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**

ANSA Phase III

Presentation of ANSA to New Sponsors

Andrew Herbert

Abstract

This presentation is design to last about 75 minutes and introduces ANSA to potential new sponsors. The focus is more on objectives and benefits, leavened by sufficient technical detail to add credibility when presented to an audience of engineers rather than managers.

APM.1057.00.03 **Draft** 23 September 1993

Briefing Note

Distribution:
Supersedes:
Superseded by:

ANSA

Andrew Herbert

ANSA Chief Architect

Technical Director
Architecture Projects Management Limited

What is ANSA

- **A collaborative industry effort to advance distributed systems technology**
- **Based on a shared architectural vision**
- **Vendor neutral**
- **Contributes to standards**
- **Produces advanced technology prototypes**
- **Enables its sponsors to deliver more effective products and services**
- **Strong focus on computer and telecommunications service integration**

Contents

- **What problem are we trying to solve?**
- **What is the ANSA architecture?**
- **Where has ANSA had an impact?**
- **How does the ANSA consortium work?**
- **What is ANSA doing now?**

What Problem are we Trying to Solve?

The problems, as seen by users

- **Incompatible systems**

Because of technical diversity of hardware, operating systems (400 Unixes), comms, programming languages, databases, guis - explosion of combinations - compatibility = hard work

- **Protecting legacy systems**

They work - need them - no business case for replacement - but legacy systems. don't support integration

- **Scaling up**

geographic - time lags - time zones - span several administrations; network size - global reboots or updates of s/w not possible; requires evolution

- **Scaling down**

computing is moving out of main-frame into pcs and workstations, how to exploit tiny computers?

- **Rate of change of technology**

Technology changes annually - changing is painful, expensive, inconvenient - not changing leaves competitors ahead - if change some, how to integrate new with old

- **Responding to organizational change**

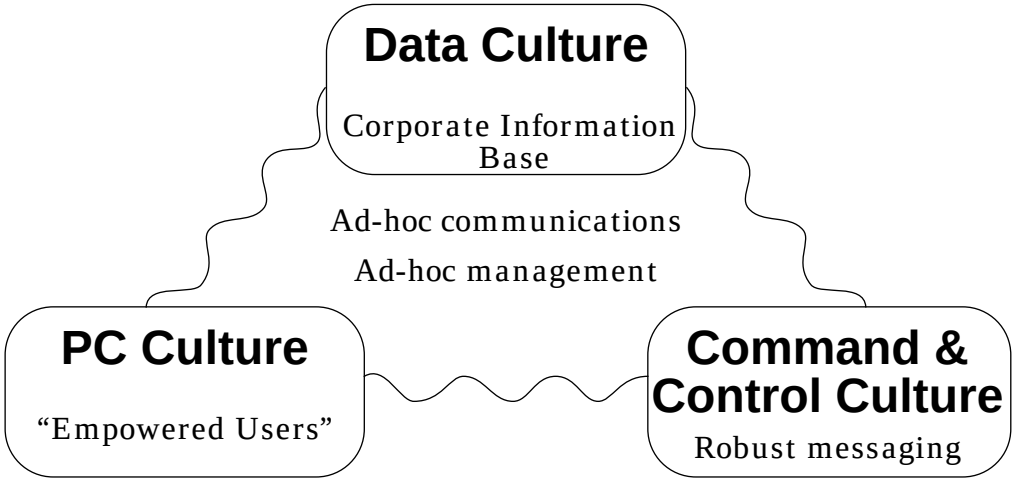
enterprise controls system / system controls enterprise? IT forces working style, inhibits change. Mergers etc. => force IT to change, but IT can't change as fast as Enterprise can. Building societies example.

- **Manageability**

Commercial pressure => fastest way to update/fix s/w. Continual patching => "structural fatigue"; can become impossible to meet change requirements (can do but not in time)

Distributed Computing Cultures

- **Three important and quite separate distributed computing cultures exist**

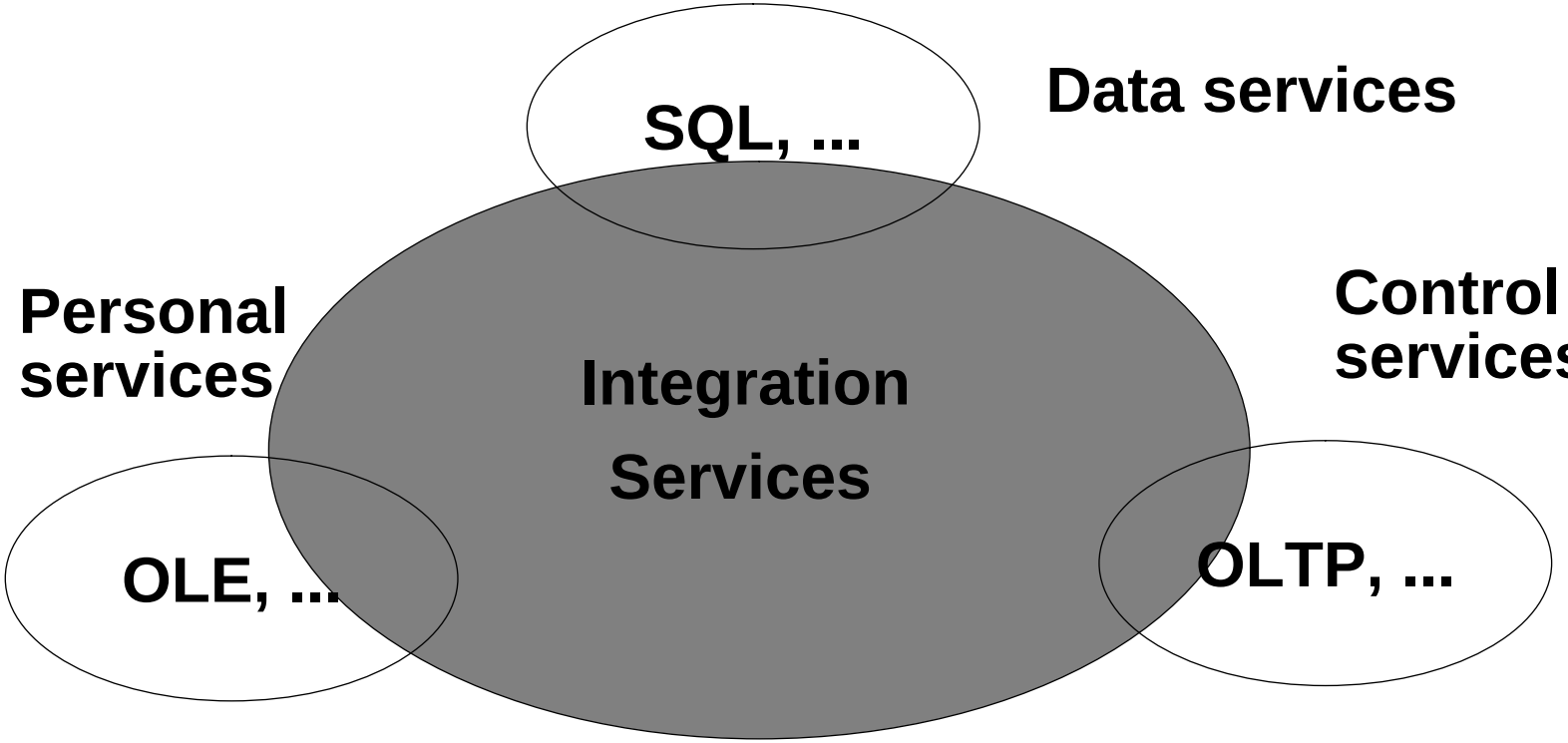


- **Each has its own techniques for distribution and application integration**
- **They meet different needs - no single one will absorb the others**
- **Legacy of existing technology choices and applications will persist**
- **Current focus on Data <-> PC axis is too narrow**

