Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (0223) 323010**
INTERNATIONAL: **+44 223 323010**
FAX: **+44 223 359779**
E-MAIL: **apm@ansa.co.uk**

Training

Building Distributed Applications with ANSA

Chris Mayers

Abstract

A one-day training course providing a technical introduction to ANSA. Originally presented to GPT Video Systems

APM.1097.00.02 **Draft** 20 May 1994

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

Building Distributed Applications with ANSA

Building Distributed Applications with ANSA

Chris Mayers

APM.1097.00.02

20 May 1994

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(0223) 323010
INTERNATIONAL	+44 223 323010
FAX	+44 223 359779
E-MAIL	apm@ansa.co.uk

Building Distributed Applications with ANSA

Chris Mayers, APM Business Unit

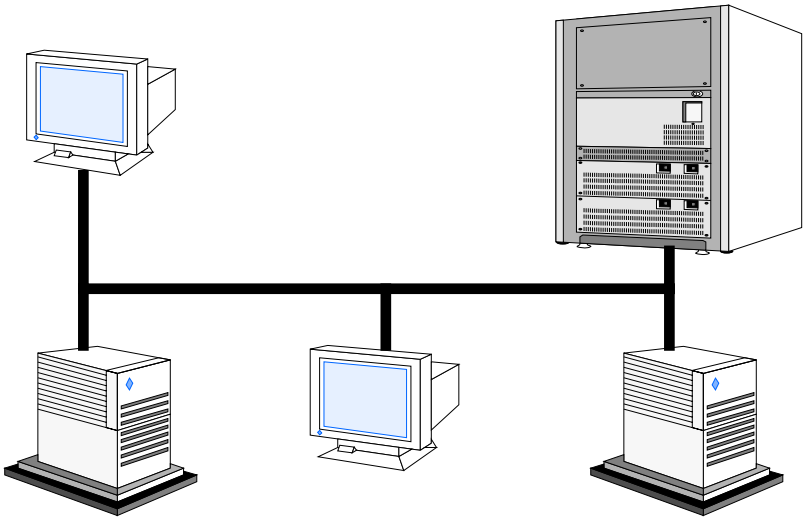
Sessions today

- **1. Introduction to Distributed Systems**
- **2. ANSA Fundamentals**

- **3. ANSA Computational and Engineering Models**
- **4. Introduction to ANSAware**
- **5. Round-up**

1.

Introduction to Distributed Systems



What's wrong with that picture?

What it leaves out...

- **People**
- **Information**
- **Applications**
- **... the business context**

Applications of distributed processing

- **Diverse business areas**
  - **Telecommunications**
  - **Airline reservations**
  - **Retail point-of-sale**
  - **Banking**
  - **Command and control**
  - **... and many more**
- **Built at the limits of the technology**

Distributed systems are fundamentally different (1)

- **Separation**
  - **remoteness**
  - **migration**
  - **no shared memory**
  - **partial failure**
  - **weak global consistency**

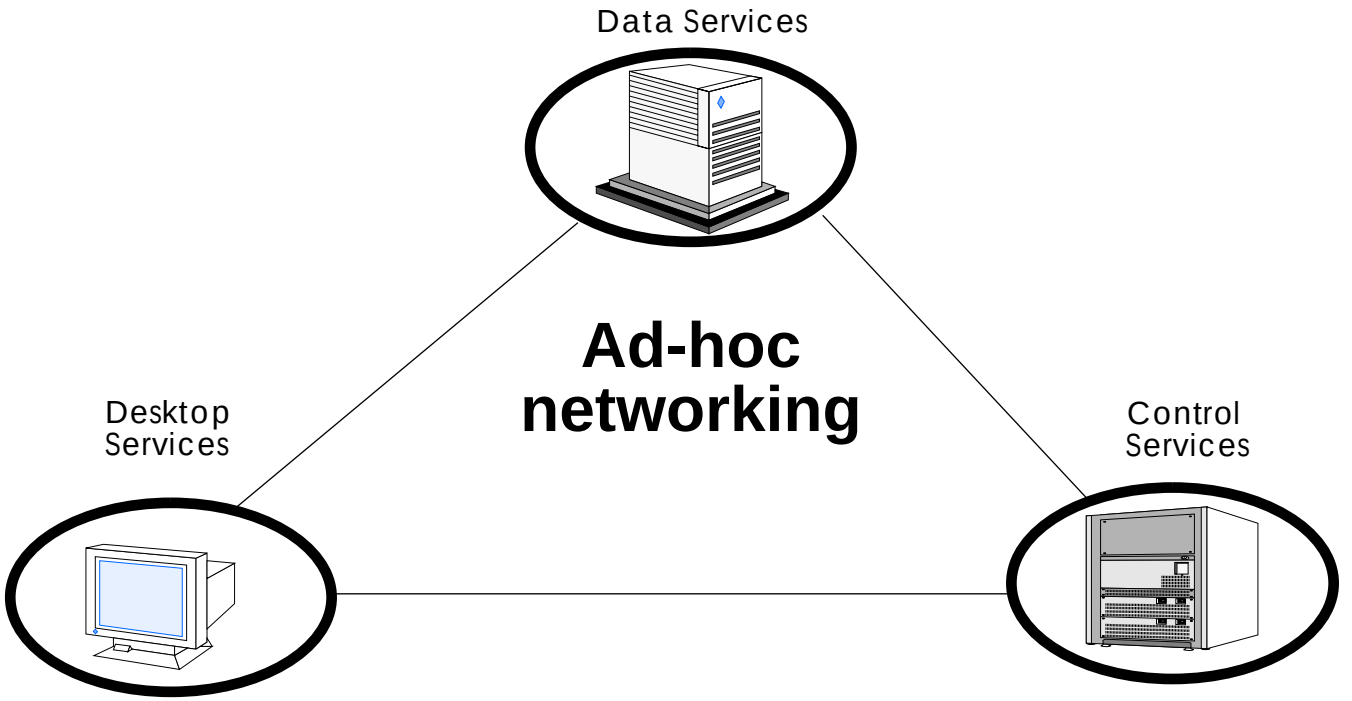
Distributed systems are fundamentally different (2)

- **Diversity**
  - **diversity of scale**
  - **diverse data representations**
  - **diverse naming schemes**
  - **diverse hardware and software**
  - **diverse communications mechanisms**

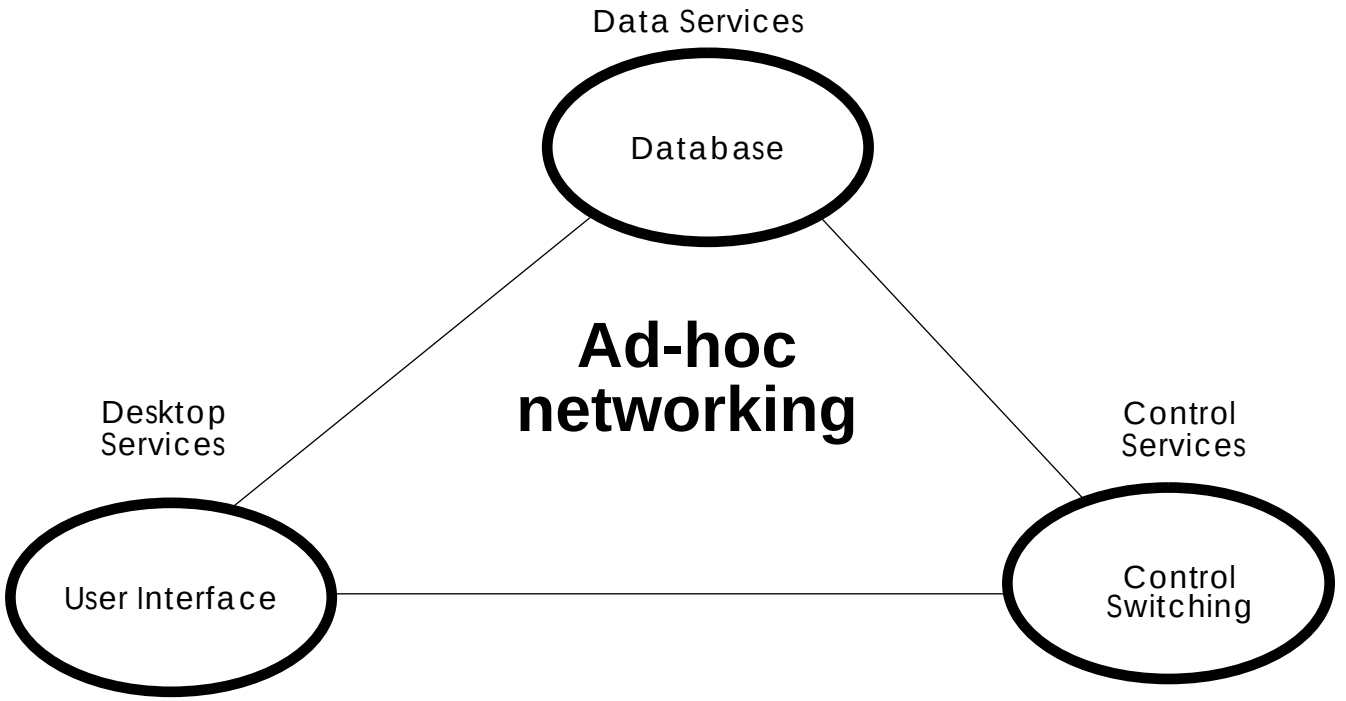
Distributed systems are fundamentally different (3)

