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

EPFL Course September 1995: CORBA Object Services

Mark Madsen

Abstract

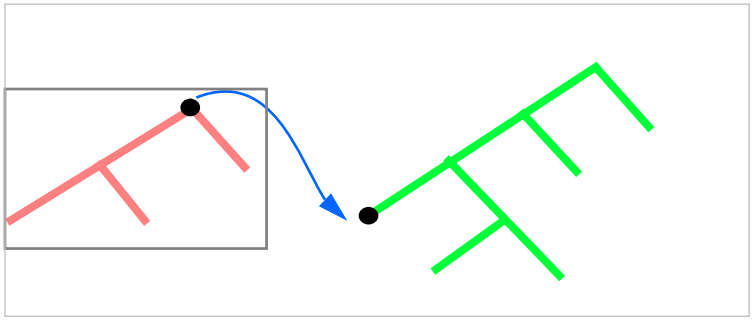
This presentation is based on APM.1349.03 “ANSAwise - CORBA Object Services”. (The Appendix is based on APM.1545.01 “ANSAwise - Persistent Data Storage with CORBA”).

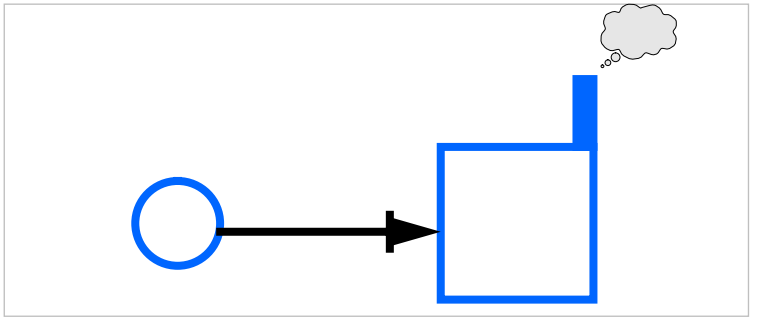
It was adapted for presentation as part of Module M3 “Distributed Systems” of the course on Communication Networks given at Ecole Polytechnique Federale de Lausanne in September 1995.

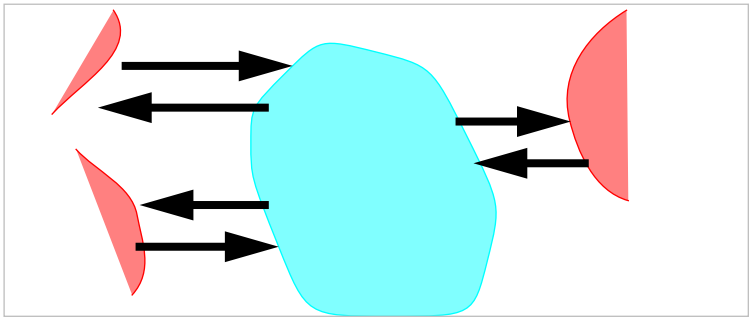
APM.1567.01 **Approved** 8th September 1995
Briefing Note

Distribution:
Supersedes:
Superseded by:

