

Technology Briefing report on distributed object technologies

By Gilbert Rondeau and Jérôme Guionnet, Cap Gemini Innovation

1. OVERVIEW	1
1.1. Corba	2
1.1.1. Object Request Broker	3
1.1.2. Object Services	3
1.1.3. Common Facilities	3
1.1.4. Interface Definition Language (IDL).	4
1.1.5. Dynamic Invocation Interface (DII).	4
1.2. OLE	4
1.2.1. Introduction	4
1.2.2. OLE structure	4
1.2.3. Discussion	5
2. ROLE IN EVOLUTION PROCESS	5
3. RELEVANCE TO RENAISSANCE	6
4. AVAILABLE SUPPORT	6
4.1. Corba availability	6
4.2. OLE Availability	7
4.2.1. OLE Strengths and Weaknesses	7
5. INAPPLICABILITY	7
6. FUTURE DEVELOPMENT	7
6.1. Corba	7
6.2. OLE	8
7. OTHER COMMENTS	8
8. REFERENCES	8

1. Overview

It has been claimed that distributed computing can improve:

- collaboration through connectivity and interworking;
- performance through parallel processing;
- reliability and availability through replication;
- scalability and portability through modularity;

- extensibility through dynamic configuration and reconfiguration;
- cost effectiveness through resource sharing and open systems.

Our experiences and the experiences of others have shown that distributed computing can indeed offer these benefits when applied properly. However, developing distributed applications whose components collaborate efficiently, reliably, transparently, and scalably is a complex task. Much of this complexity arises from limitations with conventional tools and techniques used to develop distributed application software. Many standard network programming mechanisms (such as BSD sockets and Windows NT named pipes) and reusable component libraries (such as Sun RPC) lack type-safe, portable, reentrant, and extensible interfaces. For example, both sockets and named pipes identify endpoints of communication using weakly-typed I/O handles. These handles increase the potential for subtle run-time errors since compilers can't detect type mismatches at compile-time. Another source of development complexity arises from the widespread use of functional decomposition. Many distributed applications are developed using functional decomposition techniques that result in non-extensible system architectures. This problem is exacerbated by the fact that the source code examples in popular network programming textbooks are based on functional-oriented design and implementation techniques.

So, in this world full of hollow buzzwords and slick marketing hype, it is natural to ask the question " what does object-oriented technology contribute to the domain of distributed computing?" The short answer to this question is that object-oriented technology provides distributed computing with many of the same benefits (such as encapsulation, reuse, portability, and extensibility) as it does for nondistributed computing. In fact, it is often more natural to utilize object-oriented techniques in the domain of distributed computing than it is for non-distributed computing. This is due to the inherently decentralized nature of distributed computing. In conventional non-distributed applications, there is often a temptation to sacrifice abstraction and modularity for a perceived increase in performance. For example, many programmers use global variables or access fields in structures directly to avoid the overhead of passing parameters and calling functions, respectively. In distributed computing, however, performance optimizations based on direct access to global resources are extremely difficult to develop and scale. Research and development on operating system support for distributed shared memory, for example, is not yet ready for large-scale system deployment.

Therefore, most distributed applications interoperate by passing messages. There are many variations on this message passing theme (e.g., RPC, remote event queues, bytestream communication, etc.). However, it doesn't require much of a stretch of the imagination to recognize that message passing in distributed computing is very similar to method invocation on an object in object-oriented programming. With this observation in mind, let's discuss two of the key distributed object technologies: Corba and OLE.

1.1. Corba

(Description of the technology in general terms.)

The Object Management Group's central mission is to establish an architecture and a set of specifications, to enable distributed integrated applications. Primary goals are the *reusability, portability and interoperability* of object-based software components in distributed heterogeneous environments.