- **Federalism**
  - **no central authority**
- **Concurrency**
  - **simultaneous operation**
  - **multiple copies**

How distributed applications are built now



Typical skills needed to build them



Skill cultures

- **Data Culture**
  - Remote data access
  - Distributed databases
  - Stored procedures
  - Object repositories
- **Desktop Culture**
  - Individual PC productivity services
  - File and printer sharing
  - Group productivity services
  - Mobile computers, universal personal digital communication
- **Control Culture**
  - Device control
  - Workflow
  - Robust messaging
  - Intelligent networking

Possible solutions on offer?

- **Client-server**
- **Object-orientation**
- **Open systems**
- **Rightsizing**
- **... no single approach or technology will dominate**

These are not solutions, but they are useful

Different policies for different applications

- ***Availability*** **versus** ***Consistency***
- ***Autonomy*** **versus** ***Uniformity***
- ***Security*** **versus** ***Convenience***
- **... and many other unavoidable trade-offs**

The challenge

- **Provide a framework for systems that:**
  - **integrate products from many vendors**
  - **are owned and managed by many organizations**
  - **can grow larger than the international telephone network**
  - **can evolve gracefully**
  - **allow different kinds of applications to interwork**
  - **preserve the investment in existing technology**
  - **have lower development and operating costs**
- **... This framework is an *Architecture* for Open Distributed Processing**

Other demands on the Architecture

- Must be easy to use and understand
- Must be widely applicable
- Must be durable and long-term
- Must be practical and proven
- Must be vendor-neutral
- Must be backed by the authority of international standards

meets these demands

Its principles are:

- **Distributed systems have different properties to centralized systems**
- **Different applications need different solutions**
- **Object-orientation simplifies designs**
- **Unnecessary complexity should be masked from the applications**

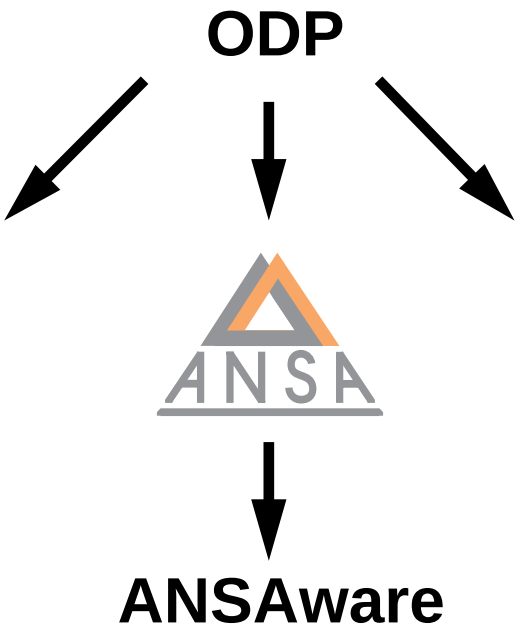
The Architecture embraces...

- **Components**
  - **standard functional building blocks**
  - **tools**
- **Rules**
  - **embodying principles and assumptions**
  - **deliberately prescriptive; to be enforced!**
- **Recipes**
  - **for satisfying commonly-occurring requirements**
- **Guidelines**
  - **for making design choices and trade-offs**
- **Concepts**
  - **clearly defined and delineated**
  - **as general as possible, and as few as possible**

The Architecture does not embrace...

- **Product recommendations**
- **Specific standards for product selection**
- **A complete operating system specification**
- **A software development methodology**
- **The user interface**

The Architecture in context



How do these fit together?

- **ODP - The Reference Model for Open Distributed Processing**
  - **The International Standard**
- **ANSA - The Architecture**
  - **One possible architecture for ODP**
  - **Other architectures may add extra rules, perhaps specific to an industry sector**
- **ANSAware - The Toolkit**
  - **One implementation of ANSA**
  - **Does not implement all the latest features of the architecture**

ODP - The International Standard

- **Is to be issued as ISO/IEC 10746, ITU-TS (CCITT) X.901-X.904**
- **Is very closely aligned with ANSA concepts and terminology**
- **Like ANSA, is a framework into which you slot existing standards**
- **Tackles the issue of conformance**

Background to ANSA

- **1985-1988: the ANSA project**
  - **Phase I sponsored under the UK Alvey programme**
  - **Focus on development of the architecture**
- **1989-1993: the ISA project**
  - **Phase II sponsored under the EC Esprit programme**
  - **Including partners from Austria, France, Germany, Greece, Holland, Italy, Sweden**
  - **Focus on completing the architecture**
  - **Example implementation: ANSAware**
- **1993 - ...: ANSA Phase III**
  - **Phase III sponsored by project partners**
  - **Focus on specific topics**

ANSA Phase III Topics

- **Federation**
- **Dependability**
- **Real-time**
- **Tool support**

Systems built with ANSA

- NASA Astrophysics Data System (ADS) - a distributed multi-database
- GESI Distributed Healthcare Patient Management System
- CTI Distributed Newspaper “Hypertext” Composition
- AEG OSI resilient network management overlay

Distributed systems - summary

- **Distributed systems have fundamentally different properties**
- **ODP (Open Distributed Processing) is the standard**
- **ANSA is the Architecture**
- **ANSA Phase III is an ongoing research project**
- **ANSAware is the toolkit**