CORBA Object Services



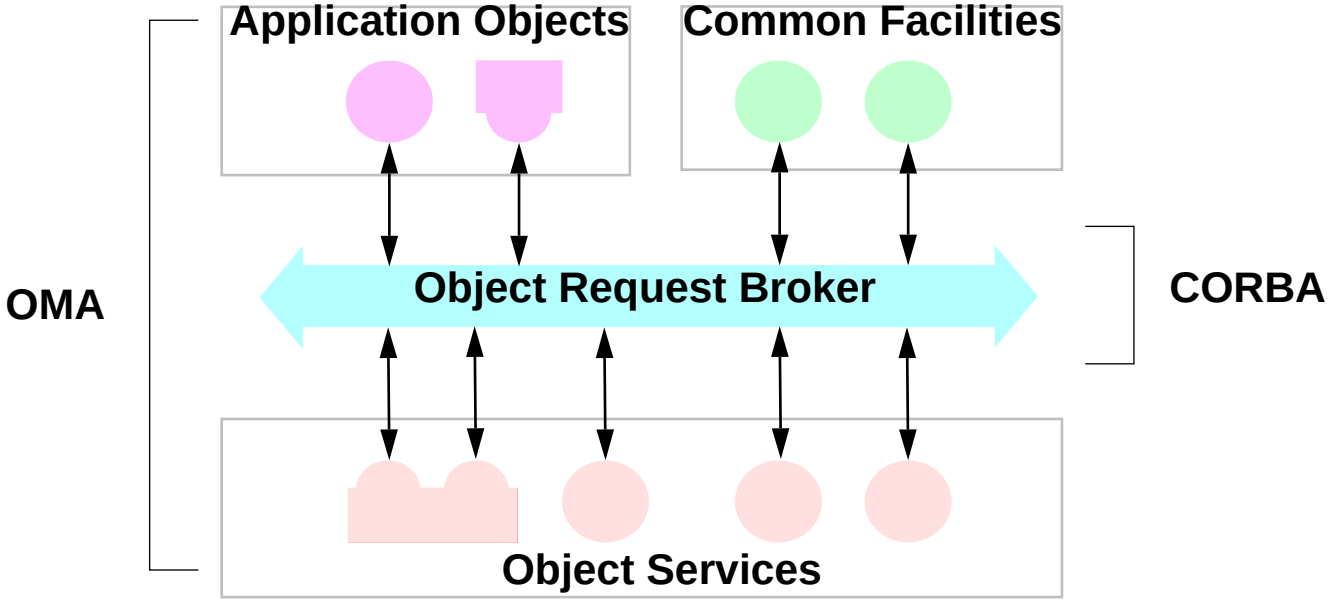




In this session

- ***Describe the general design principles for Object Services***
- ***Explain the first Common Object Services to be specified***
  - **Naming, Events, LifeCycle**
- ***Show some parts of these specifications in detail***
- ***Outline the future Object Services that will be provided***

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**

Language Independence

- ***Because the services are specified in CORBA IDL, you can call them from any programming language***

