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: Architecture for ODP, Part 1

Mark Madsen

Abstract

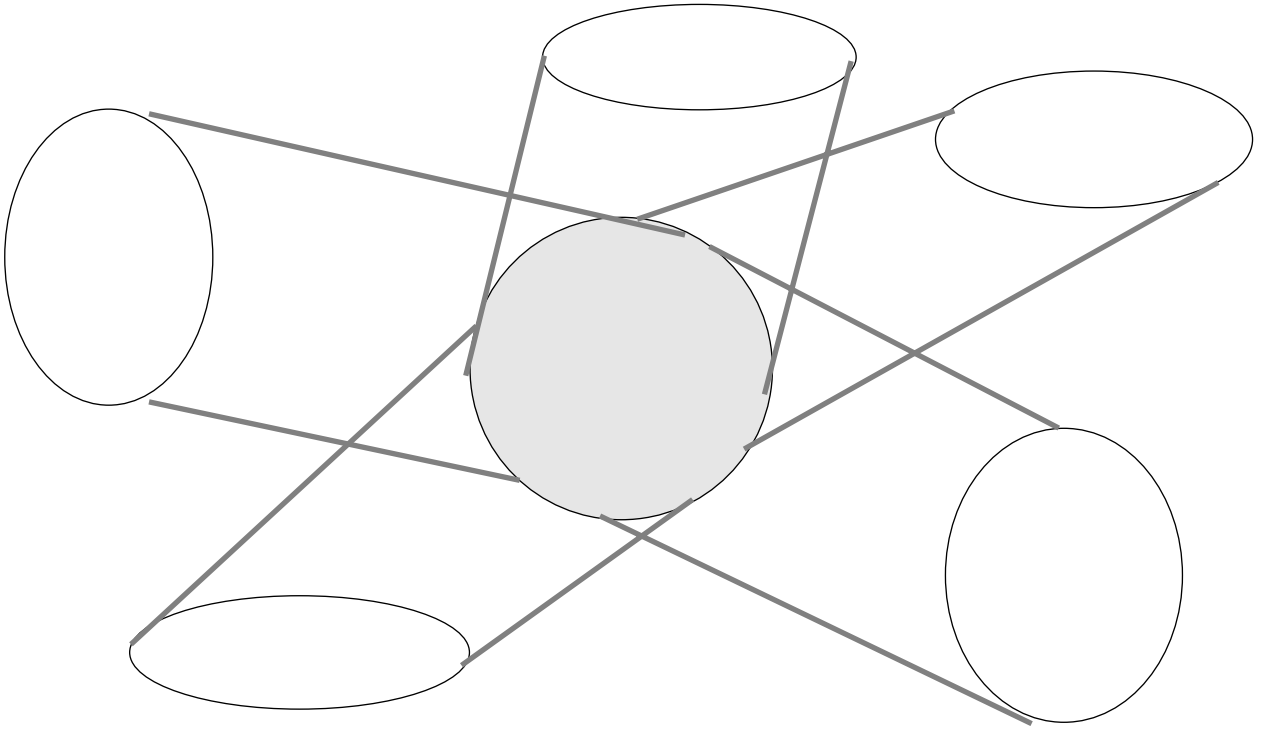
This presentation is based on APM.1336.01 “ANSAwise - The ODP Reference Model”. (The Appendices are drawn from APM.1327.01 “ANSAwise - The Computational Model” and APM.1351.01 “ANSAwise - The Engineering Model”.)

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.1564.01 **Approved** 8th September 1995
Briefing Note

Distribution:
Supersedes:
Superseded by:

Architecture for ODP, Part I: RM-ODP



In this session

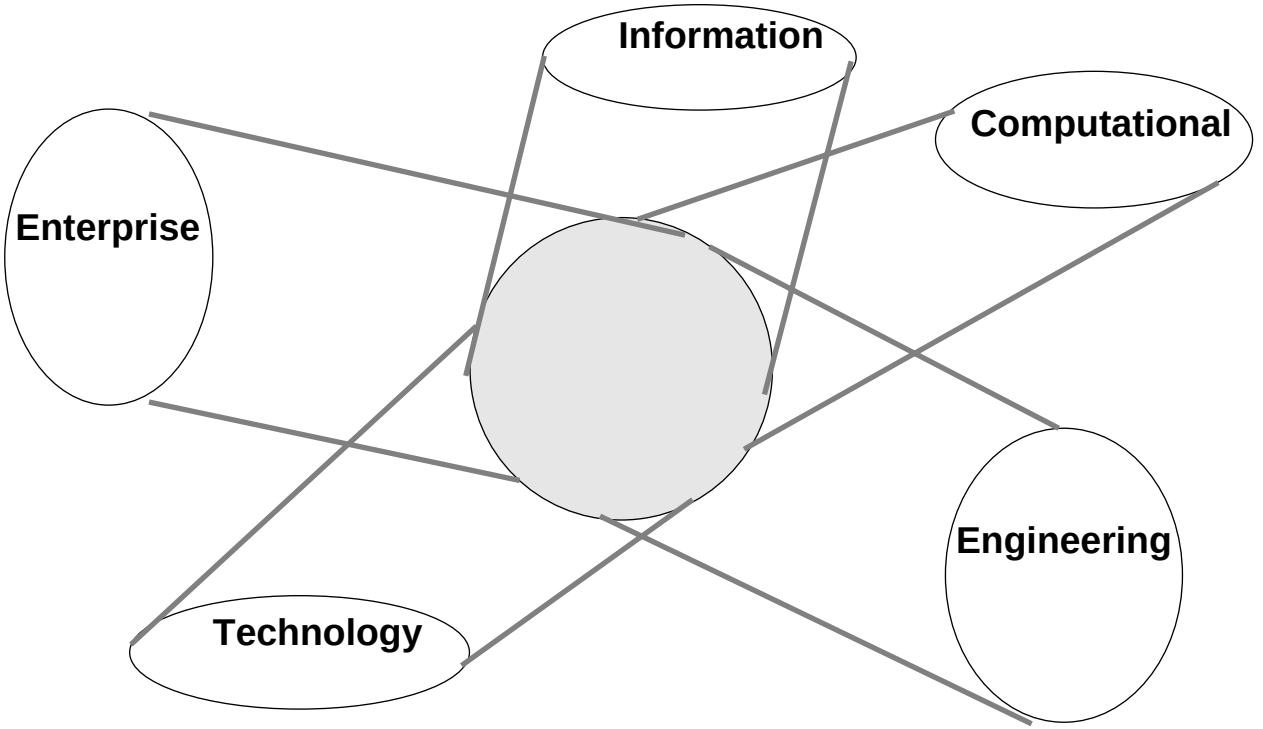
- *Show how separating the viewpoints of a system help you build open distributed systems*
- *Explain the significance of the ODP Reference Model*
- *Explain the goals of ODP*
- *Explain the key concepts of ODP*
- *Show the relationship to other standards*
- *Enable you to find out more*

Distributed Systems have many aspects

- ***Distributed systems involve many different people (the stakeholders)***
  - **business managers, users, IT managers, IT developers,...**
- ***These people are concerned with different aspects of the system***
  - **they see the system from a different viewpoint**
  - **each viewpoint is important**
- ***We need to be able to separate out these concerns when describing distributed systems***
  - **so that each stakeholder can see that their needs are satisfied...**
  - **... without being overwhelmed by descriptions of aspects that are irrelevant to them**

Five different viewpoints

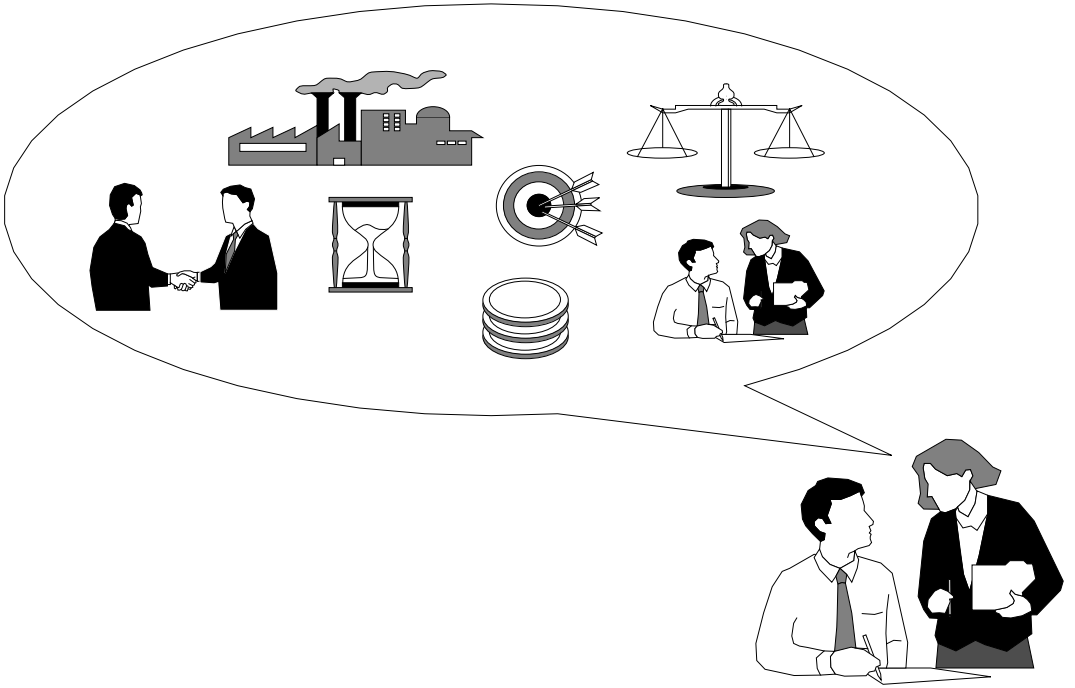
- ***These are of the same system and are <u>not</u> layered***



Content of the five viewpoints

- ***Enterprise*** **- the *purpose* of the enterprise and the system within it**
- ***Information*** **- the *meaning* of the information within the enterprise**
- ***Computational*** **- the *execution* as a model of distributed processing**
- ***Engineering*** **- the *mechanism* for realising the computational model**
- ***Technology*** **- the *conformance* of hardware, operating systems, compilers,...**

