus the choice is not a life-or-death matter as, say, a standard object model would be.

The ultimate goal of a coherent, unified universe is, fortunately, unachievable, as it would also imply the end of progress. The ORB is an important to move forward, controlling and supporting diversity and progress with encapsulation of differences, with no illusions about getting to the end.

The encapsulation of diversity inherent in object-orientation is an attempt to combine continuously changing, improving local optima into something like a global optimum. Compatibility and standards keep the global optimum as determined at any one time static and resistant to local improvements. On the other hand, they insulate complementary systems from any internal changes. Thus common interfaces allow for reuse of code -- and the ideas

AN OBJECT A DAY KEEPS STASIS AWAY!
and technology breakthroughs it implements. That is why, for one example, we're so enthusiastic about GO's object-oriented PenPoint operating system, discussed in Release 1.0, 1-91: It combines innovation with facilities for communication with the rest of the world.

**OMG ARCHITECTURE -- NOT INHERITED FROM OSF**

Although it shares many characteristics with the Open Software Foundation -- most notably, HP was an original driving force behind both of them -- the Object Management Group is a different kind of organization from the OSF.

For starters, it wasn't created to stop somebody -- Sun/AT&T -- but rather to help a broader movement towards object-oriented systems. (There wasn't anyone entrenched enough in this still immature field to stop!) The OMG has now moved well beyond that beginning and is the primary commercial organization devoted to the proliferation and standardization of object-oriented systems. Its membership of more than 120 vendors, developers and users includes almost every important computer industry player except IBM; Microsoft has just joined.

Already, the OMG is fostering not just interchange of ideas and personal contacts but technical cooperation among erstwhile competitors; witness the joint submission from Sun and Hewlett-Packard. Another benefit of the OMG may be a common language for describing its own field. Part of the difficulty in assessing the ORB submissions is the terminology: Vendors tend to offer the same things under different names, and different things under the same names.

Second, while OSF builds and sells reference implementations, OMG's role is simply to endorse technology and specs. Although the distinction is fine and one man's interface is another man's implemented layer, OMG focuses more on interoperability than on strict compatibility. Obviously, the use of objects makes that easier anyway; in theory, objects should be addressed and not seen inside during operation (although inheritance and more specifically subclassing contravene that principle).

The OMG has little staff of its own (eight people); its decisions are made by committees of its own members -- vendors, developers and users from all walks of business. This is politically benign and a good way of involving the community most affected, but it slows things down. It also requires huge commitments in time as well as money from its members.

The ORB selection will be a demonstration of the OMG at work, gearing up for the next, tougher phase. That test will come with the next selection of technology, for the Object Model. (Responses to a request for information are due May 1.) This effort is run by an amalgamation of the OMG's database special interest group with the core OMG membership, and it addresses issues that will be much harder to resolve. It covers the specs for an object manager and object structure -- the details that the ORB navigates among.
The goal with the broker is to move from point-to-point local communications to hubs: Everyone can talk to everyone through some common syntax and traffic systems. The ORB has two functions -- communications between local and remote systems, and communications across environments. These are two orthogonal tasks, although they frequently occur (and are handled) together.

The line-up

The contenders are:

- Architecture Projects Management (APM), on behalf of a project of Europe’s Esprit consortium (ANSAware);
- Groupe Bull of France (Comandos);
- Digital Equipment (ACA Services);
- DSET, a New Jersey communications and object-oriented tool company (Distributed Systems Generator);
- Hewlett-Packard/Sun, in a joint filing of political as well as technical significance (Distributed Object Management Facility);
- HyperDesk, the Data General spin-off funded by ASCII Corp. of Japan (HyperDesk);
- NCR, joined by OODB vendor Object Design (Cooperation).

All these companies have different visions of what the Object Request Broker should be, and each measures its offering by its own vision. Thus the question decision is not just, Which is the best implementation of the ORB? but also, Which is the best yardstick by which to measure implementations? Moreover, the scope of the submissions varies; each implementation must be commercially available at the time the choice is made this summer, and thus each submission includes a lot of functionality outside the scope of the ORB itself. So the question is also, How well can each implementation handle the tasks for which it is not expressly designed? (This report makes no formal recommendations, just a consideration of some issues and trade-offs.)

THE FRAMEWORK: MESSAGES WITH THE PROPER STRANGER

You talkin’ to me? Hey, you! You talkin’ to me!?

The "layers" we all work with -- hardware, operating system, "environment," user interface and the like -- are all attempts at vertical interoperability, so that different layers of a single system can be replaced invisibly. Traditional systems (even object-oriented ones) talked to the operating system as a layer; now they may talk to it as a collection of objects. The same goes for communications media and protocols, device drivers and most other external services, as well as other "application" objects, that accomplish tasks of more direct interest to end-users. Objects carve the territory up both vertically and horizontally, providing finer granularity of encapsulation and allowing different parts of a horizontally distributed system (modules rather than layers) to be replaced invisibly.

So what’s the difference between an object and a layer? They don’t look much different if you’re talking vertically. Objects, however, let you talk to peers, and (with an ORB) across environments. With distributed objects, in fact, you may be talking to a variety of implementations of the same layers in different environments. (The Bull, NCR and HyperDesk object-oriented environments, for example, explicitly treat most of the services of the local operating systems as objects, as will Patriot Partners’ Constella-
Precisely because of this variety and heterogeneity, then, you need common interfaces both for specifying (and then a system for finding) the correct object including related code, and then a general way of waking it up. Since it's not necessarily in your local environment, it may not be active at the moment you happen to summon it (although you'd like to be able to assume that it is). But you want this whole process to be invisible.

OBJECTS IN CONTEXT

The Object Request Broker is of most immediate interest to developers, integrators and ISVs. Although its existence is important to end-users, it will be most useful if they don't even know it's there. However, its effects will filter through to users in applications that seem unusually smart, flexible and aware; even the applications they've always used may suddenly get a higher IQ and better manners. How does this happen?

Basically, object-oriented systems can know about and encapsulate existing applications and data; object-oriented systems are self-aware. By contrast, procedural systems don't know about objects; they just think they're calling a function, or a user is manipulating some data or whatever. Either kind of system can call or "contain" the other, depending on your perspective (and the balance of power), but only the object-oriented system is aware of what it contains.

This is because an object is more than just some data. It's encapsulated or protected data -- bound with rules about how it behaves, what can be done with it, and so forth. It's like a formal database transaction as opposed to a simple application sequence such as editing a database item and storing the result. In the first case, the system knows what has happened; in the second, it isn't aware of the change.

For example, if you deal with the data directly, you can do what you like with it, but the system can't manage it for you. Take an editing system, you can move a set of lines over five spaces each, but if you want to add a word in the second line, everything goes awry. If you deal with it more as an "object," using the set-margin command, the system can track your moves and knows how to make the proper adjustments. It can follow the acts you (or other objects) are performing on the target object, and can keep track of its state. It can also keep track of it across applications and manage more of the work automatically. Likewise, it can undo the work, and it can maintain the state of related objects: If you change Alice's status to unmarried, for instance, it can do the same for her husband Juan -- or it could ask you, "What about Juan?"

Another example: If you specify a group of spreadsheet cells by location, watch out when someone adds a new line! Better use the spreadsheet equivalent of an object -- a named range.

None of this is magic. It still requires programming. In fact, at first it takes more programming than before. But this one-time extra effort can make thousands of users (and subsequent programmers) more productive.
Most current software systems (both traditional procedural ones and object-oriented ones such as Smalltalk) assume that you're working within an all-encompassing environment, although certainly there are mechanisms for calling outside using SQL (a standard database interface), subroutines, etc. These mechanisms are generally visible and explicit to programmers, although tools such as RPC compilers can ease the task of building them, and standard protocols such as SQL or Sybase's Open Client/Open Server allow many-to-many interchange instead of only tightly coupled client-server interactions.

Within a single environment current objects hide implementation details; an ORB will also let them hide the implementation details for communications outside their own worlds without losing the benefits of object-orientation. In essence, it lets them encapsulate the outside world and the communications mechanisms...and it also encapsulates the objects themselves for the outside world to use. In short, it extends the benefits of object-orientation across platforms and communication channels.