  - **provided there is a language mapping...**

  - **... C, C++, Smalltalk, Ada so far**

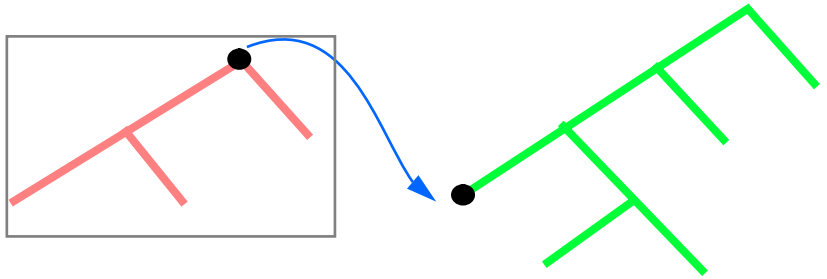
Scalable Service Implementations

- ***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**

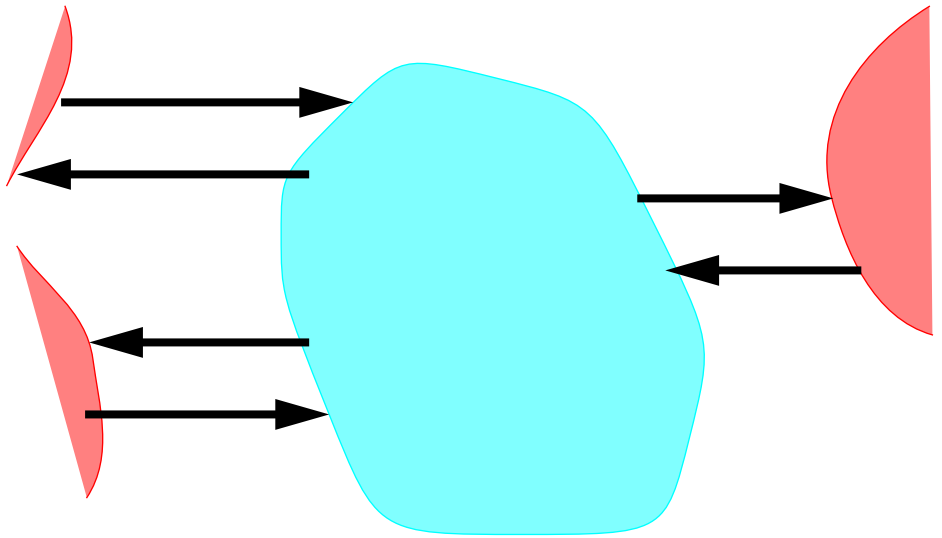
The Naming service specification

- ***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***

The Event service



- ***Events are concerned with asynchronous communication between objects***

Events

- ***Events support asynchronous notification***
  - **...'alerts', 'change notification'**
  - **for example, when disk space is getting low**
- ***Suppliers and consumers of events are decoupled***
  - **via an event channel**
- ***There can be multiple suppliers and multiple consumers***
- ***Events could be used to build an RQM (Robust Queued Messaging) interface***
  - **or to interface to an existing one**

Conspiracies - a ‘new’ object concept

- ***Recall that in CORBA, an object can only have one interface***

- ***But a service can have multiple interfaces***

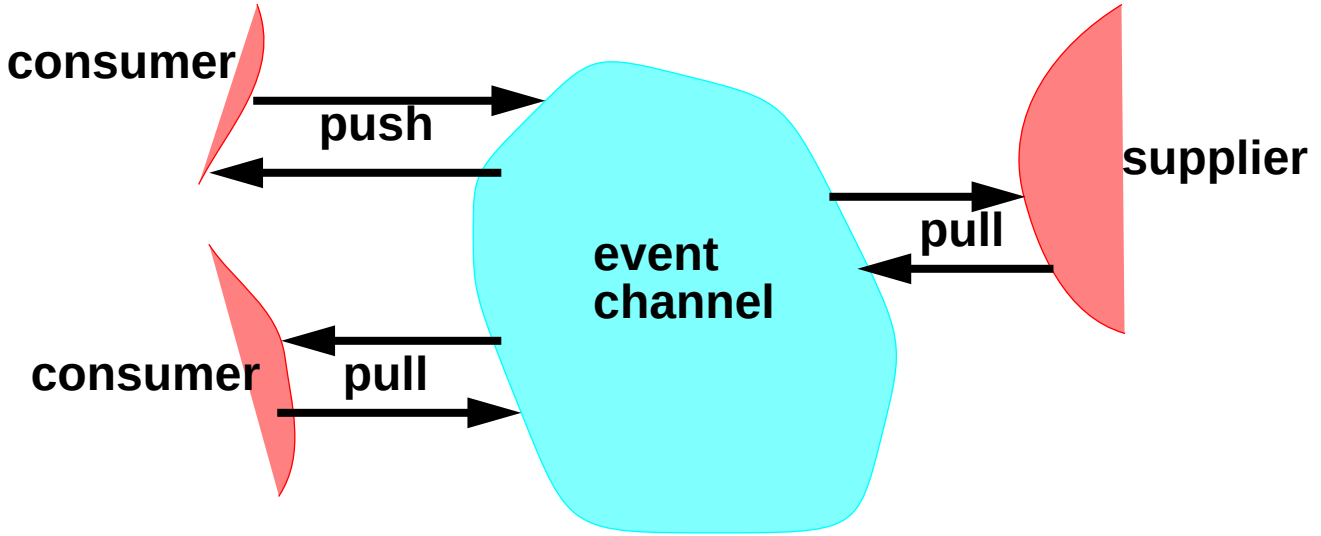
  - **provided by a set of ‘conspiring’ objects**

Push and pull

- ***Two models of communicating event data***
  - **push model: suppliers push data to consumers**
  - **pull model: consumers pull data from suppliers**

