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 - Engineering Distributed Systems

Chris Mayers

Abstract

Organizations wishing to understand how to design and build distributed systems need to appreciate engineering concepts to enable them to make trade-offs in these systems.

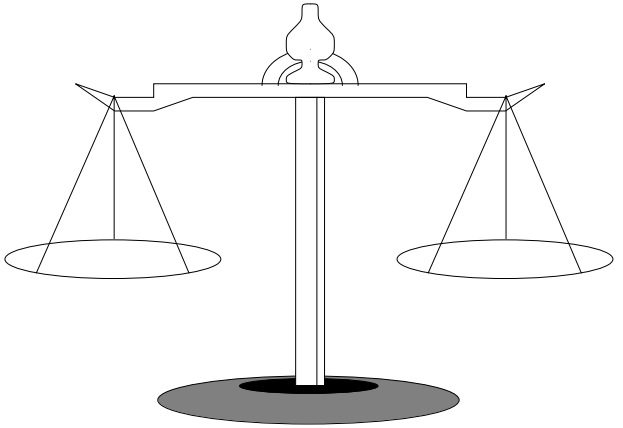
This module of the ANSAwise training programme describes techniques and mechanisms for systems designers to make these trade-offs. Transparency mechanisms are important, but many of these are not available in products, so designers will need extra techniques and mechanisms.

[This module is derived from APM.1688, but does not discuss the ANSA/ODP Engineering model directly. This module is loosely based on the paper *Hints for Computer System Design* by Butler Lampson.]

APM.1750.01 **Approved** 4th April 1996
Briefing Note

Distribution:
Supersedes:
Superseded by:

Engineering Distributed Systems



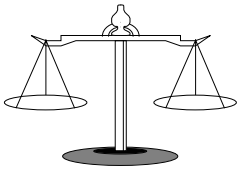
In this session

- **Identify desirable characteristics of distributed systems**
- **Describe principles that govern distributed systems**
- **Describe general techniques for engineering distributed systems**
- **Describe some mechanisms to apply these techniques and principles**

Characteristics the users want

- **Performance**
- **Predictability**
- **Reliability**

Engineering is about trade-offs



- **Trade-offs include...**
  - **flexibility versus performance**
  - **time versus space**
  - **... and many others**
- **Trade-offs must not affect interface specifications**
- **Trade-offs should be reversible**

Making trade-offs

- **Trade-offs can be influenced by**
  - **standards**
  - **product selection**
  - **configuration planning**
  - **design choices**
- **Trade-offs may require access to the distributed systems infrastructure**
  - **which may affect application portability**
- **Some trade-offs can be chosen entirely within applications**

An application trade-off - object placement

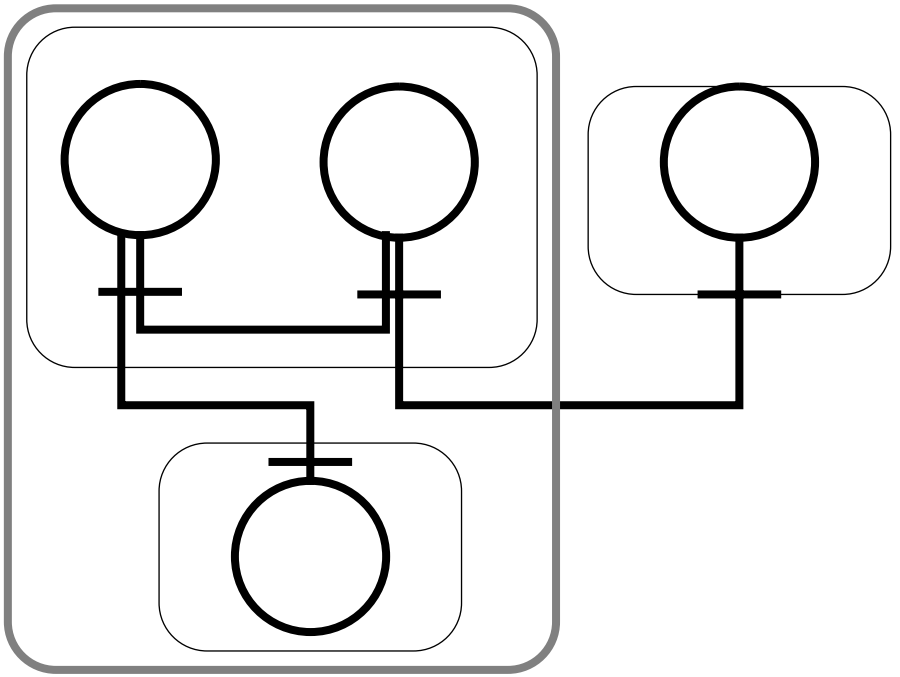
- **Place objects in the same object implementation (process)**
  - **for efficiency of communications**
  - **for efficiency by exploiting shared state**
- **Place objects in different object implementations**
  - **for robustness**
  - **for security**
  - **for flexibility of configuration**
  - **to avoid competing for same resources**