However, there's a point of view that says this shouldn't all be too smooth and invisible, just as users of distributed databases should be aware that there's a cost to instant data from around the world. Most database people have given up on the idea of impeccably consistent worldwide databases of complete integrity in favor of inclusion of local databases and processes for reconciliation of inconsistencies. The notion of the ORB as a manager of object managers, so that the ORB can talk to objects it might not know about directly, reflects this idea -- but it represents a deviation from the one true object model for everything. (Scalability per se is less important than accommodation of different systems on different scales.)

Wake up; you're on your own!

The problem -- and the advantage, in terms of agreeing on a selection -- with the ORB is that it addresses only part of the problem of interoperability among object systems. The widespread adoption of a single object request broker spec will be helpful, but it won't guarantee interoperability among systems, let alone provide much of a framework for object-oriented application development. That will generally happen at the level of the local object managers/environments.

The joint Sun/HP submission, for example, implicitly makes the valid point that the ORB is merely a spec: Sun and HP are supporting the same spec, but with different implementations. Unfortunately, there's a corollary to that point. The two implementations don't work together, since each depends on a different communications (RPC) environment. This is not unique to Sun and HP. The other systems likewise must be hooked together via some communications mechanism acceptable to both sides. The ORB can link object systems, yes, but it can't overcome incompatibilities in the rest of the world. In other words, the ORB is (appropriately) not a standard covering everything; it's a standard for a component. Just talking French doesn't let you communicate with a Parisian without a working telephone link.

Across the wires, across platforms

Likewise, while communications operates "below," there's also nothing in the ORB to ensure interoperability "above" -- that objects on the same wire will in fact talk the same language beyond a few interface calls, or mean the
same things by the terms they use. The ORB addresses the syntax of communications, not the wide range of particular methods and calls that may be made using that syntax. Those particulars can handle everything from file systems and applications to object-oriented databases and minuscule, fine-grained objects and individual methods. So the ORB doesn't guarantee much about the kinds of objects at either end, although it will deliver an exception report if the message is rejected. (More dangerous would be the case where a message is executed with unexpected results.)

This is an issue both on the level of generic commands, such as "print," specific commands, such as "issue paychecks," and the language each individual object is written in (and is part of what OMG calls common services or even application objects, page 7). You may not need to know the details of implementation if you're addressing an object, but you do need to know the details of what it will do and the parameters it needs. NCR's implementation does offer a more consistent world, with tight integration of an object model with its ORB and dynamic selection of communications methods. APM, by contrast, somewhat limits the extent of cross-dependence between systems unknown to each other. "Ideally, you want to know what the contract is," says APM's Andrew Watson. If you want to reimplement, modify or inherit from the object, you do need to know what's inside (with proper permissions, of course). You can't have children through safe sex.

A good start -- and hands off the rest!

The ORB itself generally addresses objects as discrete wholes -- i.e. it's for execution, not development (although a tenet of OO purists is that development never ends). Some of the implementations address the specifics of inheritance, IDs vs. path names, communications protocols and so forth. Others (most notably HyperDesk) say in effect: Leave it up to the local system and language and object manager.

The ORB provides interfaces for objects to talk to one another, and a runtime component to manage the process. That's all, and that's enough. As such, it's an ideal technology for the OMG to start with, since it's likely that the participants will be able to come to an agreement; it's not too confining because it's the glue among the systems rather than a spec for the systems themselves -- except perhaps in the NCR and Bull approach. (Of course, they all need to interface to it, and use its specs to define their interfaces for others. Networks, operating systems, etc., are an exercise left to the reader, or developer, or installer. They can prevent communication despite the presence of an ORB.

Correspondingly, we doubt that a standard object model is possible or desirable at this point. It would be premature to bless anything more than the ORB -- because there is no way any broader, deeper offering could be ideal. It's probably better to let a number of contenders fight it out in the market than in a ponderous standards subcomedy. Even long run, there will not necessarily be a single standard object model, but instead several models, limited in number and pervasive enough to reduce confusion and inefficiency, yet still foster the progress engendered by competition.

The ORB submissions in capsules

The most interesting material in each submission was not the ORB part per se, but the surrounding computing/object environments and the visions ex-
pressed (see pages 15 to 22). Getting anyone to standardize on any of those, however, would be a problem of a different order of magnitude. All the offerings tend to be existing technology or products dressed up for the occasion, with varying degrees of relevance. Both APM's ANSAware and the DSET technology are colored by their background in the distributed computing community and have somewhat unusual approaches to object-oriented computing.

DEC's system is an application-integration system, with enough modularity to support almost anything, but it manages only the applications and their methods rather than the objects themselves. Bull and NCR have full-fledged object-oriented systems of which the object broker is a small but well-crafted part; they both have a consistent object model and style from top to bottom. HyperDesk is also consistently object-oriented, but it interprets that phrase in a different way: It has made its system so modular and extensible that almost any part of it can be replaced; its object model is encapsulation of object models. (It's like an example of the ORB spec rather than an implementation of it.) Finally, the Sun/HP Distributed Object Management Facility (DOMF) is the clearest example of the gateway-between-object systems model, although it starts out with an application-as-object culture from its NewWave heritage. But encapsulating applications is just "a necessary migration step to the future style of true [small-grained] object-oriented applications," says a Sun/HP statement.

Is it better to fight Sun and HP one at a time, or to have them fighting each other one on one and fighting everyone else as a combined front? Although the OMG was originally most vigorously promoted by HP (which had a motivation as the vendor of NewWave, one of the few commercial OO products at that time), it is now solidly nonpartisan; in fact, it may be too diverse to be fully effective in setting a single object model standard, although it will function well in promoting the technology in general and interoperability among implementations.

OMG's Object Management Architecture

The OMG's reference Architecture includes Object Services, Common Facilities and Application Objects as well as the Object Request Broker, with some overlaps.

Object Services deals with creation, maintenance, storage and integrity of objects; it is where you go for transaction management, version control, security, and the procedures for creating, modifying (through inheritance and otherwise) and destroying objects. It manages the objects rather than what they do, and is equivalent to a repository for a software development environment -- with the difference that the repository handles dead code, while Object Services manages living objects. The Object Broker uses object naming and location services to hand on the requests it receives.

Common Facilities is akin to a shared library of commonly used functions; it includes the sorts of things that once were in applications and are now migrating into operating environments, such as link management, user interfaces, device drivers, agent facilities and e-mail (user message transport).

Release 1.0

31 March 1991
**Application Objects** is the specifics of different applications, and is unlikely to be standardized in any meaningful way; however, Application Objects will use the standard facilities of the other three parts. Technically, AOs are the same as Common Facilities, encapsulated into representation as part of a class hierarchy; it's just that they are assumed to be less "common," and thus will come from a single application vendor rather than be supplied by a Common Facilities vendor to be used by lots of applications. (Obviously, the success of this model will change things not just for software developers and users, but for software sellers and resellers.)

Application Objects will migrate into Common Facilities status as they become popular and widely used and relied on -- de facto standards. CF status, in fact, provides a convenient way for the spread of standards without copyright infringement -- once appropriate commercial arrangements are made. How much simpler life would be if 1-2-3 were a CF; Lotus would collect a small fee for each use (or user, or whatever), and no one would bother to copy it or reimplement it. We say this tongue in cheek of course; things are never that simple. And of course, a whole application would more likely be represented as a collection of facilities/objects, not a single monolithic thing. (See Patriot Partners, page 23.)

The operating system and most communications facilities are external to all of this -- unless they are represented as objects (Common Facilities) also managed by Object Services and the Object Request Broker.

**How the ORB fits into the architecture**

The ORB is a spec for a system that will manage the communications among objects -- managing location, naming and delivery services, activation and deactivation of remote objects, method invocation, parameter encoding, synchronization, exception handling and security. It sits on top of a network or RPC mechanism, but doesn't get involved in the applications or (usually) the structure, form or capabilities of the objects themselves. Nonetheless, each broker implementation makes some implicit assumptions as to the sizes of the objects, the kinds and numbers of messages and parameters sent around the system, how they are instantiated and so on.