The Object Management Architecture (OMA) provides framework which defines the functions supported by the component technology specifications within the OMG. The four major parts of the OMA are:

- The **Object Request Broker** (ORB) provides the mechanisms by which objects transparently make requests and receive responses. The ORB provides interoperability between applications on different machines in heterogeneous distributed environments.
- **Object Services** is a collection of services (interfaces and objects) that support basic functions for using and implementing objects.
- **Common Facilities** is a collection of services that provides general purpose capabilities useful in many applications.
- **Application Objects** are objects specific to particular end-user applications.

The Common ORB Architecture and Specification defines a framework for different ORB implementations to provide common ORB services and interfaces to support portable clients and implementations of objects. The CORBA specification is mainly composed of the following parts:

- An Object Request Broker(ORB).

- An Interface Definition Language (IDL).
- A Dynamic Invocation Interface (DII).

1.1.1. Object Request Broker

Client is the entity that wishes to perform a operation on the object and Object Implementation is the code and the data that implements the object. The ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the Object Implementation to receive the request, and to communicate the data making up the request. The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect which is not reflected in the object's interface. To make a request, the client can use the Dynamic Invocation Interface (the same interface independent of the interface of the target object) or an IDL stub (the specific stub generated from the interface exported by the client). The Object implementation receives the request through the generated IDL skeleton. The client or the Object implementation can directly interact with the ORB for some functions.

Definition of the interfaces to object can be defined in two ways. Interfaces can be defined statically in a Interface Definition Language or interfaces can be added to an Interface Repository service: this service represents the components of an interface as objects, permitting runtime access to these components.

1.1.2. Object Services

An Object Service defines the interfaces and sequencing semantics that are widely available and are most commonly used to support building well-formed applications in a distributed object environment. Operations provided by Object Services are expected to serve as building blocks for Common Facilities or Application Objects. The operations provided by Object Services are made available through the ORB. Many Object Services are identified but only a subset of them are completely specified:

- **Naming:** Names are human-comprehensible values that identify an object. The object Naming Service provides a mapping of names to CORBA object references.
- **Event Service:** allows objects to dynamically register or unregister interest in receiving notification when specific event occurs.
- **Lifecycle:** provides operations for managing object creation, deletion, copy and equivalence.
- **Persistence:** provides persistence of an object independent of both the lifetime of the client applications that access the object and of the implementation that realizes the object's methods.
- **Relationships:** provides for creating, deleting, navigating, and managing relationships between objects. Relationships support modeling object associations and semantics.
- **Externalization:** provides the transformation of a object into a form suitable for storage on a external media or for transfer between systems.
- **Transaction:** provides support for ensuring that a computation consisting of one or more operations on one or more objects provides some or all of the ACID (Atomicity, Consistency, Isolation, Durability) properties

1.1.3. Common Facilities

Common Facilities comprises facilities that are useful in many application domains and which are made available through OMA-compliant object interfaces. Unlike Object Services, which will be supported on all platforms, Common Facilities is optional. For end-users, Common facilities provides uniform semantics that are shared across applications, making OMA-compliant applications easier to use.

A service becomes a Common facilities when it:

- Communicates using ORB.
- Implements a facility that the OMG chooses to adopt.
- Has an OMA-compliant object interface.

Partial list of candidates for Common Facilities:

- Cataloging and browsing of classes and object.
- Link management.
- Printing and spooling.
- Error reporting or help facility.
- Electronic mail facility.
- Common access to remote information repositories.

1.1.4. Interface Definition Language (IDL).

CORBA IDL is an object-oriented interface definition language used to only specify interfaces containing methods and attributes. It supports interface inheritance and it is designed to map onto multiple programming languages: C, C++, Smalltalk, Java. It supports the following features:

modules, interfaces, methods, attributes, inheritance, arrays, sequence, struct, enum, union, typedef, consts, exceptions.

The differences from C++ are:

no data members, no pointers, no constructors or destructors, no overload methods, no Int data type, contains parameter passing modes, unions require a tag, string type, sequence type, different exception interface, no templates.