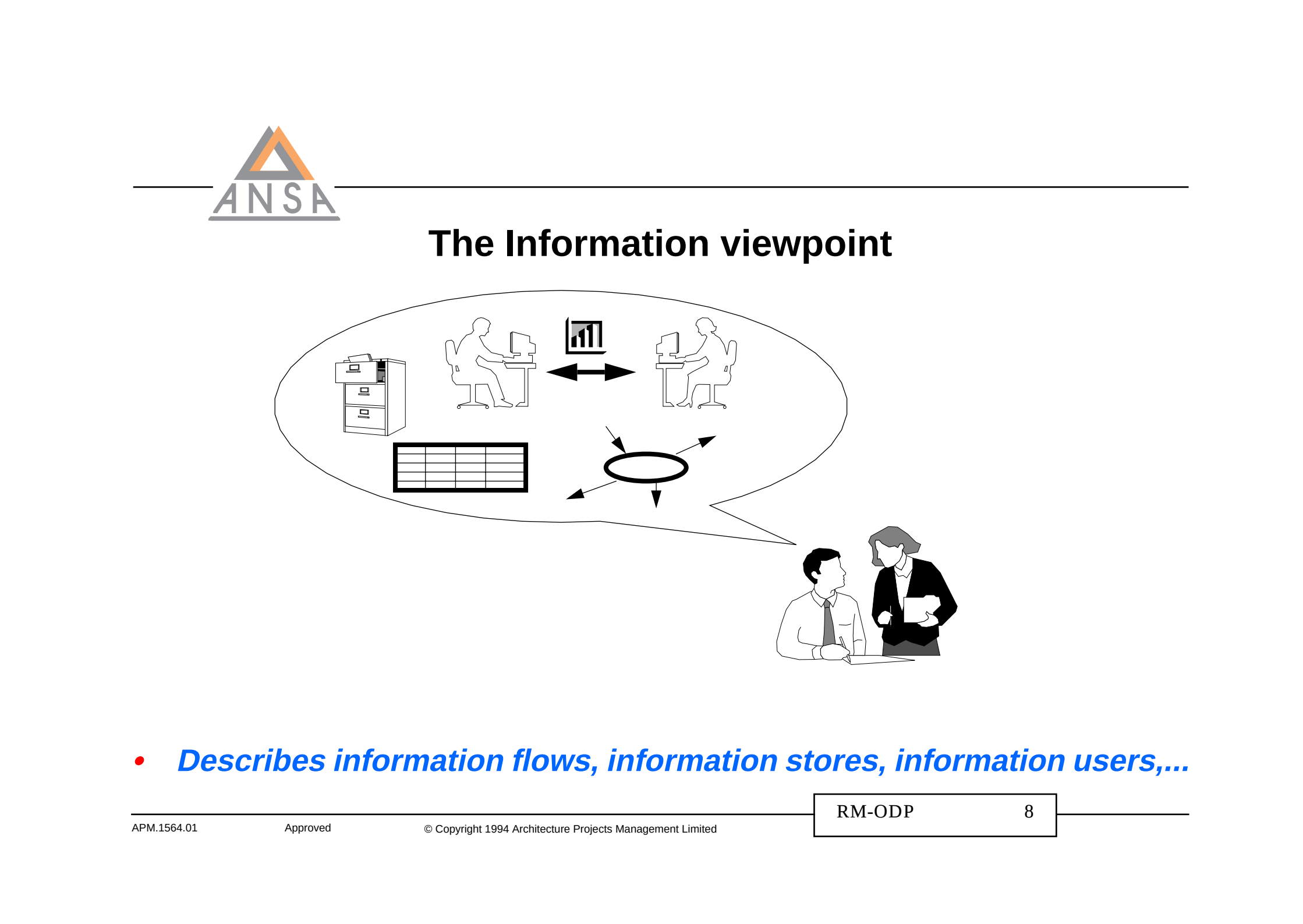
The Enterprise viewpoint



- ***Describes agreements, targets, people, time, money,...***

About the Enterprise viewpoint

- *More specifically, the Enterprise viewpoint is concerned with*
  - roles of people, organizations, and systems
  - rights, responsibilities, and obligations
  - resources



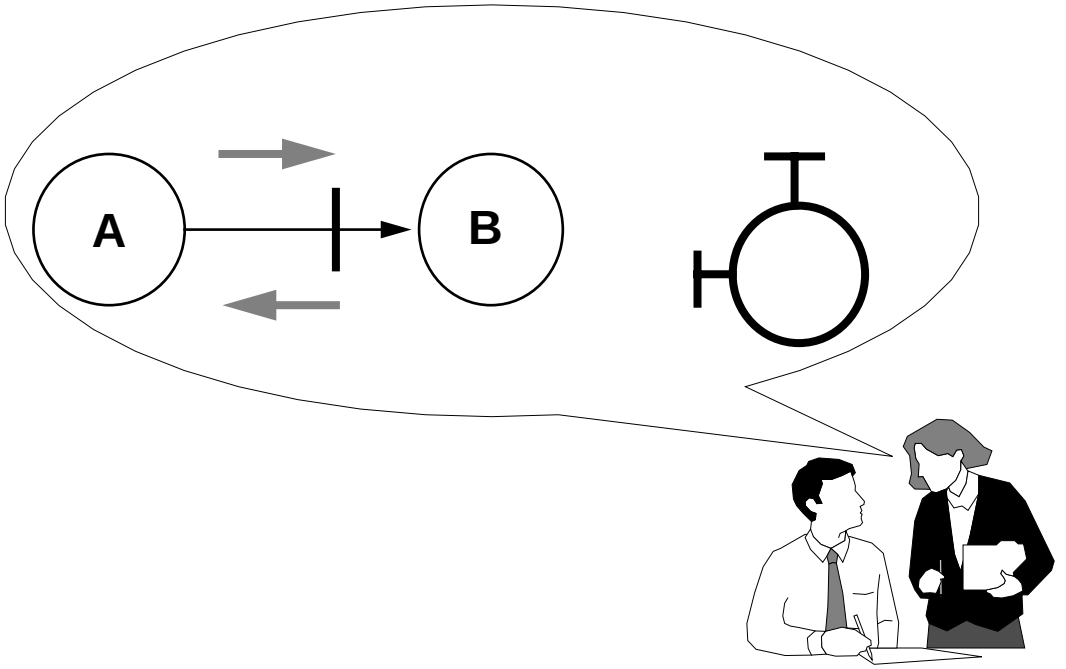
The Information viewpoint

- ***Describes information flows, information stores, information users,...***

About the information viewpoint

- ***The information viewpoint describes objects***
  - **not interfaces**
- ***This is familiar territory for the business analyst or database specialist...***
  - **...schemas, entities,...**

The Computational viewpoint

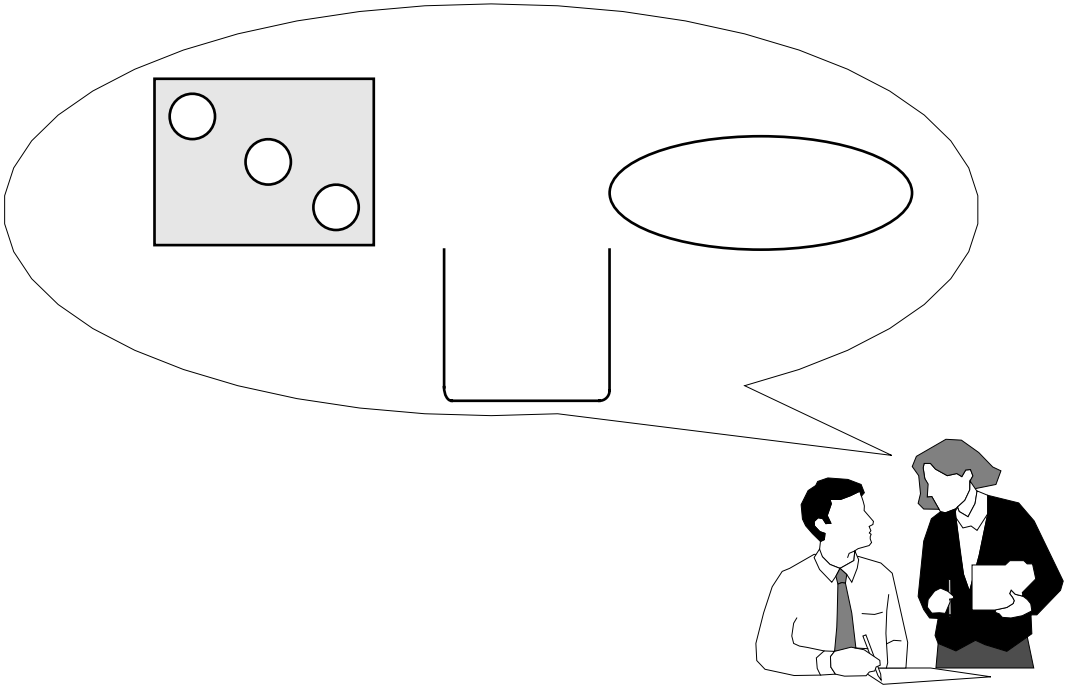


- ***Describes objects, interfaces, operations,...***

About the Computational viewpoint

- ***The distribution of a distributed system is ignored by the Computational viewpoint***
  - **it is transparent to the Computational viewpoint**
- ***From the Computational viewpoint***
  - **resources are always available when needed**
  - **communication between objects is transparent**

The Engineering viewpoint

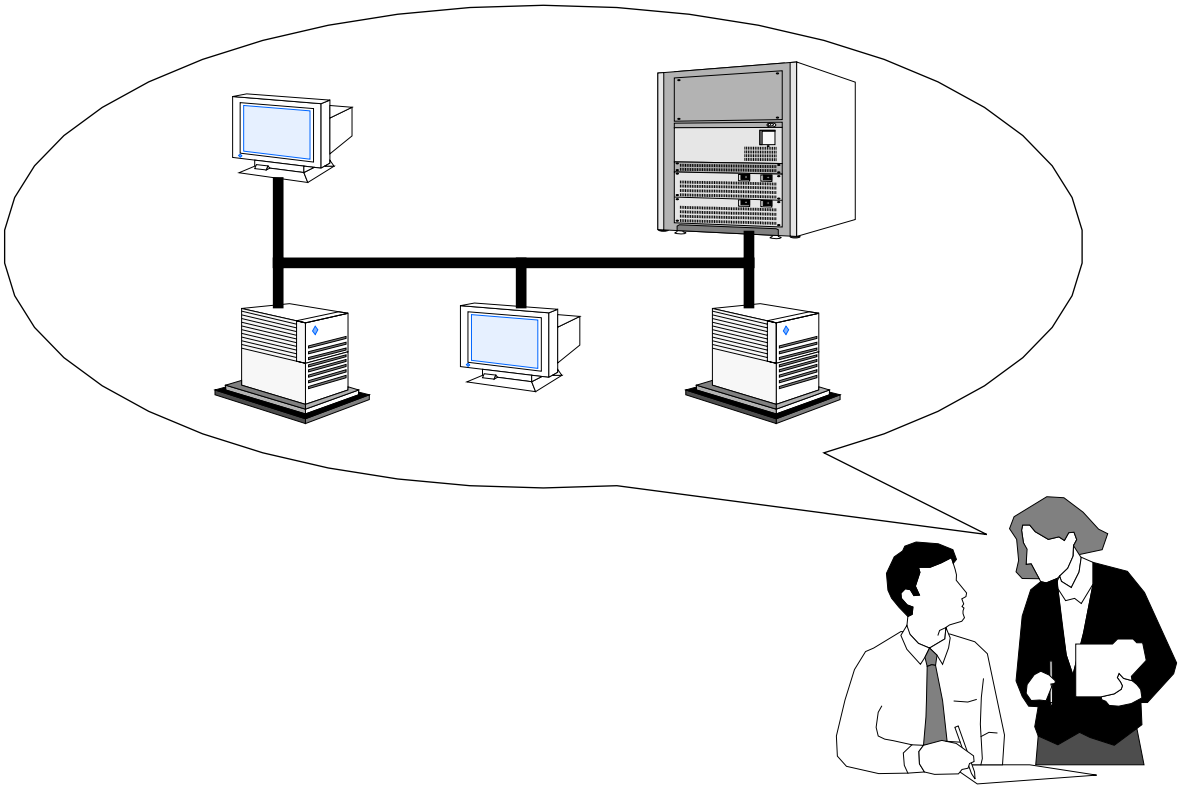


- ***Describes clusters, nodes, channels,...***

About the Engineering viewpoint

- *The Engineering viewpoint describes the infrastructure for the Computational viewpoint*
- *The infrastructure deals with*
  - communication channels between objects
  - resource management
- *It provides transparency mechanisms that hide the distribution from the Computational viewpoint*

