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**

Training

ANSAwise - CORBA Directory Services [Eurocontrol]

Mark Madsen

Abstract

Organizations that are considering using CORBA technology will wish to see examples of objects that are beginning to populate the CORBA Object Management Architecture.

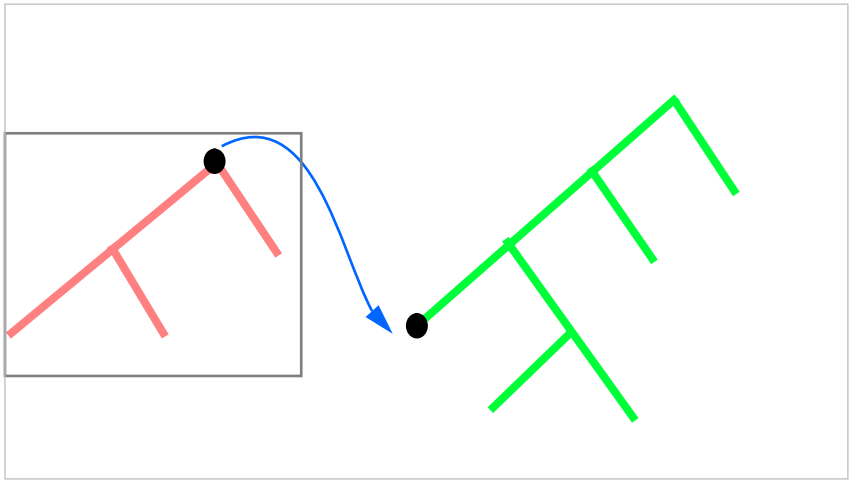
The CORBA Common Object Services are the first of these, but, being somewhat abstract, the specifications can be hard to understand.

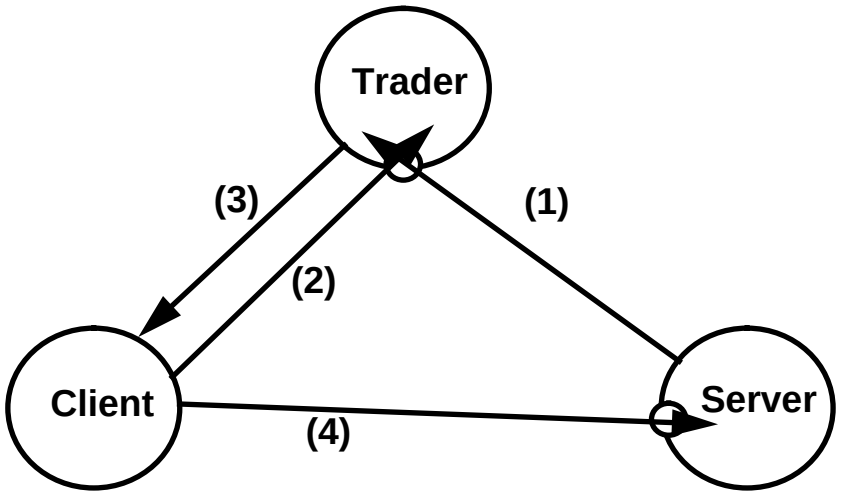
This module of the ANSAwise training programme explains the function of the CORBA directory services. These are the Naming Service specified in the CORBA Object Services Specification Volume 1, and the planned Trading Service. The presentation also relates these services to the ANSA/ODP Computational Model, and outlines the likely future of Object Services.

APM.1620.01 **Approved** 16th October 1995
Briefing Note

Distribution:
Supersedes:
Superseded by:

