
# Technical Overview

**www.2ab.com**

## *Introduction*

As computers have evolved, organizations have developed a diverse range of information systems supporting numerous applications. Applications range from legacy code written decades ago, to the latest Web-based technologies. These disparate technologies need to interact and to share information. The overwhelming demand is for distributed environments that allow the spectrum of IT resources to work together.

However, major challenges stand in the way of achieving this goal of truly distributed computing. The diversity of hardware, software information formats and protocols must all be considered and integrated. While application development tools are around to assist in this task, none can provide a complete solution and each presents a particular limitation. The outcome has been the emergence of so-called “islands of information,” where each island runs according to a different set of standards. In addition, the variety of development tools and connectivity choices can easily become part of the problem, not a solution.

A true distributed computing environment needs to display a number of features. It has to be open and expandable. It must have extensive run-time management and cover a wide-range of platforms, networks and development tools - and, crucially, such an environment must be able to access existing information, applications and services.

orb2, the 2AB Object Request Broker (ORB) architecture, addresses all these issues. The ORB is one of the most effective ways of bringing true coherency to disparate systems. The ORB provides the ability to communicate, share information and create independent layers of logic untouched by changes to operational systems — these benefits and more can and will be available when a properly executed ORB strategy has been put into place.

This white paper is an introduction to orb2 and its services, revealing how the product can be used to resolve modern enterprise-wide IT communications and integration issues. It is aimed at application developers contemplating object technology to build new systems. Another audience will be those developers looking to get the most out of an existing infrastructure with object orientation.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## What is CORBA?

The Common Object Request Broker Architecture (CORBA) defines an open architecture and infrastructure enabling applications to interact with each other regardless of the computer they are running on, the operating system being used, the programming language they are written in, or the network over which they are communicating. The CORBA architecture defines a set of components used to build applications. For the sake of this paper, objects are discrete software components and the terms are used interchangeably.

At the heart of CORBA is the Interface Definition Language (IDL), which is used to describe the interface, both the information and the functions or methods that act on that information, to each component. Development tools use IDL to automatically generate the code required to enable components to interact with each other through a runtime environment known as the ORB. Other methods being marketed have an extensive Application Program Interface's (API) that must be learned. CORBA applications are easier to create and deploy. Components can be produced using any development tool, either conventional or object-oriented, and can run on different systems.

CORBA simplifies the construction of portable, distributed applications. The combination of development tools and a runtime environment ensure that users can focus on applications and their structure without being concerned with distribution mechanisms or platform-specific APIs.

CORBA is defined by the Object Management Group (OMG), which has over 500 members and is the largest Information Technology consortium in the world. The OMG specifies that an ORB *"...provides the mechanism by which objects transparently make and receive requests and responses. In so doing, the ORB provides Interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems."*

### The CORBA architecture?

This section describes CORBA in terms sufficient to provide the reader with some insight as to how orb2 works. It is not meant to be complete. The following provide some references for further investigation:

| | |
|---|---|
| *Essential Client server Survival Guide* | Robert Orfali, Dan Harkey, Jeri Edwards Thomson press |
| *Essential Corba:System Integration Using Distributed Objects* | Mowbray et al; OMG/Wiley |
| *CORBA Fundamentals and Programming* | Jon Siegel; OMG/Wiley |
| *The Common Object Request Broker Architecture specifications* | Available from the OMG |

The CORBA architecture contains four main components:

**ORB** - The ORB is at the heart of the architecture. It offers location transparency, which allows objects to interact without concern for where their associated objects are located. The ORB defines the interfaces between objects, hiding implementation details. The analogy is plug-and-play component software.

**CORBA Services** - Provide fundamental system services that are needed by Information Technology (IT) systems. For example:

**Trader Service** – The Trader Service acts as a resource through which applications or objects can advertise the services (functionality) they offer. Using the Trader to locate


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

services means that services are located dynamically at runtime. The location of a required service is not hardwired, making it possible to reconfigure a distributed application without changing any code. You can view the Trader as analogous to using the yellow pages of a telephone directory. In the yellow pages, as in the Trader, offers are grouped together by type; a person searches for a section that advertises a relevant type of service, for example, car dealers. When a client uses the Trader to select a service, it uses *properties* to select the most appropriate offer.