The Technology viewpoint



- ***Describes how the system design uses the actual technology***

About the Technology viewpoint

- *The Technology viewpoint is mainly concerned with conformance to standards of actual hardware and software*

- *There are few rules in the Technology viewpoint*

  - rules will be implementation-dependent

Interoperability in the five viewpoints

- ***Successful interoperability requires all the viewpoints to work together***
  - **a mismatch in any one can prevent interoperation**
- ***The aim is detect and resolve this mismatch at specification time***
  - **one day there will be software tools to do this automatically**

Technology mismatch

- *Two departments wish to interconnect their LANs*
  - one uses Ethernet...
  - ...the other uses Token Ring

Engineering mismatch

- ***An engineering organization wishes to use an existing database system to store information from a real-time control system***

  - **the real-time control system delivers a periodic data feed...**

  - **... the database can't guarantee to respond in time with an acknowledgement**

Computational mismatch

- ***A company's Marketing department wishes to use the R&D department's document management system to store the master copies of its literature***

  - **the Marketing application accesses documents by filename...**

  - **...the R&D system accesses documents by reference number**

Information mismatch

- *An company wishes to integrate their Marketing and Accounts systems*

  - each keeps information about ‘customers’...

  - ... but their definition of a ‘customer’ is different

Enterprise mismatch

- *Two airlines wish to connect their reservation systems*
  - each has a policy on cancellations...
  - ...one gives automatic refunds
  - ...one automatically rebooks

What is ODP?

- ***Open Distributed Processing is a goal***
  - **the ability to create open distributed systems...**
  - **... connecting all kinds of IT systems**
  - **.... spanning organizational boundaries**
- ***Specifically, ODP aims to provide***
  - **interoperability of applications between distributed systems**
  - **portability of applications between distributed systems**
  - **... in a way that is *transparent* to the applications**

RM-ODP

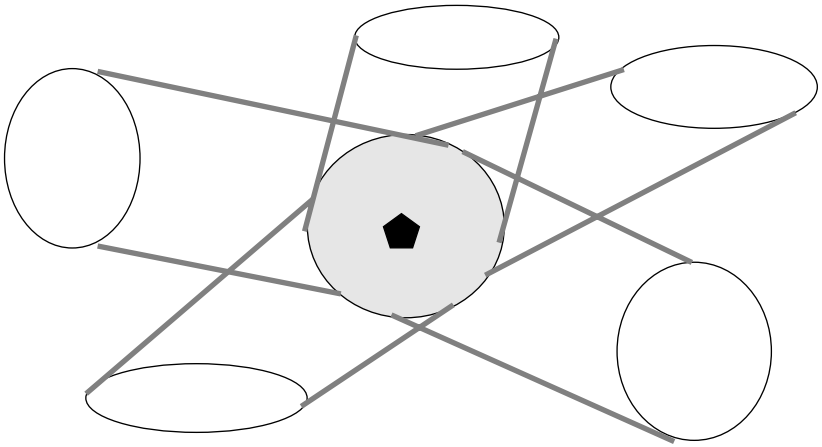
- ***The Basic Reference Model for Open Distributed Processing (RM-ODP) is...***
  - **an architectural framework for understanding the problems and concerns of distributed systems**
  - **a framework for assessing the conformance of a particular system**
  - **a forthcoming international standard**
- ***...a starting point for ODP standards***
- ***Standardization is essential for openness to be achieved***

Key concepts of RM-ODP

- ***Objects and interfaces***
- ***Transparency***
- ***Viewpoints***

The viewpoints are linked into a framework

- ***Because the viewpoints are views of the same system...***

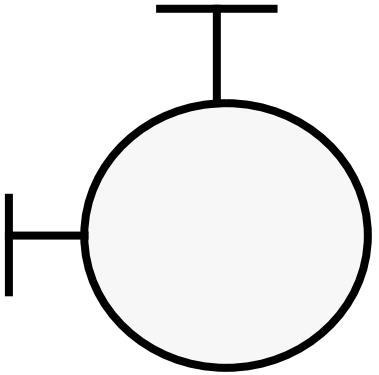


  - **to make sure the viewpoints are consistent with each other**

Service

- *To separate a system into parts, each part must offer a coherent service*
- *The service must be explicitly specified*
- *Specifications are declarative; what, not how*
- *The provider of a service agrees to meet the specification*
- *The provider does not reveal how the service is provided*
  - **it could be via a mainframe legacy system**
- *... in a word, encapsulation*

Objects for encapsulation



- ***Objects are encapsulated...***
  - **...all interactions are via defined interfaces**
  - **...all objects interact in the same way**

Examples of objects in the different viewpoints

- *Enterprise viewpoint*
  - a person, organization, or resource
- *Information viewpoint*
  - an information entity
- *Computational viewpoint*
  - an encapsulation of behaviour/state
- *Engineering viewpoint*
  - a channel controller
- *Technology viewpoint*
  - a machine

Distributed systems are different

- **Many traditional system design assumptions must be reversed**

Traditional	*Reversed*
Local	Remote
Sequential	Concurrent
Homogeneous Environment	Diverse Environment
Fixed Location	Mobile
Single Copy	Multiple Copies
Synchronous	Asynchronous
Direct	Indirect
Shared	Separate
Global	Context Relative
Complete Failures	Partial Failures
Early Binding	Late Binding

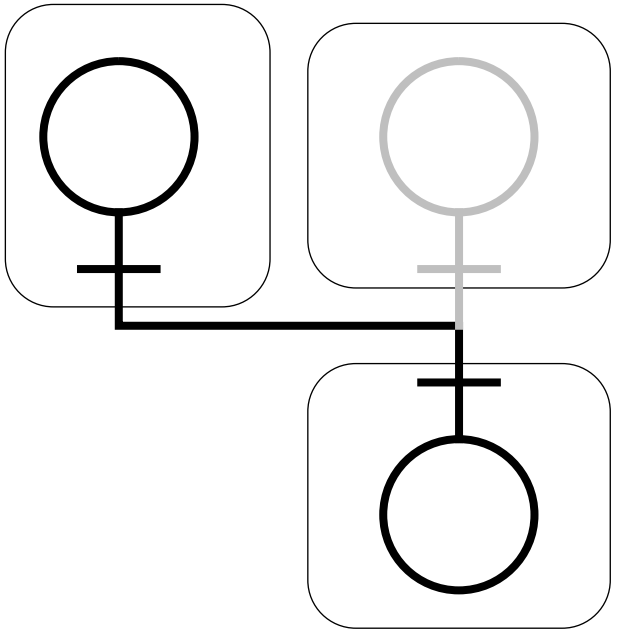
- ***A systematic approach is needed to avoid these assumptions***

The transparency approach

- ***Applications should not be burdened with the complexity of handling these reversed assumptions***
- ***Something else must handle this complexity...***
- ***... transparency mechanisms in the infrastructure***

Example Transparency - Migration

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



Exploiting the reversed assumptions

- ***Exploit positive consequences***
  - **Consider, for example...**
  - **Late binding: Trading supports choice of Quality of Service**
  - **Multiple copies: Concurrency supports parallelism**
  - **Partial failure: Replication supports availability**
- ***Mask negative consequences***
  - **Use selective *transparency* mechanisms, for example...**
  - **Migration transparency: Isolates client from service relocation**
  - **Replication transparency: Isolates client from multiple copies of service**

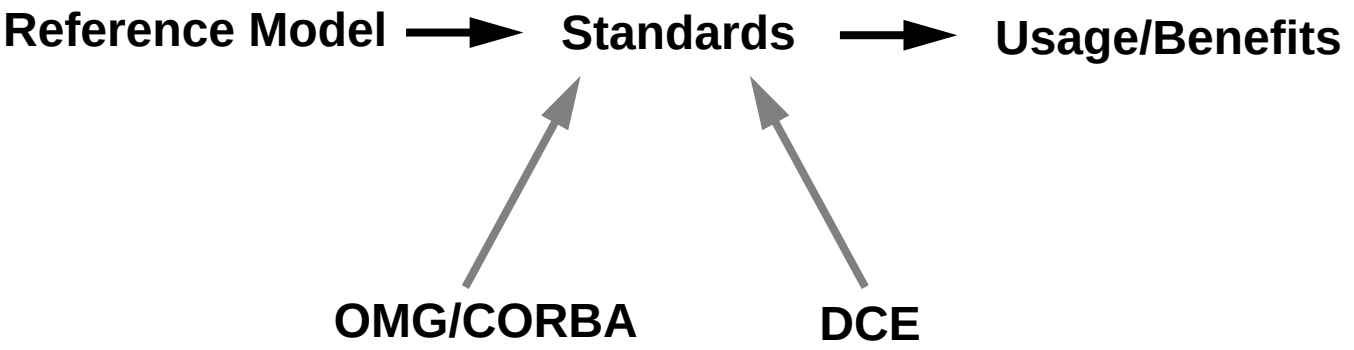
Handling the reversed assumptions - The Computational and Engineering viewpoints

- *Isolate specification of transparencies from their design*
  - Computational viewpoint defines the transparencies
  - Engineering viewpoint provides the mechanisms
  - Applications developers just state which transparencies they need
- *Automate the building of transparencies*
  - Software tools can construct transparencies from the engineering mechanisms

RM-ODP is a starting point

- *It is a framework for the development of other ODP standards...*
  - standards for specification, modelling and programming languages
  - language mappings (APIs) for ODP systems
  - functional components of ODP systems (specific services)
- *...a framework for relating the different ODP standards to each other*
- *RM-ODP contains the concepts and rules needed to write these standards*
- *Many relevant standards already exist*
  - they need to be fitted into the framework