A request and its response are the basics of ORB interaction. A request specifies an operation, usually with parameters, any of which may specify a specific object or objects to help carry it out. The ORB finds the appropriate object and methods to carry out the operation and hands them the parameters. Then the ORB conveys the results back to the requester, or on somewhere else. The ORB may not have all the necessary information itself. It may use a directory in Object Services, call on a runtime method library or otherwise rely on other system capabilities, including local object managers. It may use a trading service to search by attributes -- including such changeable attributes as not-busy or has-stationery or even within-100-feet, as long as they're defined within the system.

The issue is not just communication but how to find and assemble things. In another world, people designing far-flung systems are discovering the difficulty of operating without a global name/directory service -- a problem that will eventually be addressed by widespread adoption of the X.500 standard. Consider the ORB an X.500 of the object world; it not only delivers the mail but makes sure it gets read.
Registering objects/classes

A major function of the ORB is keeping track of the objects. The naming/directory service is not officially part of the ORB (it's an Object Service the ORB uses), but the way of registering objects is. That "register" is basically a class hierarchy that lists the available services of the classes/objects and the interfaces that another object can use to address them. It knows the relationships among them and the location of the underlying code and instances, for use in dynamic binding of methods or finding parents for inheritance. The name service, obviously, is distributed, and knows directly or by asking another name service where each object and its parents and methods can be found. (Implementation details vary). Here are three approaches to object registration:

- Create the class hierarchy automatically on a fine-grained basis when the objects are (incrementally) compiled using a special pre-processor. Bull, NCR

- Register only the objects destined for remote access separately and explicitly using a separate class or interface definition language. APM, DEC, Sun/HP

- Use an API to the broker to register remote objects. The APIs used by DEC and HyperDesk are imperative, with the result that they allow last-minute additions or changes of classes without recompilation (but with offsetting possibilities for type errors). APM, DEC, DSET, HyperDesk

HyperDesk uses an API, but it can also use external databases and name services (object managers). Although DSET uses APIs for registration, its interfaces and methods are defined in two separate languages, CTX and MSL.

To clarify, consider this (inexact) analogy to text-retrieval systems: The automatic approach corresponds to automatic indexing when you store text. The class-definition language approach is akin to creating file names and possibly key words when you store a document; the API approach is akin to creating file names or keywords at any time. You could automatically identify each paragraph, say, as a numbered component of the file; the process need not be totally manual. (A final approach to text-retrieval is full-text search where you go through files sequentially, which doesn't scale up well at all; you have to have immediate access to everything with no proxies, indexes or references. The equivalent is using a local, single-image object-oriented system without an ORB or database.)

Of course, there are advantages on either side (and simplicity for the no-index/single-image approach if you're operating on a small scale only). The file name approach makes it easy to add stuff, but you have to do so explicitly for each object. The automatic-registration approach is seamless, but works only for new objects created/compiled within the system; you can't just incorporate new things by loading them onto your machine and listing them through an API such as DEC's. Moreover, you have to have a pre-processor to do the registration that works with the specific language and in the environment of the objects registered. However, to encapsulate an application in any meaningful way, you probably have to write some object-oriented code; that is what you would compile to register the new objects automatically in the pre-processor approach.

Release 1.0 31 March 1991
Local or remote?

The assumption of the ORB is that objects are not co-located; that is, you need something to find them and mediate among them. The Broker not only helps objects find each other; it also defines how objects should represent their interfaces (register with the Object Broker, so to speak) so they can be found and addressed by other objects, agents or services (even users!). Moreover, the objects may be operating in different environments and any details of implementation should be hidden from the other objects.

As we noted, there are two separate issues (and tasks) here. One is communications over communication channels; the second is communication across environments. Although all the systems address both tasks, there is a range of emphasis among the different proposals. Some see the interoperating systems as primarily physically remote (NCR and Bull), while others see them as mostly foreign and unwilling to rely on each other (APM). The other systems range between these extremes, yet DSET focuses mostly on communications issues, while the other systems worry more about interfaces for objects and object model interactions.

Remote or foreign?

Underlying these issues is the question of control and ownership. Object-orientation requires programmers to feel comfortable using others’ code and letting objects control the flow of events rather than writing a sequentially executed program. Use of an ORB requires that programmers and users feel comfortable relying on objects created and executing elsewhere, an even bigger cultural jump.

NCR and Bull at first appear most comfortable with this notion, but they also posit systems where the ORB controls most objects directly -- or where the ORB is internalized into the objects as a result of the pre-processor/compilation step, where ORB-compliant objects are created. Thus in these systems even the remote objects are assumed to be owned by friendly forces. NCR and to a lesser extent Bull also provide explicit support for shipping objects across systems, while the others mostly send just messages or data or references to objects. That’s part of NCR’s marshaling (whereby objects package themselves) and remote method invocation, sort of the equivalent of an RPC for objects.

APM lies at the other extreme; with the benefit of substantial experience in distributed computing, it feels uncomfortable about ceding too much control or vulnerability to the other side. For that reason, it doesn’t support inheritance across boundaries. This avoids the problem of inheritance of methods across systems which may be incompatible, to say nothing of the issues of ownership of the original, etc. This does not obviate local inheritance, within a single local system. (By contrast, NCR promises support for translating objects from one object-oriented language to another as part of its support for cross-platform execution.)

Do you really want to use an object that depends on some remote class over which you have no control? asks APM. The purists would say yes, that’s the point (but it requires a lot of trust). The purists might add that "local" is not a logical distinction; maybe, but it is still one that must be made, if only to separate owned, trusted environments from outside ones. This is a fundamental argument.

Release 1.0: 31 March 1991
Wholesale or retail

You get roughly the same line-up of players when you look at a related issue: objects vs. encapsulated applications and object managers. There are two ways to build "object-oriented" environments, although you can build hybrids. One is to encapsulate existing applications, and the other is to start fresh with object-oriented systems from the ground up, usually using an object-oriented language. And from the ORB point of view, you can "encapsulate" existing object-oriented systems that don't match your object model by talking to them as or through an object manager.

You can combine the two approaches as HyperDesk and DSET do, saying in effect, "Our model is that of the local object manager." But most systems have an underlying philosophy of one or the other. Is the center of interest on developing new stuff, or encapsulating old stuff? Although some systems can do both, they generally evolved from one point or the other.

Application encapsulation

The application-encapsulation approach provides for easy upgrades, reuse, and so forth. It requires less work and commitment, and also offers less of the benefits of object-orientation. (And even encapsulation requires work: You have to define the interfaces, and it may be helpful to modify the source code in places if you have access to it.) The assumption here is that objects are large -- usually entire applications and related instances stored in files. In this approach object-orientation gives you the benefits of integrating applications, managing configurations and versions, working across environments, and so forth, but most of this is external to the applications themselves. They are still monolithic units. And they are generally treated as "foreign" to each other, as described above; the ORB is used to integrate them.

Of course, to the degree that you break an application into its component methods, more and more object-oriented features come into play and the methods integrate into a class hierarchy. For example, DEC's ACA Services, an application-integration-oriented system, allows not just for dynamic binding but also for inheritance of methods from parent classes (once they are defined as such). Others with emphasis on application integration are HP and Sun, APM and HyperDesk. All, of course, will support increasingly granular objects as the market moves that way.

Objects all the way down (or up?)

The second approach is "true" object-oriented programming, with support for multitudes of small objects, usable by all applications (or what pass for them in an object-oriented world). In this model, each "application" is in fact a collection of methods, strung together for the occasion. You needn't address a whole "application" at a time, and several "applications" can share methods. (This saves development time, increases consistency, and reduces memory use and code size of installed systems.) Here, a single object may be only a paragraph, a circle or a data element; but an entire document may consist of thousands of objects. NCR and Bull are the most vigorous practitioners of this approach (with HyperDesk, as usual, offering both approaches). Of course, they too can provide encapsulation for existing applications, but that's not their specialty.
Integration of other modules (MOMs and other strangers)

Because the systems we're considering are all modular and object-oriented, they can all talk to anything else by encapsulating it. More specifically, all the ORBs can call on the services of object managers, remote ORBs, name services, trading services (for finding objects by attribute rather than name) and the like. An object manager could also be something such as a spreadsheet, which handles cells and various other spreadsheet objects, a word-processor, or other application. Or it could be an object-oriented database. All these represent different ways of encapsulating functionality, and using it to manage objects, providing services to the ORB as well as to other objects.