CORBA Directory Services





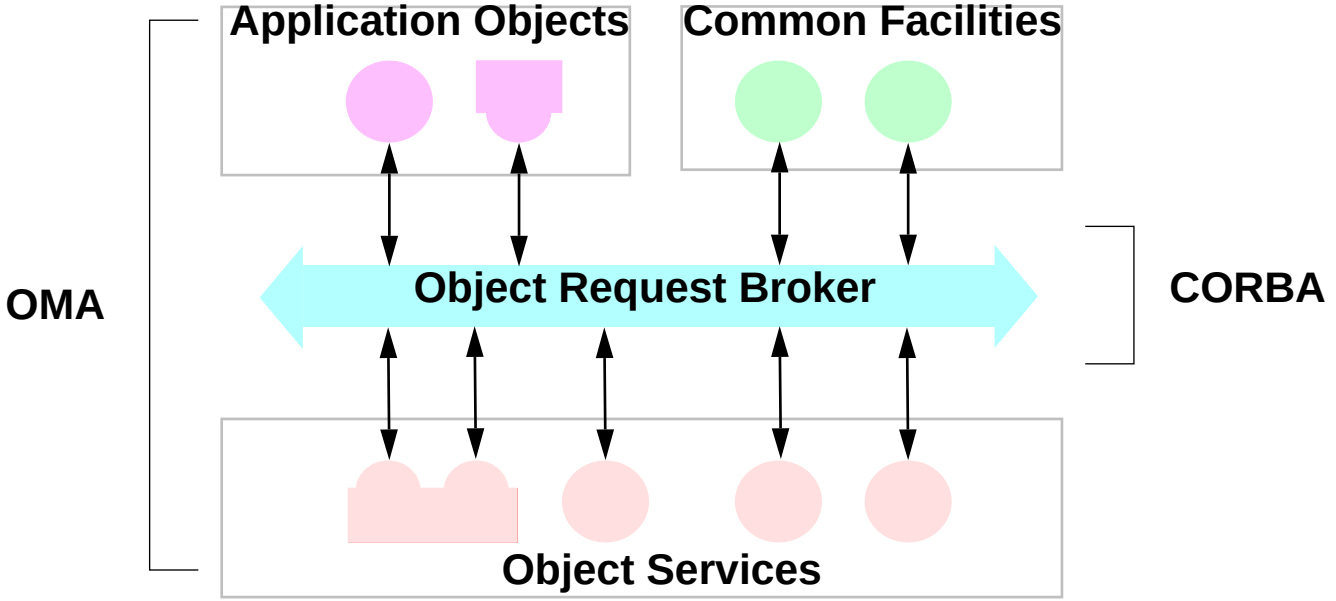
In this session

- ***Describe the general design principles for Object Services***
- ***Explain the Directory Services***
  - **Naming**
  - **Trading**
- ***Show some parts of specifications in detail***
- ***Sketch the future Object Services to be provided***

Stating service requirements

- *There are two potential ways to identify something*
  - by naming it (as in a “white pages” telephone directory)
  - by describing it (as in a “yellow pages” telephone directory)
- *These functions are sometimes combined in a directory service*
  - but they are logically separate, just as real telephone directories are
- *CORBA has separate Naming and Trading services*

The Object Management Architecture



- ***Consists of the Object Request Broker (ORB), plus objects***
  - **Objects are Object Services, Common Facilities, or Application Objects**

General Object Service design principles

- ***Build on CORBA Concepts***
- ***Specify Basic, Flexible, and Generic Services***
- ***Allow Local and Remote Implementations***
- ***Consider Reliability, Performance, Scalability, and Portability***
- ***Consider Future Object Services***
- ***Consider Conformance to Existing Standards***

General Object Service design techniques

- ***Partitioning of interfaces to a service***
- ***Inheritance of specifications***
- ***Callback interfaces***
- ***Avoidance of global identifiers***

Consequences of these principles

- ***Object service specifications are small***
  - **much smaller than a typical API**
  - **typically only 100 lines of CORBA IDL, with 20 operations**
- ***Object service specifications contain several interfaces***
  - **typically 2 or 3**
- ***There are no 'convenience functions' that gather together series of operations***
  - **these do not belong in the interface, but at a higher level**
  - **there is only one basic operation for a given function**
- ***Object service specifications are rather abstract***
  - **it can be hard to see how to apply them**

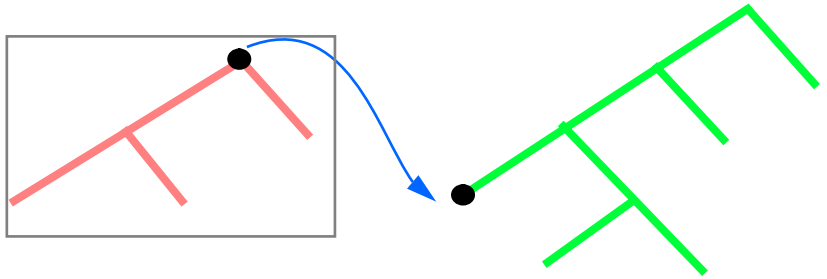
Scalable Service Implementations

- ***Because distributed systems are going to be very large...***
- ***... the US telephone network will contain***
  - **100,000,000 nodes**
  - **1000 administrative domains**
  - **100 versions of any software component**
- ***It must be possible to distribute not just access to object services***
  - **but also the implementation of the services**