Relationship to other standards



- ***Liaison is in place with OMG, and the framework is being populated***
  - **function correspondence has been identified**
- ***CORBA and DCE are working bottom-up...***
- ***...ODP is working top-down***

Summary

- ***The Basic Reference Model of ODP (RM-ODP) is a framework standard***
  - **more detailed standards are needed to populate the framework**
- ***RM-ODP simplifies the design of distributed systems***
  - **using viewpoints to separate the concerns of stakeholders**
  - **using objects and interfaces for encapsulation**
  - **using transparencies to mask distribution from applications**
- ***For more information on ODP***
  - **for more on transparency mechanisms, see *The Challenge of ODP* (TR.033.02)**
  - **for reading RM-ODP itself, suggestions are given below**

Organization of the RM-ODP Standard

- *The standard is in four parts*
  - Part 1: Overview and guide to use (ISO/IEC 10746-1, ITU-T X.901)
  - Part 2: Descriptive model (ISO/IEC 10746-2, ITU-T X.902)
  - Part 3: Prescriptive model (ISO/IEC 10746-3, ITU-T X.903)
  - Part 4: Architectural semantics (ISO/IEC 10746-4, ITU-T X.904)
- *Each part describes the Reference Model in a different way*
  - Part 1 is an informal overview and rationale in plain English
  - Part 2 is a definition of the concepts and analytical framework
  - Part 3 is a specification of the characteristics of an ODP system
  - Part 4 is a definition of the concepts in terms of other formal description techniques (LOTOS, SDL, Estelle, Z)

The general flavour of RM-ODP

- *The style of each of the RM-ODP Parts is different*
  - Part 1 contains examples of an ODP system described from each of the five viewpoints
  - Part 2 contains a list of definitions
  - Part 3 contains a list of rules for each viewpoint
  - Part 4 contains a formal description
- *RM-ODP is hard to grasp...*
  - ...not because of detail, or length, but because it is so abstract
- *Start with Part 1*

RM-ODP Part 1: Overview and guide to use

- *Contains an overview of the ODP, rationale, explanations of key concepts, and some examples*
- *The examples show how to use RM-ODP to identify where more detailed standardization is necessary*
  - at reference points for conformance identified in Part 3
- *A suggestion for understanding Part 1:*
  - Start by reading the first few sections, then look at the examples later on, to see how the five viewpoints are used...
  - ... or follow the suggestions given at the beginning of Part 1 itself

RM-ODP Part 2: Descriptive model

- ***Defines the ODP key concepts***
  - **The definitions are sufficient to support the formal semantics of Part 4**
  - **The definitions are sufficient to establish requirements for new specification techniques**
- ***These definitions are terse, highly abstract, and strongly inter-related; for example:***
  - **"*Failure*: Violation of a contract"**
- ***A suggestion for understanding Part 2:***
  - **Stick to one viewpoint at a time**
  - **Find a concept in Part 1 (or Part 3) that is of interest**
  - **Follow through the definitions in Part 2, and refer back to Part 1**

RM-ODP Part 3: Prescriptive model

- ***Specifies rules that a distributed system must follow if it is to be an ODP system...***
  - **'structuring rules' using the concept definitions of Part 2**
  - **conformance and reference points of an implementation at which these rules can be checked**
  - **consistency rules between specifications from different viewpoints**
  - **...these rules must also be followed by other ODP standards (outside the RM-ODP)**
- ***Specifies the ODP functions and transparencies***
- ***A suggestion for understanding Part 3:***
  - **Start by reading about the ODP functions and transparencies**

RM-ODP Part 4: Architectural semantics

- *Contains a formal description of the basic RM-ODP Part 2 concepts*
  - in LOTOS, SDL, Estelle, and Z
- *These formal descriptions map RM-ODP concepts to the corresponding concepts of LOTOS, SDL, Estelle, and Z*
  - sometimes there is no direct equivalent
- *A suggestion for understanding Part 4:*
  - Read the section that uses a formal description technique you already know

Status of RM-ODP

- *RM-ODP is being standardized jointly by ISO, IEC, and ITU-T (formally CCITT)*
  - RM-ODP is based on work pioneered by ANSA
- *Now has Draft International Standard status*
- *Each Part is progressing separately*

Other ODP standards

- *ODP components*
  - Trader was chosen as the first component to be standardized
  - Type Manager next to come
- *Profile*
- *Management*
- *Security*

Finding out more about ODP

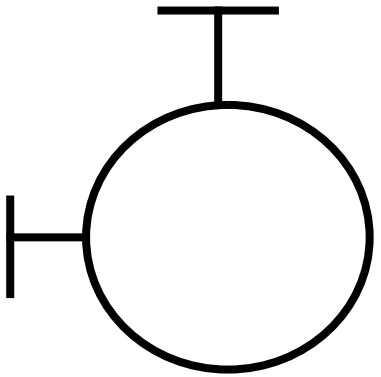
- *Via APM*
  - Andrew Herbert (editor of Part 3)
- *ISO/IEC JTC1/SC 21/WG7 Project 21.43*

  Secretariat:
  Standards Association of Australia,
  PO Box 1055, Strathfield,
  NSW, Australia 2135;

  Tel: +61 2 746 4830;
  Fax: +61 2 746 8450

APPENDIX I

The Computational Model



In this Appendix

- **Explain the concepts of the Computational Model**
- **Explain the rules that are imposed, and why they are important**
- **Explain how objects and interfaces are used**

What is the Computational Model?

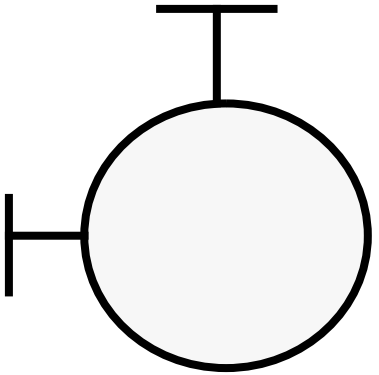
- It is a model for specifying service provision and use
  - Objects may provide multiple services to other objects
  - Objects may use multiple services from other objects
- Objects and their services may be created and destroyed dynamically
- The ability to use a service may be passed from one object to another
- The Computational Model is based on interacting *distributed objects*
  - It makes no assumptions about where the objects are located

What is meant by ‘object’?

- **Nowadays all software is supposed to be ‘object-oriented’**
  - **OOA/OOD (analysis and design methods)**
  - **OOL (programming languages)**
  - **OODBMS (databases)**
  - **... and many more**
- **All these have one key concept in common...**

Encapsulation - the key concept

- **Objects are encapsulated**
  - **every object provides a service via interfaces**
  - **the interface is public; the implementation is private and hidden**
  - **encapsulation forms a boundary; the only access to an object is via its interfaces**

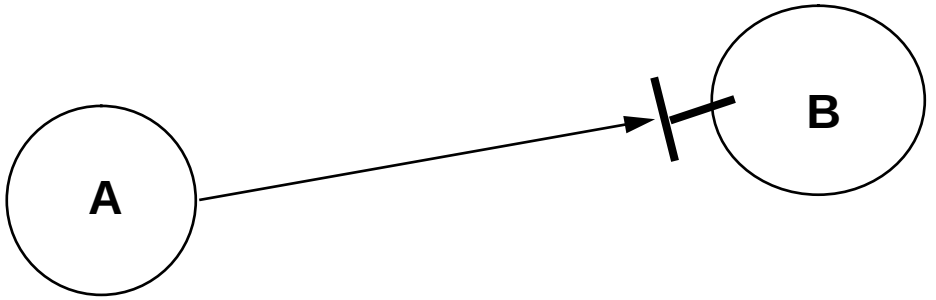


Encapsulation hides data representation

- **The boundary hides the data representation used by the object**
  - **Diversity: different objects use different representations**
  - **In a distributed system, different representations are used in different places**
  - **One object cannot manipulate another's data (only via the interface)**
  - **Data representations cannot be transferred between objects**

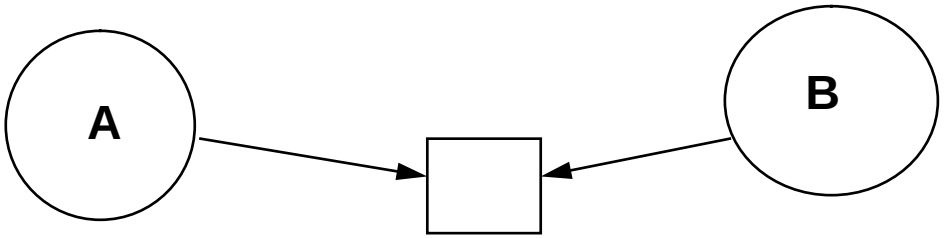
Encapsulation for distributed objects

- **We cannot assume anything about the location of distributed objects**
  - **Objects A and B could be on the same machine, or in different countries**

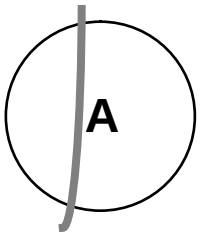


Encapsulation rules

- **Objects cannot share state directly (only access it via interfaces)**
  - ***this is not allowed***

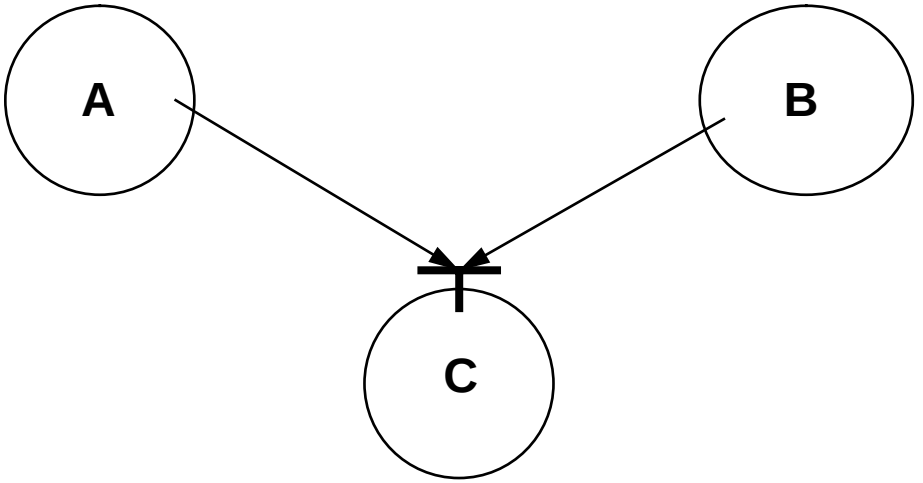


- **An object cannot be distributed in pieces; it must all be in one place**
  - ***this is not allowed***



Sharing state via an object

- No back door is needed; use an interface to a shared object instead
  - Both A and B can access their shared state stored in C