Current technology

- **Data Culture**
  - **Remote data access**
  - **Federated databases**
  - **Stored procedures**
  - **Object repositories**
- **PC Culture**
  - **Individual productivity services (based on OLE, DDE etc.)**
  - **File and printer sharing**
  - **Group productivity services**
  - **Mobile computers, universal personal digital communication**
- **Control Culture**
  - **On-line transaction processing**
  - **Workflow**
  - **robust messaging**
  - **Intelligent networking**

A Service Orientated View



Integration will not only have to link the other kinds of services, but be able to penetrate their domains

The Wish List

- **Integrate products from many vendors**

Exploit innovation, price / performance trade-offs

- **Span application domains**

Information naturally flows from one dept to another; needs to do so economically and automatically

- **Hide system boundaries**

Size should only affect performance, but not functionality

- **Protect enterprise boundaries**

To preserve autonomy and enable flexibility

- **Enable inter-organisation computing**

Controlled federation - don't want to expose everything; have to resolve policy conflicts

- **Preserve existing investments**

Commercially sensible, requires an environment which facilitates evolutionary change

- **Match IT style to Enterprise requirements**

Enterprise should drive the system, IT should respond to (facilitate) change. Can't rely on a single administrator

- **Allow rapid, low-risk adoption of new technology**

No-one wants to be first, no-one can risk being last....

- **Be manageable**

You need to know what's out there and who's using it,....

Reaching the Solution

- **Specify systems using application concepts**

then they will be durable against technical change and comprehensible to applications programmers. The technical term here is abstraction - this is a good thing in computing; consider 4GLs which abstracted away from DB access methods and COBOL, structured programmed which weaned us off assembler,...

- **Define a robust, high level model for distributed programming**

Agree what programmers can and can't do - semantics are important, not syntax; limit the number of options to increase the level of abstraction and enhance portability

- **Use tools to automatically generate the engineering detail**

because its cheaper than writing it and it doesn't have to be maintained; you can even afford to throw it away! Follow the trend towards ever more "declarative' input (i.e. say what you want, not how to do it)

- **Define structures for integration**
  - **management and monitoring**

What is going to be controllable in the system, where is it controlled from?

  - **trading and federation**

How is my system linked to yours - what services do we provide to one other - what "adaptors" do we need?

  - **transparency**

what is going to be hidden (by automation)?

- **Define models for common application domains**

so we can talk to each other - to get here we need to start from some common "describing languages"

- **In other words, an architecture**

What is an Architecture?

- **Architecture is.....**

not a laundry list, a more general notion of a set of standard ways of doing things to simplify integration and remove unnecessary differences between systems: similar to the notion of “production engineering” in manufacturing. Ultimately should link into user’s application design process.

- **A set of basic components**

For us, the services we need to do integration; Services defined by their interfaces, not their technology. Goal is smallest set that can be combined to meet all requirements and hence depends upon abstraction to reduce the set of choices to the smallest NECESSARY set.

- **Design rules**

Ways to structure and combine systems so as not to fall foul of key problems like scaling, and other traps.

- **Recipes**

Ways to build components into more complex subsystems which behave in the way you want - a cookbook for designers

- **Guidelines**

When to break the rules and what the penalties are - optimizations - trade-offs

- **.... which follow a set of architectural principles**

What is ANSA - in outline?

ANSA principles - in one slide

- **Distributed systems have different properties to centralised systems**

Differences are quite striking, and system design and implementation. techniques need to take this into account. Techniques and ideas based on central systems need revising to make appropriate. for distributed systems

- **Hide unnecessary complexity from programmers, where appropriate**

No doubt distribution more complex - more errors, more information about system to manage. Ought to make them harder to use - but it needn't if all unneccessary complexity hidden. A question of finding good abstractions (i.e. the right things to want)..

- **Different users need different solutions**

Do not expect one way of doing things will suit all uses and all technologies. Allow maximum freedom to vary and show how to cross boundaries where needs mismatch.

- **Use object-orientation to maximize simplicity and commonality**

Need a careful balance between simplicity and expressive power; don't want more complex than it needs to be so that conversion at boundaries is straight forward, but also not so simple that using it is difficult and long-winded; generic enough to apply to any component, whatever it does, or to any interaction. If you can do it for one component, it should work for all components.

Reversed Assumptions

- **Review, and reverse, many traditional system design assumptions:**

Distributed systems design is different: the are additional problems to grapple with, and new benefits to consider

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

Different Applications, Different Needs

There isn't going to be a single response to the wish list, because we are in a world of some very fundamental trade-offs, including.....