Push and pull with event channels

- ***Suppliers and consumers can use the same or opposite models...***



- ***... the models are decoupled by the event channel***

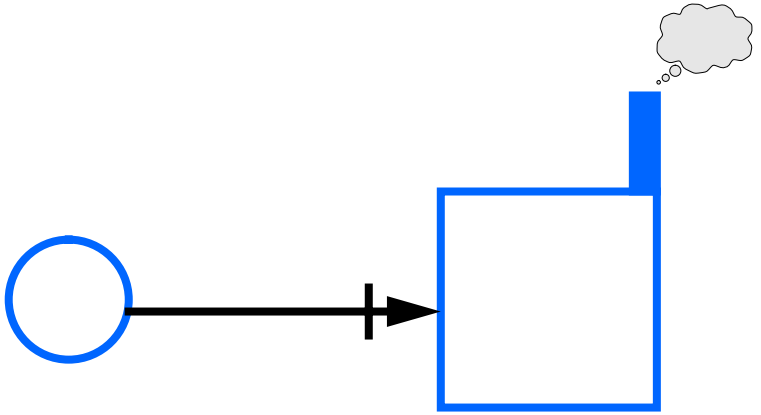
Generic and typed channels

- ***Generic event channels allow communication of arbitrary (untyped) data***
- ***Typed event channels allow communicate via an IDL interface***
  - **any IDL interface by mutual agreement between consumer and supplier**
  - **... operations in that interface are transformed into 'pull'/'try' operations, with the same parameters**
- ***Typed event channels also support generic events***
- ***Push and pull models are supported for both generic and typed event channels***

The Event specification

- ***Module CosEventComm***
- ***Module CosEventChannelAdmin***
- ***Module CosTypedEventChannel***
- ***Module CosTypedEventChannelAdmin***

The LifeCycle service

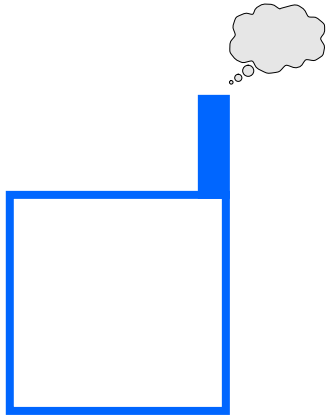


- ***Life cycle services allow applications to control***
  - **creation of objects**
  - **removal of objects**
  - **copying of objects**
  - **moving of objects**
- ***The LifeCycle service depends on the Naming service***

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



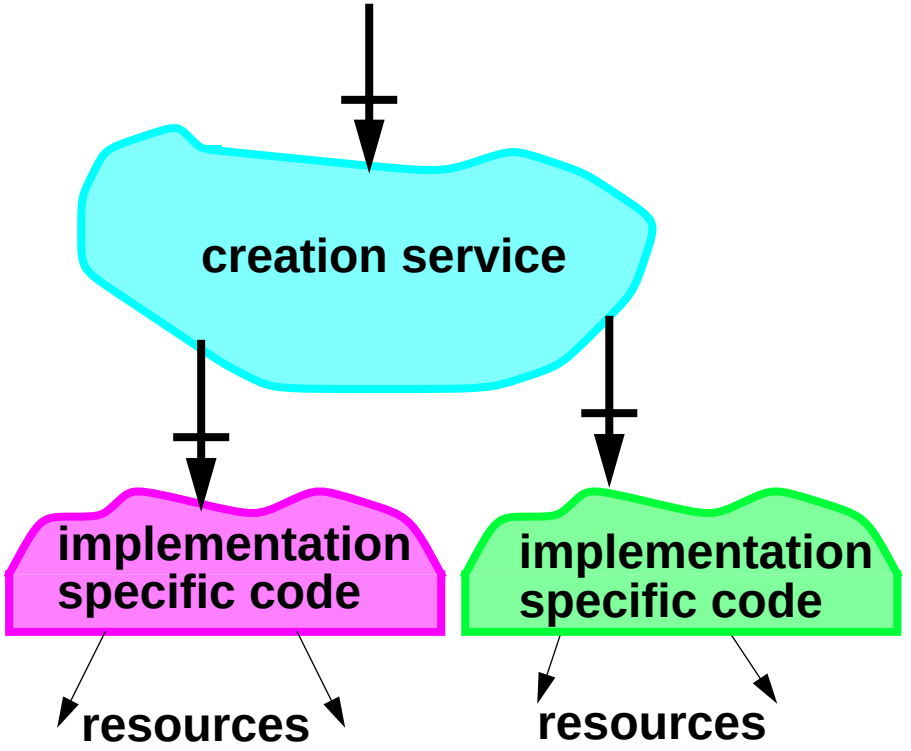
- ***A factory object is an object that can create other objects***
- ***Factories are not special***
  - **they are specified in IDL**
- ***Factories are responsible for allocating resources to objects that they create***
  - **according to client control/administrative policy**

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***



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***
  - **groups of related objects**
  - **filters to specify resource constraints when objects are created**
  - **administration**
- ***The document goes to some pains to stress that these Appendices are not part of LifeCycle specification***
  - **even though the level of detail is roughly the same**

