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 - The CORBA Object Lifecycle

Chris Mayers

Abstract

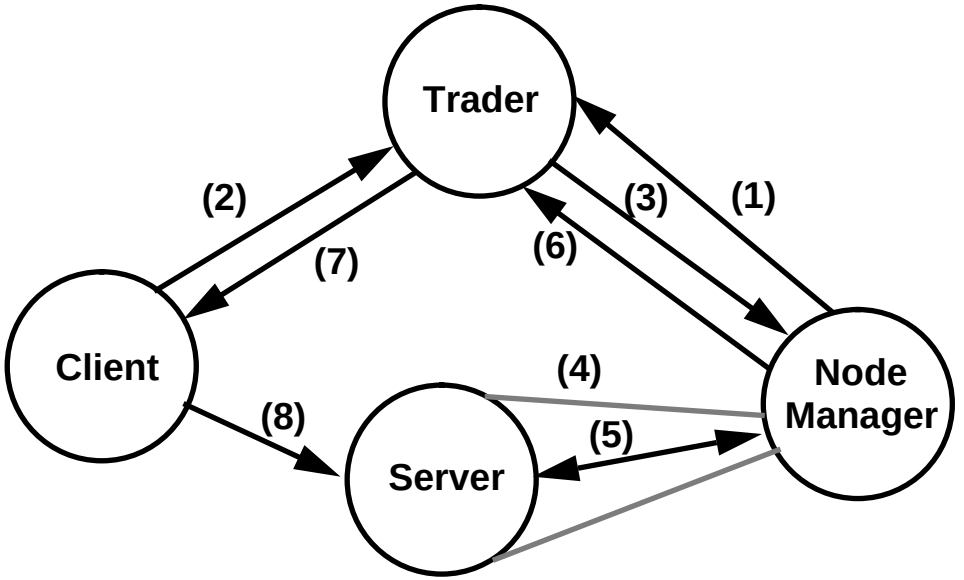
Organizations wishing to deploy CORBA applications need to understand the minimum set of object services that these applications will require.

This module of the ANSAwise training programme describes the Lifecycle/Factory and (closely-related) Relationship services. It also discusses more advanced services that can be coupled with them; the Node Manager service as delivered in ICL's DAIS (derived from ANSAware). It notes the extensions to the CORBA services that would be necessary for this.

APM.1748.01 **Approved** 3rd April 1996
Briefing Note

Distribution:
Supersedes:
Superseded by:

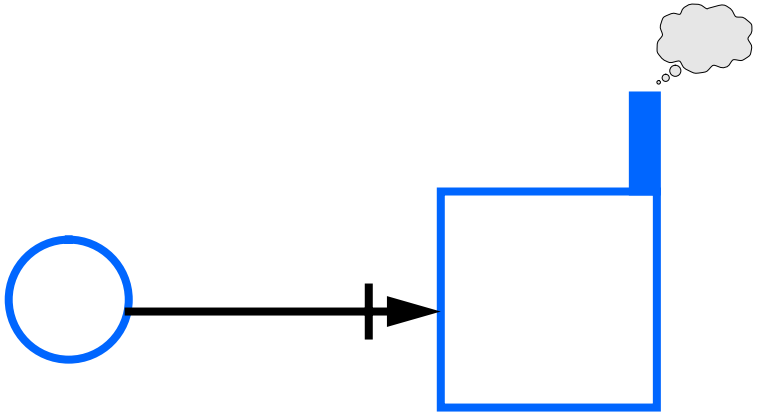
The CORBA Object Lifecycle



In this session

- **Examine the CORBA Lifecycle service**
  - **and the Relationship service**
- **Show how other services can be coupled with these services**

The LifeCycle service

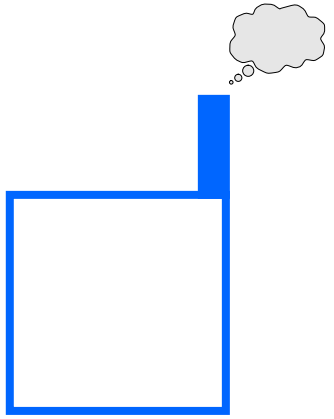


- Life cycle services allow applications to control
  - creation of objects
  - removal of objects
  - copying of objects
  - moving of objects
- The LifeCycle service depends on the Naming service

Where should objects be created?