DEC, Sun/HP, DSET, APM and HyperDesk explicitly support the use of outside object managers; Sun even defines a Manager of Object Managers. Outside object managers offer additional flexibility; the ORB can talk to an OM instead of an object directly, thus allowing a greater variety of objects and object models to interact easily. The disadvantage can be excessive overhead, plus least-common-denominator interaction among incompatible object managers. On the other hand, you get the ability to bypass the ORB for interaction within OMs and between compatible ones.

The use of object managers avoids the global unification problem, where a system optimized for all things is optimized for none. Rather, it allows the federation of individually micro-managed systems by a global administration system that keeps out of the details.

All systems support OMs (if only as objects), but with differences:

- The ORB manages through OMs, which manage local objects. The MOMs talk to each other, but operate differently locally. (This is akin to the large-object, application-oriented approach. The OMs could be local file systems, naming services, all manner of object managers, including OODBs.) The world is composed of foreign systems tied together by the ORB. HyperDesk, Sun/HP, DEC, ANSAware, DSET

- The ORB manages objects, one of which could be an object manager. Generally, it itself uses an OODB to manage/store the objects. The world is one system, but it can incorporate some foreign systems. NCR, Bull

Object-oriented databases as a special case

Object-oriented databases are a special case of object manager, since they are so powerful. There are several possibilities for ORB/OODB integration:

- The ORB knows how to talk to (can pass messages to objects within) an OODB, maybe to several. (OM-oriented)

- The ORB uses a database to manage its location and naming system; the database happens to be object-oriented, and is part of Object Services. (OODB-oriented)
The two are tightly integrated; the act of registering an object in the OODB simultaneously registers it with the ORB and vice versa. The OODB is part of the Broker (whatever that really means). The ORB's class-definition language is the same as the OODB's data-manipulation language. And in the case of NCR or Bull, they can be the same (with a pre-processor) as the local object-oriented application language. (OODB-integrated)

Finally, you could have an ORB that uses a non-object-oriented database to manage its objects (e.g. a name service). (neither OM- nor OODB-oriented)

What makes OODBs special is that they explicitly manage the creation, storage and integrity of persistent objects, which are "normally" just stored in flattened form in files on disk, and managed by the object-oriented application or environment that created them. Thus an OODB and an ORB could each use each other to make a full-fledged version of either: The ORB adds distribution to an OODB; an OODB could be a high-level name service and object store for an ORB.

The problem with a separate ORB and OODB is that you may end up with three languages: for the application, for the ORB and for the OODB. Are persistence and remoteness part of an object's nature, or are they orthogonal conditions requiring a separate language?

OODB or ORB

- finds persistent objects
- maintains relationships between objects & methods
- maintains relationships among objects (inheritance, composition, etc.)
- requires communications layer for cross-platform operations

OODB only

- manages transactions, versions, etc.
- "assumes" many objects are co-located
- finds remote/stored objects to activate and come and execute locally

ORB only

- leaves transactions, integrity, etc. up to clients
- "assumes" most objects are remote (applications)
- sends messages for remote objects to execute remotely (usually)

The point of it all is cooperation...

...among objects, among models, even among vendors. Explicitly, the ORB spec makes no assumptions about the nature of the objects themselves, or the primacy of communications protocols vs. object-model interoperability, but it is inevitable that any working system do so. Do you want application-level or object-level interoperability?

The way the ORB works, you could easily have more than one, interoperating. Each would in effect see the other as a foreign object manager. Accordingly, would it make sense to have two kinds of ORB, one for distributed...
logical-local and the other for logical-remote (or foreign)? One handles remote communications among similar objects and has a consistent object model (a la NCR); the other handles communications among different object models, with communications transports a secondary issue (the OM-oriented crowd). One is not "higher" than the other; they simply focus on different issues. One distributes a single object system, while the other federates disparate object systems.

Certain vendors on either side of the spectrum might in fact team up, offering some kind of automatic transmission with two approaches, so that there's no explicit shifting of gears. (This would be a complementary match rather than two companies refining the same proposal.) On the other hand, a single spec may be far broader in the range of situations it can handle than any single implementation might indicate; the bathwater may change around the baby. Consider Sun and HP.

In the end, most submitters have said they'll have little trouble adopting most other submissions. It's more important to have an answer than which answer is picked, and implementations can vary a lot (and won't necessarily interoperate). Besides, anything of your own that you particularly like, you can preserve one level down, in an object manager addressed by the manager of object managers. So you can support the ORB, but leave your "real system" intact.
THE SUBMISSIONS, ONE BY ONE

APM (ESPRIT) ANS Aware -- Distributed is our game

ANSAware is the work of a loose European research consortium, and is part of the Esprit program. Members include AEG, British Telecom, Chorus Systemes, DEC, Dowty/CASE, Ericsson Telecom, France Telecom, GEC, HP, ICL, Olivetti, Philips, Plessey, Racal, Siemens, STC, Televerkert, and Thomson-Syseca, with additional funding from various government bodies. The project began six years ago as part of Britain's Alvey Program with the goal of a vendor-neutral (i.e. open) architecture for distributed computing. The goal was to accommodate existing systems in an environment that could enable them to work together. The project survived the end of Alvey in 1989 to gain funding and support from Esprit. The work is actually being carried out by Architecture Projects Management Ltd., based in Cambridge, England, and devoted to this mission. APM has 20 staff members, along with seven people on loan from the project member companies. The company is involved and influential in a wide number of standards bodies and efforts.

ANSAware manages remote communications across virtually all standard protocols. The system is C-based, and works across a variety of environments, including UNIX, MS-DOS and VMS, with ports to many UNIX systems.

It is basically a distributed computing environment for objects -- with two major differences. APM started out to solve the problems of distributed computing, and quickly came to some real-world conclusions: The world cannot be a single, seamless environment, and let's not pretend that it is. (The same kind of thinking is showing up in the distributed database world, where it's becoming that the simplifications of a fully distributed, full-integrity may be more than warranted by the real world.) Sometimes it's better to have conflicts and reconcile them explicitly later, rather than eschew them entirely. Sometimes there are conflicts in the real world. In the same way, sometimes objects are remote and controlled by others; perhaps that should be acknowledged rather than hidden.

Thus, says APM's Andrew Watson, APM takes a slightly different approach to objects from others. ANS Aware does not support inheritance across environments, not because of the overhead, which may indeed be high, but because of higher-level concerns about ownership, control and reliability. Fundamentally, global consistency is nice, but it's not realistically achievable, and the thought of inheriting methods willy-nilly from objects you don't own properly appalls many real-world systems manager. (Local inheritance is fully supported within object-oriented systems connected across ANS Aware.)