- **ABSTRACTION versus SPECIALIZATION**

The more you hide, the less chance there is to tinker

- **CONSISTENCY versus AVAILABILITY**

Availability means copies, increases risk of inconsistency

- **AUTONOMY versus UNIFORMITY**

Autonomy gives more freedom, but leads to differences which increases complexity

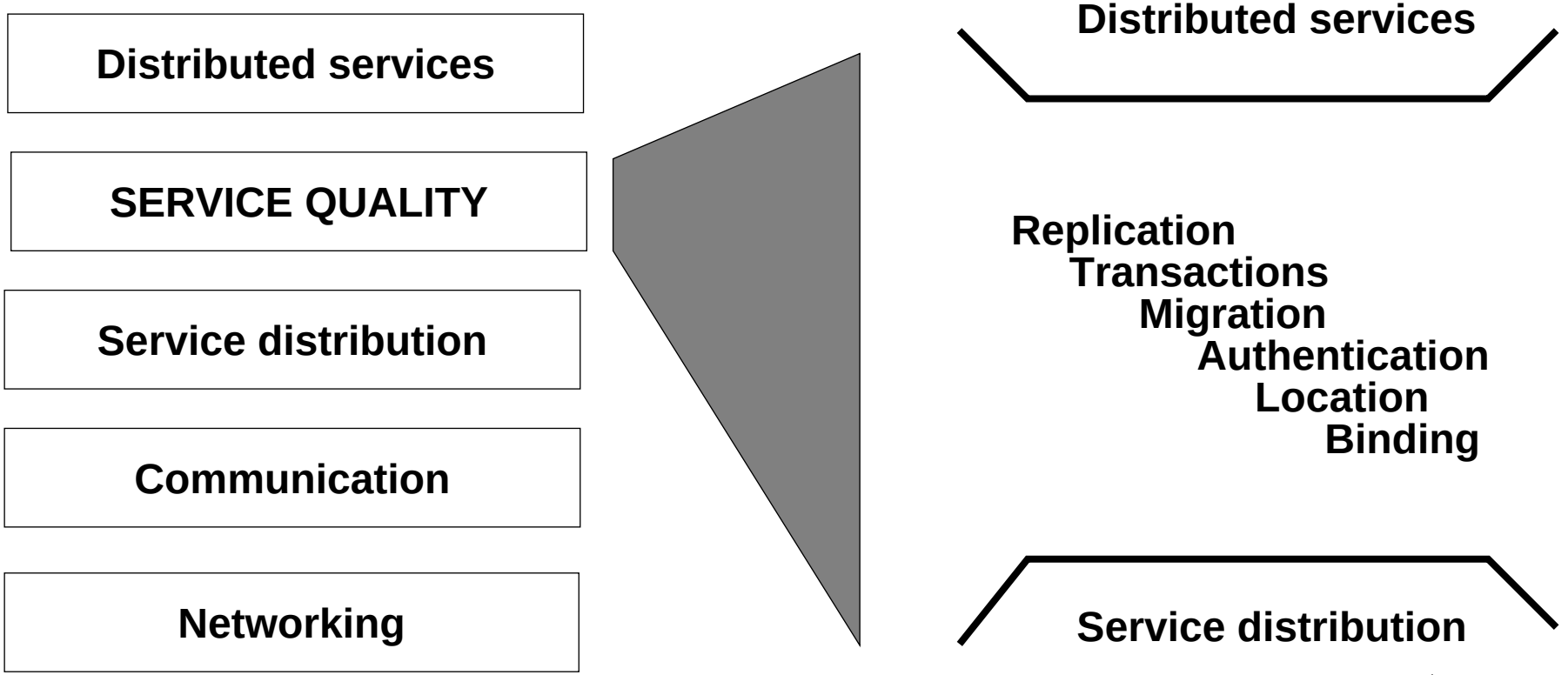
- **SECURITY versus CONVENIENCE**

Security makes things harder to do

- **any many other trade-offs**

Means that one size does not fit all.

Structure of a Distributed System



Selective Transparency

Service quality is about hiding irrelevant complexity - i.e. transparency:

- **Location**

don't need to know where it is to use it

- **Access**

don't need to know how it works to use it

- **Migration**

it can move while your using it, to balance loads or reduce latency

- **Replication**

there may be multiple copies for reliability and/or availability

- **Resourcing**

it gets resources when you use it and gives them back when you've finished

- **Partial Failure**

it always gets to a consistent state

- **Federation**

you don't have to have the same administrator to use it

The real productivity benefits come from automating transparency provision

ANSA Architectural Framework
(a.k.a.)
ISO Reference Model for Open Distributed Processing

- **concepts and rules for specifying ODP systems**

- **framework for development of ODP technology**

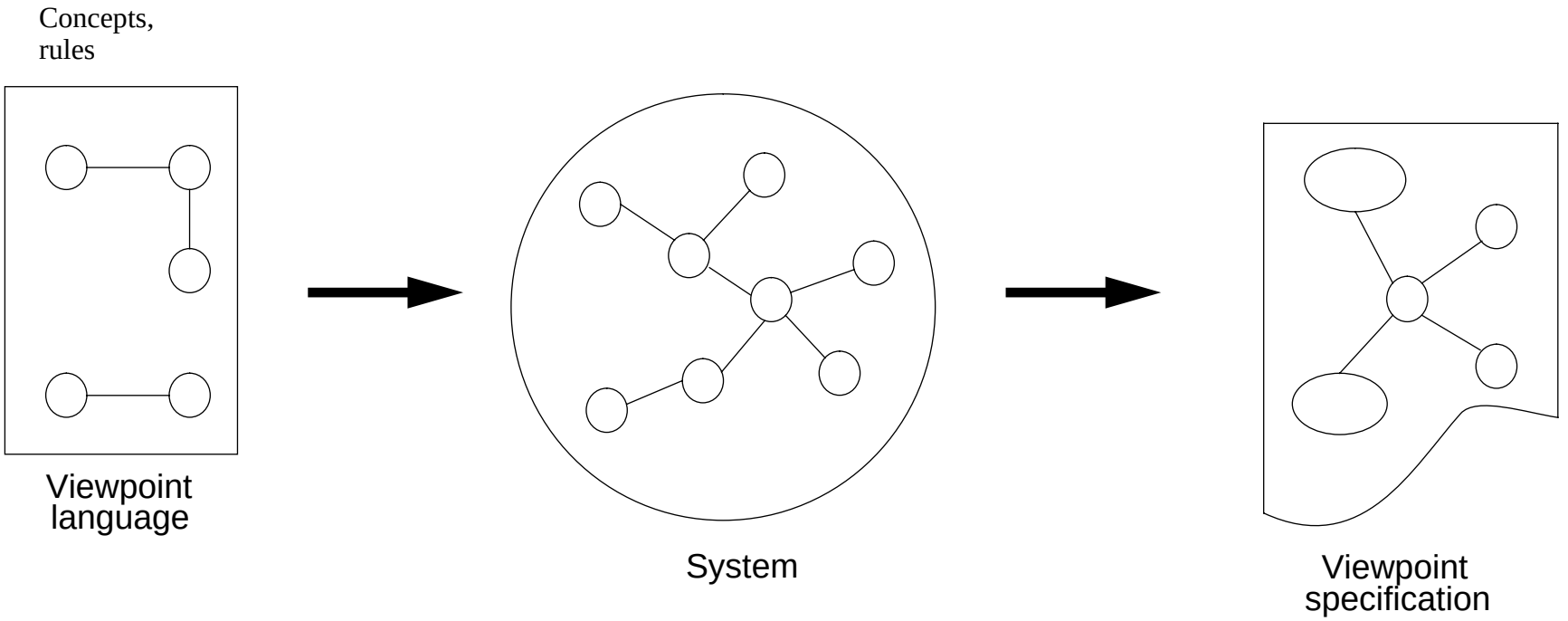
  - **basis for *specification*, *modelling* and *programming* tools**

  - **identification of crucial integration and infrastructure services**

Languages and Viewpoints (1)

- ***Framework*** **consists of five** ***viewpoint languages*** **and a set of** ***ODP functions***
- **Each viewpoint language enables specification of an ODP system or component**
- **Languages are structured so that** ***consistency checking*** **between alternative viewpoint specifications is possible**
- **The chosen set is** ***necessary*** **and** ***sufficient*** **for needs of ODP**
- **Each language consists of** ***concepts*** **(vocabulary) and** ***rules*** **(grammar)**
- **Tools for design and programming must support the concepts and enforce the rules**

Languages and Viewpoints (2)



- **Mathematical basis in *projection* of a set of concepts and abstraction/ specialisation relations over a semantic net.**

Viewpoints

- ***Enterprise*** **- purpose, scope and policies for a system**
- ***Information*** **- semantics of information and information processing in a system**
- ***Computational*** **- functional decomposition into objects suitable for distribution**
- ***Engineering*** **- the infrastructure required to support distribution**
- ***Technology*** **- the choice of technology to support distribution**
- **For engineers, the computational, engineering and technology viewpoints are of importance**
- **For designers, the enterprise, information and computational viewpoints are of importance**