- Objects must be created in some location
  - and when they are created, they use resources...
  - ... memory, disk space, CPU time,...
- So, the choice of location must be controlled
  - possibly under client control
  - possibly subject to some administrative policy
- The same issue arises when moving or copying objects
  - for their new locations

Factory objects



- A factory object is an object that can create other objects
- Factories do not have special interfaces
  - they are specified in IDL
- Factories are responsible for allocating resources to objects that they create
  - according to client control/administrative policy

Factory interfaces

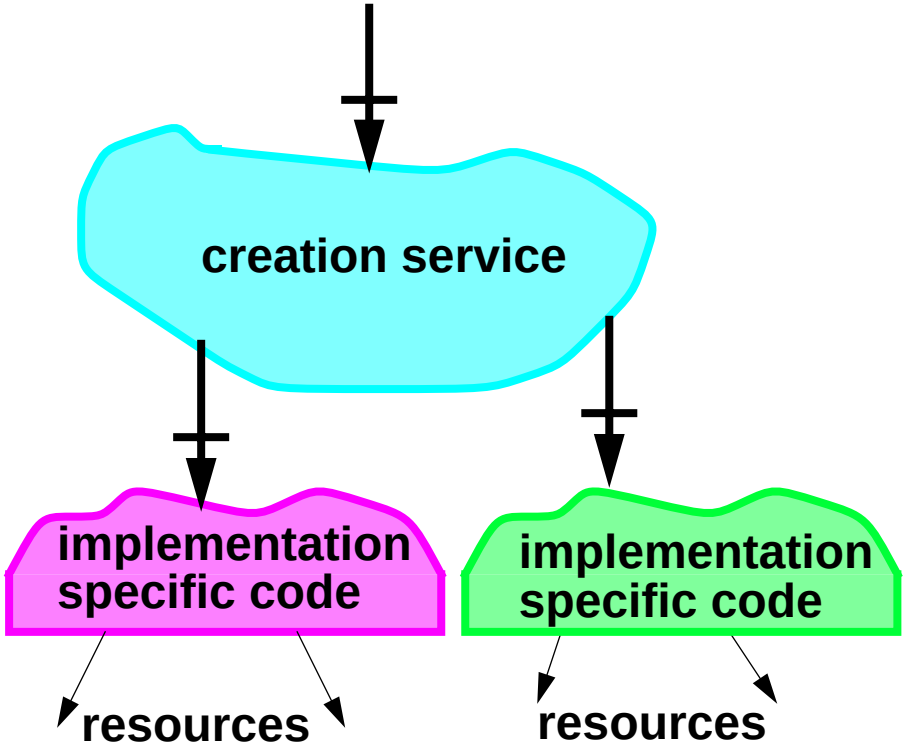
- Some services contain their own factories
  - for example, the Persistence, Externalization, and Property service do
- To create an object, an application needs the object reference of an appropriate factory
- Factory object references can be found by the usual means...
  - Naming and Trading services
- ... or be given to clients in any other way

Generic factories

- Although factories are not special, a standard interface for factories is impractical
  - there are too many possible kinds of resources
- Factories are certain to be implementation-specific
- Instead, a generic factory interface is defined
  - which invokes the implementation-specific ones
  - ... passing ‘criteria’ (name-value pairs) through as parameters
- The criteria in the LifeCycle specification are “suggestions”

Generic and implementation-specific factory interfaces

- Creations ultimately invoke implementation-specific factories



Deleting objects

- To delete an object, the client just calls the object's `remove` operation
  - this also invalidates the object reference
- A deleted object no longer exists...
  - when it's gone it's really gone
- ...if you want to *archive* an object, use the Common Facility instead
- The client is not responsible for reclaiming any resources used by the object

The LifeCycle service specification

- One module (`CosLifeCycle`), containing
  - interface `FactoryFinder`
  - interface `LifeCycleObject`
  - interface `GenericFactory`