Object Implementations and Interfaces

- Objects in the same object implementation can still invoke each other's operations
  - you are not compelled to exploit shared state
- Operations are invoked in the same way...
  - within a object implementation
  - between two object implementations on the same node
  - between two nodes

Communications Optimization

- **Infrastructure should optimize communications between objects on the same node**



Inherent properties of distributed systems

- Separation: physical and logical dispersal
- Diversity: many types of machines in the same system
- Legacy: evolution and interworking of existing systems
- Scalability: low cost of computing per machine
- Decentralization: no single point of control
- .... these differences from centralized systems are fundamental

Assumptions we must reverse

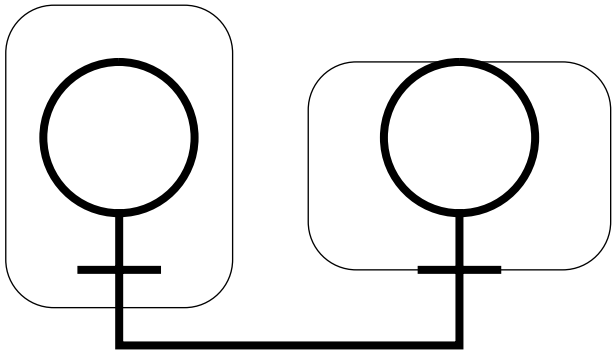
- **Because these differences are inherent, the best approach to distributed system design is to accept this**
  - **to *reverse* the traditional design assumptions...**
  - **...for example, to assume that objects *will* migrate**
- **Handling these properties explicitly would be complex for the application programmer**
- **The distributed systems infrastructure should mask these properties**
  - **using special *transparency* mechanisms**

Transparency Types

- ***Access***
- ***Failure***
- ***Location***
- ***Migration***
- ***Persistence***
- ***Relocation***
- ***Replication***
- ***Transaction***

Transparency Engineering - Access

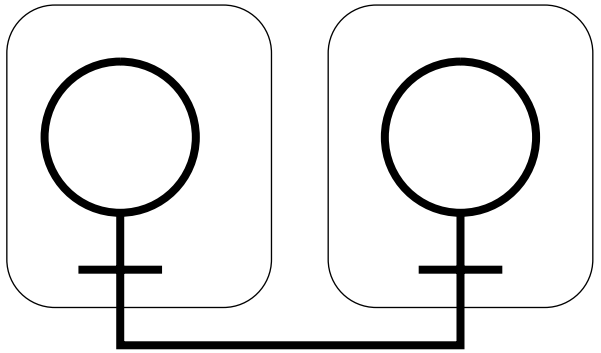
- **Access Transparency**
  - **application need not know the type of machine where the object is executing**