Computational language

- An object orientated model refined to suit distribution
- Very similar to model implied by OMG CORBA
- Interface - point of provision or use of a service
  - Operational (interrogations)
  - Transactional
  - Stream (unstructured)
- All interfaces are typed
- Operations have set of possible terminations
- Arguments and results are references to interfaces - i.e. call-by-sharing
  - abstract data types enable decoupling from underlying hardware and network
- Object - encapsulated state for a set of interfaces
  - an object can have several interfaces

Types, Trading and Binding

- **Type - description of a potential “service requirement “or “service utilization”**
  - **signature - to ensure interaction is valid**
  - **behaviour - to ensure nature of service is understood**
  - **environment - to ensure correct infrastructure is in place**
  - **polarity - client, server, producer, consumer**
- **Type matching - ensuring contracts are compatible**
  - **may be computed (type checking) or asserted (type repository relationships) or both**
- **Trading - discovering an interface that provides a service**
  - **exporters make service offers**
  - **importers make service requests**
  - **trading marries imports to exports**
- **Binding - bringing together a set of matching interfaces to enable interaction**

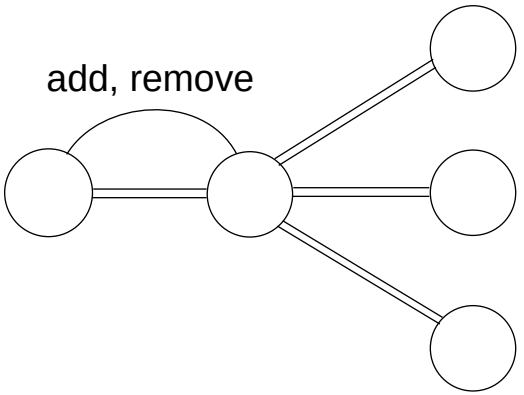
Trading

- **Each object has access to a trading context**
- **A trading context contains**
  - **services 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**
  - **volume of offers stored**
  - **accuracy of offer**
  - **dependability**
  - **local knowledge - e.g. an object can trade for parts of its infrastructure**
- **No structure is forced on context graphs, to enable federation**

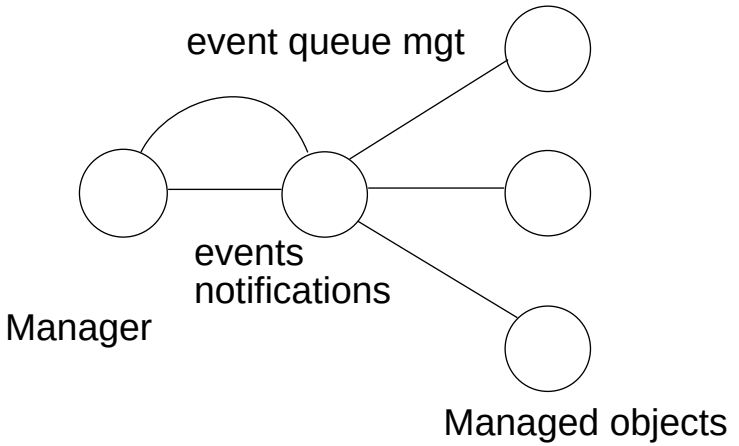
Binding

- ***Implicit*** **binding for operational interfaces**
  - **used when programmer doesn't care when resources are allocated to support the binding**
- ***Explicit*** **binding for all interfaces**
  - **used when programmer wants to bind streams**
  - **used when programmer wants to bind operational interfaces to form groups**
  - **used when programmer wants explicit control over resources**
- **BIND <binding type> {set of interfaces} - instantiates a new binding object**
- **Binding object has a control interface for supervising the binding**
  - **adding/removing interfaces**
  - **stopping/starting information flows**
  - **changing quality of service**
  - **monitoring events**
- **Explicit binding suits telecommunications and management integration**

Binding examples



Conferencing



Management domain

- **The binding is represented by a binding object that does the necessary splitting, joining, filtering and control**

Operational interface typing

- **operation invocation is a *request-reply* model**
- **type checking for safety - no surprises principle**
- **server must provide at least operations required by client**
- **client must accept all possible terminations from server**
- **client arguments “smaller” than server requires**
- **server results “smaller” than client accepts**
- **arguments and results can be interface identifiers**
- **each operation has one or more *terminations***

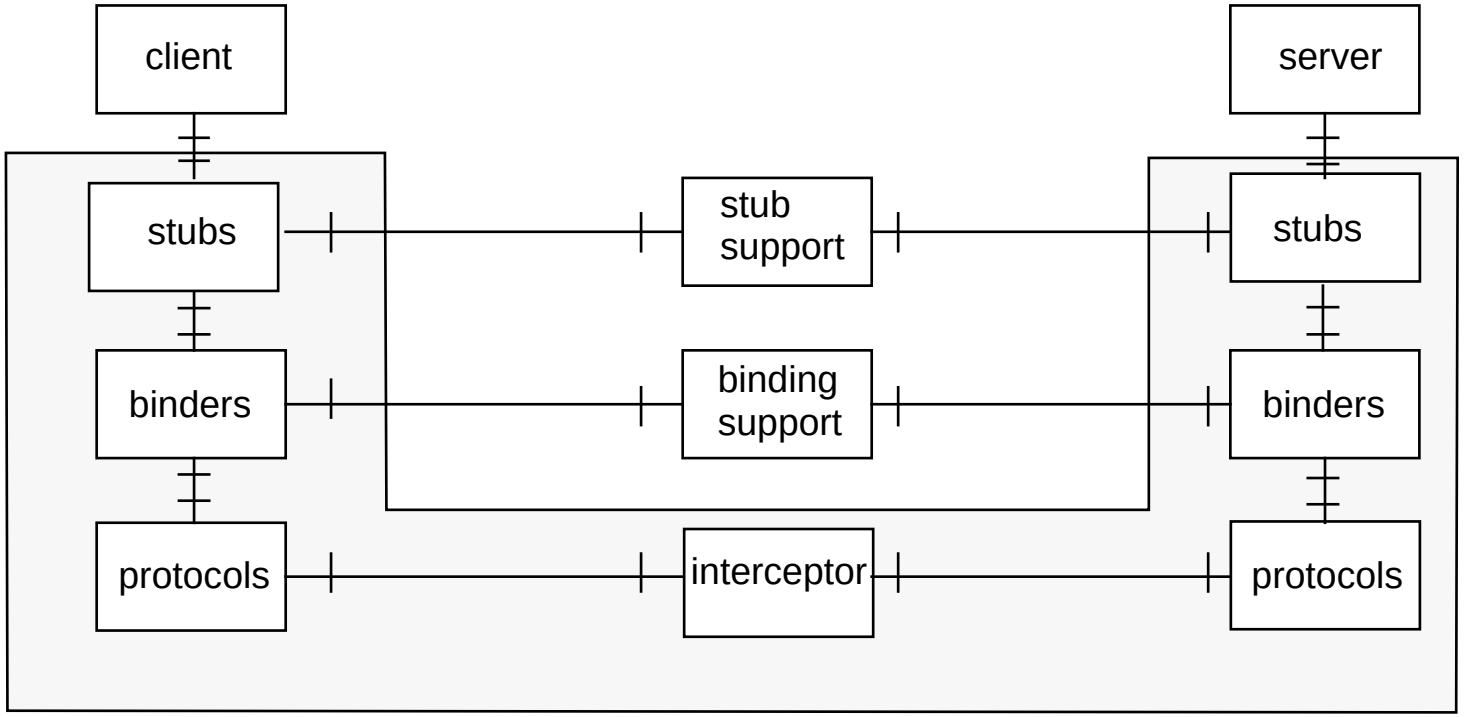
Transactions

- **Very general model - specific models defined in terms of**
- ***visibility* - how much interaction possible with objects outside the transaction**
- ***consistency* - invariant to be fulfilled at end of transaction**
- ***recoverability* - how much of transaction is undone after failure**
- ***permanence* - the degree to which failures can alter the effects of a transaction**
- ***dependency* - influence of other transactions on success or fail**
- **bracket transactions as operations**
- **label terminations *success* or *fail***
- **separation and ordering constraints as part of interface specification**