Secondly, says Watson, "classical" object-orientation accomplishes two purposes: hiding of implementation details, and provision of services. Rather than combine the two purposes, ANS suggests that you retain encapsulation for information-hiding but allow a single object to have two or more interfaces for different kinds of users (that is, you needn't stuff all the possible methods into a single interface to the object). For example, there might be one printer interface for user methods (print and various font parameters, for example, or use stationery) and another for manager methods (report on usage statistics, offload to another printer, etc.). The benefits are a better matching to the real world and the ability

Release 1.0 31 March 1991
### Table of generalizations

<table>
<thead>
<tr>
<th>company</th>
<th>object style</th>
<th>OSes</th>
<th>how to register</th>
<th>o-store</th>
<th>comms</th>
</tr>
</thead>
<tbody>
<tr>
<td>APM</td>
<td>interfaces</td>
<td>UNIX, VMS, DOS</td>
<td>IDL or API</td>
<td>any</td>
<td>mult. stds</td>
</tr>
<tr>
<td>Bull</td>
<td>objects</td>
<td>UNIX</td>
<td>pre-p</td>
<td>own or any</td>
<td>DCE</td>
</tr>
<tr>
<td>DEC</td>
<td>apps &amp; components</td>
<td>Ultrix, VMS, DOS, OS/2</td>
<td>CDL or API</td>
<td>files, DEC Name Svc, any</td>
<td>RPC</td>
</tr>
<tr>
<td>DSET</td>
<td>OMs and objects</td>
<td>UNIX</td>
<td>API</td>
<td>any</td>
<td>OSI, TCP/IP</td>
</tr>
<tr>
<td>HP/Sun</td>
<td>apps or New-Wave objects</td>
<td>UNIX</td>
<td>CDL or API</td>
<td>own or any</td>
<td>RPCs</td>
</tr>
<tr>
<td>HyperDesk</td>
<td>objects &amp; OMs</td>
<td>DOS in lab, OS/2, UNIX</td>
<td>API or OM syntax</td>
<td>OODB or any</td>
<td>RPCs</td>
</tr>
<tr>
<td>NCR/ODI</td>
<td>objects</td>
<td>UNIX, OS/2 DOS (Win)</td>
<td>pre-p</td>
<td>ODI or any</td>
<td>RMI</td>
</tr>
</tbody>
</table>

### How to read this chart...

**Object style** is the kind of object the system expects. It is not a firm requirement, but more a flavor of the system. Apps means the system is optimized for applications as objects, and calls between co-located objects usually bypass the ORB. Interfaces means it calls interfaces rather than discrete objects. Objects means it deals with "traditional," granular objects.

**OSes** shows which operating system the current implementation works on. Most vendors promise to support UNIX, OS/2 and DOS Windows eventually. (This has nothing to do with the ORB spec, but just with current implementations; likewise the communication medium.)

**How to register** shows whether the system uses a specific, separate language to register objects with its broker, or whether the registration is accomplished during a preprocessing step, or it simply calls an ORB API. IDL is APM's Interface Definition Language; CDL are the (separate) DEC and HP/Sun Class Definition Languages; CTX is DSET's Context (or interface) specification language. Pre-p is a pre-processor that automatically registers objects during the process of compilation in an object-oriented language such as C. HyperDesk uses a combination of calls to its broker API or local object manager syntax.

**O-store** shows the database or file system or name service the broker implementation typically uses to store objects or references to them. Most have a replaceable store, and can use "any."

*Release 1.0 31 March 1991*
Communication medium is the communication medium the implementation uses, but (except for NCR) it is not part of the spec. NCR's RMI could also run over RPCs, and the others could broaden the range of RPC protocols they use.

What the table leaves out: Claims of performance are hard to assess, let alone compare. Most vendors said it would take 10 to 20 milliseconds to execute an object broker call between two SPARC-range systems; APM says 5 milliseconds, but we're loath to cite numbers in a table because of the multitude of factors involved that probably are outside the range of the ORB itself. Overall, this doesn't appear to be a decisive factor, although it may matter to purchasers of the resulting products. The number of objects a single system could manage also varies and depends on the use of object managers and the resources available. It is not generally a specific number, but an issue of scale and overhead.

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

to allocate permissions for security or other reasons. The problem is knowing if you've addressed a single object instance or several, but presumably you configure interfaces so that's not a problem. (Don't have credit in one interface and debit in another.) APM states it positively: Objects are just single-interface ANSA objects.

Otherwise, this is an elegant distributed computing platform which enjoys the research support of numerous companies, including (indirectly at least) HP, DEC and Bull through their membership in various consortia. It is being used at NASA to link 1500 scientists with a number of large databases containing mostly image data from five satellites. An astrophysicist at one machine now has access through the same user interfaces to images stored in a variety of formats and environments around the world.

BULL OF FRANCE -- French elegance, misses on timing

Bull's ORB submission is part of Comandos, an Esprit project started in 1986 for which it is prime contractor. Comandos (CONstruction and MANagement of Distributed Open Systems) overall is closest to NCR's Cooperation in flavor. It is currently in its first release, and so far has been used for application development only within the 11 member companies of Esprit. Its ORB proposal describes Release 2, which Bull bills as its industrial system, for availability in early 1992 -- missing the OMG deadline. "We just wanted to take a shot at [the ORB selection], and raise awareness both inside and outside Bull of what we've got," says Bull technical staff consultant Chris Anderson. In fact, it was a pleasant surprise all around. Although Bull originally wasn't "fervently hoping to win," the company was pleased at its showing and may hasten commercialization of the product.

Comandos provides seamless object distribution for its native language, Guide, and semi-transparent support for C++ and Eiffel (fully transparent soon for C++). Guide combines elements of Modula-2,CLU and Eiffel to produce an elegant, readable but slightly academic-flavored language. Guide is implemented as a compiler preprocessor that generates straight C code. Its support for C++ and Eiffel is implemented in the same fashion, with a preprocessor that generates a class hierarchy for an object database (thereby registering objects with the ORB) automatically during the processing.

Release 1.0 31 March 1991
Comandos is focused on efficient manipulation of fine-grained objects for multiprocessor or LAN environments, and thus would fit nicely as the more local object manager inside a wider OO world -- and would meld well with (rather than compete with) an application-oriented ORB along the lines of DEC's or HP/Sun's. For example, you could put DCE (the OSF's communication system) underneath, and install a Comandos system as a manager of object managers under, say, the HP/Sun object broker.

DIGITAL EQUIPMENT ACA SERVICES -- Applications are our business

DEC's ACA (Application Control Architecture) Services is a second-generation product, the outgrowth of technology originally developed to support DEC's Compound Document Architecture. The current ACA release goes well beyond that to include support for distribution and cross-platform support. That is, CDA defines the files that store document components with related information (but not code); ACA Services implements activation and distribution of services to act on those files (or other data files) using a class/method hierarchy of applications and components. However, ACA Services does not manage the instances (with the individual object data) themselves, so it doesn't maintain the state of the objects. That is, it can call the applications' methods, and do dynamic binding and multiple inheritance and other OO functions, but it doesn't know about what the methods are acting on.

ACA Services puts wrappers around applications, targeted at inter-application communication and integration, although it can also act as the support environment for new object-oriented applications/classes in a variety of application areas. It uses a class definition language to build its class hierarchy, which by default is either files (as in typical CDA applications) or, in the forthcoming second release with an object broker for distributed implementations, the DEC Name Service or any other name service. Classes can also be added to a running system in real time by calls to the ACA Services runtime system API.

Like most of the OMG submissions, ACA contains much more than just an ORB. For example, it offers users and integrators the ability to build and execute scripts, and it manages contexts (somewhat akin to ANSA's interfaces), which set defaults in a particular situation. To do this, the system selects methods according to properties maintained in a context object (Juan likes the printer with fancy stationery and PostScript output; Alice prefers the fast, cheap one near her office).

ACA Services itself is written in C (for portability, among other things), and can support classes written in any language (but declared through the ACA class definition language or calls to the ACA API). The system is implemented in Ultrix, OS/2 and VMS, with DOS/Windows support planned. In version 2, it can also run over any transport mechanism, although the supported default is DEC's RPC, soon the OSF DCE.

ACA Services version 1 is already in use internally and in products such as DECwrite and DECdecision, with LiveLinks, with more than 10,000 licenses in force. The second release has been in field tests since December at 45 sites, including 16 ISVs, 13 CASE users, 4 manufacturers and 8 aerospace companies. We expect it to be available commercially later this year, although DEC doesn't give dates.

Release 1.0 31 March 1991
DSET (Distributed Software Engineering Tools) -- Interoperability is key

DSET is a New Jersey-based communications and object-oriented tools company with six technical people out of a total of nine, founded in 1989 by Dan Shia. He had spent the previous ten years working on UNIX internals and data communications at Bell Labs and Bellcore. DSET focuses on selling tools to vendors of OSI communication systems and network management systems, including Bell Labs and Bellcore, GM (for use in MAP installations), Boeing and CASE vendor ISSI.