Transparency Engineering - Location

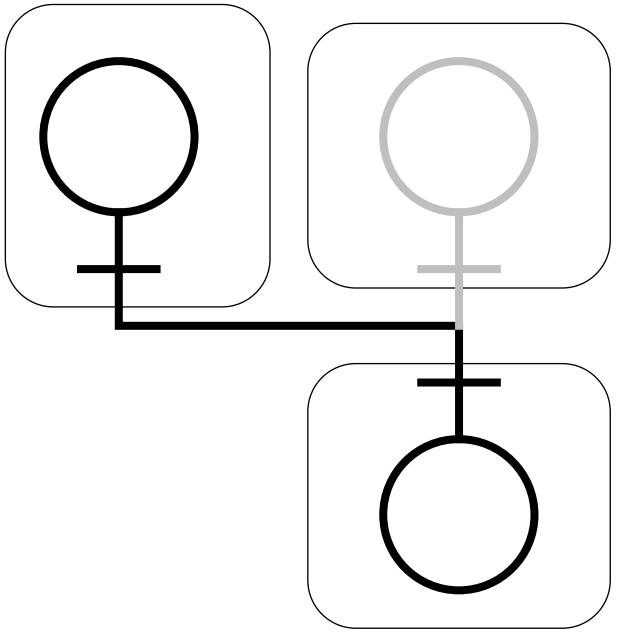
- **Location Transparency**

  - **application need not know where object is to use it**



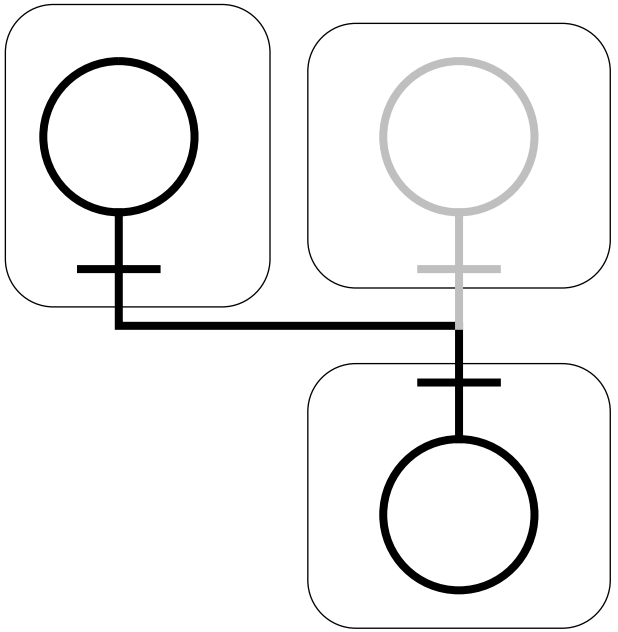
Transparency Engineering - Relocation

- **Relocation Transparency**
  - **application need not know where the object has moved to**