CORBAservices: Common Object Services Specification

- ***This covers many services***
  - **Naming, Event Management, Persistence, LifeCycle, Concurrency, Externalization, Relationships, Transactions**
- ***The OMG Object Services Task Force are working on new services***
  - **Security, Time, Licensing, Properties, Query, Trading,...**
- ***In this session we will concentrate on Naming, Events, and LifeCycle***

The Naming service



- ***The naming service is the simplest possible directory service***
  - **given a name, it will return you an object reference...**
  - **...any kind of CORBA object can be named**
- ***The naming service is a 'white pages' service...***
  - **it is not a 'yellow pages' service for finding objects from a description**
  - **... that would be the function of a separate trading service**

Names

- ***Names can be simple or compound***
  - simple names have one component (like **filenames**)
  - compound names have multiple components (like **pathnames**)
- ***Each component has***
  - an identifier (a **string**)
  - a kind (a **string**)
- ***Consider a typical filename...***

  `myfile.c`
  - the identifier is **myfile.c**
  - the kind is **c_source**
- ***The naming service does not interpret in any way the identifier or kind***
  - it does not understand what they signify; it just matches them

No name interpretation

- ***Naming conventions are not part of the naming service***
- ***The naming service does not have a syntax for names***
  - **it does not parse compound names**

    `mydir/subdir/myfile.c`
  - **there are no wildcards, or case-matching rules**
- ***Applications are responsible for structuring compound names***
  - **for example, this is a three-part structure**

    `ansa.co.uk`
- ***There are no predefined special names, not like***

  `/.:/hosts, /.:/fs, /.:/sec`

IDL for Names

```
module CosNaming
{
   typedef string Istring;// For future internationalization

   struct NameComponent {
      Istring id;
      Istring kind;
   };


   typedef sequence <NameComponent> Name;

   ...

};
```

Naming contexts

- ***A naming context is like a directory***
  - ***names*** **are bound to** ***objects*** **in a** ***naming context***
- ***Naming contexts can be arbitrarily structured***
  - **No ‘universal root’**
  - **No ‘absolute pathnames’**
- ***A naming context is itself an object***

Separation of creation and naming

- ***Creating an object, and binding a name for it...***
  - **...these are two quite separate operations**
- ***So, this is not like 'creating a file' in most file systems...***
  - **...normally, when you create a file, you must give it a name and a directory to put it in**
- ***The naming service is not involved with creating objects***
  - **except for naming context objects themselves**

IDL for Naming Contexts - 1

```
module CosNaming {
   ...
   interface NamingContext {

      ...
      // ... Exceptions omitted,see full specification

      void bind (in Name n, in Object obj)...;
      void rebind (in Name n, in Object obj)...;
      void bind_context (in Name n, in NamingContext nc)...;
      void rebind_context (in Name n,
                           in NamingContext nc)...;

      Object resolve (in Name n)...;
      void unbind (in Name n)...;
```

IDL for Naming Contexts - 2

```
        NamingContext new_context();
        NamingContext bind_new_context(in Name n)...;
        void destroy ()...;

        void list (in unsigned long how_many,
                   out bindingList bl,
                   out BindingIterator bi);

    };


    ...


};
```

Some approximate Unix equivalents

- ***bind ~= link***
- ***unbind ~= unlink***
- ***bind_new_context ~= mkdir***
- ***destroy ~= rmdir***

Uniqueness of names

- ***Objects can have more than one name***
  - **or no name at all (in which case the naming service cannot find them)**
  - **... the naming service is just one way of finding objects**
  - **... it is not the only way**
- ***Conversely, in a naming context, a name can denote only one object***
  - **simple names cannot be duplicates**

Other features of the naming service

- ***The naming service does not know about the types of objects***
- ***Names are independent of the location of name services***
- ***Existing (non-CORBA) name services can be encapsulated***

Exclusions from the Naming service

- ***No 'rename' operation***
  - **it is excluded, because it would have to rely on a transaction service to guarantee correctness**
- ***No administration of names***
  - **this is the responsibility of higher-level software**