The company is steeped in OSI communications philosophy, but it also seems to understand and handle better than anyone the practical issues of realtime communications: allocating access, managing diverse channels, dispatching, parallelism, multiple threads and the like. As practiced in heavy-duty environments such as telecommunications, these are black art rather than abstract theory. Accordingly DSET provides the broadest range of sheer communications power, with support for synchronous, asynchronous, connection-oriented and connectionless invocations (i.e. continuous or sporadic, each best for different situations).

It uses objects heavily in its implementation, Distributed Systems Generator, basing its architecture on MIT's Actor model, with strong support for concurrency. Its "Active Objects" are akin to object managers in other vendors' terminology, and could be OODBs, applications or other small-grained ORBs. (Overall DSET's use of terminology has tended to obscure understanding of its submission by non-Actor/OSI experts.)

Like HyperDesk, DSG has an overall modular flavor, and leaves object management up to object managers (although HyperDesk provides more extensive back-up services for systems without their own object managers).

Overall, Shia firmly believes that the ORB should be implemented as a two-level system, one level for communication among object managers and the other to manage the objects themselves. DSET believes it's premature to focus on anything having to do with the objects themselves, and has confined its efforts in DSG to its vision of the ORB spec, which is interaction between object managers. On the other hand, it does provide strong guidelines for the underlying communications, providing a protocol as well as an interface spec. Thus, full interoperability among object managers would be guaranteed if everyone followed this spec.

In other words, DSET specifies extensible underlying communications protocols -- over either ISO or TCP/IP -- with compiler and API support. In essence, the other vendors (except NCR) are saying, if you use a compatible communications medium, any two implementations of our ORB can communicate. DSET is saying, if you follow our spec, you will have a compatible communications medium, and thus full interoperability among multiple vendors' implementations of the ORB. (But at this point many people would rather choose their own communications layer, and the OMG has enough to do without addressing communications standards.)

Interfaces are defined by DSET's CTX (ConTeXt definition language), while behavior (methods) is defined by MSL (Manager Specification Language). However, DSET uses an API to register them with its object broker and to create and destroy instances. Both the languages support multiple inheritance and conform to various ISO standards.

Release 1.0 31 March 1991
In its more public statements about the Distributed Object Management Facility, HP tends to emphasize NewWave, with more than 100 corporations using it on a significant scale: DOMF "builds on the success of a mature, industry-acclaimed object-oriented application environment: HP NewWave." In more technical briefings, however, HP properly stresses that the DOMF is totally new technology, focused on distributed objects (unlike the so-far single-machine NewWave), and it will eventually be integrated with NewWave; it is not an outgrowth of it: "No code is shared. No design documents are shared. Only the vision..." (That is, both NCR and HP use NewWave for DOS Windows clients, but their object models are independent of it.)

In fact, DOMF resulted from two separate efforts that have been joined under a single spec. Some time ago, Sun had been working on its intertool communication facility, which is the basis of a forthcoming CAD Framework Initiative demonstration. Meanwhile, HP was working on a distributed Object Management Facility to use for a new distributed version of NewWave. Sun had networking; HP had objects. Somehow the two companies got together (it started with a conversation between Sun's Dave Cardinal and HP's Bob Frankenberg at the 1989 PC Forum). The joint project hit something of a hiatus around the Apollo acquisition, which gave HP a much stronger position in the distributed computing area. Still, by then the relationship was entrenched, although the two companies each now offer their own network underpinnings.

The process they went through is akin to what may happen as the OMG defines a spec from the selected submission implementation, although in this case defining the spec was a two-way effort. "We picked up HP's architecture document and did some intense work to abstract it," says Richard Probst, Sun's manager of distributed applications engineering. "We had many religious battles, and the result is purified by fire."

Thus the DOMF spec, announced jointly just last month, is uniquely implementation-independent. For starters, each company will ship an RPC-specific implementation for its own RPCs, with a compiler for RPCs either for NCS (Network Computing System) or ONC (Open Network Computing). NCS is the distributed computing environment now implemented in the OSF's Distributed Computing Environment (DCE); Sun's ONC is the standard for System V.4 (the AT&T/Sun camp).

They have a common vision of the ORB as a manager of object managers, which makes it easy to accommodate different local object systems. It's modular and provides for interfaces to external services, such as X.500 for object locations or a local Manager of Object Managers. To register objects with the DOMF, you use CDL, a C++-like language, or an API. This registration is a formal process, and is relatively static compared with, say, ACA Services.

The two systems are also different in their naming schemes (unique IDs for HP vs. names for Sun), and in their optimization for object size and granularity: Sun favors very small objects, such as spreadsheet cells and the ECAD gates (as in the CAD Framework), while HP favors large numbers (millions) of objects of any size. (Remember, you're sending messages to them, not necessarily shipping them around a network.) HP has a hierarchy of location services that can handle large numbers across enormous networks, with Object Region Experts, MetaObject Region Experts and Sibling MOREs. On
the other hand, its basic assumption is that an object is a file, whereas
the Sun implementation expects a file to contain many objects. (Each can
handle the non-default case, but their characters do differ tangibly.)

Finally, each uses separate security mechanisms: HP uses DCE's security and
RPCs; Sun uses ONC's Secure RPC and will later be able to use Kerberos.

HYPERDESK -- "The more a local system does, the less we do."

HyperDesk is the company that recently spun off from Data General when DG
decided to give it further funding. In fact, that's probably the best
thing that could have happened to it. It now has a reported $4 million in
funding from Kay Nishi's ASCII Corp. in Japan, and is free of the con-
straints of a large, struggling hardware parent.

HyperDesk offers one of the more complete software environments we have
seen, with plans for a full range of OA-oriented class libraries (not part
of the ORB, of course). The basis of the system is to act as glue for a va-
riety of existing applications, and then to add its own object-oriented
suite of office automation classes. Users can use their old tools, such as
XyWrite or dBASE, or switch to HyperDesk's more tightly integrated ones.
Although the positioning is different (Clarity talks about OA first, and
then objects only if you ask), the flavor of what users see will probably be
close to Clarity's Rapport (see Release 1.0, 2-91), or DEC's All-in-1 as en-
hanced with Compound Documents or IBM's OfficeVision (someday).

Technically, HyperDesk is written in C for portability; the classes it man-
ages can be written in anything supported by the local hardware. HyperDesk
fervently believes in a consistent object model, built out of objects; that
is, start with a single class hierarchy and specialize or change from that
in a modular way. In practice, the approach is so modular and extensible
that it imposes almost no constraints on what can fit inside that hierarchy.
As a small company, HyperDesk makes a point of being modestly open to third
parties. Thus you can register objects almost any way -- by declaring them
to the Broker with an API, by declaring an object-oriented database that
contains them to the API, or by registering an object manager with the API.
HyperDesk's Broker knows how to manage objects, dynamic binding, inheri-
tance, context management (so that methods are selected and bound according
to a client's or user's "context"), etc. But it is also comfortable leaving
those functions to a local object manager to the extent that the local OM
can take on the work.

HyperDesk is so modest that it leaves almost everything up to the user's
choice. This is truly a skeleton, with a little optional flesh. The flesh,
however, is very nice.

NCR/OBJECT DESIGN -- Objects' rights

NCR's ORB technology is part of Cooperation, its object-oriented environ-
ment. It uses NewWave to encapsulate existing applications and for pre-
sentation, but goes well beyond that with its own underlying object-oriented
system. For better or worse, NCR is the most aggressive in advocating its
own object model; it is the prime example of a single system distributed
across the world. It can handle other object managers, but mostly as encapsulated foreign bodies.

Whereas many of the submissions say, more or less, "Leave it up to the object manager," NCR's Cooperation says, "Leave it up to the objects themselves." It provides communication facilities at the object level, rather than funneled through a central ORB. Each object has (inherits) its own methods for rendering its messages and arguments into streams for communications and persistence (storage as in an OODB, not in flat files but as objects). This is the basis of the Remote Method Invocation communication protocol, which is linked to the object itself rather than managed as some externally imposed scheme, says NCR's Nelson Hazeltine, director of the Cooperative Computing Systems Division. This dynamic linking of communications, vs. compiled RPCs and optional registration with the broker through a separate process, allows for greater flexibility.