A CORBA IDL compiler is used to generate clients stubs and server skeletons. Stubs and skeletons automate the following activities in conjunction with the ORB:

- client proxy factories
- parameter marshalling or unmarshalling
- implementation class interface generation
- object registration and activation
- object location and binding
- per-object or per-process filters.

1.1.5. Dynamic Invocation Interface (DII).

This interface allows the dynamic construction of object invocations, rather than calling a sub routine that is specific to a particular operation on a particular object: a client may specify the object to be invoked, the operation to be performed, and the set of parameters and their types for the operation through a call or sequence of calls.

The DII is more flexible than the static IDL approach, but it is also more complicated and less typesafe and efficient. It simplifies the implementation of tools like a browser that may not know all their components to browse (operations and their parameters) at compile-time.

1.2. OLE

1.2.1. Introduction

OLE is a set of objects services commercialized by Microsoft. It was introduced in 1990. OLE 2.0 sits on top of Microsoft's Object Model (COM), which defines Microsoft's underlying concept of an object. OLE is an alternative to OMG. We only speak about version 2 of OLE, the first one having been very limited. OLE 1 allowed two Windows applications to call each other in order to offer edition functionalities for document elements. If a Powerpoint object is inserted into a Word application, the click on such object automatically invoked the PowerPoint application. Added functionalities in OLE2 increase the integration between applications. Thus, in the previous example, the word processing application temporarily becomes the graphic application that generated the graphic element. The menu is thus transformed to add needed functionalities to edit a graphic element.

1.2.2. OLE structure

OLE is implemented upon DDE (Dynamic Data Exchange) mechanisms of Microsoft. As we mentioned previously, OLE is based on the Component Object Model (COM). Objects that are written to support the COM are called in the Microsoft vocabulary *component objects*. Here are listed the major characteristics of OLE:

- OLE Automation:

Automation is the ability for one application to dynamically use the services of another application, without having been specifically designed to do so. It is the Automation mechanism, that allows for example a word processing application to be interfaced with any OLE-enabled spreadsheet application to recalculate cells, draw a graph, or perform any service the spreadsheet supports. Applications thus have to export their internal services through abstractions called: *Automation objects* *Automation controllers* are thus development tools or

applications that can drive OLE applications through Automation. Today, most Microsoft products and others offers this mechanism. There are for example: Visual Basic, Visual Basic for Applications, and Visual C++.

- **OLE Controls:**

OLE controls are OLE-enabled software components that can be purchased to extend and enhance an application's functionality. OLE controls are similar to Visual Basic custom controls (VBXes), but their architecture is based on OLE. This means, that, unlike VBXes, OLE controls can be freely plugged into any OLE-enabled development tool, application, and eventually, the windows operating system itself. These controls will be provided by software vendors. OLE controls enable the creation of 'compound applications' much like OLE enables the creation of compound documents.

- **OLE Object Linking and Embedding :**

Through Object Linking, applications can be linked to data objects within other applications. For instance, a spreadsheet table can be linked into multiple custom business reports, and as changes are made to this table within the spreadsheet document, all report documents are automatically updated. Object embedding is the ability to embed an object within another document without maintaining a link to the object's data source.

- **OLE storage:**

Through object storage, the OLE system allows applications to store embedded and/or linked objects in their native file structures, creating a persistent link between the objects and the container applications.

- **OLE visual editing and OLE Drag and Drop:**

Through these capabilities, the user has the possibility to create compound documents easily and graphically.

1.2.3. Discussion

The first commercial success in this domain was the Visual Basic Extension (VBX) model. The VBX architecture was developed as a means of extending the Visual Basic programming environment, in developing software components as VBXs. This was very popular, but not originally designed as an open standard for component software.

With VB4.0 appears OCX which are more powerful VBX build upon OLE. A large amount of OCX is available nowadays and this is one of the major base for reusing software component. They can easily be transformed in ActiveX.

With Internet appears Active X which are often more powerful OCX. ActiveX Controls are interactive objects, created by programmers, that can be embedded in Web pages to enhance the experience of a Web site. For example, an ActiveX video control could be used to enhance a Web page with real-time video sequences.