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**

Using the Events service

- ***Use the Events service if you need***
  - **queued messaging**
  - **notifications**

Using the LifeCycle service

- ***In its current form, the LifeCycle service is not detailed enough to be directly usable***
  - **the Appendices are not part of the specification**
- ***However, you may find the specification helpful***
  - **when designing your own services**
  - **when designing your own factories**
- ***For the moment, you may need to use vendor-specific interfaces***
  - **for object creation and deletion**

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**

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**

Understanding the OMG specification process

- ***Requests for Proposals (RFPs) are issued by an OMG Task Force***
  - **the Object Services Task Force tends to issue them in bundles**
- ***Potential submitters write Letters of Intent (LOIs)***
- ***Submitters supply initial submissions***
- ***Submitters supply revised submissions***
- ***The Task Force votes***
- ***The OMG Technical Committee votes***
- ***The OMG Board votes***

OMG RFPs and specifications

- ***So far, 5 RFPs have been issued by the Object Services Task Force***
  - **known as COSS (Common Object Services Specification) 1-5**
- ***The timetable doesn't always match the order in which RFPs were issued***

Object Services RFP 3

- ***Security - authentication and authorization***
  - **RFP strongly linked to security White Paper**
- ***Time - synchronized clocks***
  - **security implementation depends on (trusted) Time service**
  - **...to thwart replay attacks**

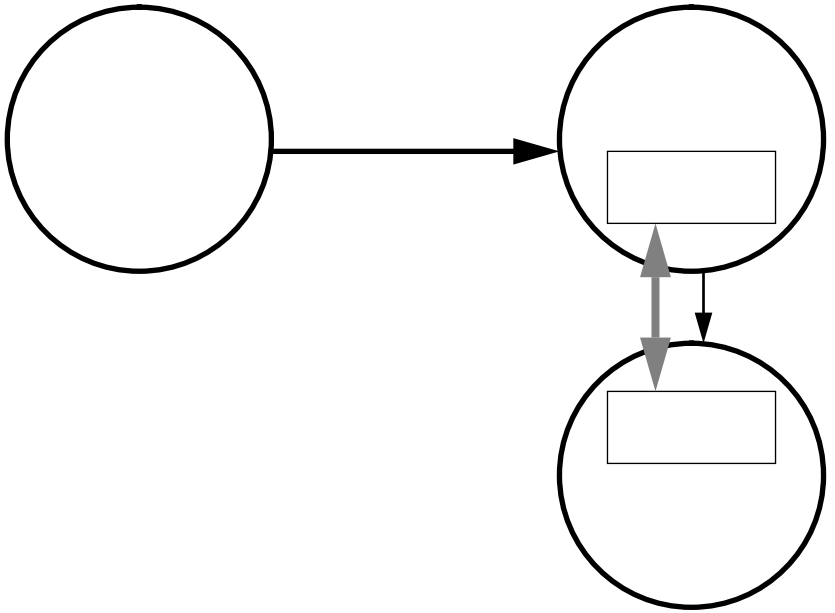
Object Services RFP 4

- ***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 RFP 5

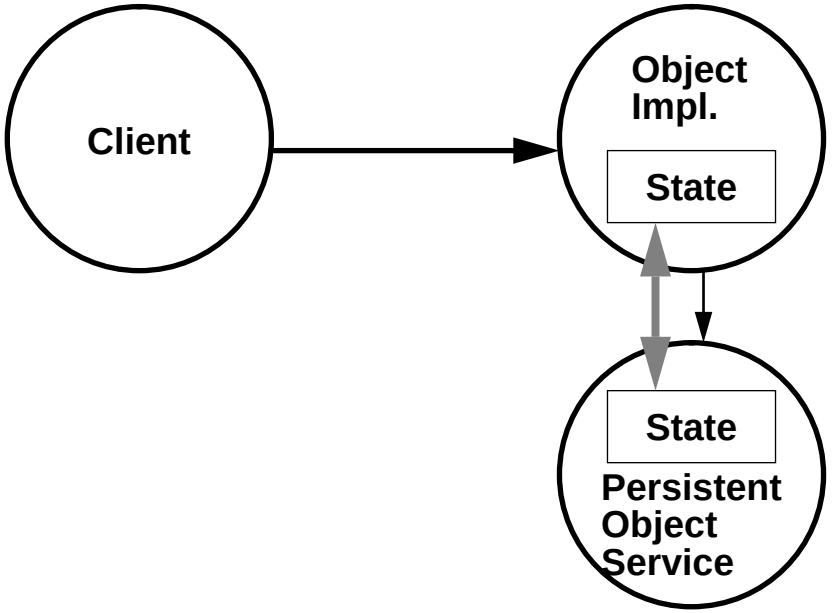
- ***Collections - collective operations on objects***
- ***Trader - "yellow pages" lookup***
  - **Trading is mentioned in several places in CORBAservices...**
  - **...particularly when federation is discussed**
  - **...particularly when life cycle issues are discussed**
- ***Startup - activating objects when the ORB is activated***