Extensions to the LifeCycle service

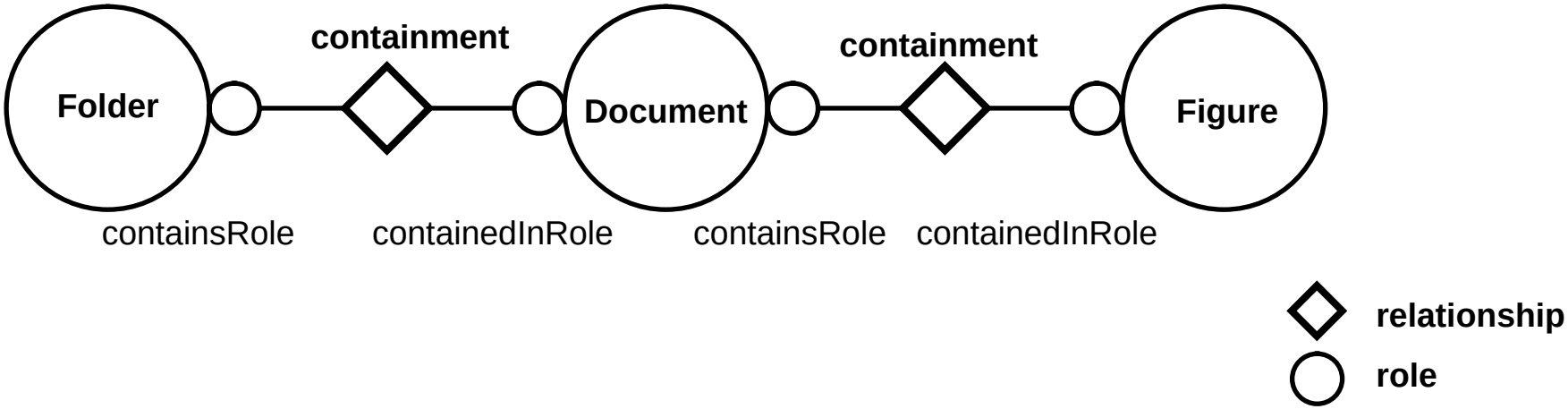
- The document includes three Appendices covering
  - A: groups of related objects
  - B: filters to specify resource constraints when objects are created
  - C: administration
- The document goes to some pains to stress that Appendices B and C are not part of the LifeCycle specification
  - they may eventually be aligned with the Properties and Trader services

The Relationship service

- **The Relationship service is a separate service**
  - **allowing the roles of related objects to be represented...**
  - **...for example a document *containing* figures...**
  - **... a figure being *contained in* a document**
- **Appendix A of the Lifecycle specification describes how the LifeCycle service handles relationships**
  - **it handles *containment* and *reference* relationships specially**
- **This allows a client to use the LifeCycle service to copy (or move) a document and all the figures contained in it, for example**

Relationships

- **An example of containment**
  - **folders containing documents, documents containing figures**



- **2 relationships, 4 roles, 9 objects involved**

When relationships are necessary

- **Object references**
  - can be stored persistently, and retrieved later, for invocation
- **Relationships**
  - can be followed in both directions
  - can relate two or more objects
  - can have their own attributes and operations
  - can be manipulated by third parties
- **Relationships are a natural fit to most analysis and design methods**

Relationships in CORBA

- **Every CORBA relationship is itself an *object***
  - **relationships are ‘first-class’**
- **Do not confuse relationships with invocations (or Naming contexts)**
  - **just because object A invokes an operation of object B, this does not imply a *relationship* between A and B...**
  - **...equally, if A is in naming context B, this does not imply a *relationship* between A and B**
- **Roles are objects too**
- **Roles and relationships can be created and destroyed separately from the objects they relate**
  - **subject to any referential integrity and consistency constraints**