The Naming service specification

- ***One module (CosNaming), containing***
  - **interface BindingIterator**
  - **interface NamingContext**
- ***A names library, containing***
  - **interface LNameComponent**
  - **interface LName**
- ***The names library is specified in pseudo-IDL***
  - **allowing names to be handled as pseudo-objects**
  - **... pseudo-objects cannot be passed across IDL interfaces**
- ***You can use CosNaming without the names library***

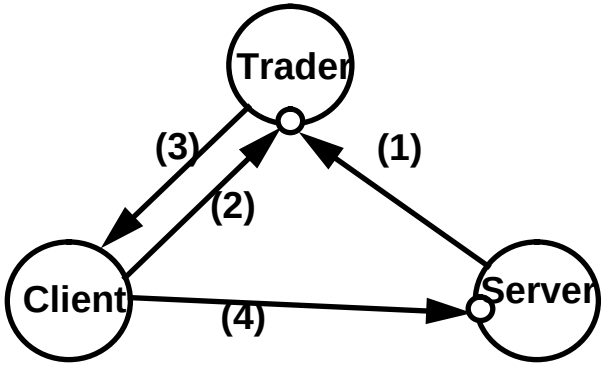
Anticipated changes to the Naming service

- ***Other forthcoming Object Services will have an impact on the Naming service***
  - **Security**
  - **Change Management/Versioning**
  - **Internationalization**
- ***The Naming interfaces will need to evolve accordingly***

Using the Naming service

- ***Almost every application will need to use the naming service***
  - **directly or indirectly**
- ***The naming service deliberately provides a bare minimum service***
  - **you may wish to build services with higher-level operations on top of it**
  - **...searching and sorting, for instance**
  - **...similar facilities to those in X.500 or the DCE Directory Services**
  - **...possibly as Common Facilities or Application Objects**

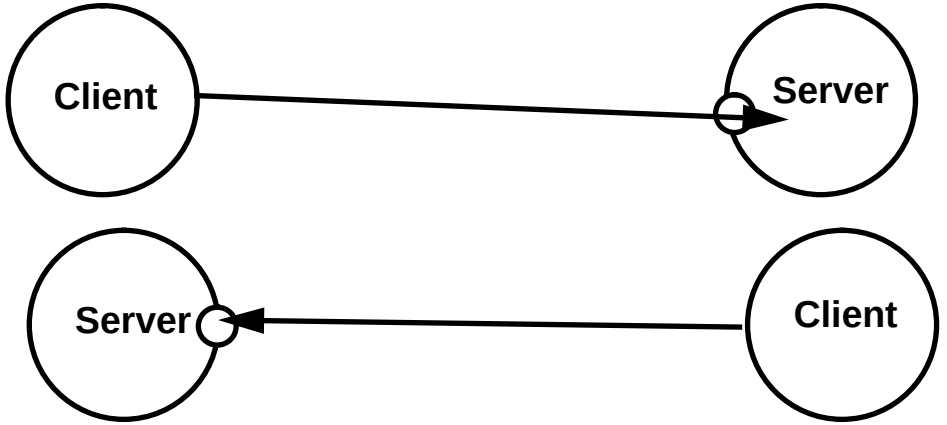
Trading and Federation

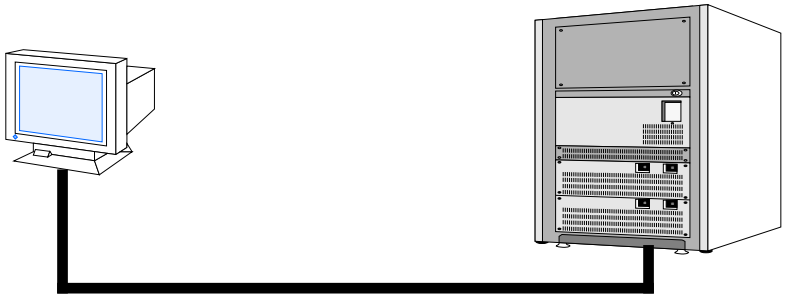


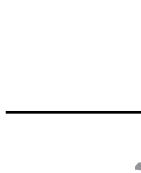
- ***Explain the purpose of trading in a distributed system***
  - **the roles of client, server, and trader**
- ***Show a simple form of trading in action***