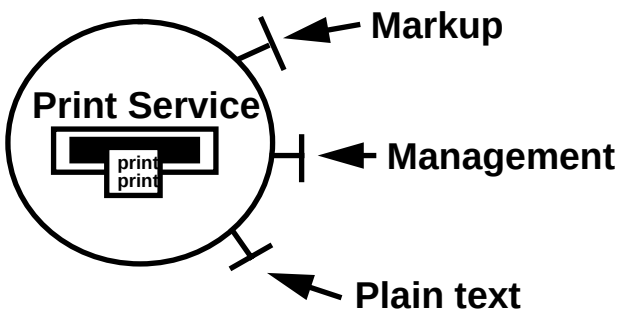


Multiple interfaces to an object

- Multiple interfaces allow flexible configurations...
- ... Consider a service for printing documents
- It must print two kinds of document
  - plain text documents
  - documents in some markup language, for example PostScript (TM)
- It must also have a management interface for
  - starting and stopping a print queue
  - delete print jobs
  - other administrative tasks

A print service with multiple interfaces

- **One object with plain text, markup, and management interfaces**



Objects and Interfaces - summary so far

- **Objects encapsulate state**
  - **and must take full responsibility for it**
- **Interfaces provide the service access points**
- **Objects can have multiple interfaces**
- **[Object-oriented programming languages:**
  - **use one construct for both encapsulation and service provision**
  - **provide only a single interface]**

with their shared state

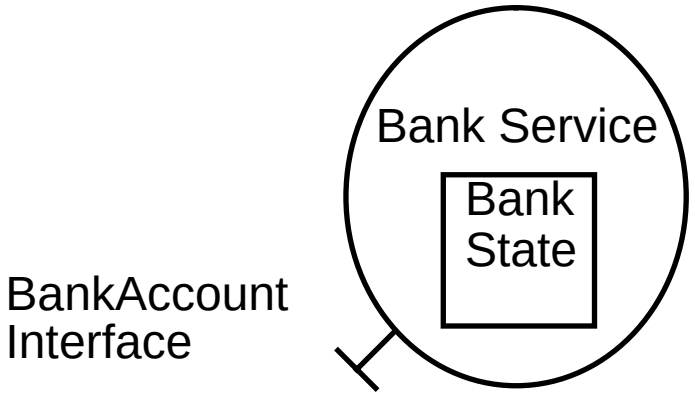
Operations - the actions in an interface

- **An interface defines one or more *operations***
- **Operations are the actions that the user of the interface can invoke**
  - **Similar to "methods" in object-oriented languages**
- **Because of encapsulation, the defined operations are the only way to act on the object**
  - **No “back door” [not even for management or debugging]**

Operations - a Simple Bank example

- Consider a bank account service...
- Each bank account has state
  - account balance
  - name of customer
  - ... and others
- Each bank account has actions that affect this state
  - Credit: depositing money, increases the balance
  - Debit: writing a cheque, decreases the balance
  - Show: enquires the account balance
- These actions are the only way of manipulating the bank account; these actions are the *operations*

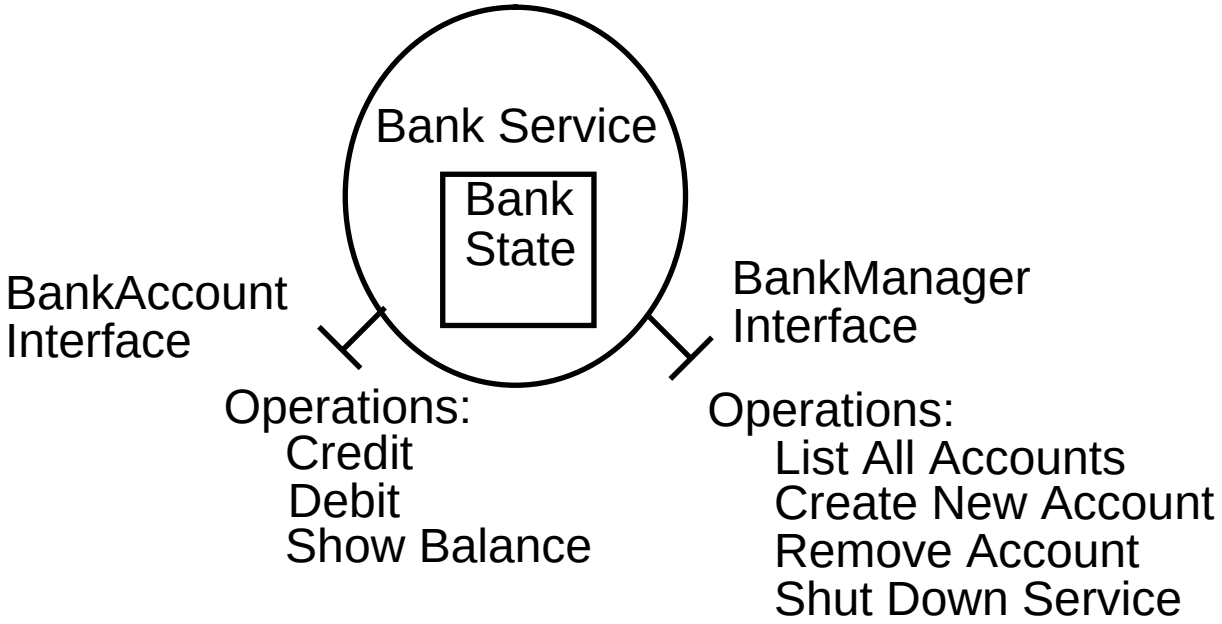
Simple Bank - Service, interface, and operations



Operations:
- Credit
- Debit
- ShowBalance

Simple Bank - Operations in multiple interfaces

- **A service can have more than one interface**
- **Each interface has its own operations**



Operations and Arguments

- **Each operation has a name, input arguments, and output arguments**
  - **for example, the Credit operation returns the new balance**

    Credit (amount : Integer) -> (newBalance : Integer)
  - **amount is a input argument; newBalance is an output argument (result)**
- **Each operation can have multiple input and output arguments**
  - **this example only has one of each**

Operations and Terminations

- **An operation can have more than one possible outcome**
- **Consider operations on the BankAccount:**

  Credit (amount : Integer) -> (newBalance : Integer)

  Debit (amount : Integer) -> (newBalance : Integer)

  - **... but the balance may be insufficient! - so**
  - Debit (amount : Integer) -> (newBalance : Integer) ->InsufficientFunds()

- **There is one Debit operation with two possible outcomes; two separate *terminations***

Named Terminations

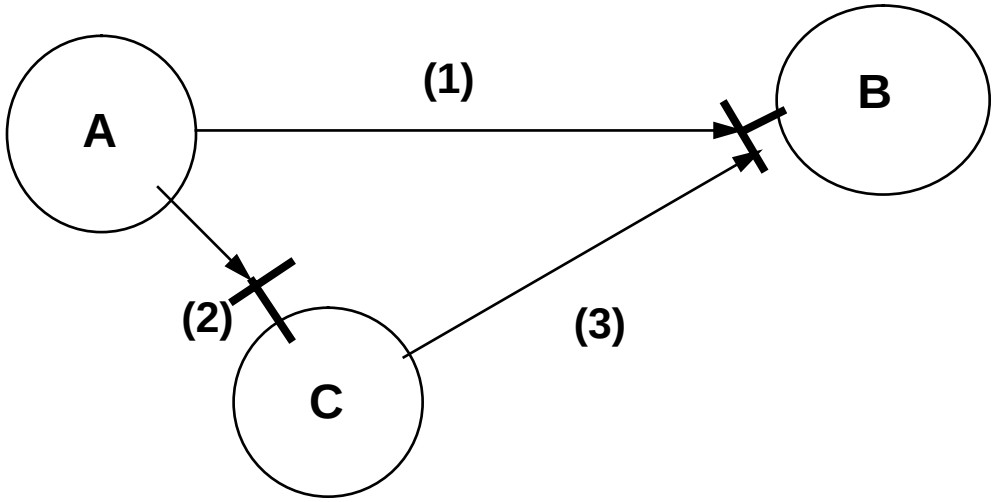
- **The InsufficientFunds outcome is called a *Named Termination***
  - **the normal outcome is an unnamed (anonymous) termination**
  - **one termination of an operation may be anonymous**
- **Each termination may return multiple results**
- **The invoker of an operation distinguishes between the different terminations by their names**

Operations, Arguments, and Terminations - summary

- An *operation* is the unit of interaction. It has:
  - a name
  - a signature (list of input arguments)
  - a set of *terminations* (possible outcomes)
- A termination has:
  - a name
  - a signature (list of output arguments)
- Terminations make operations more powerful than ‘functions’ or ‘methods’

Using Interfaces

- Object A is using one of object B's interfaces
- Suppose it needs to tell C to use the same interface



- It must be possible to pass a binding to B's interface between A and C

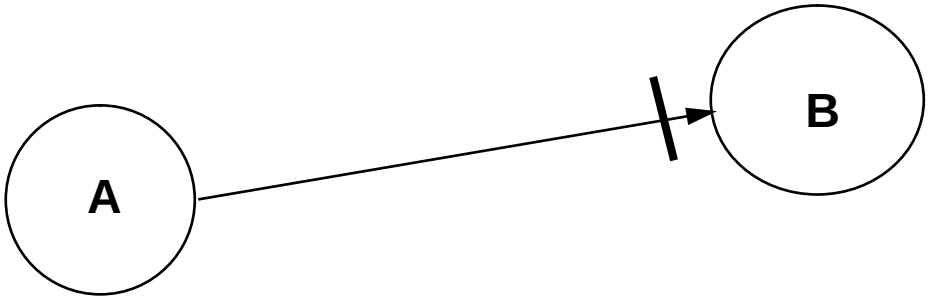
Interface Bindings

- **Interfaces are identified by *interface bindings*; these can be passed to other objects as arguments (or results)**
  - **for example, when A passes B's interface binding to C, it calls...**

    C_op (b: TypeOfInterfaceToB) ->()
  - **... with B's interface binding as the argument**
- **Only the binding is passed, not the interface...**
- **... In fact, all operations are invoked via interface bindings**

The right interface?