Creating relationships and roles

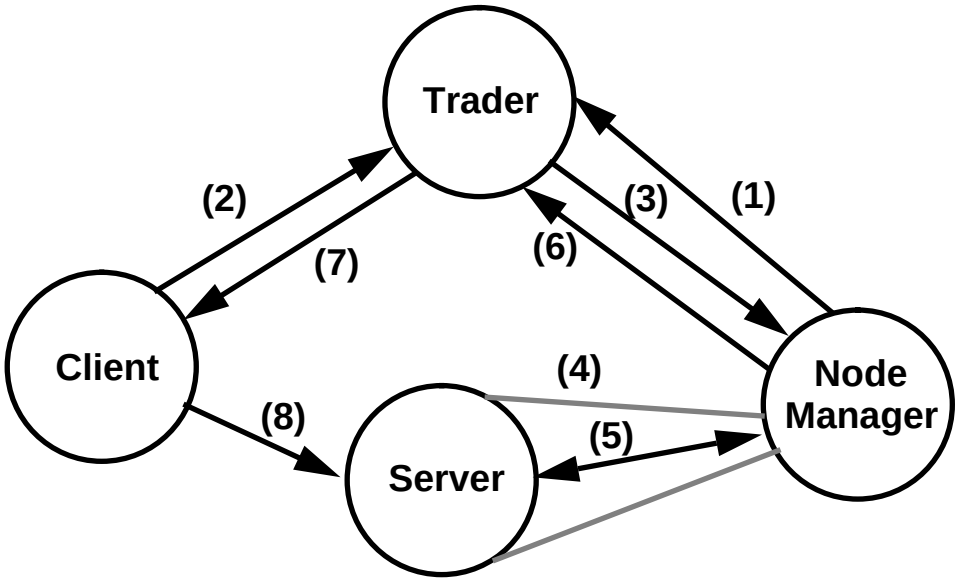
- Relationships are created via a factory interface

```
interface RelationshipFactory {
   struct NamedRoleType {
      RoleName name;
      ::CORBA::InterfaceDef named_role_type
   };
   typedef sequence<NamedRoleType> NamedRoleTypes;
   readonly attribute::CORBA::InterfaceDef relationship_type;
   readonly attribute unsigned short degree;
   readonly attribute NamedRoleTypes named_role_types;

   ...
   Relationship create (in NamedRoles named_roles) raises ...;
};
```

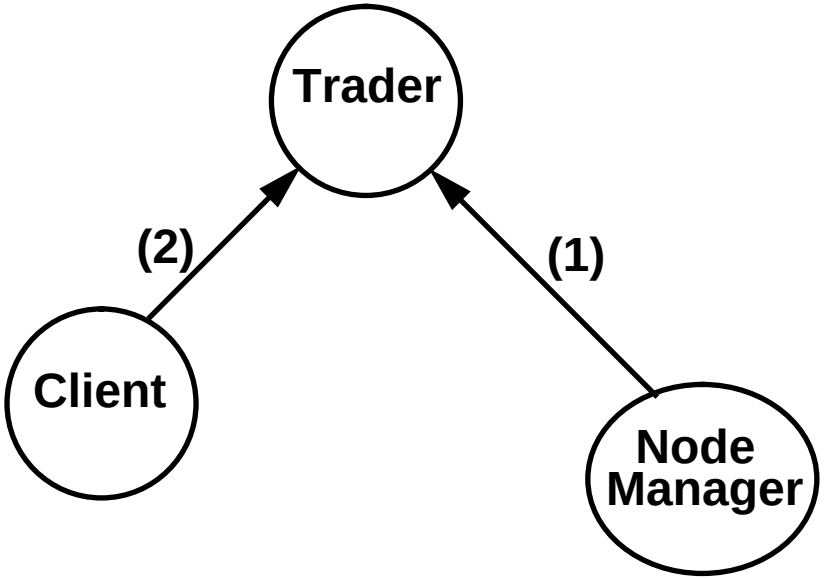
- ... and roles are also created via a factory interface (not shown here)

DAIS Node Manager



- **Provides a service for the dynamic creation and activation of other services**
  - **using an enhanced Trader service**