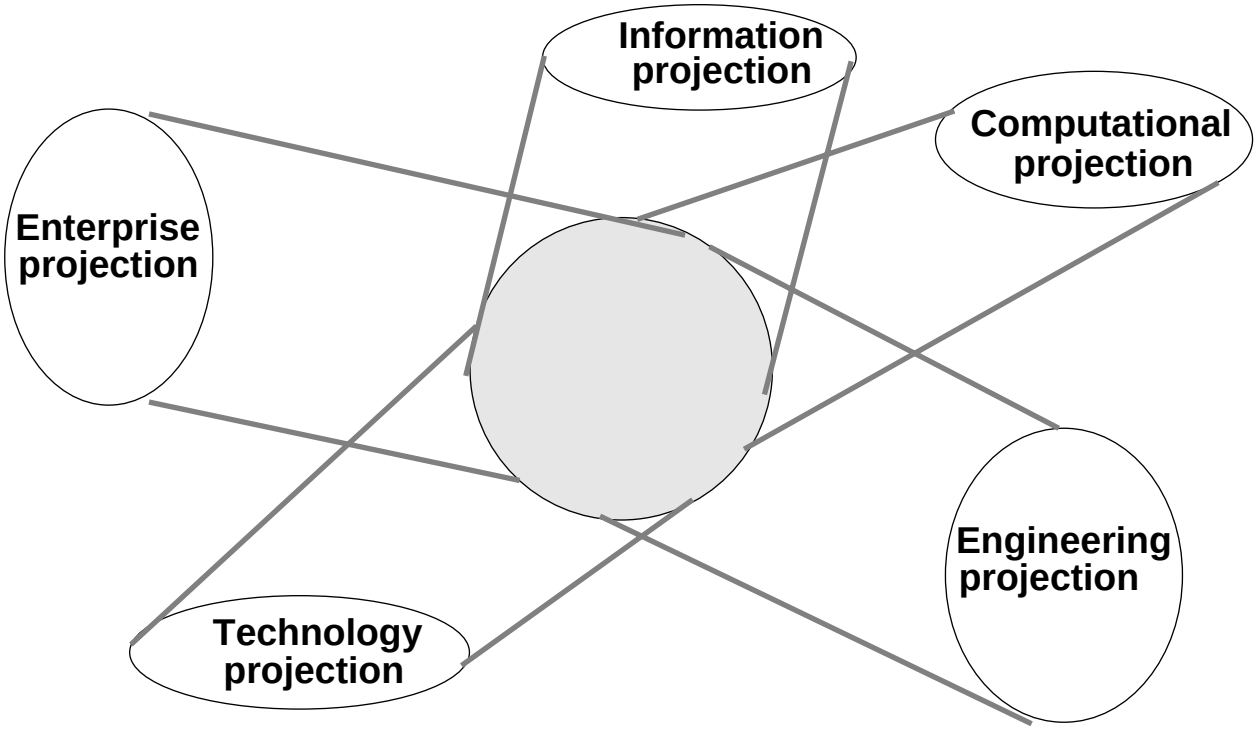
2.

Fundamentals of

The Architecture

ANSA uses 5 different viewpoints(*projections*)

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



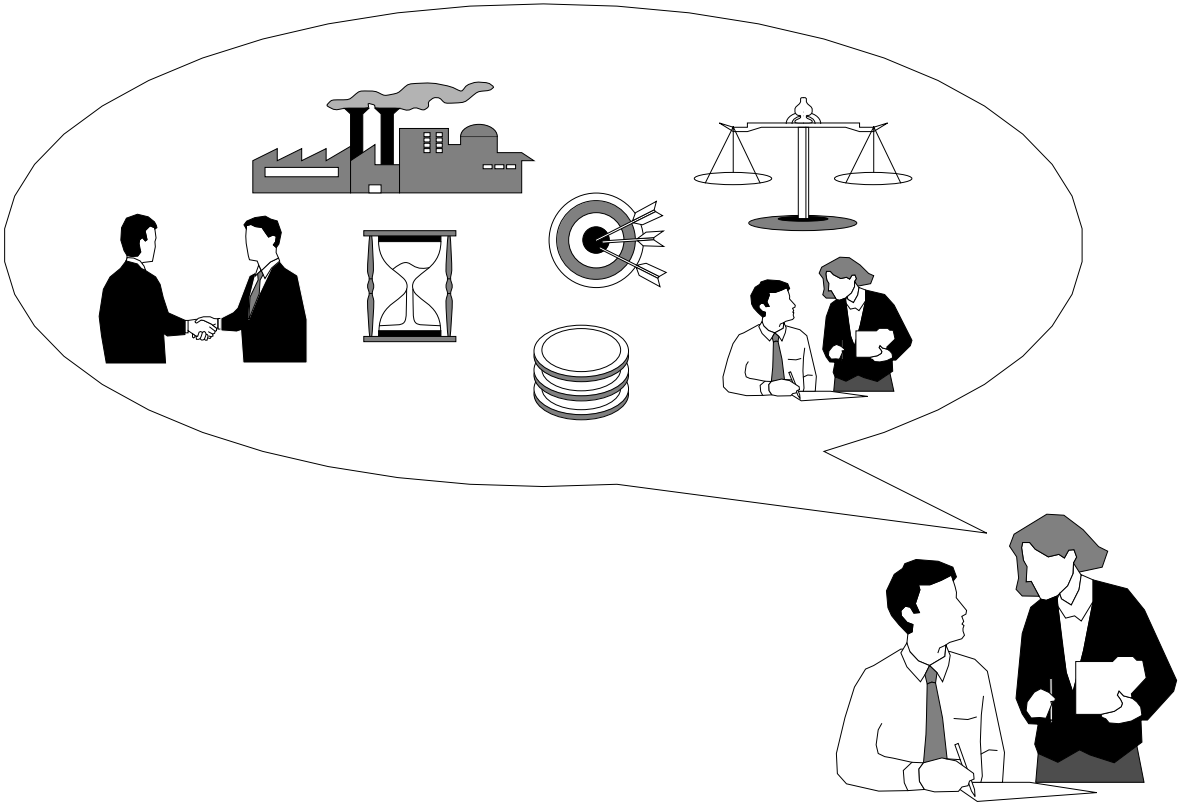
Content of the 5 projections

- ***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,...**

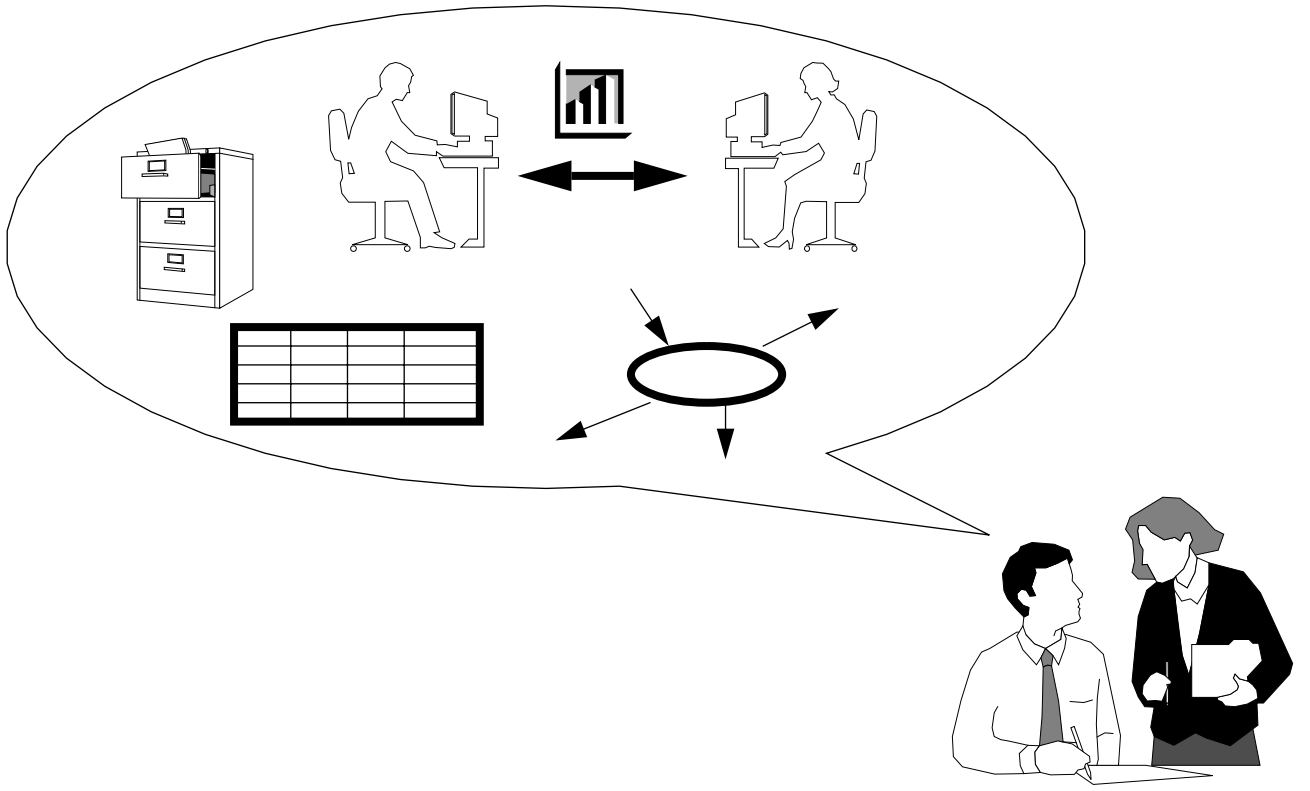
The projections reflect the different concerns of the different stakeholders