- Suppose object A has a binding for an interface of B...
- ... how does object A know that B has the right kind of interface?
  - it can't inspect the object itself
  - objects A and B could be on the same machine, or in different countries



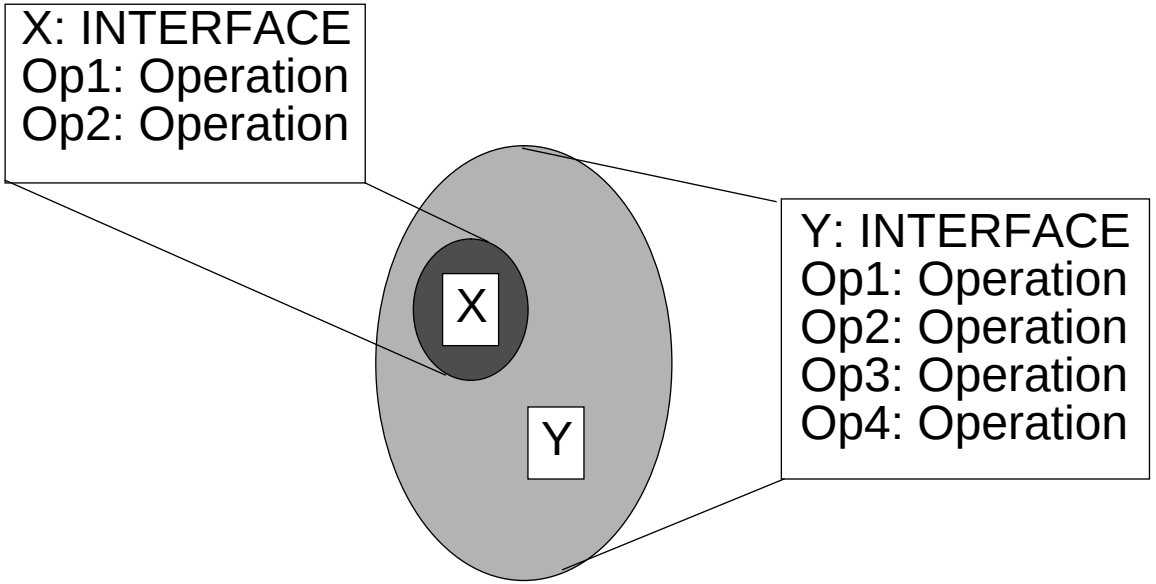
- It can tell, because interfaces have an *interface type*

Interface types

- **The interface type consists of the set of its operations**
  - **Plus all their signatures and terminations**
- **Object A can use an interface of object B, provided that the interface provided by B *conforms* with that expected by A**
- **This means that object A must state what interface type it expects**

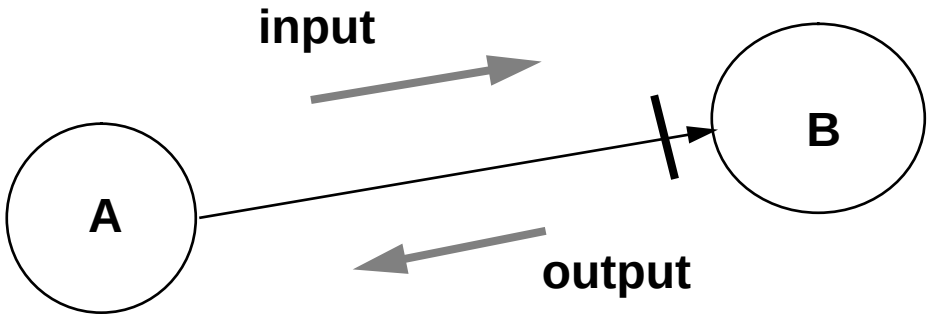
Type Conformance

- **Conforming interface types do not have to be identical...**
- **...Interface type Y *conforms to* interface type X**



Type Conformance Has Two Sides

- To conform, B must provide at least the operations expected by A...



- ...but also, A must handle at least the terminations given by B
  - same idea, but the ‘at least’ rule is the other way round for terminations
- To conform, both of these must be checked... [a two-sided *contract*]
  - B must not receive unknown operations
  - A must not receive unknown terminations

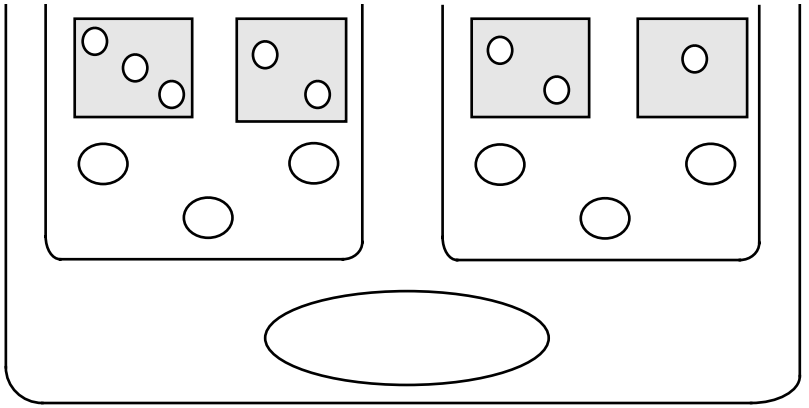
Summary

- **Objects enforce strict encapsulation**
  - **state cannot be shared between objects**
  - **only whole objects can be re-configured (*part* of an object cannot be)**
- **Interfaces provide the points of service provision**
  - **objects may have multiple interfaces**
  - **interfaces are typed**
- **Interfaces are composed of operations**
  - **operations can have multiple terminations with multiple results**
- **For more information:**
  - **for object-oriented concepts, see *Distributing Objects* (TR.018.01)**
  - **for a formal description, see *The ANSA Computational Model* (AR.001.01)**

Computational Model - topics not covered

- **Concurrency**
  - threads and sub-threads, enforcing tree-structured concurrency
- **Streams - continuous (possibly bi-directional) information flows, for example:**
  - speech, in telephony and voice processing: an audio stream
  - video, in multimedia: a video stream
  - sensor data, in telemetry: a data stream
- **Synchronous programming constructs**
  - reactive systems, bounded execution paths
  - testing, receiving, waiting for and transmitting signals
  - watchdogs

APPENDIX II
The Engineering Model



In this Appendix

- *Explain the concepts in the Engineering Model*
- *Explain the various types of transparency mechanisms*
- *Compare the Engineering and Computational Models*

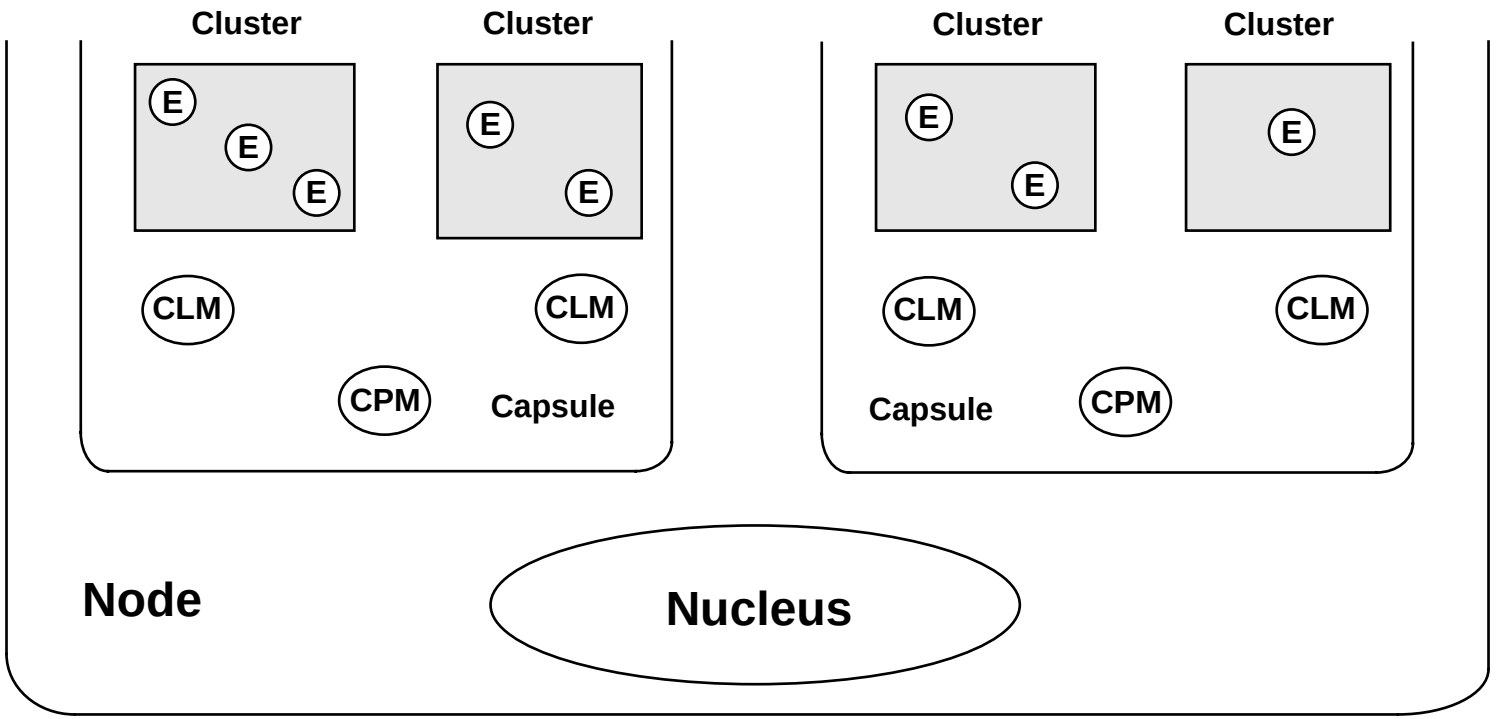
What is the Engineering Model for?

- *To allow trade-offs*
  - flexibility versus performance
  - time versus space
  - ... and many others
- *But without affecting the Computational Model*

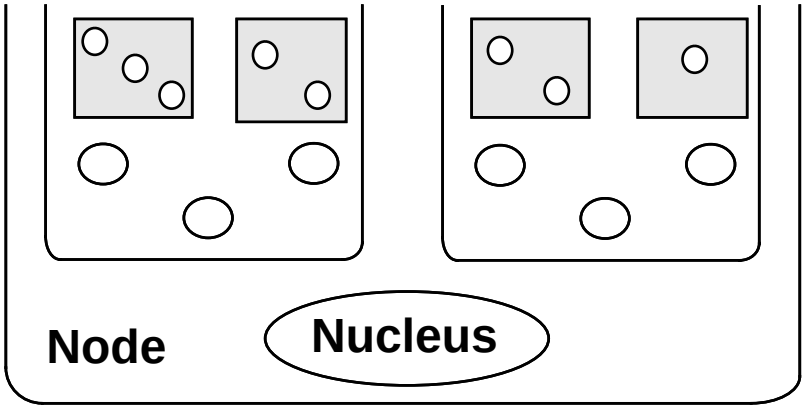
Engineering is infrastructure

- ***Provides infrastructure for encapsulation***
  - ***capsules*** **for resource allocation and protection**
  - ***clusters*** **for collective activation, deactivation, and migration**
  - ***nodes*** **for network addressing**
- ***Provides channels for communication***
  - **point-to-point (simple client-server)**
  - **point-to-multipoint**
  - **streams**
- ***Provides concurrency mechanisms***
  - ***threads*** **for concurrent execution**
- ***Provides transparency mechanisms***