APPENDIX:
Persistent Data Storage with CORBA



Roles in the Persistent Object Service

- ***Persistent Object Service stores persistent state***



Persistent state

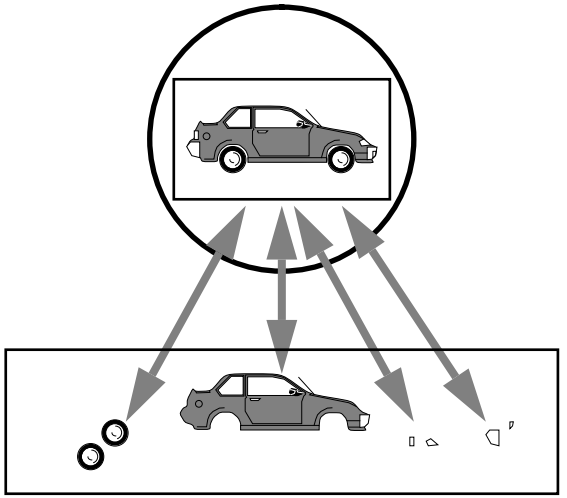
- *The state of an object can be treated as two parts*
- *Dynamic state*
  - typically in memory
  - lost if the object crashes
- *Persistent state*
  - typically on disk
  - preserved over crashes, and can be used to reconstruct the dynamic state

Preserving state

- ***It is an object's responsibility to preserve its dynamic state as persistent state***
  - **and to recover it after a crash**
- ***The object may use the Persistent Object service for this purpose***
  - **or any private mechanism it chooses instead...**
  - **... flat files**
  - **... direct access to a relational database**
  - **... direct access to some other kind of database**
- ***The object may delegate the responsibility back to its client***

Advantages of using the Persistent Object Service

- *Typical data storage mechanisms do not have object characteristics*
  - uniform interfaces, self-description, and abstraction