Client and Server Roles

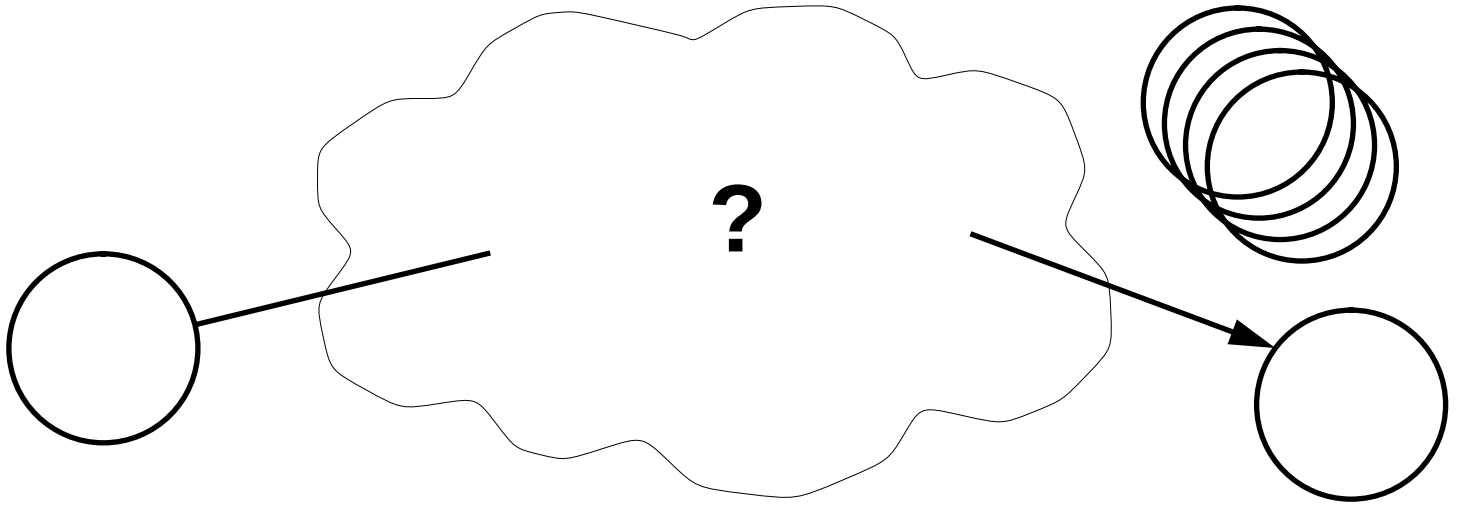
- ***The roles are determined by the objects themselves***







The need for Trading



- ***How can clients find servers that provide the services that they need?***
  - **in the future, there will be millions of interconnected servers around the world**
  - **clients will come and go dynamically**
  - **servers will come and go dynamically**

Naming is not enough

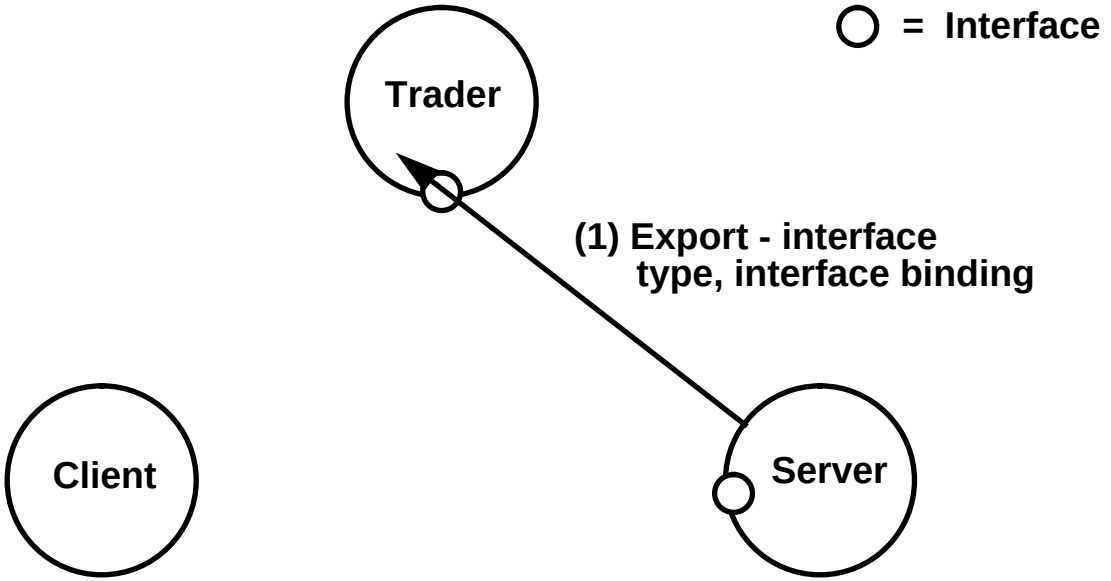
- ***Trading is necessary***
  - **we cannot rely on clients being able to name servers...**
  - **....the server may not even exist when the client was created**
- ***Trading works by matching descriptions provided by clients and servers***