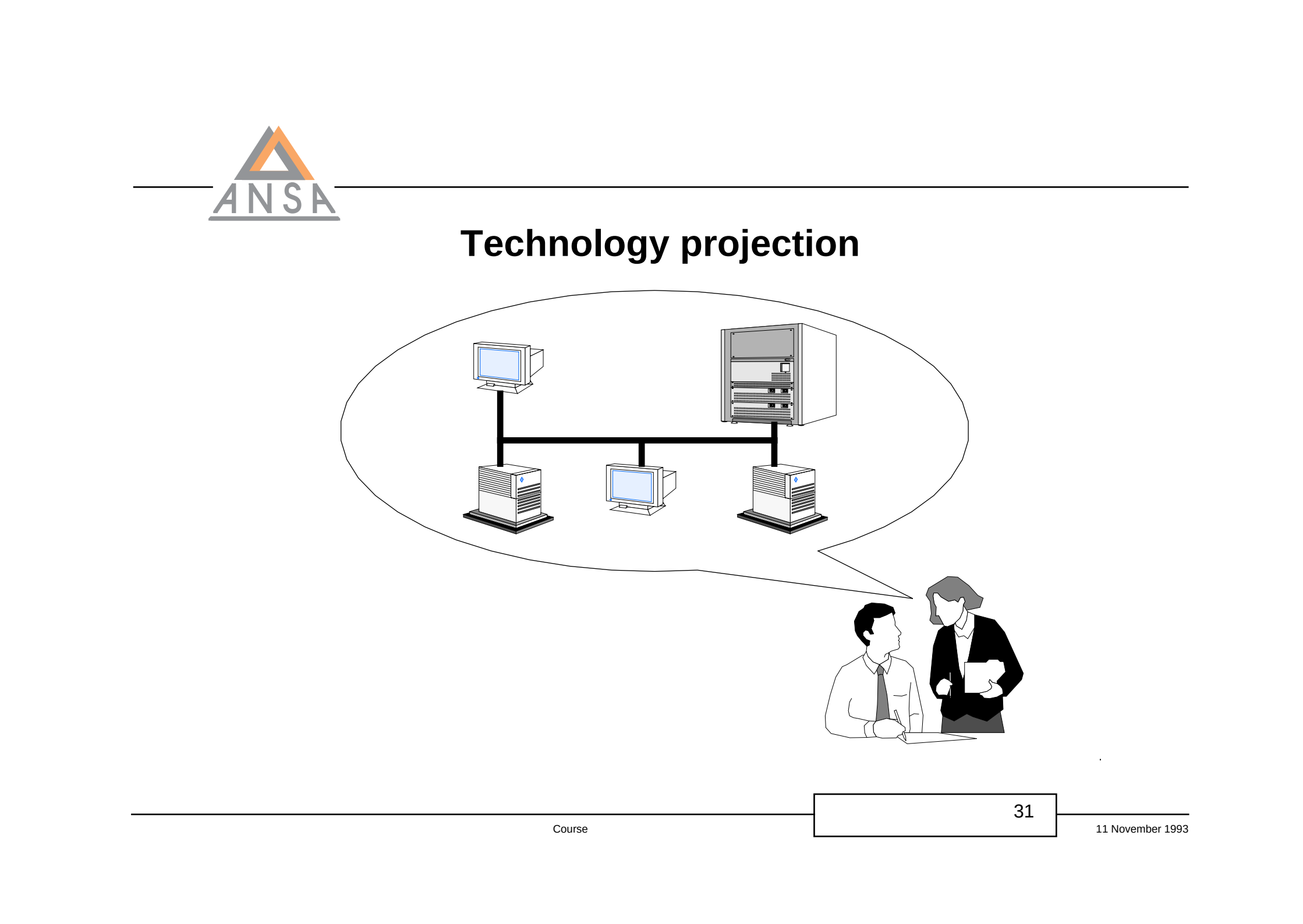
This course focuses on the Computational and Engineering projections

Enterprise projection



Information projection





Technology projection

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*

Quality of Service

- Alternative providers may offer the same service functionality
- ... but with different Quality of Service (QoS), e.g.:
  - Cost
  - Throughput
  - Availability
- Need to choose a service that matches both specification and QoS
- In a distributed system, this must be done at run-time
  - providers may come and go
  - users may come and go

Trading

- **Provider of a service posts details of the offer with a *Trader***
- **Potential user asks Trader for list of matching offers**
- **User selects ‘best’ offer, and talks directly to provider**
- **Trader then no longer involved in interaction**
- **User is the *client*, provider is the *server***
- **... but client and server are only roles in this interaction**
- **Traders are key components**

Distributed systems are different

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

Traditional	*Reversed*
Local	Remote
Sequential	Concurrent
Single 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

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 projections

- **Isolate specification of transparencies from their design**
  - **Computational projection defines the transparencies**
  - **Engineering projection 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**

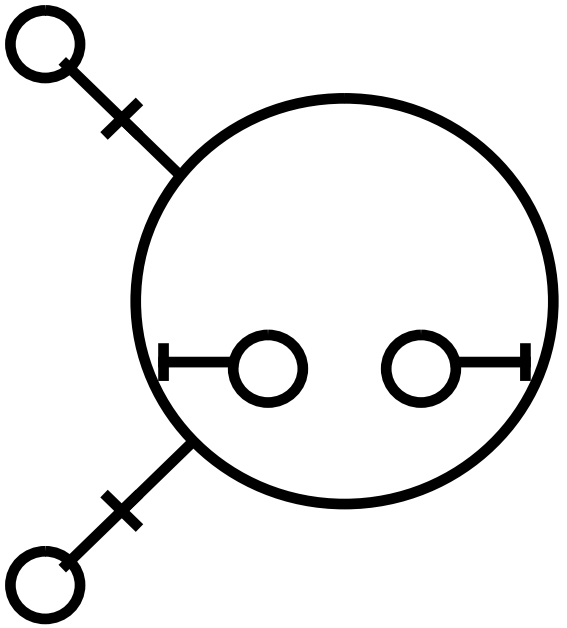
ANSA Fundamentals - summary

- **5 projections: Enterprise, Information, Computational, Engineering, Technology**
- **Services are encapsulated**
- **Traders hold the details of service offers, for clients to choose servers**
- **Client and server are roles, not hardware or software products**
- **Transparency mechanisms simplify the construction process**
- **For more on ANSA Fundamentals, see *An Overview of ANSA* (AR.000.00)**
- **For more on transparency mechanisms, see *The Challenge of ODP* (TR.033.02)**

3.

