Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

Comparison of CORBA-compliant platforms

Mike Beasley

Abstract

CORBA is an important standard to the ANSA sponsors, many of whom are members of OMG. Products that claim to be CORBA-compliant have been appearing in the marketplace for nearly two years now. They come from a number of vendors, including some of the sponsors.

Those sponsors who have an ORB need to know how their product compares with others; other sponsors need to know which ORB to choose for research prototyping or for real-life projects. The comparison focuses on CORBA compliance and ease of use.

This paper presents the results of an evaluation of a number of such products; the intention is to include further products, and further versions of existing products, as they become available.

APM.1194.02 **Approved** 22nd March 1995
Technical Report

Distribution:
Supersedes:
Superseded by:

Comparison of CORBA-compliant platforms

Comparison of CORBA-compliant platforms

Mike Beasley

APM.1194.02

22nd March 1995

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	(01223) 515010
INTERNATIONAL	+44 1223 515010
FAX	+44 1223 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 in the first evaluation (Jun-Aug 93):

- 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.

The second evaluation (Sep 94 - Feb 95) added:

- Orbix 1.2, and a beta release of 1.3 on HP/UX
- ILU (Inter-Language Unification) 1.7, free from Xerox
- ORBeline from PostModern Computing Inc., Mountain View, CA
- DAIS 2.1 (pre-release) from ICL.

It did not include:

- the latest version of DOME (because Object-Oriented Technologies did not seem to be sufficiently interested in CORBA compliance)
- DEC's Object Broker 2.5 (because of difficulties with getting a non-disclosure agreement)
- Expersoft's product (they failed to respond to email)
- HP's ORB Plus (internal use only)
- any products that run on platforms unavailable at APM
- any products which don't claim to be CORBA compliant.

It is expected that further products will be included in the evaluation when they become available, 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 also 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.

The products are considered in terms of their suitability for prototyping in a research environment, and also for implementing real production-quality software.

It is considered more desirable if clients and servers can be implemented in C++ rather than only C. The reason for this is that it is that C++ makes it much easier to distinguish between different objects of the same class. Support for C++ is probably more important in a research environment, where its use is more common, than in industrial software production.

The size of the generated binaries is considered under the general heading of 'ease of use'.

An attempt was made to construct a multi-threaded server using each platform. 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* ? (We are concerned here with functionality rather than with compliance).

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[1]:

Table 1.1: Supported hardware and operating systems

HW/OS	Orbix	ACAS	DOME	DAIS	ORB Plus	ILU	ORBeline
SunOS 4	Y (4.1.3)	Y	Y	Y		Y*	Y
Solaris 2	Y (2.2, 2.3)		Y	Y (2.3)		Y*	Y (2.3)
HP/UX	Y (9.0)	Y (8.x)	Y (8.x)	Y (9.x)	Y (9.x)	Y†	Y
SVR4 Sparc				Y		Y	
SCO Unix				Y		Y†	
OSF/1	Y (3.0)	Y		internal		Y	Y (1.3)
Ultrix	Y	Y				Y	
AIX	Y			soon	Y (3.1)	Y†	Y
Stratus			Y			Y	
SGI Irix	Y (4.0, 5.1)					Y*	
UnixWare	Y			internal		Y	Y
Sinix	Y					Y	
DG/UX						Y†	
Pyramid				internal			
Linux						Y*	
VxWorks						Y	Y
LynxOS						Y	Y
pSOS						Y	Y
MS Win 3	Y	Y	Y	Y (3.11)			Y (3.1)

Table 1.1: Supported hardware and operating systems

HW/OS	Orbix	ACAS	DOME	DAIS	ORB Plus	ILU	ORBeline
OS/2			?	Y (2.1)			
NT	Y	Y	?	soon			Y (3.5)
VMS		Y	Y	Y (6.1)			
VME				Y (SV 294)			
Unicos							Y

1. Note that ILU is supplied as source, and therefore listed as supported on any Unix system; those that Xerox have tested themselves are marked with a *, and those specifically tested, but not by Xerox themselves, are marked witha †.