Engineering Encapsulation Infrastructure

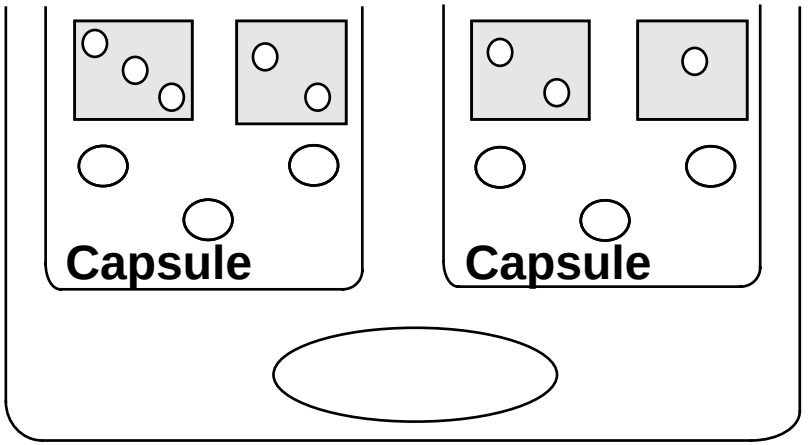


Node and Nucleus



- *A node is the unit of network addressing*
- *The nucleus handles communications between nodes and capsules*
  - it may be part of the operating system (if there is one), or a layer on top of the operating system

Capsules



- ***Capsules contain objects***
  - **objects must be encapsulated**
  - **every object is contained in a capsule**
  - **different capsules separate the objects**

Capsules can contain more than one object

- *Objects of the same type, or of different types*
- *Put more than one object in a capsule to*
  - share resources
  - share state information

Capsules and Shared Memory

- *In a distributed system, two objects cannot usually share memory*
  - the objects could be distributed anywhere in the world!
  - but if particular objects do, their implementation may be more efficient
  - ... for example, you might keep objects of the same type together in the same capsule
- *Objects in the same capsule can share memory for efficiency - but then there is no protection boundary between them*
  - resources: they compete for the same resources (physical and virtual memory)
  - robustness: object failure within the capsule can cause the whole capsule to fail
  - security: there is no security between objects in the same capsule
- *Sharing memory between different capsules is not permitted*

Capsules and Objects

- ***So, the Computational and Engineering Models of an object are different:***

  - **in the Computational Model, every object is separately encapsulated; objects cannot share state**

  - **in the Engineering Model, objects can share a capsule, and share state**

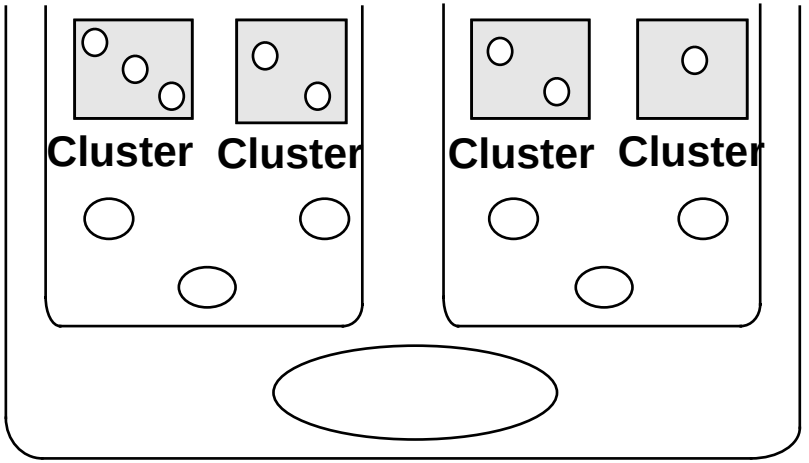
Capsules and Interfaces

- ***Objects in the same capsule can still invoke each other's interfaces***
  - **you are not compelled to use shared memory within a capsule**
- ***Object interfaces are invoked in the same way...***
  - **within a capsule**
  - **between two capsules on the same node**
  - **between two nodes**
- ***...the infrastructure will optimize communications between objects on the same node***

Nodes and Capsules

- ***Most systems can support more than one capsule per node***
  - **using the multi-tasking and memory protection features of the operating system**
  - **Unix can...**
  - **...DOS can't (only one capsule per node)**

Clusters



- ***A cluster is a collection of objects that must be 'kept together'***
  - **if an object moves (*migrates*) between capsules, objects that share memory must move together with that object**
  - **if an object fails, you might want to shut down other objects automatically**
  - **when starting up one object, you might want to start up other objects automatically**

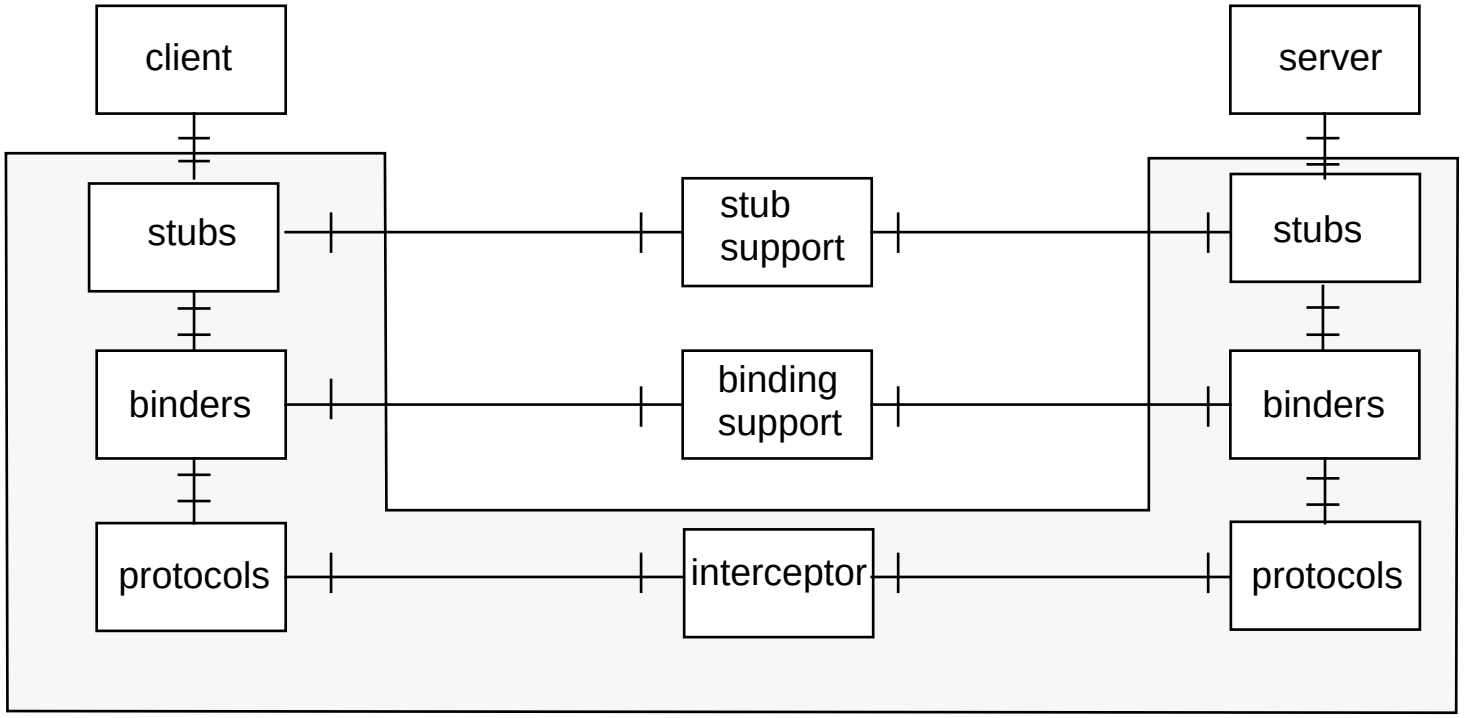
Clusters and Capsules

- ***A capsule can support multiple clusters***
- ***A Cluster Manager handles this for each cluster***
  - **it coordinates *checkpoints* for the cluster (snapshots of its state)**
- ***If there are no clusters, no Cluster Manager is needed***

Channels

- ***Channels are communication paths between objects***
- ***Channels may be:***
  - **1 to 1 (point-to-point)**
  - **1 to many (point-to-multipoint)**
- ***Channels may be:***
  - **operational**
  - **stream**
- ***Channels are layered:***
  - **built from *stubs*, *binders*, and *protocol objects***
  - **there may be multiple protocols in a particular infrastructure...**
  - **...layering hides the diversity from the application**

Point-to-Point Client-Server Channel



Stubs, Binders, and Protocol Objects

- *Stubs provide data conversion*
- *Binders manage end-to-end integrity and quality-of-service*
- *Protocol objects provide communication*
- *... Most application developers will only be aware of stubs*
  - and even these will probably be generated automatically

Concurrency

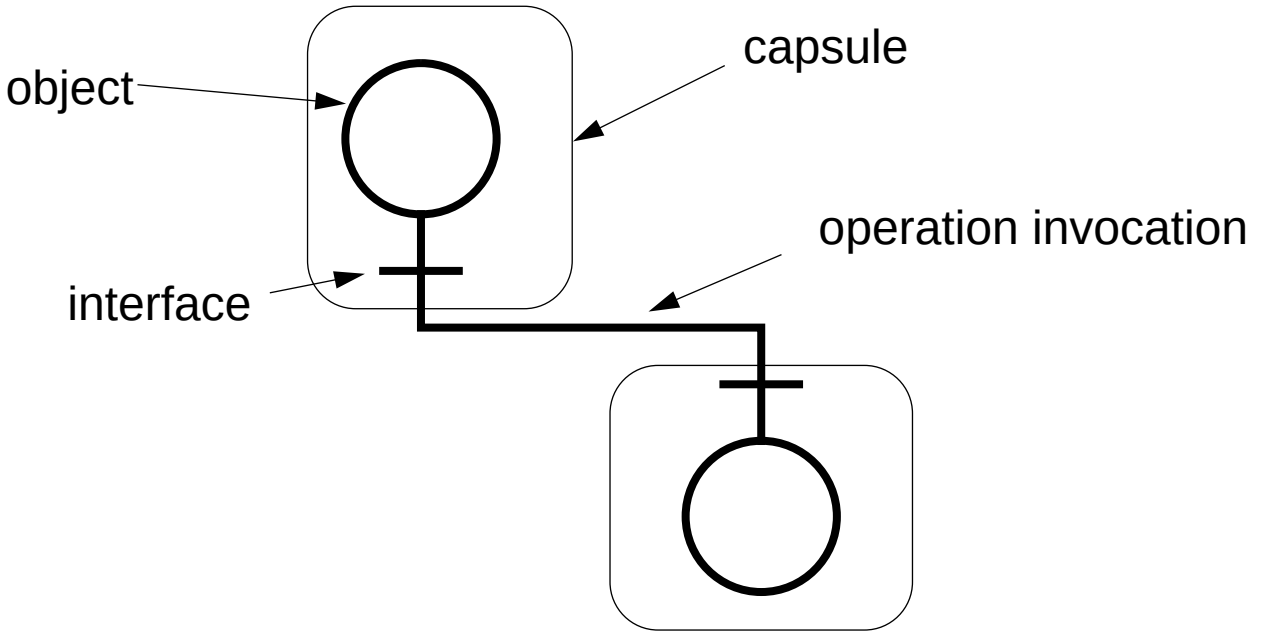
- ***Capsules on the same node run concurrently***
- ***Within a capsule, you can use threads to achieve concurrency***
  - **these may be supported by the operating system, or by the nucleus**
  - **... threads are always available, even under DOS!**
- ***On a multi-processor system, different threads in the same capsule can run simultaneously on different CPUs***