Trading - Basic needs

- ***Server must state what it provides***
  - **it must export a service offer**
- ***Client must state what it requires***
  - **it must import a service offer**
- ***Trading must find a service offer that matches the request***
  - **there may be many such offers...**
  - **...there may be none**

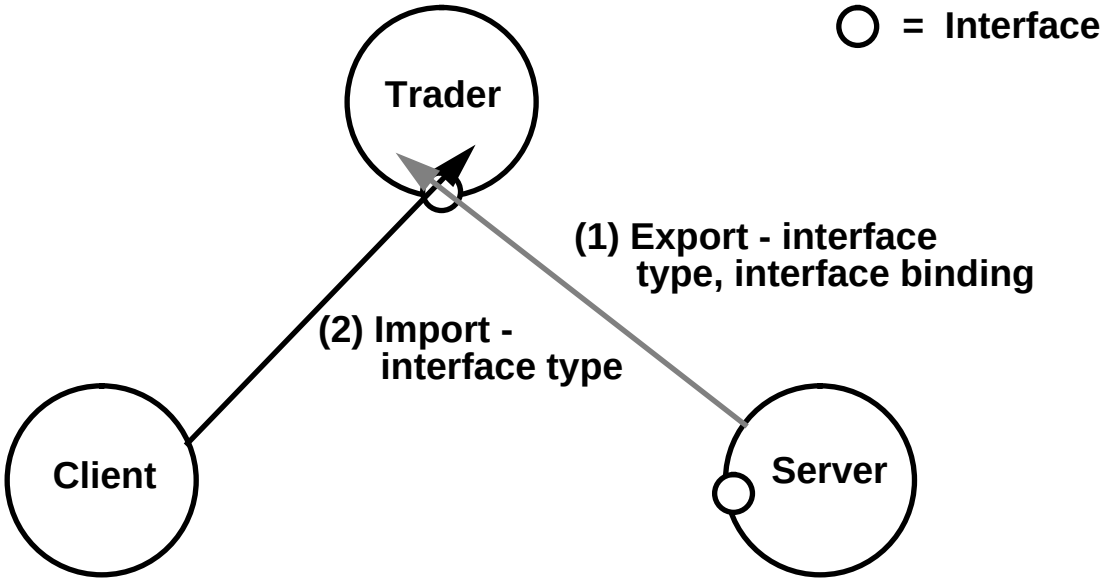
Steps in Trading - (1)

- *(1) Server exports a service offer to the Trader*



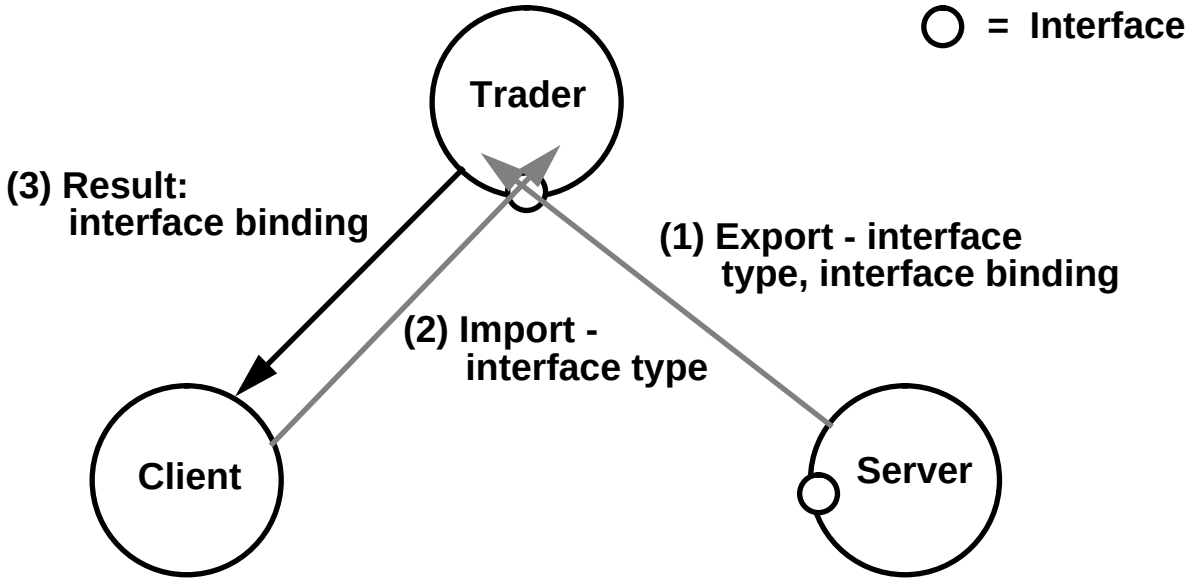
Steps in Trading - (2)