2 Orbix

2.1 Vendor/Product Information

Vendor:

IONA Technologies Ltd.

postal address: 8-34 Percy Place, Dublin 4, Ireland

phone: +353 1 668 6522

fax: +353 1 668 6573

email: info@iona.ie

anonymous ftp: ftp.iona.ie

www: http://www.iona.ie/

Versions evaluated:

- 1.1 on SunOS 4.1 using g++
- 1.2 on SunOS 4.1 using g++
- 1.2 on HP/UX 9.0 using HP C++
- 1.3β on HP/UX 9.0 using HP C++

Comments below apply to all the versions evaluated except where explicitly stated.

2.2 CORBA Compliance

Orbix implements all of CORBA. This includes exception handling and the dynamic invocation interface.

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

2.3 Programming

2.3.1 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++.
- servers can easily be made to start automatically.
- binaries on HP/UX with Orbix 1.3β are very small - typically 100-150Kb,
- there are two different mechanisms for relating implementation to IDL interfaces, which gives a certain amount of flexibility.

On the negative side:

- in some environments, binaries can be large (typically 1Mb). This problem is caused by some compilation systems, notably g++ where the run-time library does not come as a shared library.
- the programmer needs to be aware to some extent of the way in which exception handling is implemented. 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).

2.3.2 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, with an example using the lwp (light-weight processes) package on a Sun.

2.3.3 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.
- 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, and this can be done more easily than with ANSAware.

The Orbix method of trading 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.

Another problem is that is not possible for two persistent servers to support different objects of the same class, and for the client to find the object via the Orbix daemon, without the client needing to know that the objects are in different 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 by an explicit function call, 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.

2.4 Administration

2.4.1 Repositories

The *Interface Repository* is included in Orbix 1.2, and is implemented simply as a directory containing all the IDL source. This choice of implementation has the advantage of simplicity, but disadvantages in performance terms compared with more sophisticated implementations, as the IDL has to be analysed again when the Interface Repository server starts up. The server also seems to consume excessive quantities of virtual memory.

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.

3 ACAS

3.1 Vendor/Product Information

Vendor:

DEC

contact via the usual sales channels.

Version evaluated:

- 2.1 on SunOS 4.1 using gcc

Information from DEC about Object Broker 2.5 is also included.

3.2 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 Object Broker 2.5 has 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 lack of support for IDL and stubs suggests strongly that the product has been built from existing software and ORB glue rather than being designed from the beginning to be CORBA-compliant.

3.3 Programming

3.3.1 Ease of Use

- servers can be made to start automatically.
- binaries are small (24K for echo client/server). This is accomplished by using shared libraries.
- exception handling is straightforward with either the ACAS interface or the ORB interface.
- 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.

- 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).

On the negative side:

- it is difficult for a beginner to navigate through the documentation, which is quite large compared with other similar products.
- clients and servers can be written in C but not C++.
- 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 (arrays, structures, sequences) also makes a lot of work.

3.3.2 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.

3.3.3 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.

3.4 Administration

3.4.1 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 are separated in Object Broker 2.5.

4 DOME

4.1 Vendor/Product Information

Vendor:

Object-Oriented Technologies Ltd

postal address: 118-120 Warwick Street, Royal Leamington Spa, Warwickshire, England CV32 4QY

phone: +44 1926 313133

fax: +44 1926 422165

Version evaluated:

- a pre-release, built on site at APM, on SunOS 4.1 using g++.

4.2 CORBA Compliance

DOME implements some of CORBA; however, this is such a small subset of the CORBA specification as to suggest strongly that CORBA compliance has been added 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).

4.3 Programming

4.3.1 Ease of Use

- the documentation is quite small, and not difficult to navigate.
- there are demonstrations and examples.
- the product was produced by a two-man team in the UK, so support should be straightforward.
- clients and servers are written in C++.
- 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:

- servers cannot be started automatically.

- 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.
- the product was in such a state that someone had to be sent to help APM install it.
- if you start a server when its port is in use, it doesn't inform you that there's a problem. This deficiency caused some confusion during the evaluation.
- 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. Other products manage 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.

4.3.2 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.

4.3.3 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.

4.4 Administration

4.4.1 Repositories

No mention of any such thing in the documentation.

5 ORBeline

5.1 Vendor/Product Information

Vendor:

PostModern Computing, Inc.

postal address: 1897 Landings Drive, Mountain View, CA 94043, USA

phone: +1 415 967 6169

fax: +1 415 967 6212

email: info@pomoco.com

anonymous ftp: labrea.stanford.edu/pub/pomoco

Version evaluated:

- two pre-release versions on HP/UX 9.0 using HP C++.