The controls are language-independent, and can be programmed using programming languages such as C++, future versions of the Microsoft Visual Basic programming system, or Java in the future. Over 1,000 ActiveX Controls are available today from a wide variety of software vendors. Using the ActiveX Control Pad, Web developers can now easily incorporate ActiveX Controls into an HTML document using a simple point-and-click process.

2. Role in evolution process

(The part played by that technology in the general evolution process.)

One of step in the migration of a legacy application is to modeling and design the revised application. Object Oriented modelisation provides a interesting solution for this important phase and this solution is now more and more often selected. Once this modelisation technologie is used, it is a great simplification for the implementation phase to use a middleware and a programming language which support the Object Oriented concepts, otherwise it will necessary for the programmer to developpe its own support for these concepts. Actualy, CORBA is the only industrial product, supported by different tool providers, which provides the framework and the foundation for higher-level distributed object collaboration, and which is available on a large set of platforms.

3. Relevance to Renaissance

(The specific relevance of the technology to Renaissance and how it might be part of the Renaissance method.)

Developing distributed applications whose components collaborate efficiently, reliably, transparently, and scalably is a complex task. However, the evolution of application is often a migration from centralized legacy system to a distributed client/server system. This kind of evolution offers some interesting features:

- The migration may be incrementally done, using wrappers to encapsulate some legacy components.
- It forces the definition of clear interfaces between distributed components (simplify next evolution).
- The partitioning of the application and its distribution on different platforms allows to adapt the resources and the performances to the need (load balancing).
- Simplify the reuse of components and allows composition of services.
- User interface may be more independent of the implementation of the application and will be more easily customized.

4. Available support

(A discussion of available tool support with examples if appropriate.)

4.1. Corba availability

This is an incomplete list of different implementations of CORBA:

Note: all these implementations are announced to be compliant with CORBA 2.0: one important aspect of the CORBA 2.0 specifications is that it enables objects developed using ORB from different vendors to interoperate transparently (by using the Internet Interoperability Protocol (IIOP)). Moreover, most of these implementations offer a mapping to Java.

Orbix provided by IONA Technologies.

The first version of ORBIX was released in June 93. It is supported on more than 10 Unix platforms (HP, SUN, IBM, DEC,...), Windows (3.1, NT), OS/2, MAC, VMS, QNX, LynxOS, and embedded real-time system such as VxWorks. It provides a mapping for C++, Smalltalk, ADA, Java. Today Orbix runs on top of TCP/IP, IPX/SPX, SNA, Tuxedo and ISIS.

ORBIX-OLE is an automatic way of generating the automation server/Orbix client code from a IDL specification.

ORBIX-MT : Multithreaded ORBIX offers threading compatibility with various third party threads packages.

ORBIX+ISIS : integration with ISIS software fault tolerance infrastructure offers high availability CORBA facilities.

ORBIX+ObjectStore: integration of ORBIX with a database system provides for persistence of ORBIX objects.

We develop a mock up using Orbix+Isis for the CNES: French National Spatial Agency. We were very impressed !

PowerBroker provided by Expertsoft Corporation

PowerBroker runs on SunSoft Solaris/SunOs, Hewlett-Packard HP-UX, IBM AIX, SGI IRIX, Digital UNIX for Alpha, Microsoft Windows/NT. It provides a mapping for C++, Smalltalk. It does not include threads. One of Expertsoft's greatest differentiators from other CORBA compliant products is its commitment to asynchronous messaging: an object can continue processing while it waits for the result from a request to another object. It provides a complex asynchronous programming model. PowerBroker includes also in its first version a set of Object Services (Naming Service and replicated Federated Naming Service, Event Service, Object Life Cycle Service).

PowerBroker CORBA/OLE maps CORBA objects into OLE objects for inclusion into Windows OLE applications.

Object Broker provided by Digital.