**Naming Service** – The underpinnings of the Naming Service works akin to the Trader Service with an analogy to the *white* pages of a telephone directory, where a person searches for a single telephone number and uses the name and address details to locate it.

**CORBA Facilities** - Provide application objects that are used by user applications. Two types are available

**Horizontal**

- User interfaces
- Systems management
- Document management

**Vertical,**

- Financial
- Manufacturing
- Healthcare

**Application Objects** - Represent applications written by users that provide the functional and informational requirements of the specific industry in which the application is written for.

The CORBA architecture, which includes, the ORB, CORBA Services, CORBA Facilities and Application Objects (which use the IDL to communicate over the network) enable a true software component market to exist. For each component of the architecture, the OMG provides a formal specification that defines the component interface in terms of their informational and operational semantics. Figure 1 illustrates how the components fit into the CORBA architecture.

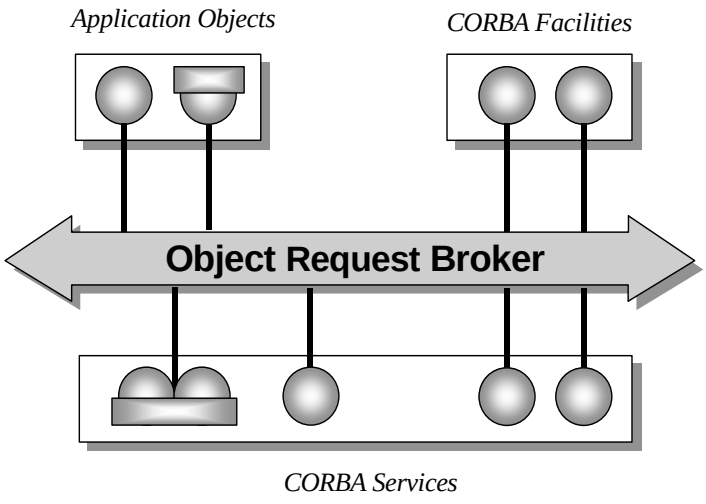


Figure 1


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## What is orb2?

orb2 is a set of CORBA-based software tools designed for the creation and deployment of distributed applications. orb2 enables you to design, implement, operate, manage and maintain distributed applications that run on any combination of Windows or UNIX, proprietary midrange systems or mainframes. The facilities offered by orb2 allow you to focus on application integration. Once enabled, orb2 manages the system issues associated with writing software that can be distributed across a heterogeneous computing environment.

Interoperability in a distributed environment is desirable and the benefits to users are considerable. However, achieving these levels of communication can pose major challenges. You must consider

- Authentication/Authorization
- Hardware
- Software
- Operating systems
- Data representations
- Communications protocols

There are application development tools available that address some of the issues but few, if any, cover the task completely and all have their limitations. Islands of technology have emerged; each island observing a different set of standards with different connectivity and different development tools. Such systems typically need expensive new skills and constant care. Therefore, they are expensive to maintain and change.

Real solutions must have the following characteristics, which orb2 addresses

- Open and expandable architecture
- Extensive runtime management features
- Wide choice of platforms, networks and development tools
- An ability to access existing data, applications and services

### How Does orb2 Handle Changes?

orb2 processes changes in the following ways

- System components establish connections to each other, dynamically, at runtime
- Replacement of service components even if the system is running
- Identification of components by type and property as well as by name

Naming is like using the White Pages in a telephone directory. orb2 also offers Yellow Pages, known in CORBA terminology as the Trader. You can test new components in their own context with access restricted so that untested components can be loaded without fear of use by the production system.