- *(2) Client requests a service offer from the Trader*



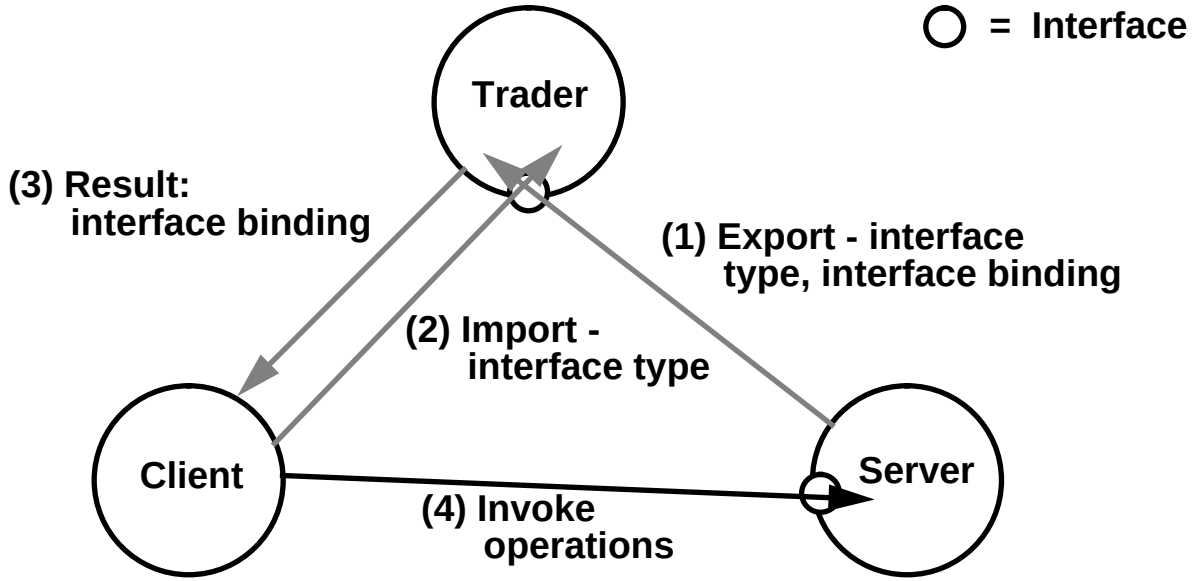
Steps in Trading - (3)

- *(3) Trader returns a matching service offer to the client*
  - it returns the interface binding given by the server



Steps in Trading - (4)

- *(4) Client uses interface binding to invoke the server's operations*
  - Trader takes no further part in the interaction
  - an interface binding from the Trader is invoked just like any other



Matching requests with offers - type conformance

- ***How does Trading decide whether a client request matches a server offer?***
  - **it uses the interface *type conformance* concept from the Computational Model**
- ***If the client request and server offer interface types do not conform, they are incompatible, and cannot match***

Other matching criteria

- *Type conformance is not sufficient*
  - who owns the service?
  - what will it cost to use?
  - where is the service, and can it be reached?
- *These criteria are known as properties*
  - (name, value) pairs
- *Preference criteria sort matching offers into order*
- *Scope criteria control where to look for offers*

Using properties

- *Property name/value pairs are specified when offer is exported*
- *Clients importing services may specify:*
  - Acceptable property constraints, for example

    `(PaperSize=='A4')`

  - Selection of offer with maximum (or minimum) value of a property
- *Arbitrary combinations of constraints may be specified in the form of a logical expression, for example*