5.2 CORBA Compliance

ORBeline implements all of CORBA, including exception handling and the dynamic invocation interface.

ORBeline has obviously been designed from the beginning to be a CORBA-compliant system.

5.3 Programming

5.3.1 Ease of Use

- ORBeline is a well thought-out and well documented product, with an adequate supply of demonstrations and examples, and good support by email.
- clients and servers are written in C++.
- servers can easily be made to start automatically.
- binaries (on HP/UX with the native HP compiler) are impressively small (typically around 300Kb). This is achieved by the use of shared libraries. In other environments, using g++ for example, it would not be possible to achieve such small binaries; it does depend on support of shared libraries by the compilation system.
- there is claimed interoperability with Sun and HP's ORB products, by implementing their on-the-wire protocols.
- ORBeline handles the _bind call flexibly: by default, a connection is established at bind time, but this default can be changed so that the connection is not established until first use.

- ORBeline has support for event handling, object migration and object replication.
- IDL strings are mapped to a string class. This makes the use of strings easy.

On the negative side:

- the C++ mapping may need some changes to conform to what has recently been agreed by the OMG. It is more difficult to use than it might be, because it has explicit accessor methods rather than public data in its structures and sequences.
- the use of nested classes makes C++ programming somewhat long-winded.
- exceptions have to be checked for by an explicit call; unlike Orbix, there are no exception handling macros to make life easier.

5.3.2 Concurrency

There is a separate thread-safe version of ORBeline, packaged separately.

5.3.3 Trading and Type Safety

The functions of a trader are supported in ORBeline by:

- the *smart agent*: this keeps track of all active objects. There only needs to be one on a LAN; clients and servers find it by broadcasting.
- the *object activation daemon*: this activates object implementations when required.

It is possible to run two servers handling the same class of object without any problems, and without any need for the client to know which server the object will be in.

The use of broadcasting to find the smart agent may cause problems on large LANs. If separate LAN segments are connected by bridges, a smart agent will need to be run on each segment.

The following features help with type safety:

- a cast from ‘Object*’ to a pointer to an IDL class ‘Fred*’ is performed by using a static member function ‘_narrow’ in class ‘Fred’; this function checks that the cast is permissible.

5.4 Administration

5.4.1 Repositories

ORBeline supports both the Interface Repository and the Implementation Repository. The Interface Repository represents the IDL on some binary, indexed form.

There is a significant difference in the implementation of ‘any’ compared with Orbix; with Orbix, if an ‘any’ is a structure, the value must be cast from a void* to a pointer to the appropriate structure, but with ORBEline, the value is an object with operations to find the names of the structure members and obtain their values.

6 ILU

6.1 Vendor/Product Information

Vendor:

Xerox Corporation

email: ilu@parc.xerox.com

anonymous ftp: parcftp.parc.xerox.com

www: ftp://parcftp.parc.xerox.com/pub/ilu/ilu.html

Versions evaluated:

- 1.6.4 (built at APM) on HP/UX 9.0 using HP C++
- 1.7 (built at APM) on HP/UX 9.0 using HP C++

6.2 CORBA Compliance

ILU has its own Interface Specification Language (ISL), but includes an IDL-to-ISL translator, which is invoked automatically by the various stub generators when they are asked to handle a file with a .idl extension.

The IDL-to-ISL translator uses the public domain IDL compiler front end from Sun, and therefore accepts the whole IDL language, with the following exceptions:

- the IDL types `Object` and `any` are disallowed by the translator
- use of `context` clauses on operations is prohibited.

The C++ language mapping is intended eventually to be compatible with the CORBA mapping.

There is no Dynamic Invocation Interface or Repositories. The Basic Object Adapter interface is not supported, although the required functionality is all present.

Obviously CORBA compliance has been added as an afterthought to an existing system, but one which was quite close to CORBA in many ways.

6.3 Programming

6.3.1 Ease of Use

- support by email was good when problems had already been found and fixed by Xerox, but when a new problem was found, there was quite a long wait (remember that it is a free product, however).

