Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (0223) 323010**
INTERNATIONAL: **+44 223 323010**
FAX: **+44 223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

Comparison of CORBA-compliant Platforms

Mike Beasley

Abstract

This paper compares three currently available CORBA-compliant products in terms of their CORBA compliance, and programmer and administrative facilities. The intention is to include further products and versions as they become available.

APM.1048.00.03 **Draft** 1 November 1993

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

Comparison of CORBA-compliant Platforms

Request for Comments (confidential to ANSA consortium for 2 years)

Comparison of CORBA-compliant Platforms

Mike Beasley

APM.1048.00.03

1 November 1993

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(0223) 323010
INTERNATIONAL	+44 223 323010
FAX	+44 223 359779
E-MAIL	apm@ansa.co.uk

Contents

1 Introduction

1.1 Products Compared

The following products, claiming some degree of CORBA compliance, were compared:

- Orbix 1.1 from Iona Technologies Ltd., Dublin
- ACAS (Application Control Architecture Services) 2.1 from DEC
- DOME from Object-Oriented Technologies Ltd., Leamington Spa

Where information about future versions of the products is available, it is included.

It is expected that further products will be included in the evaluation when they become available, for example ORB Plus from HP and DAIS from ICL, as well as new versions of the products currently reviewed.

1.2 Basis of Comparison

The platforms were compared against the CORBA specification for compliance - in particular their support for IDL, the BOA and ORB interfaces, and repositories (though the last is considered under a separate heading). Such compliance is important to APM and to the sponsors, many of whom are members of OMG.

The ANSAware 'echo' and 'simple bank' examples were implemented on each platform, 'echo' because it is about the simplest possible and 'simple bank' because it involves passing interface references and exceptions. This practical experience of the platforms gave an impression of their ease of use and of the quality of the documentation and of the software itself.

An attempt was made to construct a multi-threaded server using each platform (even though only Orbix mentioned threads in its documentation). The objective is to write a server which will fire off a separate thread to handle each request, so that the main thread can continue to listen for the next request.

Features relating to trading and type safety were looked for. What is the equivalent of the trader ? (i.e. how do clients find servers ?) If an attempt is made at runtime to use an interface reference of the wrong type, what happens ?

What equivalents are there, if any, to the CORBA *Interface Repository* and *Implementation Repository* ?

1.3 Hardware and Operating Systems supported

Table1.1 shows the combinations of hardware and operating systems which the vendors claim to support in their current product versions:

Table 1.1: Supported hardware and operating systems

HW/OS	Orbix	ACAS	DOME	DAIS	ORB Plus
SunOS 4	Yes (4.1.3)	Yes	Yes	Yes (4.1.2)	
Solaris 2	Yes (2.2)		Yes	Yes (2.1)	
HP/UX	high priority (8.x)	Yes (8.x)	Yes (8.x)	Yes (8.0)	Yes (9.x)
SVR4 Sparc				Yes	
SVR4 Intel				Yes	
SCO Unix	some interest			Yes (3.2.4)	
OSF/1	some interest	Yes			
Ultrix		Yes			
AIX	some interest				Yes (3.1)
Stratos			Yes		
SGL Irix	Yes (4 + 5)				
MSDOS				Yes (5.x)	
MS Win 3	Yes	Yes	Yes	Yes (3.1)	
OS/2	started		Nov 93	Yes (2.0)	
NT	Yes	Yes	Nov 93		
VMS		Yes	Yes	Yes (5.5)	
VME				Yes (SV 293)	

2 Orbix

2.1 CORBA Compliance

Orbix implements most of CORBA. This includes passing interface references, exception handling and the dynamic invocation interface. Indeed, the CORBA specification does not state clearly whether the passing of interface references is permitted, and some would argue that it is an extension. The type 'Object' is currently not included in IDL, but this will be in 1.2.

Orbix has obviously been designed from the beginning as a CORBA-compliant system.

2.2 Programming

2.2.1 Ease of Use

Orbix seems to be a well thought-out product with good documentation, an adequate supply of demonstrations/examples, and impressive support by email.

Clients and servers are written in C++.

There are two different mechanisms for relating implementation to IDL interfaces, which gives a certain amount of flexibility.

Servers can easily be made to start automatically.

There will be facilities in a later version of Orbix (not 1.2) for interacting with applications running under Microsoft Windows using DDE (Dynamic Data Exchange), and for writing servers as shell (or other) scripts.