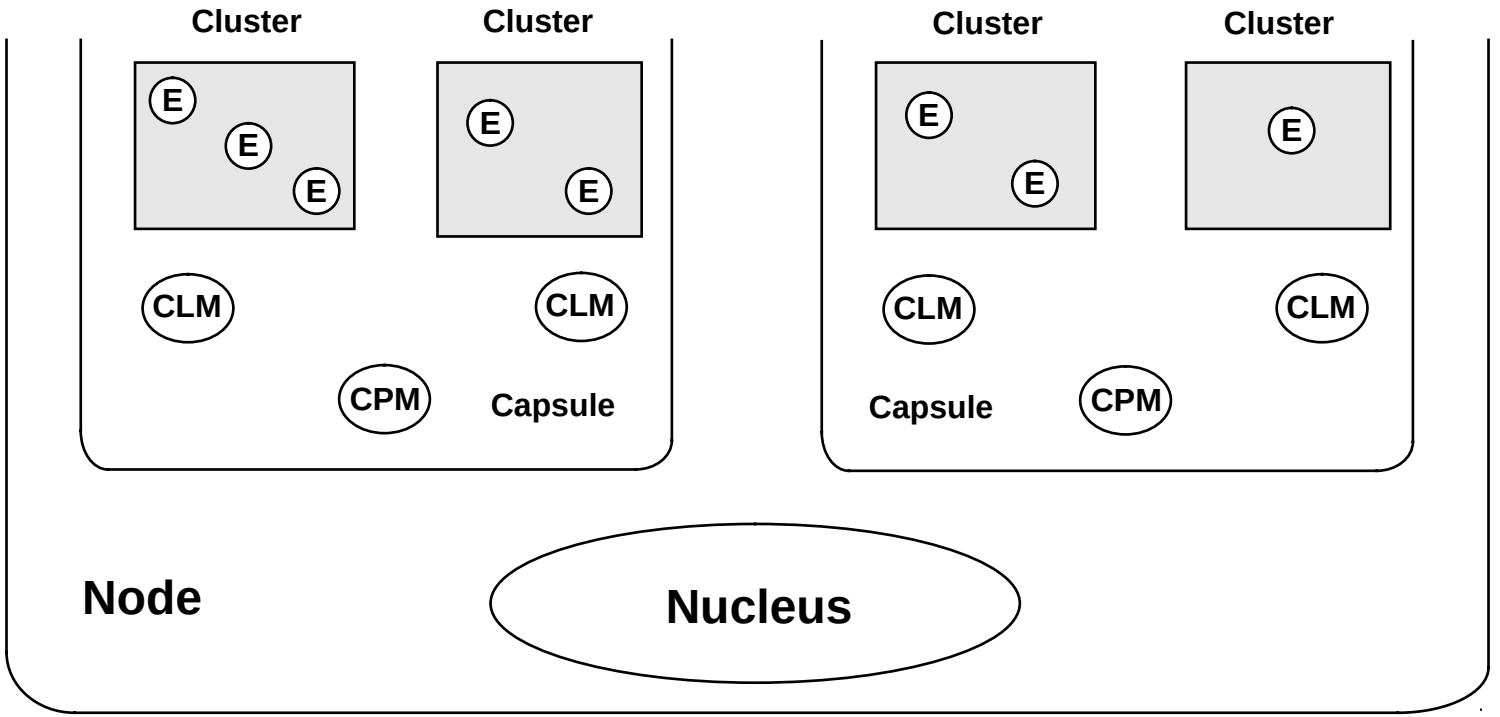
Engineering language

- **structures for encapsulation - basis of failure and security management**
  - **engineering objects for program modularity**
  - **clusters for activation, deactivation, migration**
  - **capsules for resource allocation, protection**
  - **nodes for network addressing**
- **structures for implementing infrastructures supporting *selective transparency***
  - **stubs, binders, protocols, interceptors**
- ***channels* for communication**
  - **simple client server**
  - **multipoint channels**
  - **stream channels**

Client-Server Channel



Engineering Structures



Functions

- **Management (object, cluster, capsule, communications, interface reference)**
  - **objects manage themselves, with the help of supporting services**
- **Coordination (transactions, groups)**
- **Repository (Storage, relocation, types, trader)**
  - **autonomous repositories federate as systems are interconnected - no assumptions about single name space**
  - **trader enables services to be discovered by type, context and properties**
- **Security**
  - **objects secure themselves, with the help of supporting services**
- **Transparency**
  - **recipes for using other functions to extend transparency of the infrastructure**

How does ANSA relate to other standards?

- ANSA shows where to use technology standards, and their strengths and weaknesses
- ANSA stimulates development of “missing” technology, better interfaces and tools
- e.g. OSF DCE & DME
  - mostly engineering => programmers interface is low level and cluttered
  - interworking protocol is included
  - oriented towards migration from mainframe to minicomputer => modularity and performance are not major issues
- e.g. OMG CORBA
  - mostly computational => nice programming model
  - no engineering => no interworking
- e.g. OSI Management
  - GDMO is an information model of network elements
  - CMIS/CMIP is engineering
  - where’s the computational viewpoint - “management applications”?

Where has ANSA had impact?

Impact

- **Standards**
  - **ISO Basic Reference Model for Open Distributed Processing**
  - **OMG CORBA, OMG JOSS (Lifecycle, Naming)**
  - **OSF DCE, DME**
  - **TINA-C (architecture, DPE?)**
- **Users**
  - **NASA Astrophysics Data System**
- **Sponsors**
  - **Reliable, high performance OSI routers**
  - **Integrating Banking and Healthcare applications**
  - **Technology feed into “distributed object” product lines**

How does the ANSA Workprogramme Operate?