- the range of programming languages supported is impressive - not only C and C++, but also Common Lisp, Modula-3 and Python(the evaluation confined itself to C++, however).
- SimpleBank was very easy to get working with version 1.7 (1.6.4 had a serious error in C++ stub generation). In particular, returning an object reference as a result worked first time.
- the evaluation was the first attempt to run ILU on HP/UX. The problems with version 1.6.4 were certainly no worse than those encountered when porting ANSAware to a new version of UNIX; 1.7 was easier.

On the negative side:

- the documentation was found to be lacking in certain areas, notably support for C++.
- there is no way that servers can be made to start automatically.
- binaries are large (1Mb typically). This is because of a lack of shared library support (which is very difficult to achieve in a product which is freely issued in source form, and portable to various UNIXes with a variety of compilers). However, anyone who was really concerned about the size of binaries could build their own shared libraries, unless they were using g++ which has a non-shared run-time library.

6.3.2 Concurrency

The manual mentions a 'lightweight process' system for Common Lisp, but nothing similar is available for any other of the supported languages.

6.3.3 Trading and Type Safety

ILU supports a 'simple binding method', where objects can register their string binding handles and type IDs in whatever name service or registry is convenient. The supplied implementation of this method stores the string binding handles in files. Potential clients have to specify the 'object ID', which includes the host name and a dozen or so hexadecimal digits (some of which are the process ID of the server), so this is of limited usefulness. The other drawback of the supplied implementation is that the files which hold the bindings need to be accessible using NFS, with the same names from all machines.

In the evaluation, servers were made to output their string binding handles to their standard output; the binding handles were then supplied to clients on invocation.

If an operation is invoked on an inappropriate interface, a 'protocol error' exception is returned to the client.

6.4 Administration

6.4.1 Repositories

ILU contains no interface repository, and no implementation repository.

7 DAIS

7.1 Vendor/Product Information

Vendor:

ICL

postal address: DAIS Product Centre, ICL Corporate Systems, Wenlock Way, West Gorton, Manchester, England M12 5DR

phone: +44 161 223 1301 x3771

fax: +44 161 223 0482

email: dais@wg.icl.co.uk

www: http://www.icl.co.uk/ or http://www.cityscape.co.uk/ICL

Version evaluated:

- pre-release of 2.1 on HP/UX 9.0 with HP C.

7.2 CORBA Compliance

DAIS implements most of CORBA. The exceptions are:

- there is no Dynamic Invocation Interface (DII)
- the only implementation activation policy supported is ‘persistent server’
- the ‘Object’ and ‘ORB’ interfaces are incomplete.

DAIS has been developed from ANSAware; CORBA compliance has been added later.

7.3 Programming

7.3.1 Ease of Use

- DAIS is well documented and supported, and comes with an adequate supply of examples.
- servers can be made to start automatically by using the node manager, as in ANSAware.
- in addition to the CORBA Basic Object Adapter, DAIS has an *Extended Object Adapter*. Extended Objects have better control of state and concurrency, and additional facilities such as object containment, life cycle, relocation and migration.
- DAIS supports multiple communications protocols, and is very flexible in this respect. In addition to TCP and UDP, there is support for OSI protocols and for named pipes for communication within a process.

- the *Alert Service* allows system management information to be channelled to a single point
- the *Configuration Service* allows configuration information to be distributed
- there is support for *transactions* via the DAIS/TI optional product (interworking with TPMS on ICL mainframes running VME) and DAIS/OTS (supports the X/Open Distributed Transaction Processing Reference Model)
- *security* features are provided via the DAIS/SE optional product
- transparent access to hetergeneous *databases* is possible using DAIS/IS.

On the negative side:

- clients and servers have to be written in C at present.
- binaries are large (500Kb or so); indeed DAIS binaries are larger than ANSAware ones. The problem is that DAIS does not use shared libraries. The structure of the ANSAware code makes the use of shared libraries difficult, but not impossible.

7.3.2 Concurrency

DAIS includes that same thread support as is present in ANSAware. It therefore does not depend on any other threading package being available.

DAIS also supports event counts, sequencers and mutexes.

7.3.3 Trading and Type Safety

DAIS contains a full ANSAware trader, with properties/constraints, contexts and types, local/master traders and trader federation.

If you try to use an object reference as if it were another type of object reference, you will typically receive an exception because the interface does not support the operation you asked for.

7.4 Administration

7.4.1 Repositories

DAIS includes no CORBA-compliant repositories. However, the Node Manager database can be regarded as a non-compliant Implementation Repository.