### How Does orb2 Handle Growth?

orb2 provides common pre-emptive multithreading and configuration facilities on most supported platforms, which simplifies the task of porting components from one platform type to another. You can split an orb2 network into separate domains to allow easier management.

orb2 is highly scalable and can support all sizes of applications from those running on a single PC up to enterprise-wide networks. orb2 automatically decomposes data to make it more manageable so that large data items, such as images, can be passed around the network easily.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## How is the System Managed?

Users can observe and change the status of individual components at runtime. They can be activated on command or automatically when called; multiple copies of components can be activated to share the load. You can specify how many copies are loaded and can vary this at runtime to provide optimum throughput.

orb2 automatically logs information as it runs. Users can use this statistical feature for diagnostics, audit trails, billing or for passing information to system management monitors.

## Maximizing Availability

You can distribute orb2 applications and services around the network to avoid any single point of failure. You can also multiplex essential data for added resilience. orb2 detects when components have failed and can take remedial action. orb2 offers facilities for standby components to be constructed. Users can map requests for a specific service onto any other service, for example, to avoid an overloaded or failing service. orb2 provides dependable messaging over connectionless transport protocols.

## Optimizing Resources

orb2 minimizes the use of communication resources by multiplexing and reusing wherever possible. orb2 uses static and Dynamic Invocation Interfaces (DII) that allow users to choose the optimum method for their application.

- Static invocation is easier to program and more efficient at runtime because the call logic is built and checked at compile time.
- Dynamic invocation allows more flexibility at runtime. orb2 automatically detects when components are unable to handle network traffic and reacts accordingly to reduce retries and network congestion.

## Accessing Existing Data, Applications and Services

You can integrate existing applications into orb2 systems by writing a program called a wrapper that makes them appear to be orb2 components. You can connect any type of application in this way even if the target application does not run on a platform supported by orb2.

orb2 can also connect to applications written using another distributed standard, for example, Microsoft's Common Object Model (COM) technology, OLE, X/Open's Distributed TP or OSF's DCE, using gateways. Currently, gateways are unique, written for each new application, with the exception of OLE/COM. orb2 supports communications between the CORBA IDL and OLE/COM Automation mapping specified in the COM-CORBA interworking specification produced by the OMG.

orb2 also supports CORBA 2.0 interoperability protocols, the Internet Inter-Orb Protocol (IIOP), enabling applications to interwork with those based on other CORBA ORBs. The ability to work with other ORBs enables orb2 to adopt a comprehensive, flexible approach to supporting objects distributed across multiple CORBA-compliant ORBs.

2AB has developed various prototype wrappers and gateways for use with orb2 that can be developed further to suit your specific application.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## Language Mappings, Development Environments, Operating Systems and Transports

**Language Mappings**

- C, C++
- Java
- COM2CORBA

**Development Environment**

- Visual C++
- All native C and C++ compilers
- Java IDEs including Symantec Visual Café, SUN JDK, JBuilder
- Visual Basic, Visual Basic for Applications, VBScript, Delphi
- Informix, Oracle and Sybase RDBMS
- Paradigm Plus, Select Enterprise, System Architect and Galaxy

**Operating Systems**

- AIX
- HP-UX
- OpenVMS
- SCO UnixWare
- Stratus VOS
- Sun Solaris
- Microsoft Windows 2000 and NT

**Transports**

- IIOP
- TCP/IP
- UDP/IP
- IPC


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## *Writing an orb2 Application*

Application development in orb2 is described in detail in our Users Guide for the respective programming languages such as C, C++ and Java and this section is meant only to provide a high-level overview.

### Designing an Application

The majority of the design work for an orb2-based system is originated using any suitable design methodology; for example, structured design or object-oriented design. In general, such techniques lead to a design with a number of components interacting through well-defined interfaces. Users decide where there is potential to distribute application components, which is typically done for one of several reasons

- Part of the application is accessed over a network
- Part of the application requires changing without affecting other parts
- Part of the application is used in other applications
- An existing application needs to be encapsulated.