- *This is sometimes known as an 'impedance mismatch'*

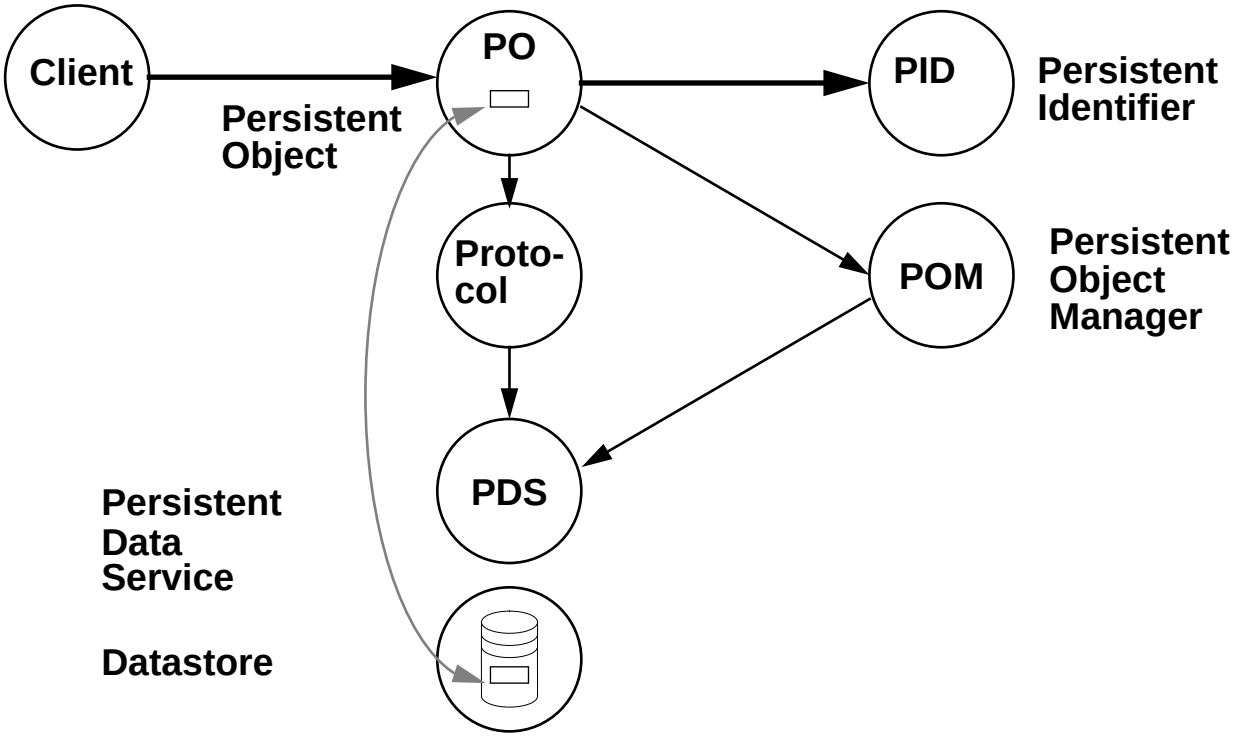
Client Control

- *Clients may need to control or manage persistence of the objects they use, in particular*
- *In particular, they client may need to control*
  - exactly when persistent state is saved and restored
  - which copy of persistent state is to be used
- *Object implementations do not provide client control*
  - they may choose to hide the complexity from the client

Persistent Object Service Implementations

- ***As with all CORBA Object Services, a wide range of implementations is envisaged***
  - **on top of relational databases**
  - **on top of object-oriented databases**
  - **on top of specialized compound document storage mechanisms**
  - **on top of lightweight storage mechanisms**
  - **... or any combination of these**

Major Components of the Persistent Object Service



Persistence Components - 1

- ***Persistence Identifier (PID)***
  - **describes the location of an object's data in some Datastore**
- ***Persistent Object (PO)***
  - **an object whose persistence is controlled externally (by some client)**
- ***Persistent Object Manager (POM)***
  - **supports both internally and externally controlled persistence**
- ***Persistent Data Service***
  - **coordinates basic persistence operations between Datastore and Protocol**

Persistence Components - 2

- ***Protocol***
  - encapsulates a general kind of data storage
- ***Datastore***
  - basic access to the underlying data storage mechanisms

Persistent Identifier (PID)

```
module CosPersistencePID {

   interface PID {
      attribute string datastore_type;
      string get_PIDstring();
   };

};
```

- *An object requires a PID to store data persistently*
- *Do not confuse a PID with an object reference*
  - object reference identifies an object
  - persistent identifier identifies an object's state

Specialized PIDs

- *There also may be specialized PIDs*
  - with additional operations and attributes...
  - ... with specifications inherited from PID
- *For example*

```
interface PID_DB : CosPersistencePID::PID {
   attribute string database_name;
};

interface PID_SQLDB : PID {
   attribute string sql_statement;
};
```


PID Factory example

```
interface PIDFactory {
   CosPersistencePID::PID create_PID_from_key(in string key);
   CosPersistencePID::PID create_PID_from_string (
      in string pid_string);
   CosPersistencePID::PID create_PID_from_string_and_key (
      in string pid_string, in string key);
};
```

- *To create a PID, you need to identify a PID implementation to use*
  - identified by key, string, or both
- *Supplying both may be useful when moving persistent data between Datastores with different interfaces*
- *This specification is only an example; others are also possible*

Persistent Object

- ***Has three interfaces***
  - **PO: for allowing a client to control persistence externally**
  - **POFactory: for creating POs**
  - **SD: for the persistent object to maintain internal consistency**

interface PO

```
interface PO {
   attribute COSPersistencePID:: PID p;
   COSPersistencePDS::PDS connect (
      in CosPersistencePID::PID p);
   void disconnect (in CosPersistencePID::PID p);
   void store (in CosPersistencePID::PID p);
   void restore (in CosPersistencePID::PID p);
   void delete (in CosPersistencePID::PID p);
};
```