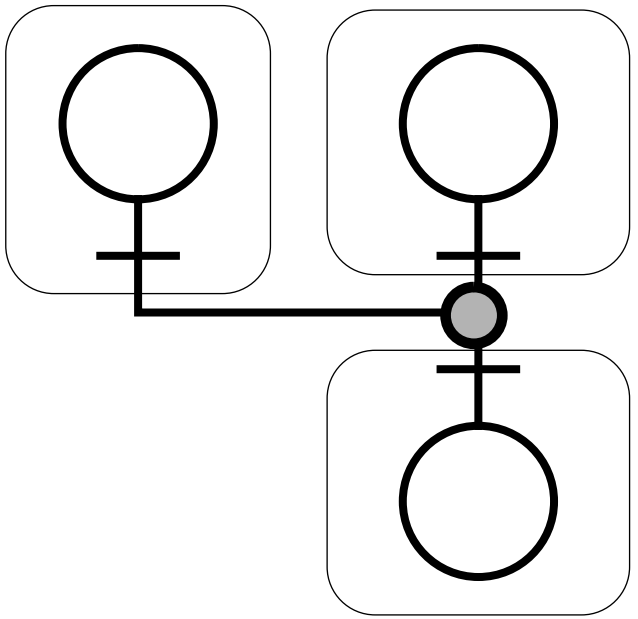
Transparency Engineering - Migration

- **Migration**
  - **server need not know that it has itself moved**



Transparency Engineering - Replication

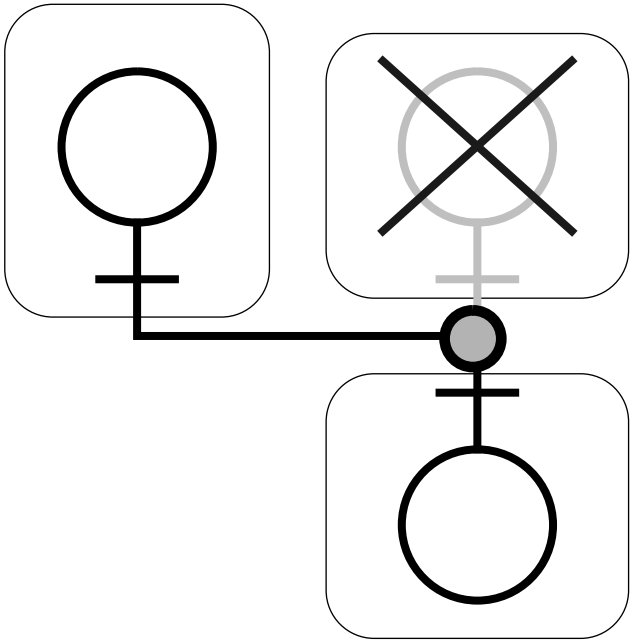
- **Replication Transparency**
  - **application need not know how many copies...**



  - **...application only sees a single interface**

Transparency Engineering - Failure

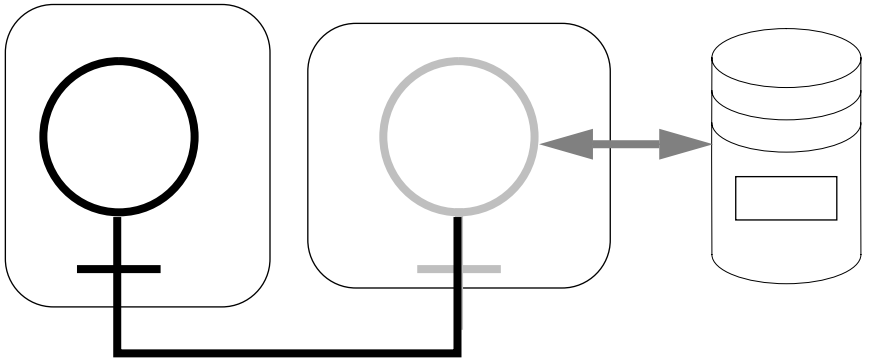
- **Failure Transparency**
  - application need not know when an object fails



  - may use replication transparency to achieve this

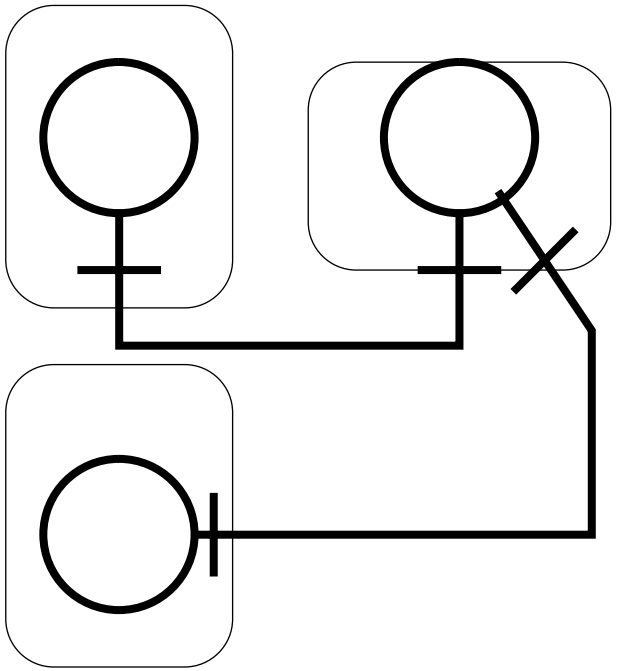
Transparency Engineering - Persistence

- **Persistence Transparency**
  - **application need not know of object activation/deactivation**



Transparency Engineering - Transaction

- **Transaction Transparency**
  - **applications need not be aware of inconsistent states**