History of ANSA

- **1985-88 ANSA Project started in UK under Alvey Programme**
  - **developed understanding of *architecture***
  - **developed basic ANSA computational and engineering models**
  - **built prototype DCE like distributed platform over MSDOS, VMS and Unix (ANSAware)**
  - **twelve partners, based in US and UK, coordinated by a management consultancy**
  - **goal was to build up capability in distributed computing**
- **1988-93 ESPRIT Integrated Systems Architecture Project**
  - **developed architecture for major transparencies**
  - **extended ANSAware with CORBA like capabilities and support for major transparencies**
  - **twenty one partners, based in Europe and US coordinated by APM**
  - **goal was to widen constituency, exploit the architecture and demonstrate benefits**

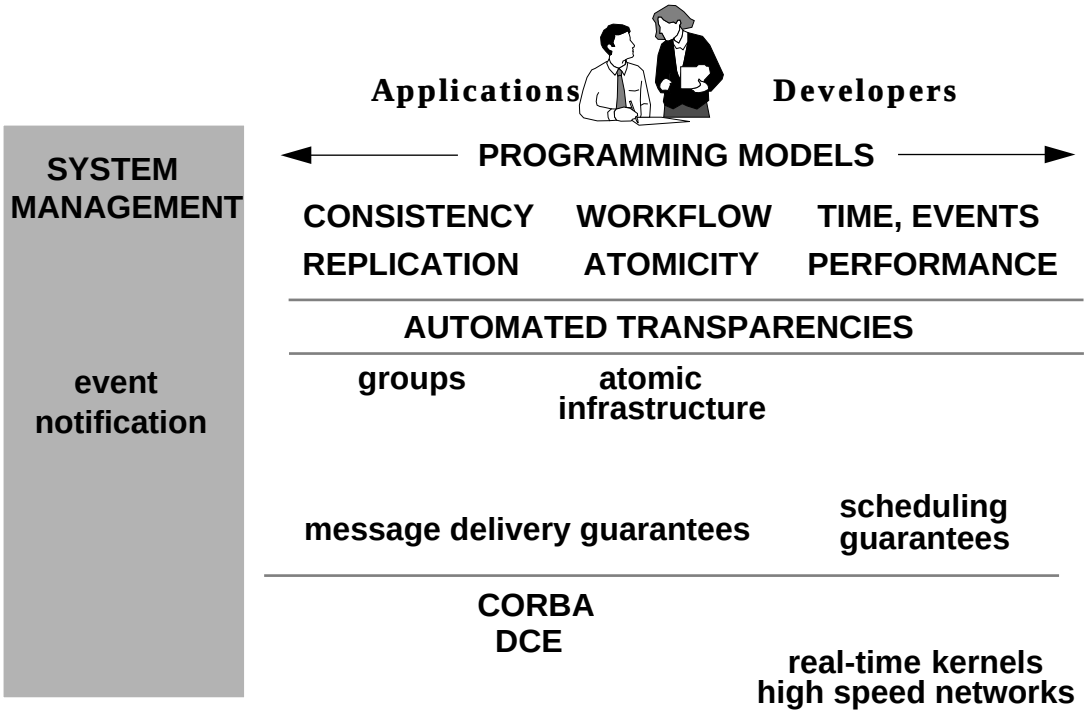
Founding Sponsors

- Bellcore
- Bell Northern Research - Europe
- British Telecom
- Digital Equipment Corporation
- France Telecom
- GEC
- GPT
- Hewlett-Packard Company
- ICL
- Open Connexion
- Additional sponsors identified

ANSA Phase III

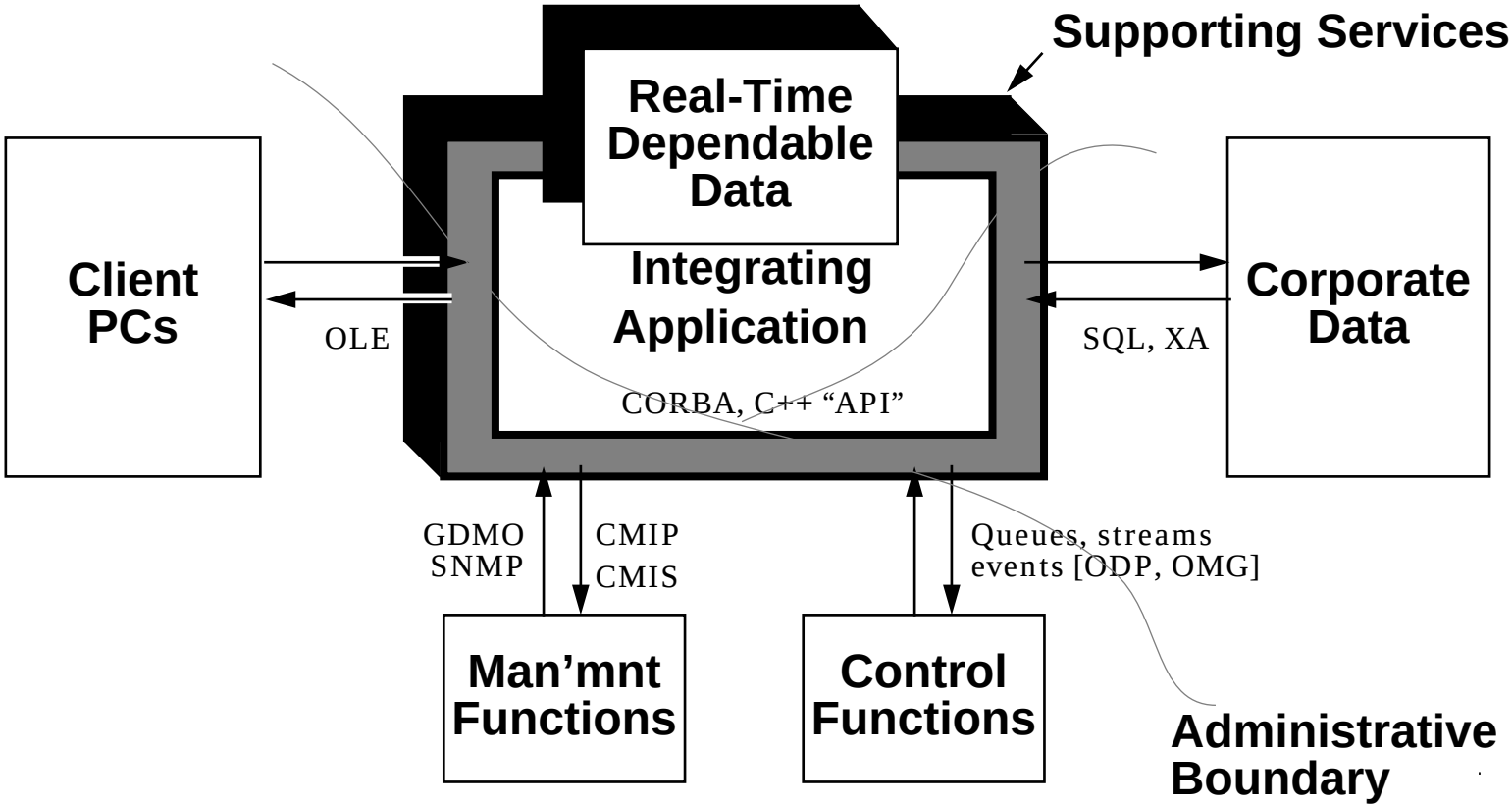
- **Jointly agreed programme of research and development between sponsors**
- **Harvest and integrate leading edge research into context of advanced product development**
- **Validation in applications done by sponsor's field trials**
- **Work with teams in sponsors on requirements, architecture and prototypes**
- **Focus on International Standards (ODP) for architectural framework, industry standards (OMG, OSF, UI, X/Open) for technology**
- **Strong input on technology from vendor sponsors**
- **Strong input on requirements from user sponsors and telecommunications sponsors**
- **Strong link into TINA Consortium**

Key Areas of Focus



- **Performance** **Dependability** **Federation & Tools**

The “Management Engine”