On the negative side:

The programmer must be aware that exception handling does not use setjmp and longjmp. Although this makes the code tidier, whatever is between `TRY` and `CATCH` does not automatically stop executing when an exception occurs (though any remote invocations will not be executed if an exception has already occurred).

Binaries are large (1Mb typically) - but smaller than DCE binaries which are typically 2Mb! This is because shared libraries are not used by either Orbix or libg++. There will be shared libraries in Orbix 1.2, and smaller binaries as a consequence.

Some facilities - the locator, the implementation repository interface and threads - do not work in this version.

2.2.2 Concurrency

The Advanced Programmer's Guide tells you how to create a thread to handle a request. Orbix does not itself contain a thread package, but contains the necessary facilities to do so yourself, using whatever threads you have.

Unfortunately the thread support doesn't work in Orbix 1.1, but Iona are working on it.

2.2.3 Trading and Type Safety

Trading in Orbix works as follows:

- the nearest equivalent of a trader export is a call of 'Orbix.impl_is_ready'. This informs the Orbix daemon that the server, already registered in the implementation repository, is running; it does not tell it about any specific classes supported.
- the equivalent of a trader import is a call of the '_bind' member function of the specific class of which an object is required, passing as a parameter the name of the server required. Servers can be started automatically, which represents an important difference between CORBA's use of repositories and the ANSA trader.

One point that must be made about the Orbix method of trading is that it does not scale well; looking on multiple hosts for a specific server is done by following a circular chain of hosts. As the network gets larger, this search will take a proportionately longer time; the effect of any particular host being unavailable could be catastrophic. Some way needs to be found of combining ANSA/ODP trading with the automatic starting of servers.

Orbix performs the following checks which help to ensure type safety:

- a cast from 'Object*' to a pointer to a derived IDL class 'Fred*' must be done via a static member function 'narrow' in class 'Fred', which ensures that the cast is type-safe.
- the 'request' class has an 'assert' operation, taking as argument a string representing the signature of the operation being invoked. A call to 'assert' is generated in the stubs, to check that the correct operation is being invoked.
  Iona say that 'assert' is only used for runtime type checking of requests from the dynamic invocation interface, though this is not obvious from the code.
  If you try connecting an 'echo' client to a 'simple bank' server erroneously, for example, you tend to end up with an invalid object reference. If you set up an object reference yourself and use the dynamic invocation interface, you will typically receive an exception because the interface does not support the operation you asked for.

2.3 Administration

2.3.1 Repositories

The *Interface Repository* is not currently included in Orbix. It is 'currently undergoing its final tests and will be shipped as part of Orbix in the very near future'; Iona now say that it will be included in Orbix 1.2.

The *Implementation Repository* is a set of text files which map server names to code images. An example of such a text file is :

```
Name : SBank
Comms : xdr/tcp
Activation : shared
Launch Command : /usr/users/mdrb/orbix/SBankServer
```

IDL for the Implementation Repository is included, and the Orbix daemon 'orbixd' is the server for this interface. Unfortunately it does not work, but Iona say that it is fixed in Orbix 1.2.

3 ACAS

3.1 CORBA Compliance

ACAS supports the CORBA dynamic interface, and includes exception handling with the usual CORBA 'environment'.

There is no support for CORBA IDL, and consequently no stubs. Interfaces are defined in a proprietary Class Repository Language (CRL). Arguments can only be of built-in datatypes; there are no structures, arrays, sequences or unions, though it is possible to include item lists in a list.

DEC say that ACAS 2.5 will have support for IDL, and therefore the generation of stubs. CRL will be replaced by IDL, in conjunction with two other languages: IML for implementation definition in servers, and MML for method mapping in clients.

The product looks very much like existing software with some ORB glue rather than something designed from the beginning to be CORBA-compliant.

3.2 Programming

3.2.1 Ease of Use

Methods can be implemented as shell (or other) scripts as well as code - this makes ACAS very flexible. Clients can also be implemented using the shell, via an interface 'acasinv' which invokes a method.

Servers can be written which can respond to requests running under Microsoft Windows and using DDE (Dynamic Data Exchange).

Servers can be made to start automatically.

It is possible, without too much work, to achieve a similar effect to passing an interface reference; an 'instance handle' contains 'instance reference data' which can be used, for example, in an ACAS implementation of 'simple bank' to pass a reference to a specific account.

Exception handling is straightforward with either the ACAS interface or the ORB interface.

Binaries are small (24K for echo client/server). This is accomplished by using shared libraries.