Of course, Cooperation objects and the Cooperation broker can communicate with other systems as foreign objects, using the NCR ORB as a sort of neutral carrier. Because it is modular and encapsulating, Cooperation can handle anything, but its real value comes with tight integration and wholesale adoption. For other vendors to use and benefit from this system would require close adherence to an all-encompassing spec that could devalue existing work. That's always the trade-off; it's the same price you pay for using object-orientation in general.

Although it is C++-based, the system will offer the same kind of automatic registration of objects for other object-oriented languages such as C++, Smalltalk and C. Moreover, NCR plans to provide cross-language translation so that objects can automatically rewrite themselves from language to language as they move from environment to environment. (Most other systems concentrate on sending messages; Cooperation, because of its object marshaling, is able to send objects, not just pointers or references or data streams, around with ease.)

NCR has been joined in its submission by Object Design Inc. (see Release 1.0, 9-90). Cooperation can use ODI's ObjectStore OODB for superior performance where clustering (efficient storage) of small-grained object structures is appropriate. ObjectStore also brings transaction management and a nice versioning system to the party, says Hazeltine.

Meanwhile, there are some interesting complementary moves in this field outside the ambit of the OMG. On the one hand, there's Microsoft's Object Linking and Embedding, which is about as powerful as the equivalent feature in NewWave, scheduled for wide availability in Windows 3.1. It allows you to embed an object or a link to one within a Windows application file, but it assumes the applications exist on the same machine. OLE right now typically loads an entire application, but it can handle single methods through a dynamic link library. It uses the DOS file system to manage objects, and users must take the trouble to track the links when objects are moved.

Right now, OLE is Windows-specific; that is, Windows supports it, and nothing else does. However, an OS/2 version is planned, and there's no reason

Release 1.0
31 March 1991
you couldn't put OLE support into, say, UNIX or GO's PenPoint. Microsoft has already developed a version of OLE for the Mac, if only for its own applications, although we're sure it will prove popular with other cross-platform developers. However, you'd still need translation of executable binaries for the code, and a transport substrate (for cross-platform execution, as opposed to separate versions of an application for each platform).

Microsoft has the opportunity, based on its control of the de facto operating system standard for PCs, to establish a local standard without worrying about standards bodies. It has recently joined the OMG and could also easily adopt whatever the OMG proposes for an ORB; OLE could operate nicely as an object manager under an ORB. If Microsoft adopted the OMG's ORB standard, other systems could easily address and use Microsoft's objects (and vice versa). However, Microsoft may have other plans -- and it will certainly create a superset of the OMG spec if it uses it at all (as will every vendor, of course).

Open Sybase

Another potential player is Sybase, not a member of OMG. It does not have anything close to an ORB, but with its Open Server/Open Client interface spec, it probably has fostered one of the largest installed bases of open cross-platform object-style interfaces (much richer than SQL). A call to a stored procedure through the Open Server interface on a Sybase server is basically a call to an interface to perform a method (a transaction, typically) on an object. It's not quite true 00, and it would take some work to extend Sybase's technology into an ORB, but it might be part of one. At least, you could call Open Server an object manager and fit it in nicely under an ORB.

And the two Constellations

Patriot Partners, the IBM-Metaphor joint venture (see Release 1.0, 9-90, 2-91), has recently gone public with its object architecture model, although the implementation details are still under wraps. (This system is code-named Constellation, not to be confused with Constellation Software, below.) The goal is to build an object-oriented, cross-platform environment that would treat the local operating system as a collection of objects and allow developers to build the next generation of applications without worrying about DOS? or OS/2? or UNIX? or NT? Of course, they would have to standardize on Patriot... (They would also get the benefits of object-orientation, with easy integration of applications and so forth. All these are fairly familiar by now, at page 23!)

The Patriot model has end-users in mind. As envisioned, it will let users construct their own customized applications by visual programming, connecting graphical screen objects that have already been designed by programmers to interact according to specified protocols. (An initial implementation of this idea is Metaphor's Data Interpretation System capsules, which allow you to connect various analysis components to build a data-analysis task. In actuality, you're instructing objects to send messages to each other.) Any objects that use the Patriot protocols/interfaces can effectively work together in what Patriot calls a "builder," an object manager with constituent classes, which provides the environment within which components are con-

Release 1.0 31 March 1991
nected, and methods inherited or selected. "Components" are simply objects, built in C++ by default but optionally encapsulations of objects built elsewhere some other way.

Patriot also addresses some interesting commercial questions, about how all these things should be defined, packaged and marketed. We've discussed what is technically possible, but what is commercially possible? Builders with constituent components are what users buy instead of applications; sets of components within a builder provide the functionality for specific tasks, such as text-editing, spreadsheets, and data manipulation. As you buy more builders, you increase the range of activities you can perform on the same data rather than increasing the number of kinds of files you have to manage; that is, you could have a group scheduler, a personal calendar and a form tool that could all handle appointments (time & date objects with duration) as appointments rather than as strings.

Patriot could of course build its own ORB for distributed computing and develop gateways to other object managers. However, a standard ORB would allow Patriot to link easily to other systems. Moreover, if Patriot adopted the OMG's ORB, it could either save itself some work (if its own efforts aren't far along), or at least avoid future redundant efforts. After all, every vendor with a cross-platform system has an ORB of some form, even if it's not abstracted out into a spec. (Patriot's OMG membership is an expression of interest and a desire to have honest access to accurate information rather than an endorsement of anything, notes president David Liddle.)

Constellation Software

Constellation Software is a reconstitution of the client-server architecture group at Prime much as HyperDesk comes from the object-oriented group at Data General. It was formed early last year by six Prime employees with a simple mission -- to reuse applications to build effective integrated systems. The best way to do that is to use object-oriented technology, encapsulating existing applications, and extend them with object-oriented modules (C++ mostly, as a pragmatic decision). The company's first product, being resold by an unidentified hardware company, is now in pilot use at customer sites. It incorporates a number of different hardware and software platforms and imaging technology, but appears to users as a single system.

The Constellation team has long applied object technology to solve interoperability problems. It used existing technology and implementations to the extent possible, but the group also developed its own general object transfer mechanism. It submitted that mechanism to the OMG as part of the ORB technology in hopes that it could find a partner (or the OMG would find it a partner) to complement its piece. That hasn't happened, and because the OMG is focused on complete submissions, Constellation has withdrawn from the bidding. However, it remains active in the process, and is eager to become a user of whatever technology is adopted. "We've been vocal in the process," says principal architect Lee Scheffler, "because we'd rather buy it than build it, and we want some say in what it is. We represent a well-informed viewpoint as consumers of this technology."

Release 1.0 31 March 1991
RESOURCES & PHONE NUMBERS

John Bull, Michael Eyre, Architecture Projects Management Ltd., 44 (223) 323-010; fax, 44 (223) 359 779; or 44 (293) 51-88-55 x 401
Chris Anderson, Groupe Bull, (508) 294-5773; fax, (508) 294-2619
Roland Balter, Gerard Vendome, Bull of France, 33 (76) 54 49 12
Lee Scheffler, Constellation, (508) 620-1651; fax, (508) 879-8594
Liz Freburger, DEC, (603) 881-2669; fax, (603) 881-0120
Dan Shia, DSET, (908) 832-6533; fax, (908) 832-6523
Mike Mathews, Hewlett-Packard, (408) 447-1959; fax, (408) 447-4729
Jim Waldo, Hewlett-Packard/Apollo, (508) 256-6600; fax (508) 256-2384; waldo@apollo.hp.com
Herb Osher, Bill Andreas, HyperDesk, (508) 366-5050; fax (508) 898-3841
Greg Whitten, Ed Jung, Microsoft, (206) 882-8080; fax, (206) 883-8101
Nelson Hazeltine, NCR, (803) 796-9740; fax, (803) 739-7745; nelson. hazeltine@columbia.ncr.com
Tom Atwood, Object Design, (617) 270-9797; fax, (617) 270-3509; tom@odi.com
Chris Stone, John Slitz, Object Management Group, (508) 820-4300; fax, (508) 820-4303
David Liddle, Patriot Partners, (415) 961-3600; fax (415) 966-1637
Jacob Stein, Servio Logic, (503) 629-8383; fax, (503) 629-8556; steinj@slc.com
Dave Cardinal, Sun Microsystems, (415) 336-3703, 960-1300
Richard Probst, Sun Microsystems, (415) 336-2999; fax, (415) 961-0992
Bob Epstein, Sybase, (415) 596-3500; fax, (415) 658-9441

Each vendor provides ample background information with its submission to the OMG. And of course the OMG's Object Management Architecture Guide provides the necessary broad, impartial overview. In addition, we found useful insights in several technical backgrounders from Constellation Software about some practical concerns in object-oriented development.