- ***Allows a client to connect PO with a Datastore***
  - **when disconnected, the data in the PO and the Datastore are the same**
- ***Allows a client to move data between PO and Datastore in either direction***
  - **store/restore**

Synchronized Data

```
interface SD {
   void pre_store();
   void post_restore();
};
```

- ***Some objects have persistent and transient data***
  - **for example transient data in a cache, or redundant data structures**
- ***Operations in the Synchronized Data interface are only invoked by the POM***

Persistent Object Manager

- *Interface is very similar to the PO interface*
- *Externally-controlled POs invoke the POM when they are themselves invoked*
- *Internally-controlled POs invoke the POM according to their internal policy*

Persistent Data Service

- ***Interacts with the object to get data in and out of the object***
  - **using a Protocol**
- ***Interacts with the Datastore to get data in and out of the object***
- ***A PDS may support several Protocols***
- ***A PDS may use either a standard or a proprietary interface to its Datastore***

Protocols

- *The Persistent Object Service specifies three protocols*
  - Direct Access (PDS_DA) Protocol: uses attributes to store the data - the DDL is a subset of CORBA IDL, for example

```
interface MyDataObject {
   attribute short my_short;
   attribute float my_float;
};
```

  - ODMG-93 Protocol: similar to PDS_DA, but uses ODMG ODL (Object Definition Language)
  - Dynamic Data Object (DDO) Protocol: a Datastore-neutral protocol
- *Other protocols could be integrated*

Datastores

- *The Persistent Object Service specifies one Datastore*
  - Datastore_CLI: for record-oriented databases, based on the X/Open Data Management Call Level Interface
- *Other Datastores are not specified because*
  - other standard interfaces already exist
  - Protocols can drive their underlying database directly if they wish

Externalization

- *Externalization is a separate Object Service*
- *Externalization records an object's state in a stream of data*
  - in memory, on disk, across the network
- *The stream owns the externalized form*
- *The externalized form can be stored indefinitely and transported outside an ORB*
- *The stream can be later internalized*
  - into a new object...
  - ...in a different ORB, not necessarily connected to the previous ORB

Externalization Portability

- *There are many possible storage media and data formats*
- *To ensure data portability, the Externalization Service defines a standard external representation*
  - and an interface to externalize to a file

Externalization - the client's view

- *The client must first acquire an object reference for a stream*
  - possibly creating it via a StreamFactory
- *The client then invokes externalization for an object on a stream*
  - this may cause other objects to be externalized, but the client is unaware of this
- *A client later invokes internalization for an object from the same stream*
  - supplying a factory finder...
  - ... which will create a new object with the data from the stream

Externalization - the object's view

- ***Every object that wishes to be externalizable must support the Streamable interface***

- ***When the object receives an externalize_to_stream request it must write its state to the stream***

  - **using write_<type> operations for each of the CORBA basic types**

Externalization - the stream's view

- *The stream co-operates with the externalizable object*
  - using the object's Streamable interfaces
- *The stream supplies a StreamIO object to the externalizable object*
  - StreamIO actually writes the data to the externalized form

Externalizing Multiple Objects

- ***Multiple objects can be externalized to the same stream***
  - **by bracketing within a begin/end 'context' pair**
- ***This requires the objects to coordinate their externalization***
  - **for compound externalization of graphs of related objects**
- ***There is also specific support for externalizing relationships and roles***

Persistence, Externalization, and Lifecycle

- ***Saving and restoring data with the Persistent Object Service affects a single object***
- ***Externalizing and internalizing with the Externalization Service creates a new object***
- ***Copying an object with the LifeCycle Service creates a new object***

Summary

- ***The Persistent Object Service provides considerable flexibility***
  - **and supports both external (client) and internal control**
- ***The Externalization Service provides storage outside the ORB***
  - **and supports data portability between ORBs**
- ***For more information***
  - **about these services, see *CORBAservices* by the Object Management Group (Wiley)**
  - **about ODMG, see *The Object Database Standard: ODMG-93* by R. Cattell (Morgan Kaufmann)**