ObjectBroker is available on UNIX platforms (OSF/1, ULTRIX/RISC, SunOS, HP-UX, IBM-AIX) and Microsoft Windows, Open VMS VAX/AXP, Macintosh. It has been the first CORBA implementation to provide integration with OLE and DDE on Windows.

Sun's DOE

IBM's DSOM

...

4.2. OLE Availability

OLE2 is available on Windows and Windows NT, as well as for MacIntosh. Microsoft claims that ActiveX/OLE will eventually work within Apple and Unix architectures, but it has not given any specific dates.

4.2.1. OLE Strengths and Weaknesses

4.2.1.1. Strengths

- High-level of application integration on and off screen,
- Rock-bottom price, free run times,
- Will be built into future Windows operating systems,
- Multi-platform support now or soon: Macintosh, Windows, Windows NT.

4.2.1.2. Weaknesses

- Very few documentation for OLE programming and few development tools,
- Container and objects must execute on the same desktop,
- No real plan for UNIX and OS/2 support,
- Faced with a very competitive technology OpenDoc investigated mainly by IBM, Apple, Novell, Borland.

5. Inapplicability

(Areas where the technology should NOT be applied (e.g. 4GLs for real-time systems))

The use of CORBA will be really efficient only if the re-design of the legacy applications will be done with Object Oriented Modeling and Design, and with the use of an Object Oriented Programming Language.

CORBA is not available on all platform mainly for the mainframe. So the evolution of legacy applications which use large database on some mainframe, need to be re-implemented on new platforms which support CORBA otherwise the database access through CORBA will be impossible. Each time that the encapsulation of the existing application in a CORBA wrapper, or that the full reimplementations of the legacy application is impossible, the use of CORBA as middleware is inapplicable.

CORBA, as any distributed system, is not relevant for applications with strong real-time properties. However, some implementations of CORBA (like the version of ORBIX-RT provided by IONA Technologies) are available on real-time system as VxWorks, QNX and LynxOS.

6. Future development

(An assessment of possible future developments of the technology.)

6.1. Corba

One of main evolution of this technology will be the completion of the specifications and implementations for the Object Services and Common Facilities. The interaction with the WEB (IDL mapping to Java) or multimedia technology is also started and available for example in Netscape One product (see report on internet technologies)

6.2. OLE

The next version of OLE will be 'distributed' and will run on Cairo. It will allow you to invoke documents even if they are on different platforms. OLE will add a mechanism to keep track of unique names of objects on any platform in the network (nearer to the ORB mission). It will also incorporate a security system based on Kerberos.

Through a recent agreement, Microsoft and Dec are integrating the Common Object Model technology with the Digital Object Broker. This will enable the Microsoft OLE running on Windows or MacIntosh to interoperate with objects on SunOS, IBM AIX, HP-UX, DEC ULTRIX, OSF/1 and OpenVMS through DEC's CORBA-based Object Broker ORB.

7. Other comments

To see how it can be connected to web application, read the technology briefing report on internet.

8. References

[C++Report Jan 95] "Introduction to Distributed Object Computing", C++ Report, Douglas Schmidt, Steve Vinoski, SIGS, Vol. 7, No. 1, January 1995.

[OMG-CORBA1.1] 'The Common Object Request Broker: Architecture and Specification', Revision 1.1. OMG Document 91-12-1

[OMG-CORBA1.2] 'The Common Object Request Broker: Architecture and Specification', Revision 1.2. OMG Document 93-12-43

[OMG-COSS2] 'Common Object Services Specification, Volume 2'

[OMG-OSA] 'Object Services Architecture'

[OMG-OMA] 'Object Management Architecture Guide', OMG Document 90.9.1

[OLE-PR-V1] OLE Programmer's Reference Volume I 'OLE Guide to Programming'

[OLE-PR-V2] OLE Programmer's Reference Volume II 'Creating Programmable Applications with OLE Automation'

[OOS] Object Oriented Strategies 'The monthly newsletter for manager & developers of object oriented systems', monthly issue from 1993 to 1996