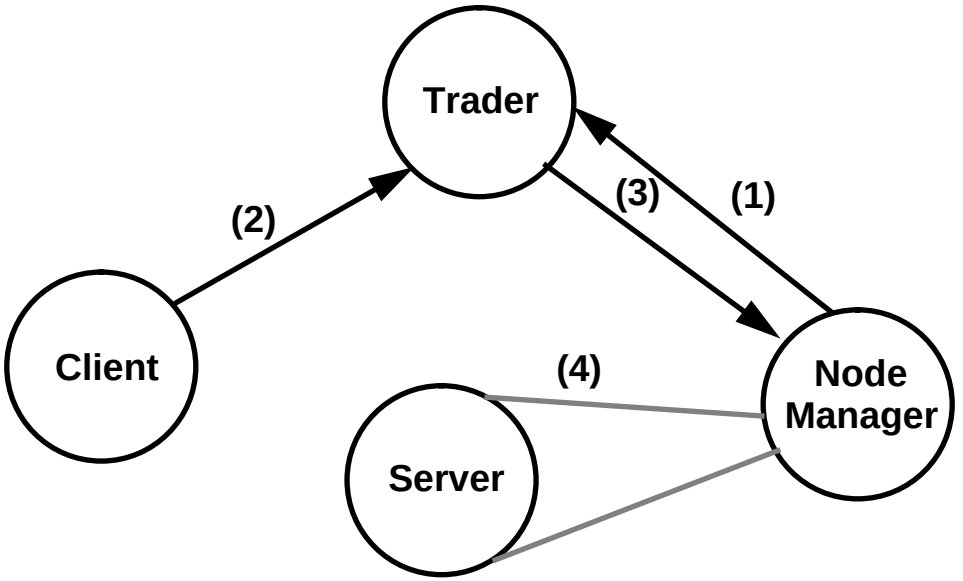
Node Manager - Registering the proxy offer



1. Node manager registers a proxy offer with the trader

2. Client performs an import

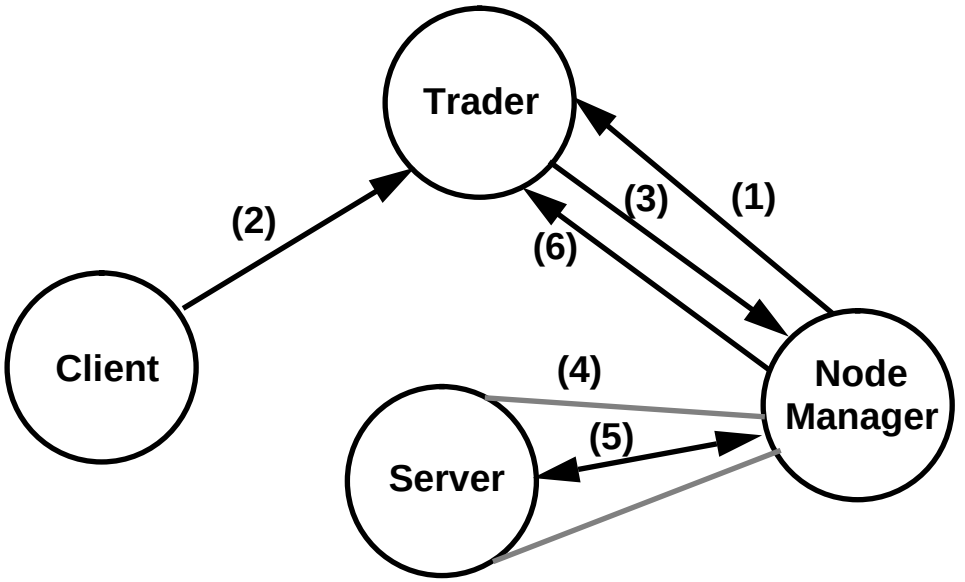
Node Manager - Creating the server capsule