When an object-oriented implementation language is used, it may not be necessary to use orb2 for every programming language object. Most objects are only used locally by one program. Therefore, you should use the language's object linking. You should only consider an orb2 link for those objects that could be used outside a local program. This balance provides the best performance.

### Using IDL

orb2 applications contain clients and servers. IDL is used to describe what a service is, so that a client knows how to use it and the server knows what to implement. Both client and server share the same Interface Definition so the interactions between them are guaranteed to be consistent.

IDL has a text file that contains

- The name of the service, the Service Type
- A definition of which operations are supported
- A definition of the arguments required for each operation, both input and output
- A definition of the datatypes used in the arguments to the operations
- A definition of the exceptions that can be generated as a result of incomplete operation execution


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## Examples

The following sections contain a simple IDL mapping example, an orb2 client mapping example, an orb2 server mapping example, performance considerations, and how to handle failures and debug orb2 applications.

## IDL

This is a simple IDL example. A service called BankAccount has operations to create a new account, deposit or withdraw money and obtain the balance. This IDL defines a service of type BankAccount with the three operations **create**, **credit** and **query**; it does not define how the service is implemented, just how to use it.

***Here is the IDL without Comments***

```
interface BankAccount
{
        typedef unsigned long Number;
        struct Record
        {
                string owner;
                float balance;
        };
        exception ErrorCall { string reason; };

        void create
        (
                in Record cust,
                out Number account
        )
        raises ( ErrorCall );

        void credit
        (
                in Number account,
            in float amount
        )
        raises ( ErrorCall );

        void query
        (
                in Number account,
                out Record cust
        )
        raises ( ErrorCall );
};
```


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

***Here is the IDL with Comments, the Comments are explained below:***

```
// Comment 1
interface BankAccount
{

// Comment 2 - Parameter datatype definitions
        typedef unsigned long Number;
        struct Record
        {
                string owner;
                float balance;
        };

// Comment 3 - Exception definition
exception ErrorCall { string reason; };

// Comment 4 - Creation definitions
        // Create new account with balance
        void create
        (
                in Record cust,
                out Number account
        )
        raises ( ErrorCall );

        // Credit (or debit if amount is negative) the account
        void credit
        (
                in Number account,
                in float amount
        )
        raises ( ErrorCall );

        // Query balance and account name
        void query
        (
                in Number account,
            out Record cust
        )
        raises ( ErrorCall );
};
```

***Explanation of Comments:***

1. BankAccount is the typename of the interface being defined; analogous to a class in object terminology.

2. IDL has some predefined datatypes, such as short, long, float and char. From these basic types, you can create more complex types using a variety of constructs.

3. Operations can return exceptions if they detect an error while processing. Each exception can have its own type.

4. The operation definitions are similar to C++ function definitions.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

a. The return type is defined before the operation name; each argument can be defined as client to server only (in), server to client only (out) or both ways (inout).
b. Any exceptions that the operation can return are listed in the raises expression.
c. IDL operations are analogous to methods in object terminology.

## orb2 using IDL Mapping

The IDL is processed by orb2 to produce source files. Each IDL type and exception definition is converted into a type definition. Each operation definition is transformed into a method.

The IDL in the BankAccount example produces the following definitions

**IDL Statements**

```
// Interface TypeName
    interface BankAccount {


// Parameter datatype definitions
    typedef unsigned long
            Number;
    struct Record {
            string owner;
            float balance; };


// Exception Definition
    exception ErrorCall {
            string reason; };


// Operation definitions
    void create
        (
            in Record cust,
            out Number account )
    raises ( ErrorCall );
    void credit
        (
            in Number account
            in float amount )
    raises ( ErrorCall );
    void query
        (
            in Number account,
            out Record cust )
    raises ( ErrorCall );
};
```

**Statements**

```
/* Object type definition */
    class BankAccount :
        public virtual CORBA::Object


/* Datatype definitions */
    typedef CORBA::ULong Number;
    struct Record {
        CORBA::String_var *owner;
        CORBA::Float balance; }


/* Exception definition */
class ErrorCall :
    public CORBA::UserException


/* Method definitions */
    virtual void create
                            (
        const Record&,
        Number& ) = 0;

    virtual void credit
                            (
        Number,
        CORBA::Float) = 0;

    virtual void query
                            (
     Number,
     Record*& ) = 0;
```

For each IDL statement there is a corresponding statement. The IDL definition of BankAccount relates to the class BankAccount.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## Writing an orb2 Client

An orb2 client is a program that wants to use orb2 servers. You can write it in any language supported by orb2; for example, C, C++, Java or any language that is able to call functions written in the supported languages.