Transparencies - the bad news

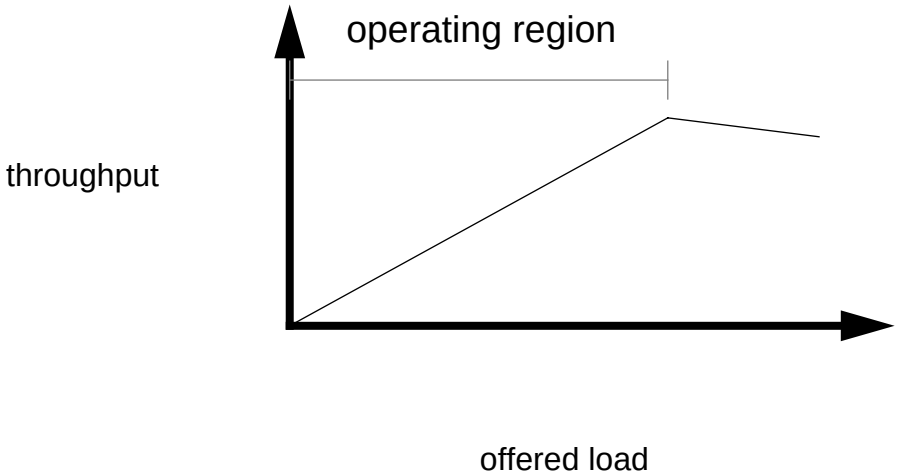
- **Only access and location transparency are supported by CORBA**
- **Other transparencies are only available as part of vendor-specific extensions**
- **Transparency mechanisms must be implemented as part of the ORB infrastructure**

Loaders in Orbix

- If an invocation arrives at the destination, but the object cannot be found, an *object fault* occurs
- Loaders allow applications to intercept the object fault and activate the object
  - reading its state from persistent storage
  - creating the corresponding data structures
- Loaders can be per class, per process, or per database
- Loaders can be an alternative to the CORBA Persistence service

Scalability

- **Ideal scalability would look like this**



Implications of scalability

- **Each request must use the same amount of resource**
  - **however many other simultaneous requests there are**
  - **however much stored data there is**
- **This implies that any data access methods must require *linear access time***
  - **such data access methods do not generally exist...**
  - **... unless the access characteristics of the data set are known in advance**
- **In practice, logarithmic rather than linear scalability may be an achievable target**

Scalability principles

- Do not allocate resources that are never used
- Allocate resources as late as possible
- Share resources as much as possible
- Release resources as early as possible
- Match the distribution of resources to the scale of the demand

Quality-of-service considerations may constrain them

General guidance for scalability

- **Separate the normal from the worst case**
  - **the normal case must be fast (enough)**
  - **the worst case must make some (enough) progress**
- **Do the processing on the client side**
- **Design out bottlenecks**
  - **avoid shared state between potentially simultaneous requests**
- **Design out hard numeric limits**

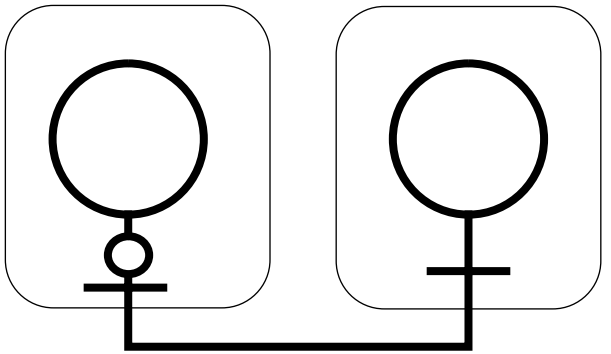
Caching

- Caching comes in two common guises
  - keeping a *local* copy of master information
  - keeping a *pre-computed* copy of master information
- Caches can anticipate
  - read-ahead, write-behind
- Caches can be persistent, but usually are not
- Persistent caches also support off-line working

Cache Implementation Challenges

- Caches must be kept consistent with the master information
  - this is difficult to achieve if the accesses can bypass the caching mechanism
- Caches must be tuned to avoid thrashing
  - this is usually best done adaptively
- Caching has security implications