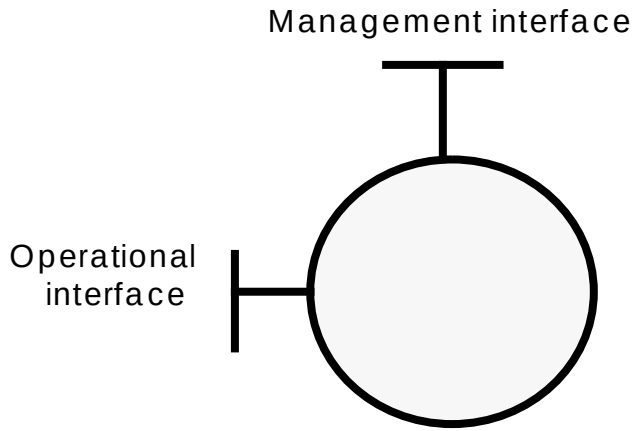
The ANSA Computational and Engineering Models

The ANSA Computational Model



Basic concepts - Objects and Interfaces

- **Objects encapsulate services**
- **Objects interact only via interfaces**
- **Objects can have multiple interfaces**
- **Objects have the full responsibility for providing the specified service**



Interfaces

- **Interface defines all valid interactions**
- **Interfaces are defined in an interface definition language (IDL)**
- **Interfaces consist of *operations***
- **Interfaces are identified by *interface references*; these can be passed between objects**
- **More than one client can use the same interface concurrently**

Operations and Terminations

- **An *operation* is the unit of interaction. It has:**
  - **a name**
  - **a signature (list of input arguments)**
  - **a set of *terminations* (possible results)**
- **A termination has:**
  - **a name**
  - **a signature (list of output arguments)**
- **Terminations make operations more powerful than ‘functions’ or ‘methods’**
- **Each operation embodies a contract**
  - **Server must handle all calls of operations**
  - **Client must handle all resulting terminations**

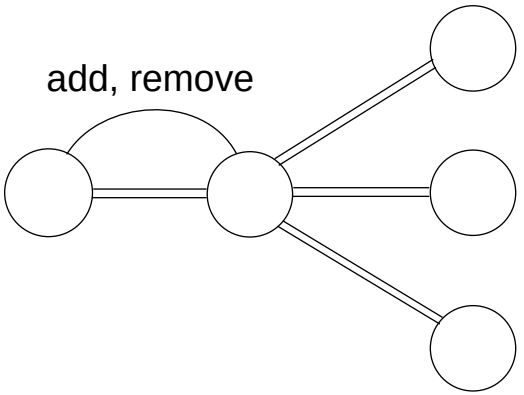
Types and Bindings

- **Arguments in signatures have the usual basic *types***
  - **boolean, integer, string, ...**
- **Operation type is defined by its name, signature, and terminations**
- **Interface type is defined by its operations**
- **A *binding* is established at an interface between a client and server**
- **Interface types are checked when a binding is established**
  - **client and server must match as subtypes**
  - **extra rules for streams (directions must match)**

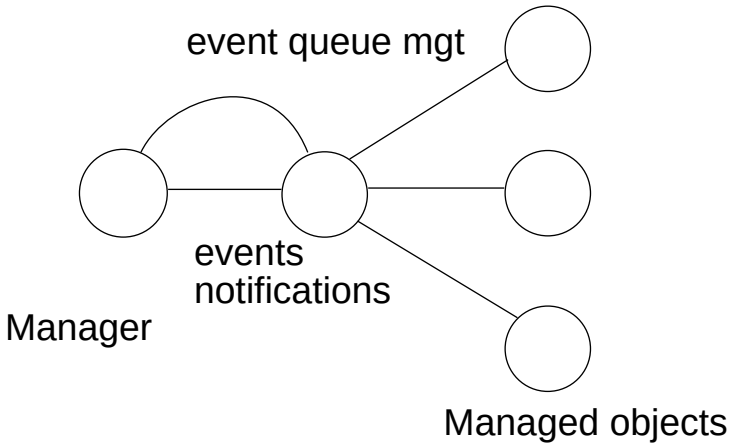
Bindings

- ***Implicit*** **binding for operational interfaces**
  - **Resources will be automatically allocated**
- ***Explicit*** **binding for all interfaces**
  - **For explicit control over resources**
  - **To form groups**
  - **To bind streams**
- **A binding is itself a object with a control interface for**
  - **adding/removing interfaces**
  - **stopping/starting information flows**
  - **changing quality of service**
  - **monitoring events**
- **The binding object does the necessary splitting, joining, filtering and control**
- **Explicit binding suits telecommunications and system management needs**

Examples of bindings



Conferencing



Management domain

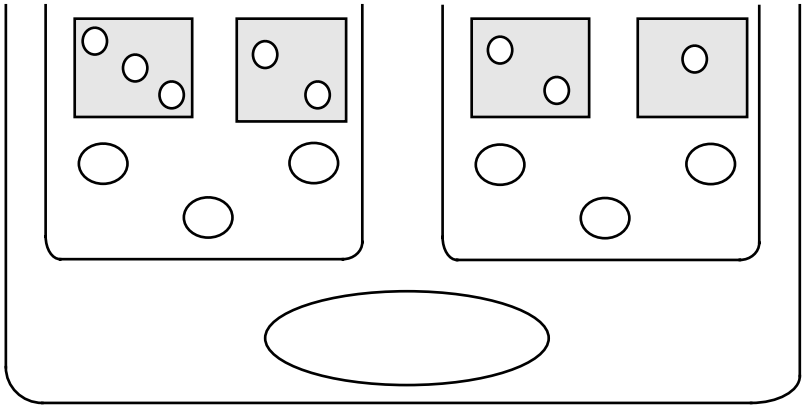
Trading

- **Each object has access to a *trading context* containing**
  - **service offers; type, properties, interface reference**
  - **links to other contexts; name, properties**
- **A service offer can be tied to an export policy**
  - **to monitor imports**
  - **to allocate resources**
- **A context may be optimized for**
  - **speed of look-up**
  - **number of offers stored**
  - **accuracy of offers**
  - **dependability**
  - **local knowledge; parts of the infrastructure can be treated as service offers**
- **No predefined naming structure for contexts; allows federation**

Trading and Federation

- **Trading is simply a particular kind of service**
- **Traders can be interconnected to provide federation**
- **Federation raises policy issues including**
  - **Authority**
  - **Billing**
  - **Security**