In addition to producing method definitions from the IDL, orb2 also generates complete client functions for server operations, stubs. Stubs do not implement operations; they pass them into the ORB. The ORB delivers requests to a server and returns any results. When the client needs to call the server, the appropriate function is called just as if it were a local procedure. To call the **create** operation in a BankAccount server, users write a call to the **BankAccount::create** function, which is in the stub. The stub delivers the operation request to the appropriate server that is specified by the object reference argument for execution.

It is not necessary for clients to know where the server is, what communications protocols need to be used or the data representation. The client does not need to be concerned with communications error recovery or to know how the server implements the operation, it just makes what appears to be a local call and orb2 does the rest. All that users have to write is the application logic.

## Writing an orb2 Server

A server must implement the operations defined in the IDL. For the BankAccount example, the server would contain three functions, **create**, **credit** and **query**. The arguments to these functions are passed from the client through the ORB and are transformed, if necessary, to take care of local data mappings. Finally, orb2 takes the IDL and generates skeletons. The stubs are linked with the code that implements a server. They are responsible for receiving operation requests from the ORB and passing them to the server code. The server is oblivious to the details of its clients, it is also unaware as how to drive the communications layer to retrieve arguments or return results, orb2 handles this.

When the server initializes, it calls an orb2 function to say that it is an implementation of the BankAccount service. In response, orb2 returns an object reference that clients can use to access the service. The server normally passes this to the Trader using the Trader's **export** operation to announce itself to the world. Clients import this reference, and then use it to call the server operations.

Many clients can import the reference and use the same server simultaneously. The object reference is an IDL data type and can therefore be used just like any operational argument. The server can pass its object reference to further servers as an input argument (which is how the Trader **export** operation works) or it can return it to a client (which is how the Trader **import** operation works) as well as, or instead of, using the Trader. The server can use orb2 multi-threading to execute several incoming calls concurrently, if appropriate, or it can choose to execute the calls synchronously. orb2 takes care of any queuing requirement.

## Performance Considerations

orb2 application performance is governed by how well the application is designed. There is a small overhead introduced by orb2 on each client/server interaction. Generally, the processing done by the server should be simplistic. For example, when accessing a database through orb2, designing a server that returns every record to the client does not provide optimum results. It is more efficient for the server to interpret the data and pass it back to the client as consolidated results; put the processing where it is most effective.

Network requests impact performance. Because orb2 runs on many types of system and across several types of network, the amount of data transferred is reliant on the operation definitions and is difficult to codify. When designing the application, it may be necessary to model the system and to take some actual measurements.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## How are failures handled?

The failures within orb2 that must be proceduralized are those relating to client/server interaction. Application function calls either work or return an exception generated by the called function. In a distributed application, the client must expect additional failures when the call cannot be delivered or the results cannot be returned. The types of failure that could occur are

- orb2 cannot create the message to be sent to the server; for example, memory is unavailable.
- Basic communications failures, preventing the request from being delivered to the server. orb2 performs communications retries, but it eventually abandons the request.
- Server cannot process the request.
- Errors could occur in the transmission of the results, either communications or processing failures.

All types of failure are reported to the client as exceptions. They also indicate whether the request was successfully executed or if an error occurred. There are two classes of errors

- **Server-generated**
  Reports a user exception, which is one of the exceptions described in the IDL. In the BankAccount example, any operation could return an ErrorCall exception that has a string element. The values are defined as part of the server implementation; any recovery from these errors is application dependent.
- **orb2-generated**
  Reports a system exception. You can use various recovery strategies. You can examine the system exception to determine the underlying cause of failure. It also indicates whether the request was completed or if it only returns **don't know**.