Performance

- **Context**
  - **open, federated, scalable and heterogeneous**
  - **control performance in an integrated open architecture, not a closed RT system**
  - **high level supervisory control, NOT low level interrupt handling**

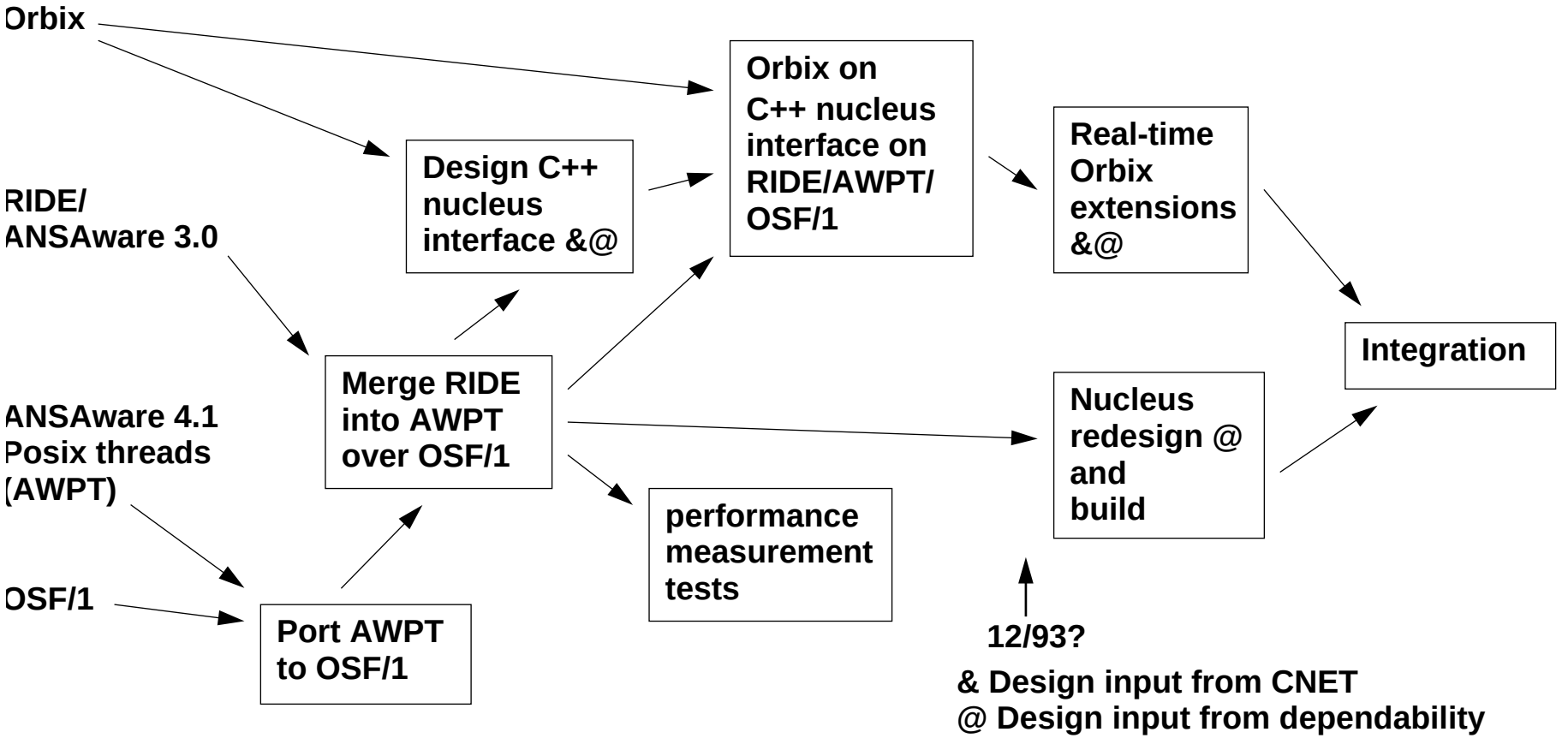
- **Performance requirements of scenario applications**
  - **need to manage real-time services from management services with less stringent performance requirements**
  - **therefore must interact between different scheduling domains whilst preserving performance guarantees**

- **Issues:**
  - **resource pools, scheduling points, choice of scheduling policies**
  - **QoS guarantees (deadlines, criticality, etc.)**

Performance Design

- **Computational - RT primitives for application programs**
  - **style: predictability, user control, mission orientation**
  - **QoS: deadlines, criticality, temporal synchronisation, resource requirements**
  - **application controlled resource management, scheduling policies and domains**
- **Engineering - RT nucleus extensions**
  - **pre-emptive - for responsiveness**
  - **resource pools**
  - **tasks, channels, buffers**
  - **scheduling points**
  - **interfaces, activities, operations**
  - **choice of scheduling policies**
  - **map scheduling points to resource pools**
  - **QoS negotiation during binding**
  - **bounded RPC protocol**

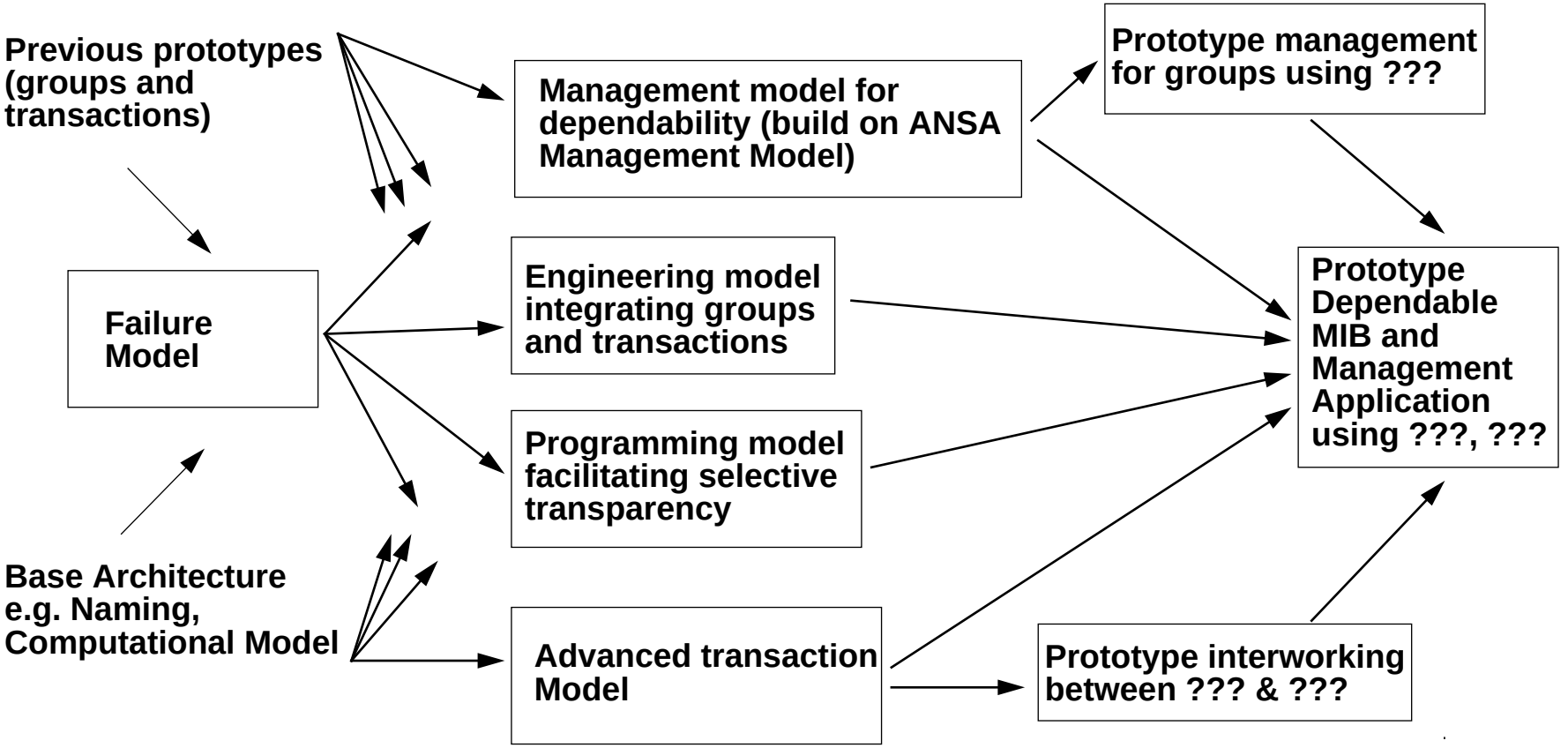
Plan for Building a Performance Platform