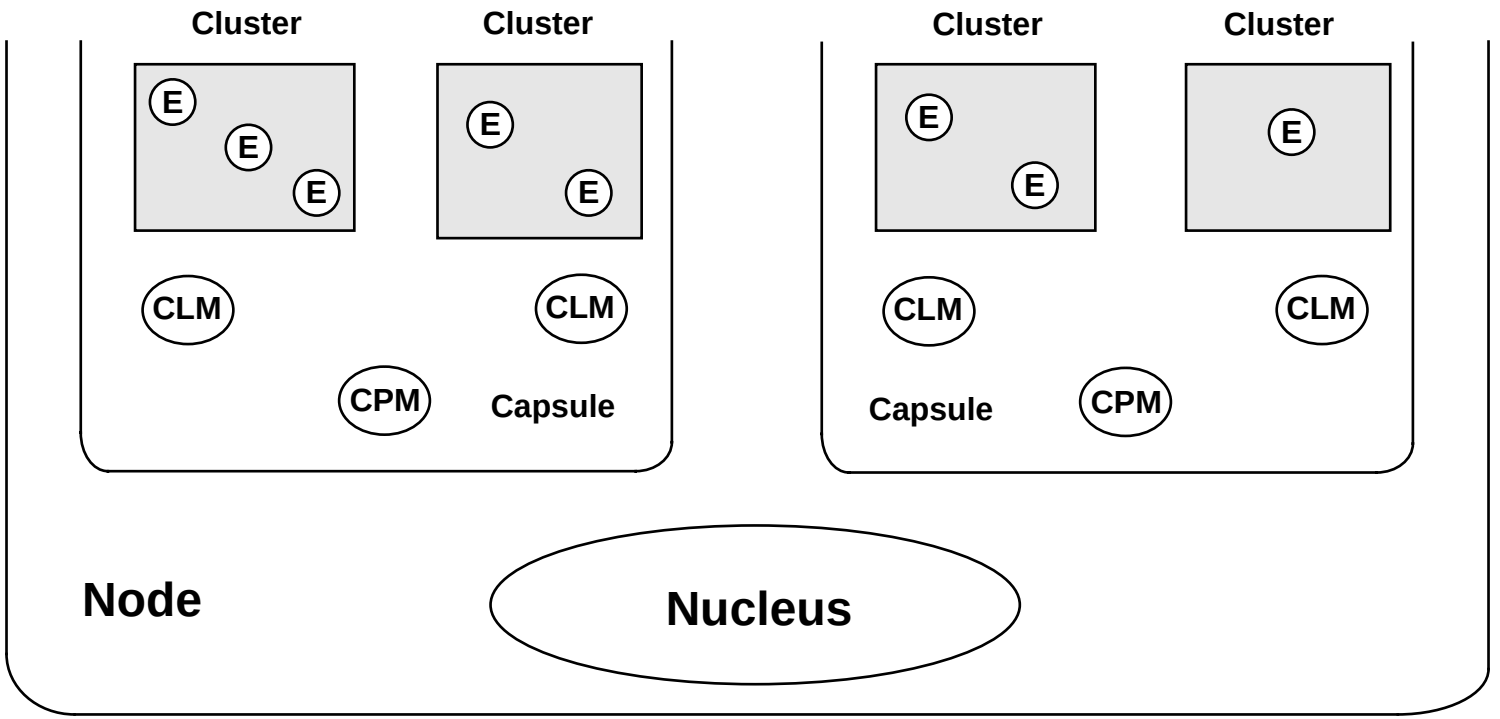
The ANSA Engineering Model

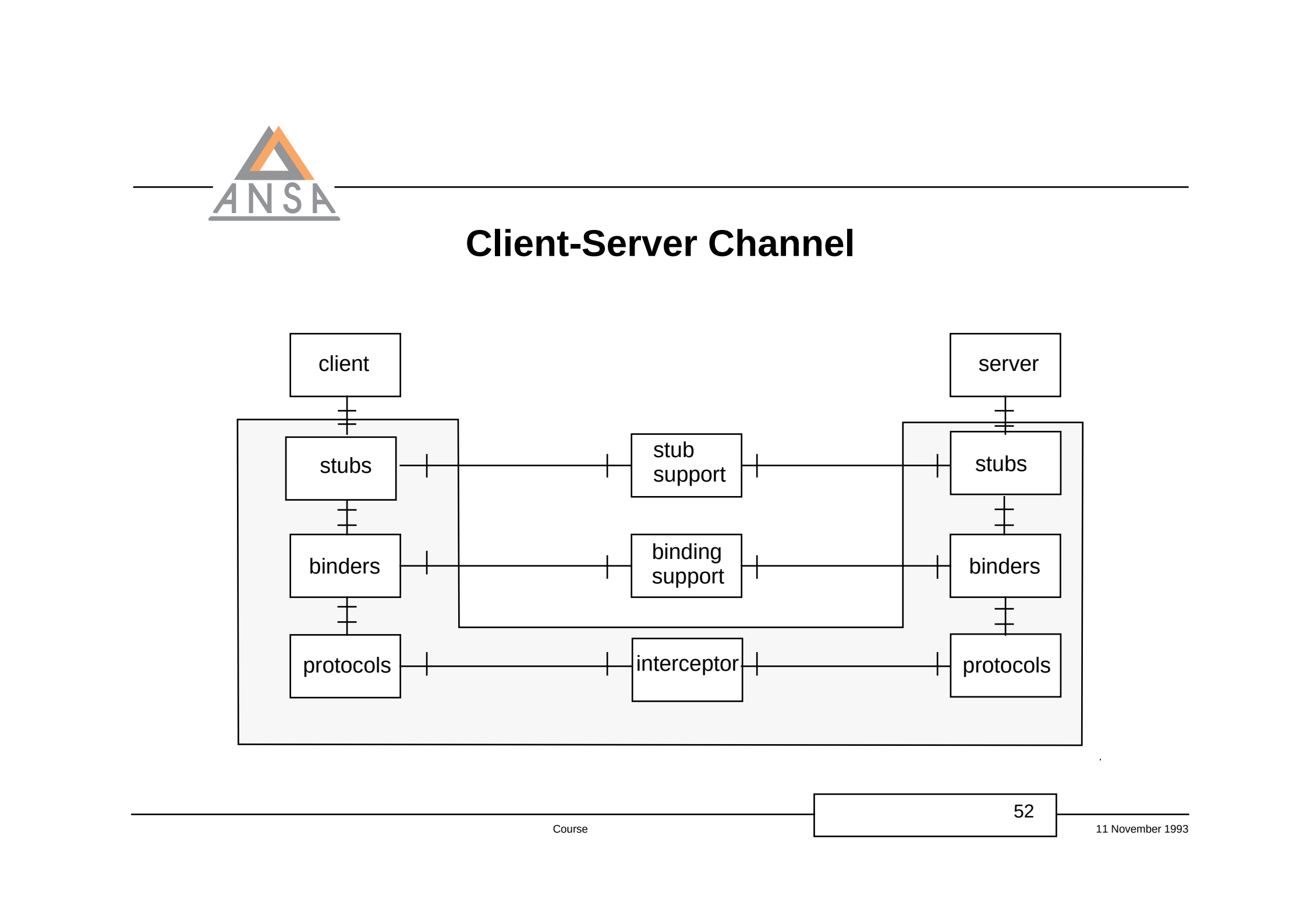


Engineering is about infrastructure

- **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 transparency mechanisms**

Engineering Encapsulation Infrastructure

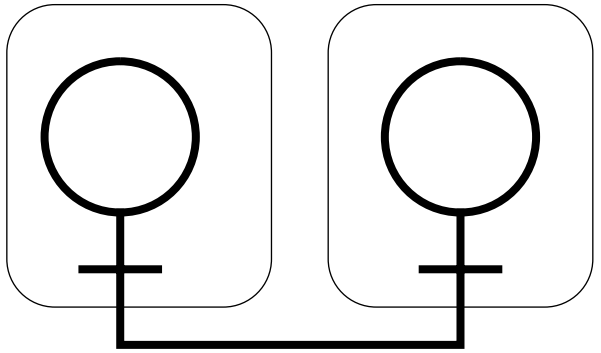




Client-Server Channel

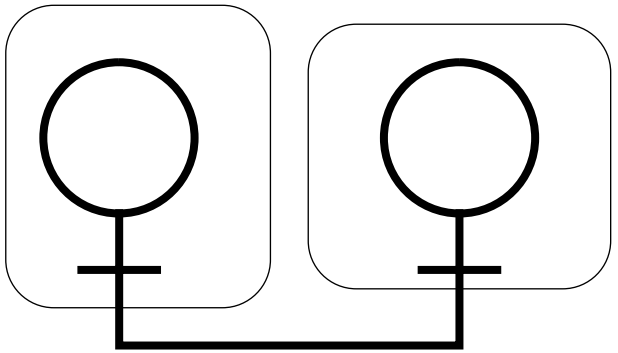
Selective Transparency Engineering - Location

- **Location Transparency**



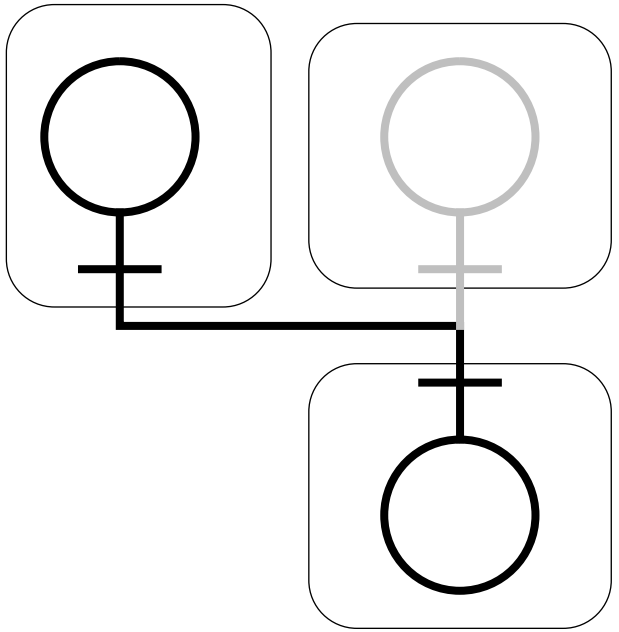
Selective Transparency Engineering - Access