  `(Node == 'basilisk') and (Price <= 200)`

Trading in large distributed systems

- ***Because there will be millions of servers in the world:***
  - **there will be many Traders providing the Trading service**
  - **... Traders must be interconnected**
  - **... the Trading service must itself be distributed for scalability**
  - **... and cannot be centralized**
  - **this is called *federated trading***
- ***And also because organizations will wish to control their own Traders:***
  - **to determine who sees which services**

Implementing Traders

- ***Traders can be implemented in several ways***
  - **as small, self-contained objects: optimized for locality and speed**
  - **as databases: optimized for large numbers of offers, and sophisticated query facilities**
  - **integrated with the naming service: optimized to share the federation facilities of the naming service**
- ***Whichever implementation is chosen, traders can be federated***
- ***The research community has used all these implementations and proved the scalability of traders***

Standards in Trading

- ***Standards work has been carried out in ISO/IEC using the ODP Reference Model***
  - **issued as DIS 13235 (ODP Trading Function)**
  - **Annex A includes a CORBA IDL Specification**
- ***OMG CORBAservices COSS RFP 5 is for a Trader of exactly this kind***

Summary

- ***For a description of the service design principles***
  - **see *CORBAservices* by the Object Management Group (Wiley)**
- ***For the specifications of the services, also see CORBAservices***
- ***For future Object Services***
  - **await future publications**

Summary

- *The Naming Service provides the basic infrastructure necessary to support object referencing*
- *Trading is necessary to support large distributed systems*
- *The Trading service (trader) is provided just as any other service*
- *A trader manages a database of service offers and matches requirements to offers*
- *Once a trader has introduced a server to a client, it plays no further part in the interaction*

Other Specified Object Services

- ***Persistence - object storage when an object is inactive***
- ***Externalisation - object storage on (removable) media***
- ***Relationships - associations between objects***
- ***Concurrency - control of concurrent operations***
- ***Transactions - serializable operations***

Obtaining Object Service implementations

- ***Initially, ask your ORB vendor***
- ***In the future you should be able to 'shop around'***
  - **and buy different Object Services implementations from wherever you wish**
  - **... with the performance and quality-of-service that you require**
- ***Or you could implement them yourself!***
  - **to integrate with your own existing event handling software, for instance**

Trading and Federation - more information

- *For a general discussion of trading and federation, see The ANSA Model for Trading and Federation (AR.005.00)*
- *For naming issues, see The ANSA Naming Model (AR.003.01)*
- *For security issues, see A Framework for Federating Secure Systems (AR.008.00)*
- *For a discussion of Quality of Service issues in multimedia applications, see Integrating Multimedia into the ANSA Architecture (TR.028.00)*

Trading - Extra questions

- ***If Trading is a service, how does a client find the Trading service to start with?***
  - **Yes, there has to be a way to bootstrap the client. This can be done by building a Trader interface-reference into the infrastructure, by broadcasting, or some other special means. This is really an Engineering issue**
- ***How does Trading understand what the service offers mean?***
  - **It doesn't need to. It just needs to decide whether they match service requests**

Advanced trading - negotiation

- ***The 4-step approach described earlier will satisfy many needs, but not all...***
  - **...if several offers match, which one should be chosen?**
- ***Different offers may have a different quality-of-service, or cost***
  - **This will require more sophisticated patterns of *negotiation*...**
  - **... for trade-offs between quality and cost**
  - **... involving another service, a *negotiation service***
  - **the Trading service is neither accountable nor responsible for the quality of service described in service offers**

Advanced trading - different lifecycle epochs

- *The description so far implies that trading happens at run-time*
  - this is typical
- *Does it make sense for trading to happen at earlier epochs during the service development lifecycle?*
  - at application link time?
  - at application compile time?
  - at design time?
  - at specification time?

Advanced trading - compile/link time epoch

- *At compile/link time, special tools could be used...*
  - to test that a set of services could trade successfully
  - to determine an optimum set of bindings within this set
  - to replace the trading operations by this set of bindings
- *... to optimize a group of closely-related services*

Advanced trading - design/specification epoch

- ***Applications and service designers need to know what services already exist***
  - **to avoid re-inventing the wheel**
- ***This information is already available in the trading service***
  - **intelligent browsing tools can aid the designer**

Advanced trading - new service wanted...

- *Suppose the designer does not find the service they require...*
  - ...but they do not want to implement the service themselves
- *They can simply place a service request for this non-existent service*
  - ...like a Service Wanted advertisement
- *A special service can intercept this request*
  - and create a dummy service

Use of MatchMaking Service