8 Conclusions

8.1 CORBA Compliance

Orbix and ORBeline are ahead of the others on CORBA compliance. They are full CORBA implementations, with IDL, stubs and the dynamic interface. Unlike the others,they have clearly been designed with CORBA compliance in mind from the start.

DAIS has no dynamic interface (and no repositories), but is otherwise CORBA compliant.

ACAS has no IDL, and only the dynamic interface.

DOME has limited support for IDL, and little CORBA compliance in other areas.

ILU has good IDL support, but limited CORBA compliance in other areas.

Object Broker 2.5 is believed to be a considerable improvement on ACAS 2.1; because of problems with non-disclosure agreements, however, we have been unable to obtain a copy for evaluation.

8.2 Programming

8.2.1 Ease of Use

Orbix and ORBEline win again here.

ACAS and DAIS only support C at present; ILU supports C, C++, Common Lisp, Modula-3 and Python; the others all support C++ only.

Orbix, ORBeline, ILU and DAIS all have good support by email.

All the products except DOME support exceptions.

All the products except DOME and ILU support the automatic starting of servers.

DAIS has some features which are very important in ‘real-life’ commercial systems - support for multiple protocols, the alert and configuration services, and support for Transaction Processing, Databases and Security.

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

On the size of binaries, ACAS has the advantage, because of the difficulties of using shared libraries with C++. ORBeline and Orbix 1.3β, however, support shared libraries on HP/UX.

8.2.2 Concurrency

DAIS has ANSAware threading, and does not depend on any threading package being provided.

Orbix and ORBeline have thread support designed in.

DOME can easily be made to work with Sun's lwp thread package.

ACAS and ILU do not support concurrency at all.

8.2.3 Trading and Type Safety

Only DAIS has a proper trader, to enable clients to find servers. Orbix, ACAS and ORBeline all have daemons; DOME relies on /etc/services, and ILU relies on binding handles being passed by some unspecified means. ORBeline also has a 'smart agent'.

DAIS therefore wins on trading - it has the power and flexibility of a full ANSAware trader. The 'trading' schemes in Orbix and ACAS do not scale well. Some means needs to be found of integrating ANSA/ODP trading with the starting of servers on demand.

Orbix and ORBeline are again the clear winners as far as type safety is concerned. Type safety is easier to support in C++ than in C, and Orbix and ORBeline have most of the checks that can easily be made without going for a full conformance-based scheme. ACAS, ILU and DAIS have good checking at run-time.

8.3 Administration

8.3.1 Repositories

Orbix and ORBeline 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 and ILU have no repositories at all. DAIS has the Node Manager database, which is a non-compliant implementation repository, but no interface repository. Again, Object Broker 2.5 is believed to be more CORBA-compliant than ACAS 2.1 in this respect.

8.4 Recommendations

Orbix and ORBeline have many advantages over the other products for prototyping in a research environment. ORBeline has a slight edge in performance terms over Orbix, because it implements the interface repository more efficiently.

DAIS has the best trading of all the products evaluated, and also many useful facilities for real-life applications.

ILU has the advantage (and disadvantages !) of being free; being issued as source it is potentially available on a wider range of platforms. It also supports more languages.

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 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.

[ICL 95a]

'DAIS: Installation Guide', 55350/002, ICL, Manchester, 1995.

[ICL 95b]

'DAIS: System Administration Guide', 51217/002, ICL, Manchester, 1995.

[ICL 95c]

'DAIS: Application Development Guide', 55218/002, ICL, Manchester, 1995.

[ICL 95d]

'DAIS: Programmers Reference Guide' (draft), ICL, Manchester, 1994.

[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.

[POMOCO 94a]

'ORBeline Reference Manual', PostModern Computing, 1994. ftp://labrea.stanford.edu/pub/pomoco/orbeline.docs.tar.gz

[POMOCO 94b]

'ORBeline User's Guide', PostModern Computing, 1994. ftp://labrea.stanford.edu/pub/pomoco/orbeline.docs.tar.gz

[XEROX 94]

'ILU 1.6.4 Reference Manual', Xerox Corporation, 1994. ftp://parcftp.parc.xerox.com/pub/ilu/1.6.4/ilu-manual-1.6.4.ps.Z