3. Trader, recognising that the offer is federated, forwards the import to the node manager

4. Node manager creates the server capsule (using the factory)

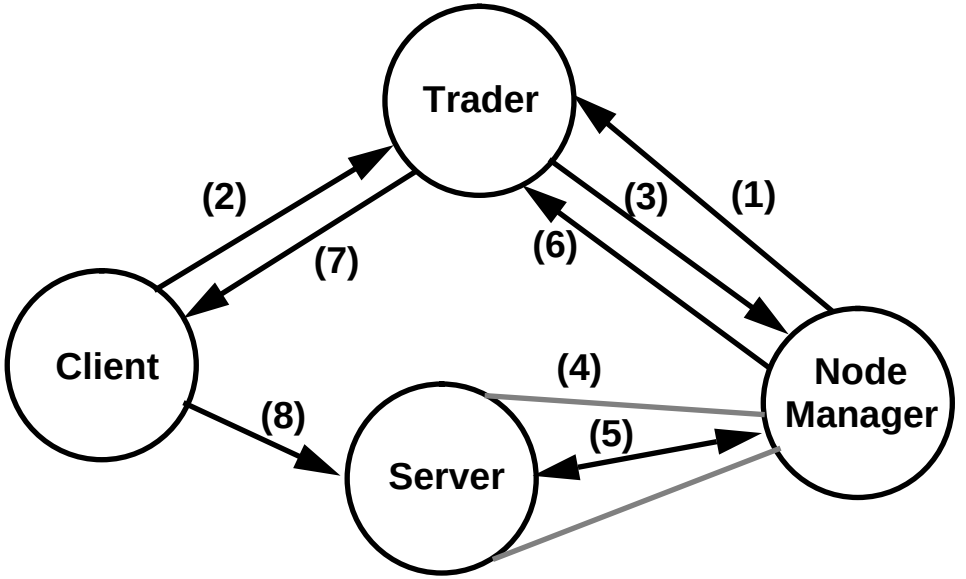
Node Manager - Instantiating the capsule



5. Node manager invokes the Instantiate operation on the newly created capsule's Capsule interface

6. Node manager returns the interface (result of Instantiate operation) to the trader

Node Manager - Return of interface

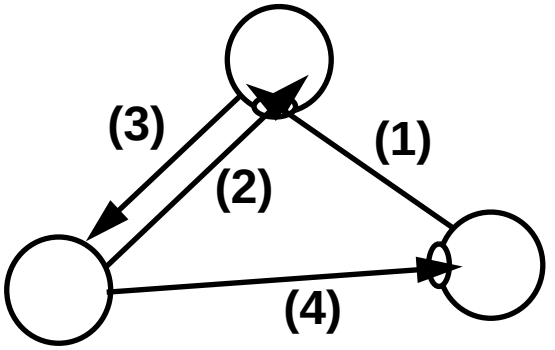


7. **Trader returns this interface to the original client**

8. **Client can now invoke operations on the server**

Trader, Factory, Node Manager

- Each of these services has interfaces that can be used independently
- These services also cooperate to provide dynamic service creation
  - transparently to the client application...
  - ... it only saw the ordinary trader interactions



Using the LifeCycle service

- The LifeCycle service shows how to use service-specific factories
- Generic factories, and `move` and `copy` operations will require an implementation of the LifeCycle service itself
- You may also find the specification helpful
  - when designing your own services
  - when designing your own factories
- You may need also to use vendor-specific interfaces
  - for object creation and deletion

Using the Relationship service

- Use the Relationship service to model real-world relationships between objects

- Use the Transaction service to enforce referential integrity

  - otherwise, a failure during relationship construction could leave one end of a relationship dangling

Summary

- The LifeCycle service allows objects to create and destroy other objects
  - via (inherited) factory interfaces
- The Relationship service allows real-world entities and relationships to be modelled
- Other services can be coupled with these services
- For more information
  - on the LifeCycle and Relationship services, see *CORBAservices* (OMG)
  - on using roles and relationships in information modelling, see *Information Modelling: Practical Guidance*, by Richard Veryard (Prentice Hall)

Other Specified Object Services

- Naming - “white pages” lookup
- Events - asynchronous communications
- Persistence - object storage when an object is inactive
- Externalization - object storage on (removable) media
- Concurrency - control of concurrent operations
- Transactions - serializable operations

Other Specified Object Services (not yet published in the CORBAservices manual)

- Security - authentication and authorization
- Time - synchronized clocks
- Licensing - support for controlling and charging for service usage
- Properties - associating named data with an object
- Query - query language used to select objects from collections

Object Services Still Being Specified

- Collections - collective operations on objects
- Trader - “yellow pages” lookup
- Startup - activating objects when the ORB is activated