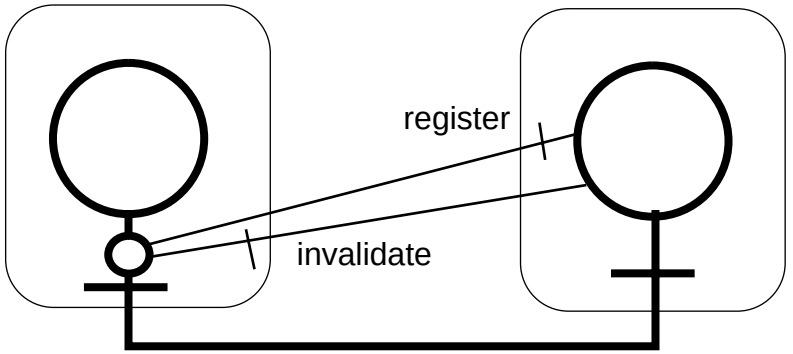
Smart proxies in Orbix



- **Smart proxies allow the client side of an interface to do some of the work of the server (object implementation)**
  - **effectively an extension of a stub**

Using smart proxies

- **Smart proxies can be used for**
  - **caching**
  - **access to local physical resources**
  - **load balancing among replicas**
- **Callbacks are used to update caches**



Hints

- A hint is a piece of information that...
  - ...improves performance if correct
  - ...has no negative consequences if incorrect
- To be worthwhile, hints must be almost always correct
- Hints must be self-validating
- Hints are often used for location and routing information in distributed systems
  - if location information is incorrect, this is readily detectable when used
- Hints are often cached

Locators in Orbix

- Locators are used at bind time when the destination of an invocation is not yet known
- Note that an inefficient locator can waste time
  - it can never cause the wrong object to be invoked
- Locators can be used when control is needed over selection of the destination
  - in some systems, static lookups may be best...
  - ... in others, random lookups may be best

Coping with inconsistency

- **Sometimes inconsistency is inevitable**
  - **consistency may be too expensive or impossible to enforce**
- **Aim for convergence instead**
  - **for example, by forwarding messages that have been incorrectly routed, but also telling the sender the new destination**
  - **thus converting a consistency issue into a performance issue**

Passing The Buck

- When there's work to be done...
  - do it in the background (when the machine is idling)
  - do it somewhere else (at the client)
  - do it later (write-behind)
  - do it in batch mode (off-line)
- If you must do it, use brute force
  - a straightforward implementation of a simple algorithm

Predictability

- **Control demand by shedding load**
  - **never thrash by operating in overload**
- **Force the (ultimate) originator of the request to wait**
- **Load control is harder in distributed systems**
  - **because of latency, and the remoteness of clients**

Reliability - the end-to-end principle

- **Error checking must always be done at the highest level of applications**
  - **because an error can always occur at the last moment, applications must always check**
- **Therefore, the only purpose of error checking at lower layers is to improve performance (by reducing the cost of retries)**
  - **it can never mask errors completely**

Summary

- **Engineering is concerned with trade-offs**
  - **and the efficient allocation of resources**
- **When making engineering decisions**
  - **strive to avoid bad choices, rather than pursue optimum choices**
  - **isolate the choices to particular system components**

More information

- **For more information**
  - see *Hints for Computer System Design*, by Butler Lampson (IEEE Software, January 1984)
  - see *End-To-End Arguments in System Design*, by J.H. Saltzer, D.P. Reed, and D.D. Clark (ACM Transactions on Computer Systems, November 1984)
  - for the ANSA principles for system design, see *An Overview of ANSA* (AR.0)

Related Topics

- **Optimization of distributed applications**
  - **essentially the same issue as for optimization of single programs...**
  - **... in practice requires more sophisticated tools and instrumentation**
- **Design of distributed algorithms**
  - **a very different issue from single-machine algorithms...**
  - **...requires highly specialist expertise**
- **Validation, verification, test, and debugging**

Filters in Orbix

- Filters allow application code to intervene at specific invocation points
  - before and after marshalling, on requests and replies, at client and server
- Filters are implemented as C++ objects
- Filters can also be attached to individual CORBA objects
- Filters can be used for
  - monitoring
  - debugging
  - audit trails
  - ...