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

Release 1.0 is published 12 times a year by EDventure Holdings, 375 Park Ave., New York, NY 10152; (212) 758-3434. It covers pcs, software, CASE, groupware, text management, connectivity, artificial intelligence, intellectual property law. A companion publication, Rel-EAST, covers emerging technology markets in Eastern Europe. Editor & publisher: Esther Dyson; associate publisher: Daphne Kis; circulation & fulfillment manager: Lori Mariani; executive secretary: Denise DuBois; editorial & marketing communications consultant: William M. Kutik. Copyright 1991, EDventure Holdings Inc. All rights reserved. No material in this publication may be reproduced without written permission; however, we gladly arrange for bulk purchases or reprints. Subscriptions cost $495 per year, $575 overseas.
## RELEASE 1.0 CALENDAR

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 16-17</td>
<td>LAP &amp; PALMTOP ’91 - New York City. Sponsored by Laptop Expositions. Keynote by Kathy Vieth, IBM. Call Peter O’Connor, (212) 682-7968.</td>
<td></td>
</tr>
<tr>
<td>April 17</td>
<td>Information technology and coordination in organizations of the 1990s symposium - Cambridge, MA. Sponsored by MIT’s Industrial Liaison Program. Speakers include Tom Malone, Lester Thurow, Stuart Madnick, John Rockart and Esther Dyson. Call Maria Clara Martin, (617) 253-0213.</td>
<td></td>
</tr>
<tr>
<td>April 22-24</td>
<td>Reverse engineering forum: Capturing value - St. Louis. Sponsored by Washington University. With Elliot Chikofsky; Charles Bachman; David Sharon, CASE Associates; Stephen Errico, Price Waterhouse; Dina Bitton, DB S’ware; James Miller, Andersen Consulting. Call Elizabeth Hartog, (314) 889-5875.</td>
<td></td>
</tr>
<tr>
<td>April 24-26</td>
<td>*National Venture Capital Association annual meeting - Washington, DC. Panel with Esther Dyson, Dick Shaffer. Call Bruns Grayson, (301) 727-1700.</td>
<td></td>
</tr>
<tr>
<td>April 26</td>
<td>Third Computer Bowl - San Jose. With Bill Gates, Heidi Roizen, Philippe Kahn, Dave Liddle, David House and Ed Juge from the West (including Fort Worth!); Pamela McCorduck, John Armstrong, James E. Clark (AT&amp;T, not Silicon Graphics), Sam Fuller and John Markoff from the East. Sponsored by the Boston Computer Museum. Call Gail Jennes, (617) 426-2800.</td>
<td></td>
</tr>
<tr>
<td>April 28-May 1</td>
<td>*Borland languages conference - San Francisco. Come C what’s so exciting about languages! Call Kathy Bentley, (408) 438-8400 or (800) 946-TURBO.</td>
<td></td>
</tr>
</tbody>
</table>

Release 1.0 31 March 1991
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Location</th>
<th>Sponsor and Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 30</td>
<td>TELECOMMUTE '91: Laying the foundation - Oakland. Sponsor: Nichols International. Call Larry Nichols, (415) 797-4560.</td>
<td>Oakland</td>
<td></td>
</tr>
<tr>
<td>May 1-3</td>
<td>APPLlCASE '91 - San Francisco. Sponsored by Ernst &amp; Young and Information Week. Keynotes by DuWayne Peterson, Merrill Lynch; Vaughn Merlyn. Call Ellen Douglas, (201) 402-2637.</td>
<td>San Francisco</td>
<td></td>
</tr>
<tr>
<td>May 12-13</td>
<td>THE THIRTEENTH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - Austin, TX. Sponsored by ACM, IEEE Computer Society. Call Barbara Smith, (512) 338-3336.</td>
<td>Austin</td>
<td></td>
</tr>
<tr>
<td>May 15</td>
<td>PC USER GROUP MEETING - New York City. With Jerry Kaplan and Robert Carr, GO. Call John McMullen, (914) 245-2734.</td>
<td>New York City</td>
<td></td>
</tr>
<tr>
<td>May 19-22</td>
<td>INTERNATIONAL MARKUP '91 - Lugano, Switzerland. Sponsor: Graphic Communications Association. SGML etc. Keynote by Esther Dyson. Call Joy Blake, (703) 519-8160.</td>
<td>Lugano</td>
<td></td>
</tr>
<tr>
<td>May 19-23</td>
<td>INTERNATIONAL DB2 USERS GROUP: DISTRIBUTING THE EXPERIENCE - San Francisco. Speakers include Chris Date, Michael Stonebraker. Call Larry Fleischman, (312) 644-6610.</td>
<td>San Francisco</td>
<td></td>
</tr>
<tr>
<td>May 20-23</td>
<td>SPRING COMDEX - Atlanta, GA. Sponsored by the Interface Group. Call Elizabeth Moody, (617) 449-6600. Includes Windows World; coincides with Interface/91.</td>
<td>Atlanta</td>
<td></td>
</tr>
<tr>
<td>May 21-23</td>
<td>UNIX &amp; OPEN SYSTEMS: APPLICATIONS, TOOLS &amp; SOLUTIONS FOR THE '90S - Santa Barbara. Sponsored by Patty Seybold, UniForum and X Open. With David Stone, DEC; Peter Weinberger, AT&amp;T USL; Ira Goldstein, OSF; Pete Peterson, WordPerfect; Charles House, HP. Call Deborah Hay, (617) 742-5200.</td>
<td>Santa Barbara</td>
<td></td>
</tr>
<tr>
<td>May 21-23</td>
<td>SILICON GRAPHICS DEVELOPER'S FORUM - San Francisco. Sponsored by Silicon Graphics. Call Debbie Chen, (415) 335-1392.</td>
<td>San Francisco</td>
<td></td>
</tr>
<tr>
<td>June 3-7</td>
<td>OBJECT WORLD - San Francisco. Co-sponsored by The Object Management Group and World Expo Corp. Businesspeople's answer to OOPSLA. Call Dave Bradway, (508) 820-8123.</td>
<td>San Francisco</td>
<td></td>
</tr>
<tr>
<td>October 6-11</td>
<td>*OOPSLA '91 - Phoenix. Sponsored by ACM. Call John Richards, (914) 784-7731.</td>
<td>Phoenix</td>
<td></td>
</tr>
</tbody>
</table>

*The asterisks indicate events we plan to attend. Lack of an asterisk is no indication of lack of merit.

Please let us know about any other events we should include. -- Denise DuBois
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 rate on request. Satisfaction guaranteed or your money back.

Name ____________________________________________________________

Title ____________________________________________________________

Company _________________________________________________________

Address _________________________________________________________

City ___________________________ State ______ Zip ____________

Telephone ________________________________

How did you hear about **Release 1.0**?

[ ] Payment enclosed.

[ ] American Express #_________ - ___________ - _______ Expires _______

Signature _______________________________________________________

[ ] Please send me additional information on your multiple-copy rate.

Please fill in the information above and send to:

EDVENTURE HOLDINGS INC.
375 PARK AVENUE, SUITE 2503
NEW YORK, NY 10152

If you have any questions, please call us at (212) 758-3434.

Daphne Kis
Associate Publisher