Some failures may be recoverable; for example, a communications failure occurred. However, it may be more suitable for the client to call the Trader to find a new copy of the server. Alternatively, it is possible to create resilient services so that if a service fails, an alternative can be used. The server creates a special object reference to indicate that an alternative is available; orb2 updates the client's object reference with one that now points to the alternative server. In a future release of orb2 , resilience will be extended so that a replacement server can be created, if necessary, either on the same system, as the one which hosted the failed server, or on an alternative. When an alternative server takes over from an existing one it needs to recover the state of the failed server.

If an error is attributable to the host system, such as memory problems, any recovery action is dependent on the general application recovery facilities on that specific system.

The orb2 C++ and Java mappings provide native exception handling capabilities.

## How do I debug orb2 applications?

Since orb2 allows you to determine the location of clients and servers at runtime, it may be appropriate to locate the entire application on a single host as an aid to development and initial debugging. Parts of the application can then be distributed after they are debugged. You can use standard host system debugging facilities with orb2 applications.

You can debug each part of the application even when it is fully distributed. When using interactive debugging in a server, time-outs may occur in clients that are waiting for results.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## *Incorporating Existing Code and Data*

You can make existing applications into orb2 servers if a method exists to call them from a program. If your applications have a programming interface that can be called from any programming language supported by orb2, for example, C, you can use it directly. Applications that only have an interactive interface can be called using a virtual screen protocol such as HLLAPI (a standard programming interface for accessing virtual terminals by program). You must write IDL to reflect the interface that will be presented to orb2 clients by the existing application.

Incorporating an existing application involves writing an integration layer, which has two parts

| | |
|---|---|
| Front end | A set of C, or any other language directly supported by orb2, functions that implement the server IDL. |
| Back end | Makes calls on the existing application interface or drives the interactive screens to achieve the required effect. The latter technique is sometimes known as screen scraping. |

You can use a similar technique to screen scraping if an existing application needs to use new orb2 servers. Here, you must modify the application code and call C functions or any other language directly supported by orb2. The integration layer is a set of functions that call the appropriate orb2 stubs for the service to be used. You can also use such integration techniques to access applications residing on systems not directly supported by orb2. Here, the front end of the integration layer has a C interface, or any other language directly supported by orb2, and runs on an orb2-supported system. The integration layer makes a call across the network to the non-orb2 system using any convenient technique; for example, X.25 or TCP/IP. An integration layer running on OS/2 could use SNA protocols to provide orb2 access to an MVS application.

### How do I incorporate existing Tuxedo applications?

You can incorporate existing Tuxedo (TP on UNIX systems) applications by

- Writing a server that accesses Tuxedo through an existing application's screen interface using a screen driving protocol
- Writing a server that uses the Tuxedo C programming interfaces directly so that the orb2 server then becomes a client of a Tuxedo service

### How do I access existing data?

New applications can access existing data directly. If the data resides on a system directly supported by orb2, you can write an orb2 server to provide access to the data. The orb2 server uses the native data access programming interface directly.

If the data resides on a system not directly supported by orb2, you can still access it through an appropriate gateway product that does run on an orb2-supported system.


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

## How do I get further information?

This White paper has intentionally only scratched the surface of orb2. Further information can be obtained by phone, fax or e-mail


Phone: USA +1 877 334-9572
Fax: USA +1 205 621-7455
E-mail: info@2ab.com.com


Comments on this White paper or on the orb2 product are welcome. Please use the above contacts.

General information about orb2 or any other 2AB products and professional services can be found at:

Web: http://www.2ab.com/

General information about the Object Management Group and CORBA can be found at:

Web: http://www.omg.org/


Copyright 2001, 2AB Inc. All rights reserved.
All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.