- **Access Transparency**



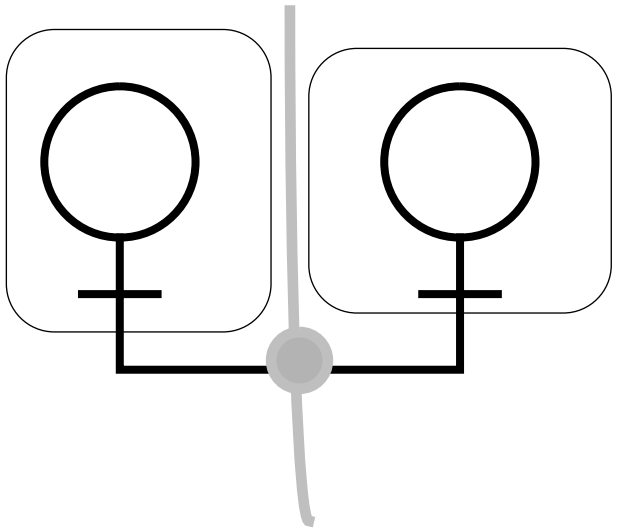
Selective Transparency Engineering - Migration

- **Migration Transparency**



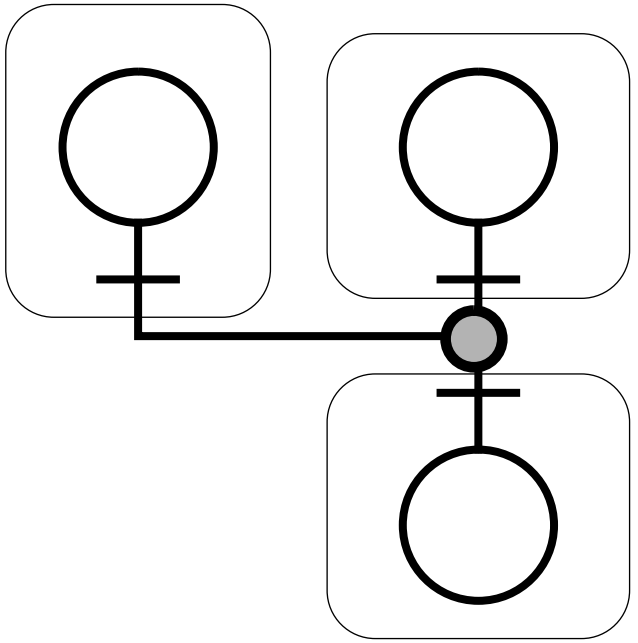
Selective Transparency Engineering - Federation

- **Federation Transparency**



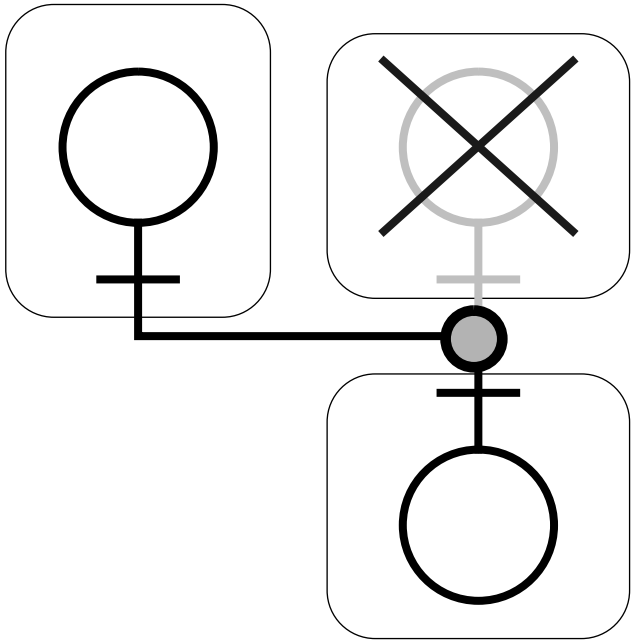
Selective Transparency Engineering - Replication

- **Replication Transparency**



Selective Transparency Engineering - Failure

- **Failure Transparency**

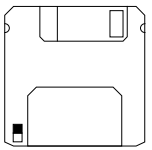


ANSA Computational and Engineering Models - summary

- **Computational Model hinges on encapsulation via interfaces**
  - **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**
  - **transparency mechanisms**

4.

Introduction to ANSAware



Distinctive ANSAware features

- Not a shrink-wrapped product - you can experiment with it
- Implements ANSA transparencies
- Supports concurrency
- Supports light-weight implementations
- Supports group execution
- Third-party extensions available
- Ported to many environments
- Based on long-term research

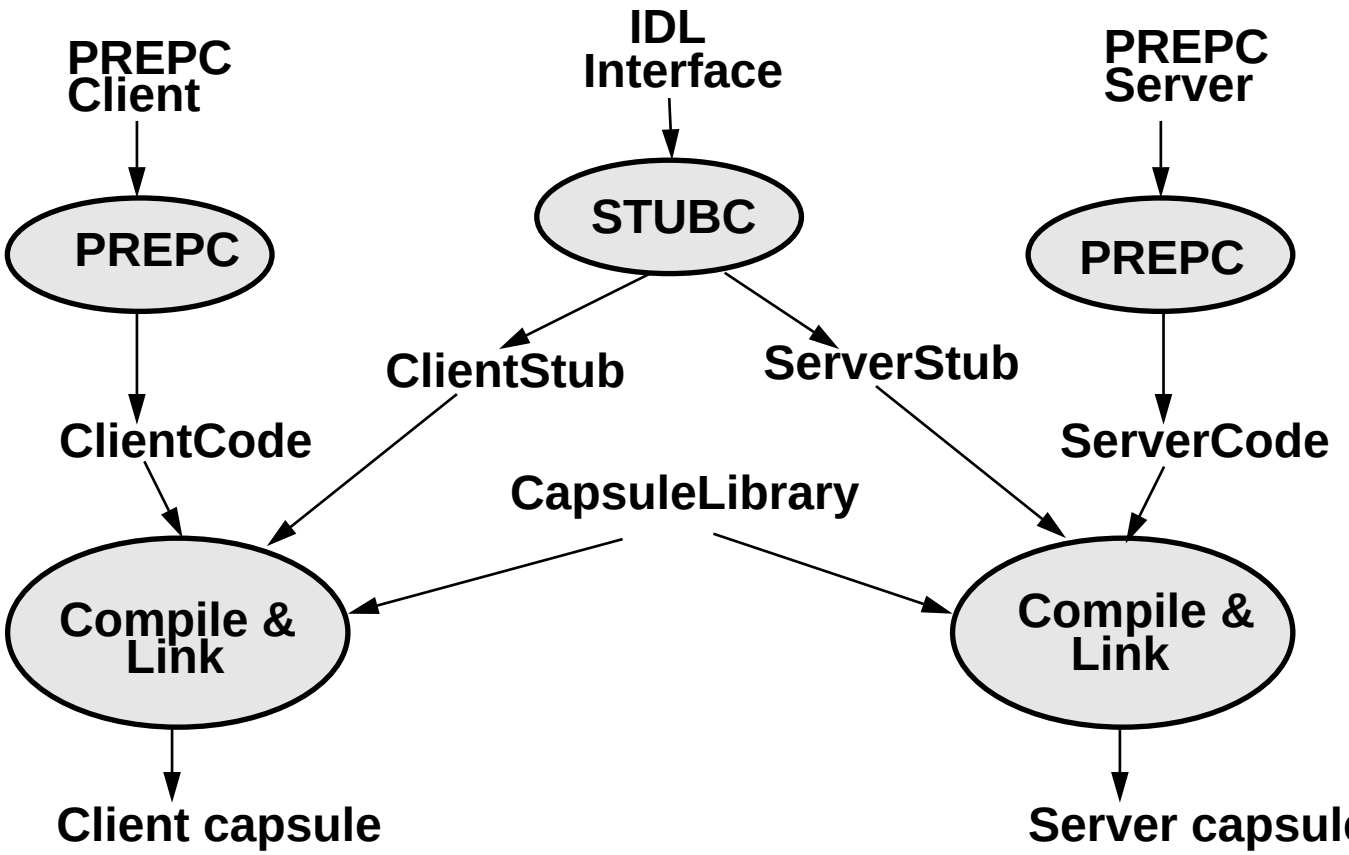
Content of ANSAware 4.1

- **Basic infrastructure support**
  - **known as the nucleus, capsule library, or run-time system**
  - **efficient, transport-independent and portable RPC protocol**
  - **light-weight threads**
  - **synchronisation operations**
  - **timer handling**
  - **support for interworking with other systems - e.g. X11**
- **Tools**
  - **STUBC: Compiler for Interface Definition Language (IDL)**
  - **PREPC: Preprocessor for IDL calls embedded in C**
- **And...**