On the negative side:

Clients and servers are written in C.

It is difficult for a beginner to navigate through the documentation, which is quite large compared with other similar products.

The lack of support for CORBA IDL and stubs makes a lot of work for the programmer, who has to use the dynamic interface.

The lack of constructed datatypes also makes a lot of work.

3.2.2 Concurrency

No occurrences of ‘thread’ or ‘scheduling’ in the index.

However, you can write your own event notification and dispatcher functions, and there ought to be all sorts of possibilities there for setting off new threads. Unfortunately, the thread dispatcher seems to be called once, and you then need to wait for an event to happen. It is difficult to see how this mechanism can work in the unthreaded case without having access to the list of file handles that ACAS will do a ‘select’ call on, never mind the threaded case.

3.2.3 Trading and Type Safety

The server exports an interface by calling BOA_create_implementation. The information that needs to be supplied to this call involves a method server name and UUID and the address of a dispatcher function. If this information is supplied inconsistently, the results could be catastrophic.

The client imports an interface by calling BOA_create_interface and BOA_create, passing the class name. The latter returns an ORB_InterfaceDef. The class name is not looked up in the *Class Repository* until invocation is attempted. If the client attempts to invoke operations on the wrong type of interface, the server dispatcher code (generated by ‘acasgen’) will check the validity of the operation name, and of the parameter names and types, and any errors reported will appear in the client as the result of its call of ORB_Request_invoke.

As with Orbix, ACAS has a method of finding servers on other hosts which does not scale well. System (or user) context objects have a list of hosts on which any particular server will be looked for. This has the advantage over the Orbix method that the absence of any particular host does not prevent the server being found on other hosts, but the disadvantage that the addition of a host to the network requires updates on all existing hosts.

As mentioned with reference to Orbix, some compromise needs to be found between the scalability of ANSA/ODP trading and the flexibility of being able to start servers on demand.

According to the CORBA specification, BOA interfaces are for the use of servers rather than clients, so ACAS is non-compliant in this respect.

3.3 Administration

3.3.1 Repositories

There is a *Class Repository* and an API for accessing it. This combines the CORBA *Interface Repository* and *Implementation Repository,* though the API is proprietary. The repositories will be separated in ACAS 2.5.

4 DOME

4.1 CORBA Compliance

DOME implements some of CORBA, though CORBA compliance has quite clearly been added as an afterthought to an existing product.

Support for CORBA IDL at present is very, very limited. No typedefs, structures, arrays, strings, exceptions.

Those features of IDL which are present are not processed in a CORBA-compliant way; none of the standard CORBA interfaces are present (with the exception of the ORB 'obejct_to_string' and 'string_to_object' operations).

4.2 Programming

4.2.1 Ease of Use

The documentation is quite small, and not difficult to navigate.

Clients and servers are written in C++.

There are demonstrations and examples.

The product was produced by a two-man team in the UK, so support should be straightforward.

Even though the support for CORBA IDL is very limited, it is easy to adapt the code generated by the IDL compiler so that it marshals whatever argument types you like. This is facilitated by the fact that the C++ stream operators '<<' and '>>' are used for marshalling and unmarshalling requests.

On the negative side:

The product is in a state in which someone had to be sent to APM to install it.

Binaries are quite large (750 Kb or so). The main reason for this is that libg++ does not make use of shared libraries; DOME itself uses archive libraries too, and the size of binaries could be reduced if DOME changed to use shared libraries.

There is no daemon; port numbers for servers are obtained from /etc/services.

A rather irritating bug is that if you start a server when its port is in use, it doesn't inform you that there's a problem.

The IDL compiler generates C++ source files with names ending '.c'. This behaviour is non-standard.

The supplied class 'CopyString' suffers from the unfortunate (but defined) behaviour that:

```
CopyString fred("SBank");
if (fred == "SBankMgmt")
{
     ...
}
```

goes down the 'then' path. (This has been reported to the developers, and is due to be fixed).

There seem to be more classes than are strictly necessary, including a class in the client and a class in the server with the same name, which is a potential source of confusion. For example, a class 'Account' in the IDL ends up as classes 'Account' and 'Account_I' in the server, and a class 'Account' in the client which is different from 'Account' in the server. Orbix manages with only two classes. The developers have explained that the server class 'Account' is the existing implementation and the client class 'Account' is the proxy class, and that these need to have the same name so that an application can be split into client and server with minimum change.

Servers cannot be started automatically.