Dependability

- **Two important principles:**
  - **only use what you need**
  - **only provide what is required**
- **Select transparency engineering by stating an appropriate requirement**
- **Architect engineering and reuse of mechanisms**
  - **what is the kit of parts?**
  - **how do the mechanisms interact?**
  - **how are they divided between application and infrastructure?**
  - **what is the role of trading and binding in assembling the infrastructure?**
  - **can you select appropriate transparencies statically and dynamically?**
- **Based on this define the programmer's interface**

Dependability outline plan



Federation and Tools

- **Apply ideas for re-use in existing small scale language based tools (SmallTalk, C++, Objective C, Eiffel) to programming in the very large**
- **Use of existing services essential**
- **Characterise system components as services**
- **Service contracts to describe services**
- **Exchanging and negotiating service contracts**
- **Maximize checking, use repositories when available**
- **Tools to generate interworking code from service contracts**

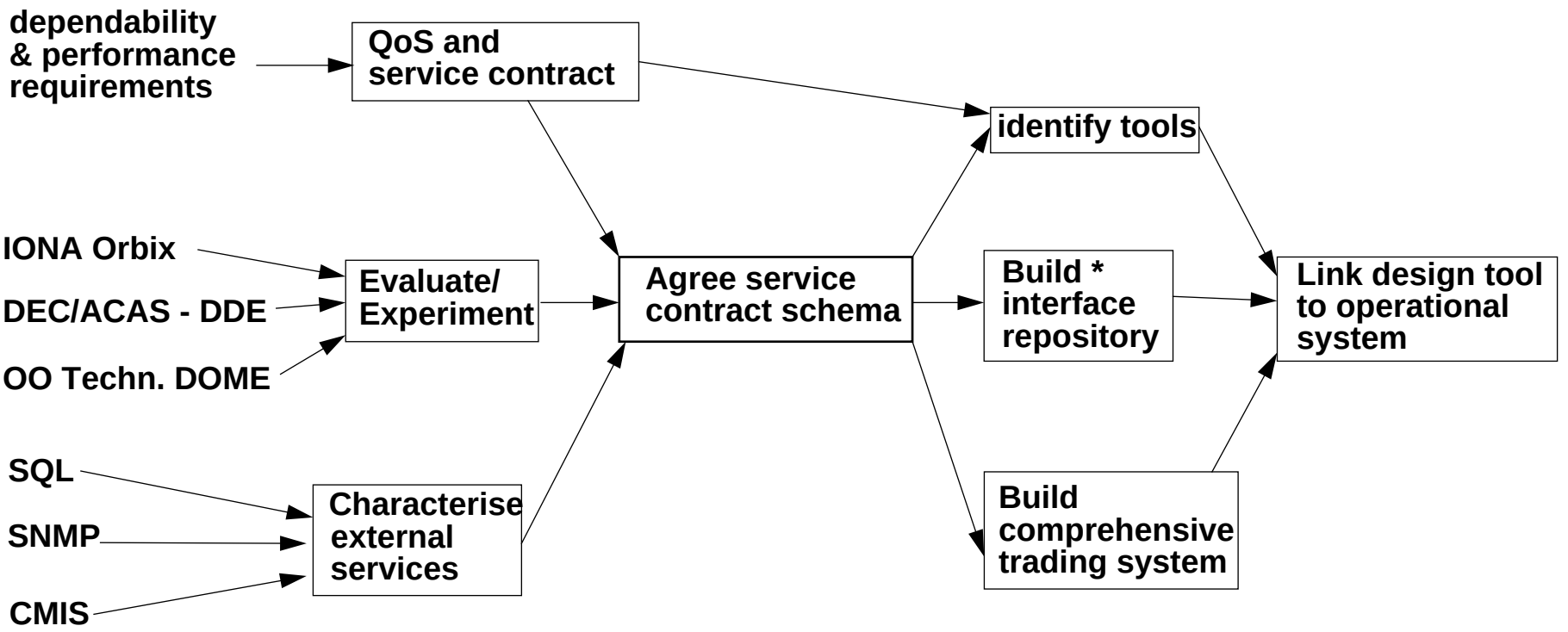
Federation and Tools
- service contracts (1) -

- **Client needs to know about service before it can use the service:**
  - **semantics**
  - **type**
  - **QOS**
  - **transparency mechanisms**
  - **communications protocols**
  - **naming/address**
  - **other service design decisions which affect interworking?**
- **Server needs to know about clients (e.g. for billing and security)**
- **Specialisation leads to technology domains: need interception code**
- **Contract enables management to ensure contractual obligations are met**

Federation and Tools
- service contracts (2) -

- **Service contracts are used to achieve interworking**
  - **at analysis time: determine common purpose and meaning**
  - **at design time: determine common infrastructure**
  - **at build time: generate interceptors - determine common engineering**
  - **at run time: binding - select compatible engineering**
- **Service contract representation**
  - **completeness: an IDL is not enough: what about QoS?**
  - **representation: DCE IDL, CORBA IDL, Abstract Data Types, Abstract Syntax Trees**
  - **type systems act as context in which the service type information is interpreted**
- **Service contract negotiation and exchange:**
  - **requires more comprehensive trading service**
  - **start from interface repositories & implementation repositories, not name servers**

Federation and Tools plan



* OMG input

Sponsor Involvement

- **Management Committee: strategy, budgets, resources**
- **Technical Committee: focus, quality, technology transfer overall, standards**
- **Topic Review Groups: working with the ANSA team members in depth**
- **Field trials: joint between sponsors**
- **Secondees: in the ANSA team**
- **Workshops: both on requirements, and results**
- **Consultancy: helping individual sponsors use ANSA**
- **Reports: the audit trail and repository of knowledge**

Why are we here?

We'd like you to join in the ANSA Workprogramme

to become technical contributors,

and users

of ANSA