Transparencies - Simplifying distribution

- ***Remember that in a distributed system, traditional design assumptions must be reversed***
  - **for example, mobility: objects do not stay in one place, they can migrate**
- ***Must isolate the specification of transparencies from their design***
  - **Computational Model just assumes the transparencies are provided when needed**
  - **Engineering Model must show how transparencies are provided from engineering mechanisms**
- ***Applications developers simply state which transparencies they require***
  - **software tools construct them automatically from the engineering mechanisms**

Transparency examples

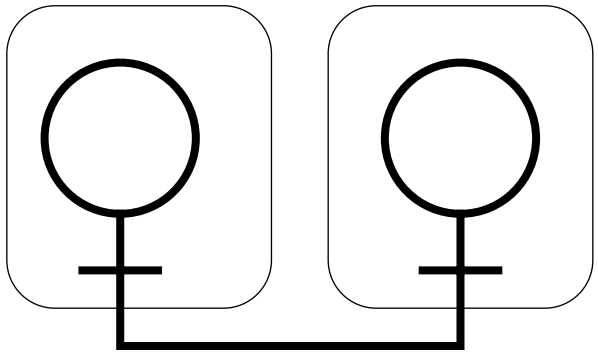
- *In these examples the diagrams are slightly simplified*
- *This shows an object invoking an operation from another capsule*



Selective Transparency Engineering - Location

- **Location Transparency**

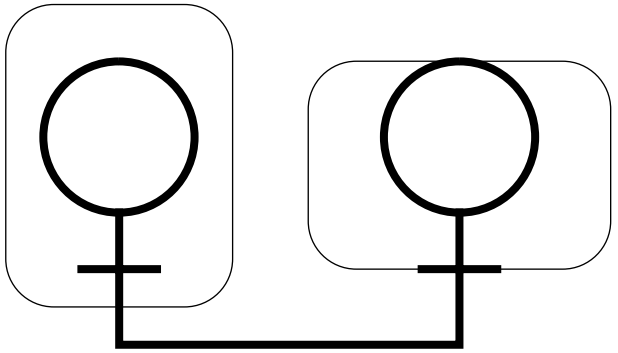
  - **application need not know where object is to use it**



  - **objects may be in the same capsule, different capsules, or different nodes**

Selective Transparency Engineering - Access

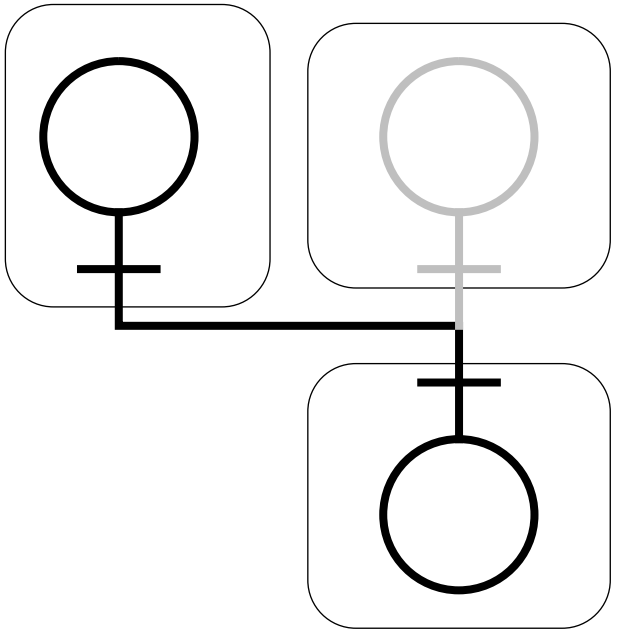
- **Access Transparency**
  - **application need not know the type of machine where the object is executing**



  - **objects may be in capsules on different operating systems, on different processor types (mainframe, workstation, or PC),...**

Selective Transparency Engineering - Migration

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

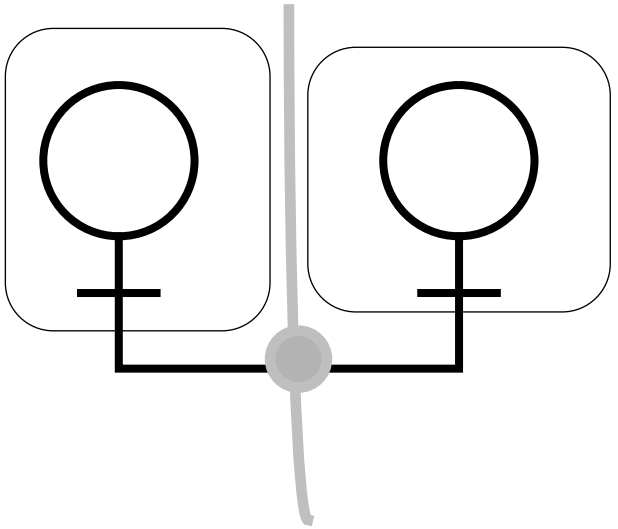


Migration Transparency

- ***Object migration needed:***
  - when a node fails, and its capsules have to be moved to another node
  - for load-balancing between capsules
- ***Like a stronger form of location transparency***
  - relies on location transparency mechanism

Selective Transparency Engineering - Federation

- **Federation Transparency**
  - **application need not know where administration boundaries are**



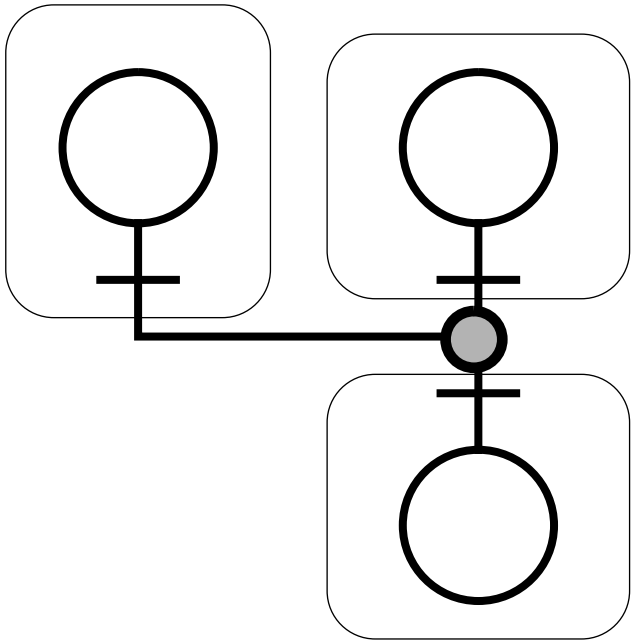
  - **interception may happen at the boundary, but this is not visible to the application**

Federation Transparency

- ***Federation is an Enterprise issue***
  - **there are many different kinds of federation boundaries: administration, organizational, contractual, and so on**
  - **constructing the transparency requires Enterprise knowledge**
- ***Federation is an ANSA research area***
  - **how it relates to trading**
  - **part of ANSA Phase III**

Selective Transparency Engineering - Replication

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



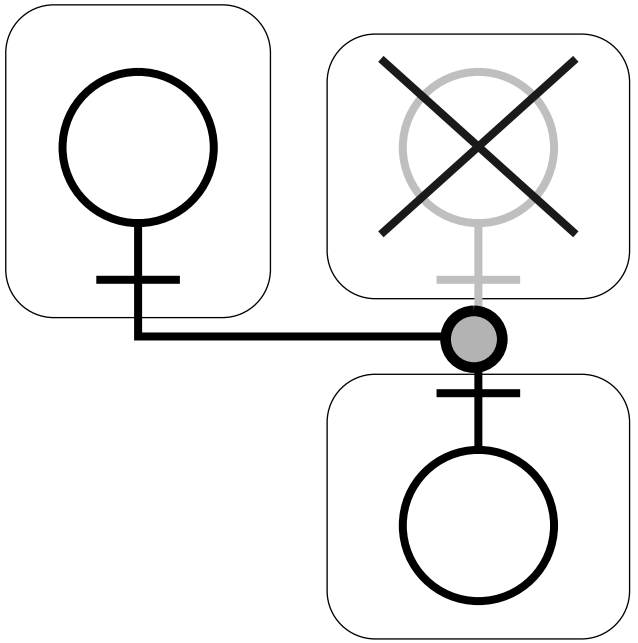
  - **application only sees a single interface**

Replication Transparency

- ***Server objects are members of a group***
  - **do not confuse with a cluster!**
- ***Replication transparency uses special mechanisms to make sure the group members are consistent***
  - **for instance, it may use multi-point channels and special protocols**
- ***Implementing replication transparency efficiently is difficult***
  - **it may need information from the application**
  - **it is under active research in the distributed systems community**

Selective Transparency Engineering - Failure

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



  - **may use replication transparency to achieve this**

Other transparencies

- ***Security***
  - application need not be aware of security policy
- ***Concurrency***
  - application need not be aware of other concurrent operations
- ***Transaction***
  - applications need not be aware of inconsistent states

Computational and Engineering Models - comparison

- ***Computational Model is for specifying interfaces for distribution***
  - **interfaces consist of operations**
  - **operations have (multiple alternative) terminations**
  - **everything has a type**
- ***Engineering Model is the infrastructure for Computational Model***
  - **encapsulation using capsules, clusters, and nodes**
  - **communication using channels**
  - **concurrency using threads**
  - **transparency mechanisms**
- ***The Computational Model is pure...***
- ***... the Engineering Model supports trade-offs***

Summary

- ***This has described the Engineering Model***
  - **ANSAware implements some, but not all of the Engineering Model...**
  - **... and not always as described in the Engineering Model**
- ***For more information:***
  - **on transparency mechanisms, see *The Challenge of ODP* (TR.033.02)**
  - **on replication transparency and groups, see *A Model for Interface Groups* (AR.002.01)**
  - **on federation, see *The ANSA Model for Trading and Federation* (AR.005.00)**
  - **on streams, see *Integrating Multimedia into the ANSA Architecture* (TR.028.00)**