4.2.2 Concurrency

The most promising approach seems to be to change the 'dispatch_request' methods of the various implementation classes to fire off a separate thread to do all the work that they currently do, and then return. This should work, because they handle a complete request, including replying to the client, and they do not need to return any indication of success or failure.

If the server is built without the non-blocking I/O library, it works; if it is built with it, it does not work. This is unfortunate: DOME's 'select' call presumably needs to block, whereas any activity in a request-handling thread must not block. However, it seems unreasonable to expect a product to work perfectly in a multi-threaded environment when it was written without any thread support built in; DOME came out of this test surprisingly well, all things considered.

The 'dispatch_request' methods are generated by the IDL compiler, and it might therefore be thought undesirable to change them. However, given the small subset of IDL that is currently supported, that is hardly a problem, because extensive editing of generated code is already required.

4.2.3 Trading and Type Safety

There is no trader: the client finds the server by creating an ORB object and initialising it using /etc/services (or supplying a port number explicitly). The client then checks that an object of the required type can be created, and then does so. This involves the constructor of a specific class, and thus checks are possible; unfortunately, no such checks are done.

There are few checks at runtime: if a server is passed the wrong number or types of arguments (for example because the client is using an object reference of the wrong type), chaos will ensue. An experiment along these lines showed that the server's create_object method noticed (from the class name) that it was being asked for an unsupported object, but then the client and server both continued to run, producing core dumps.

4.3 Administration

4.3.1 Repositories

No mention of any such thing in the documentation.

5 Conclusions

5.1 CORBA Compliance

Orbix is way ahead of the other two on CORBA compliance. It is a full CORBA implementation, with IDL, stubs and the dynamic interface. Unlike the other two, it has clearly been designed with CORBA compliance in mind from the start.

ACAS has no IDL, and only the dynamic interface; DOME has limited support for IDL, and little CORBA compliance in other areas.

ACAS 2.5 will, however, be a considerable improvement on 2.1.

5.2 Programming

5.2.1 Ease of Use

Orbix wins again here.

Both Orbix and DOME have the advantage, compared with ACAS, of supporting C++ as the implementation language.

Orbix has the most convenient support route.

Both Orbix and ACAS have exception handling.

Orbix and ACAS both have a daemon, which helps clients to find servers; DOME relies on /etc/services.

Orbix and ACAS both support the automatic starting of servers.

ACAS has some nice features - the implementation of methods as shell (or other) scripts, and support for Microsoft Windows and DDE (Dynamic Data Exchange). Orbix 1.2 will also have these features.

ACAS is the clear winner on the size of binaries - this is mainly because of the difficulties of using shared libraries with C++. Orbix 1.2 will be better than 1.1.

5.2.2 Concurrency

DOME probably does best here now, as it is the only product of the three with which lwp threads work today. However, Orbix has threads designed in, and will offer better support when the facilities are working.

5.2.3 Trading and Type Safety

Orbix is again the clear winner. Type safety is easier to support in C++ than in C, and Orbix has most of the checks that can easily be made without going for a full conformance-based scheme. ACAS has good checking at run-time.

However, both Orbix and ACAS have trading schemes which do not scale well. Some means needs to be found of integrating ANSA/ODP trading with the starting of servers on demand.

5.3 Administration

5.3.1 Repositories

Orbix will support both Interface and Implementation Repositories as in the CORBA specification. In contrast, ACAS has its own Class Repository, with a proprietary API, which combines the functions of the two CORBA repositories; DOME has no repositories at all. ACAS 2.5 will be more CORBA-compliant than 2.1 in this respect.

5.4 Recommendations

Orbix has so many advantages over the other two products that we ought to be making use of it in our scenario.

ACAS has some useful features - support for Windows DDE especially - and is the only product of the three that can be used now for integrating PC-based applications. However, Orbix 1.2 will contain these facilities too.

References

[DEC 93a]

'DEC ACA Services: System Integrator and Programmer Guide', DEC, Maynard, Massachusetts, USA, 1992.

[DEC 93b]

'DEC ACA Services: Reference Manual', DEC, Maynard, Massachusetts, USA, 1992.

[IONA 93a]

'Orbix: Programmer's Guide', Iona Technologies Ltd, Dublin, Republic of Ireland, 1993.

[IONA 93b]

'Orbix: Advanced Programmer's Guide', Iona Technologies Ltd, Dublin, Republic of Ireland, 1993.

[OOT 93]

'DOME: Introduction and Reference Manual', Object-Oriented Technologies, Leamington Spa, 1993.