Content of ANSAware 4.1 (2)

- **Services**
  - **Trader: provides service import/export matching**
  - **Factory: creates and destroys capsules**
  - **Node Manager: provides per-node service management**
- **Sample demonstration applications**
- **... all supplied as ANSI C source code**
  - **complying to POSIX 1003.1**

Component construction



IDL Example

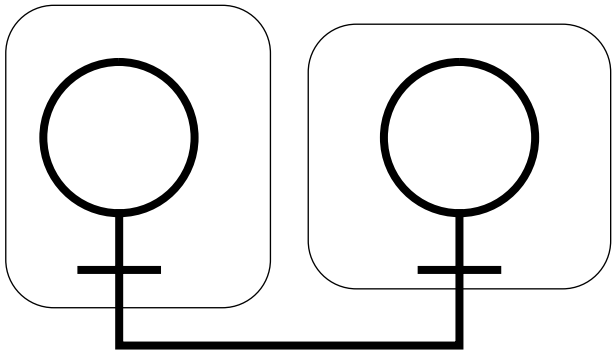
```
Echo : INTERFACE =

-- Comment lines start with two dashes

BEGIN

   Echo : OPERATION [ Src: STRING ] RETURNS [ STRING ];


   Reverse:OPERATION [ Src: STRING ]RETURNS [ Res: STRING ];


END.
```

IDL types

- **IDL provides a set of built-in concrete types:**
  - **BOOLEAN**
  - **[SHORT] CARDINAL**
  - **[SHORT] INTEGER**
  - **[LONG] REAL**
  - **OCTET (signed)**
  - **CHAR**
  - **ENUMERATION**
  - **STRING**
- **IDL provides concrete type constructors:**
  - **ARRAY OF Type (fixed size)**
  - **SEQUENCE OF Type (variable size)**
  - **RECORD [Types]**
  - **CHOICE (discriminated union)**

STUBC

- **“Stub Compiler”**
- **Generates stub code from IDL interface specification to**
  - **marshal/unmarshal IDL parameters**
  - **transmit/receive via the nucleus**
- **Stubs optimize calls within the same capsule**
- **In essence, implements the Access transparency**



PREPC

- **“Preprocessor for C”**
- **Clients and servers written in C with embedded PREPC statements**
- **PREPC provides**
  - **access to all services: i.e. the trader, factory, node manager, in addition to user written services**
  - **simple exception handling mechanism**
  - **creation/destruction of interface instances**
  - **creation/destruction of objects**
  - **some static type checking (e.g. number of arguments/results), remainder is left to the underlying C language**
- **PREPC generates calls to STUBC stubs**

PREPC Echo client

```
! USE Echo
! DECLARE { intRef } : Echo CLIENT
...
ansa_InterfaceRef intRef; /* or EchoRef intRef; */
...
/* import service */
! { intRef } <- traderRef$Import ( “Echo”, “/”, “”)
...
/* invoke operations */
! { obuf } <- ir$Echo(ibuf)
...
/* discard service */
! intRef$Discard
```

PREPC Echo server

```
! USE Echo
! DECLARE { intRef } : Echo SERVER
...
ansa_InterfaceRef intRef
body(int ac, char **argv) {
/* create & export interface-instance */
! {intRef} :: Echo$Create(16)
! {} <- traderRef$Export( “Echo”, “/ansa/testservices”, \
   “”, intRef )
...
/* operations invoked */
...
}
```

PREPC server operations

```
Interface_Operation (
_attr, /* always required */
args,  /* arguments */
results /* results (pointers) )
{
/*  do the operation */
return SuccessfulInvocation;
/* return UnSuccessfulInvocation; would indicate a failure */
}
```

Factory service

- **Provides a service for creating and destroying capsules**
- **It also:**
  - **monitors the continued existence of the capsules it has created and will notify an interested party when they terminate**
  - **provides an “IsAlive” operation to determine if a specified capsule is still alive**
- **And so avoids relying on:**
  - **a centralised notification/failure detection service**
  - **a local file system to log the existence of capsules**

Node Manager service

- **Provides a stable, per-node database of commonly used services**
- **Can be used to:**
  - **create new service instances**
  - **destroy existing services**
  - **dynamically create services on first use**
  - **implement simple resource management**

Supported platforms

- **Unix**
  - **HP/UX (various versions and platforms)**
  - **SunOS (Sun3 and Sun4)**
- **DOS/Windows**
  - **DOS**
  - **Windows 3.1 386 Enhanced Mode**
- **VMS**

Unsupported platforms

- **Some in the ANSAware distribution**
- **... others available on request**

Platform implications

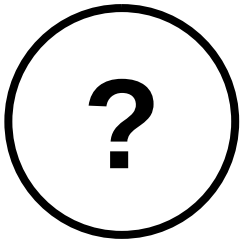
- **DOS/Windows: PC/TCP 2.05 and 2.2 are supported**
- **DOS has inevitable memory limitations**

ANSAware 4.1 - summary

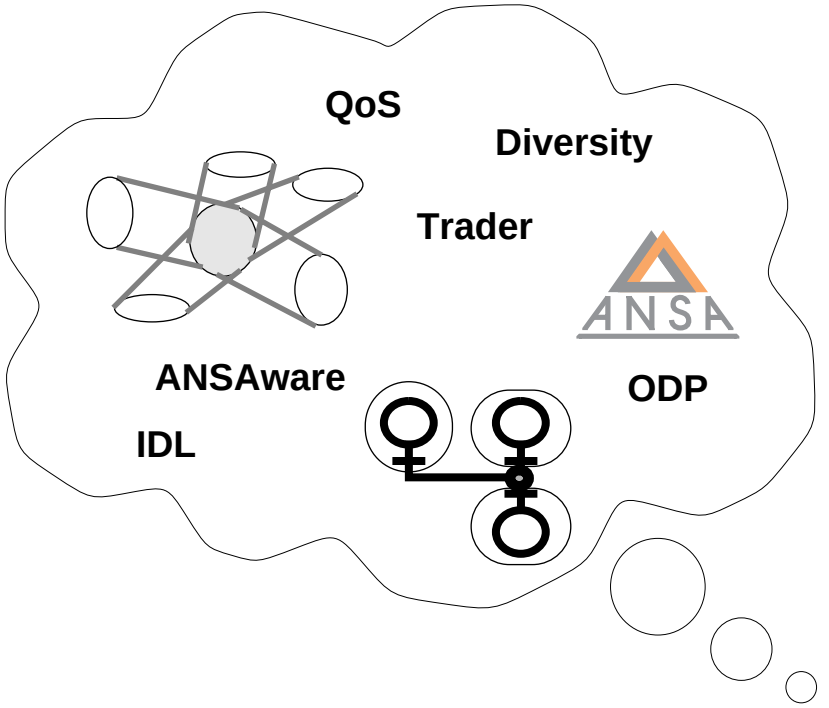
- **A light-weight, portable implementation of the ANSA Architecture**
- **Applications written in C with embedded ANSAware calls**
- **Tool support for ANSA transparencies**
- **For more detail, see *Application Programming in ANSAware* (ANSAware Volume B)**

5.

Distributed Systems Round-up



New Ideas



References

[ANSA 91]

ANSA: A Systems Designer's Introduction to the Architecture, APM Ltd., Cambridge U.K., April 1991.