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

ANSA: From Soup to Nuts

Andrew Herbert

Abstract

A 60 minute talk on ANSA starting from the ISA results, through service management issues, to an architectural summary and introduction to Phase III.

APM.1105.00.01 **Draft** 7 December 1993

Briefing Note

Distribution:
Supersedes:
Superseded by:

ANSA

from Soup to Nuts

Andrew Herbert

ANSA Chief Architect

Technical Director
Architecture Projects Management Limited

What is the ANSA Workprogramme?

- A collaborative industry effort to advance distributed systems technology
- Based on a shared architectural vision (ANSA)
- Vendor neutral
- Contributes to standards
- Produces advanced technology prototypes
- Enables its sponsors to deliver more effective products and services
- Provides a technical resource for the sponsors to use
- Strong focus on computer and telecommunications service integration

History of ANSA

- **1985-88 ANSA Project, started in UK under Alvey Programme**
  - developed basic understanding of architecture
  - developed distributed programming and system structuring models
  - built prototype DCE-like platform over MSDOS, VMS, Unix (ANSAware)
  - twelve partners 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**
  - made programming model more object-oriented - c.f. OMG CORBA
  - selective transparency in system structuring model
  - twenty one partners in Europe and US, coordinated by APM
  - goal was to widen constituency and demonstrate benefits

ISA Results (Available to all Sponsors)

- **Applications / Products**
  - NASA Astrophysics Data System, Banking, Healthcare, Reliable OSI communcations servers, others
- **Standards**
  - ISO / CCITT Basic Reference Model of Open Distributed Processing (CD-1)
  - Acknowledged input to OSF DCE
  - Supported sponsors in OMG ORB and Object Services RFP process
- **Reports to replace *ANSA Reference Manual***
  - Architecture Reports on Naming, Replication, Transactions, Trading, ...
  - Technical Reports on Streams, Programming Tools, Interoperability, ...
- **ANSAware 4.1: a full function *Object Request Broker* with:**
  - Trading and resource management
  - Object persistence and migration
  - Transactions (Arjuna) and Replication (GEX)
- **Approximately 150 registered ANSAware licencees**
  - Results now appearing in the journals

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 (linked to TINA auxiliary projects)
- Work with teams in sponsors on requirements, architecture and prototypes
- Focus on International Standards (ODP) for architectural framework, industry standards (OMG, OSF, X/Open) for technology
- Strong input on technology from vendor sponsors
- Strong input on requirements from user sponsors and telecommunications sponsors
- Strong architectural input into TINA Consortium

Key Areas of Focus

Applications Developers

SYSTEM MANAGEMENT

event notification

PROGRAMMING MODELS

CONSISTENCY WORKFLOW TIME, EVENTS

REPLICATION ATOMICITY PERFORMANCE

AUTOMATED TRANSPARENCIES

groups

atomic infrastructure

message delivery guarantees

scheduling guarantees

CORBA
DCE

real-time kernels
high speed networks

Performance **Dependability** **Federations & Tools**

Founding Sponsors

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

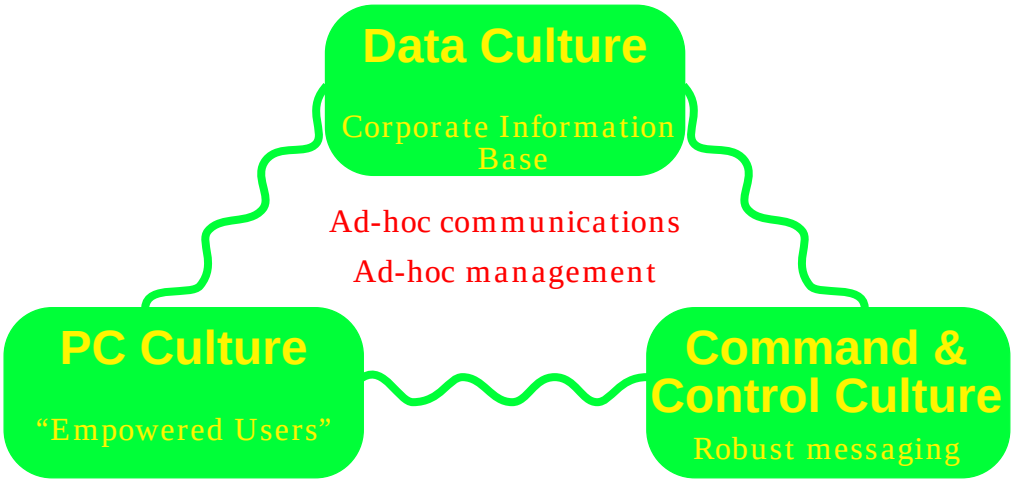
Distributed Computing

What are the issues?

Distributed systems used to be an end in their own right, but that picture has changed in recent years

Distributed Computing Cultures

- Three important and quite separate distributed computing cultures exist

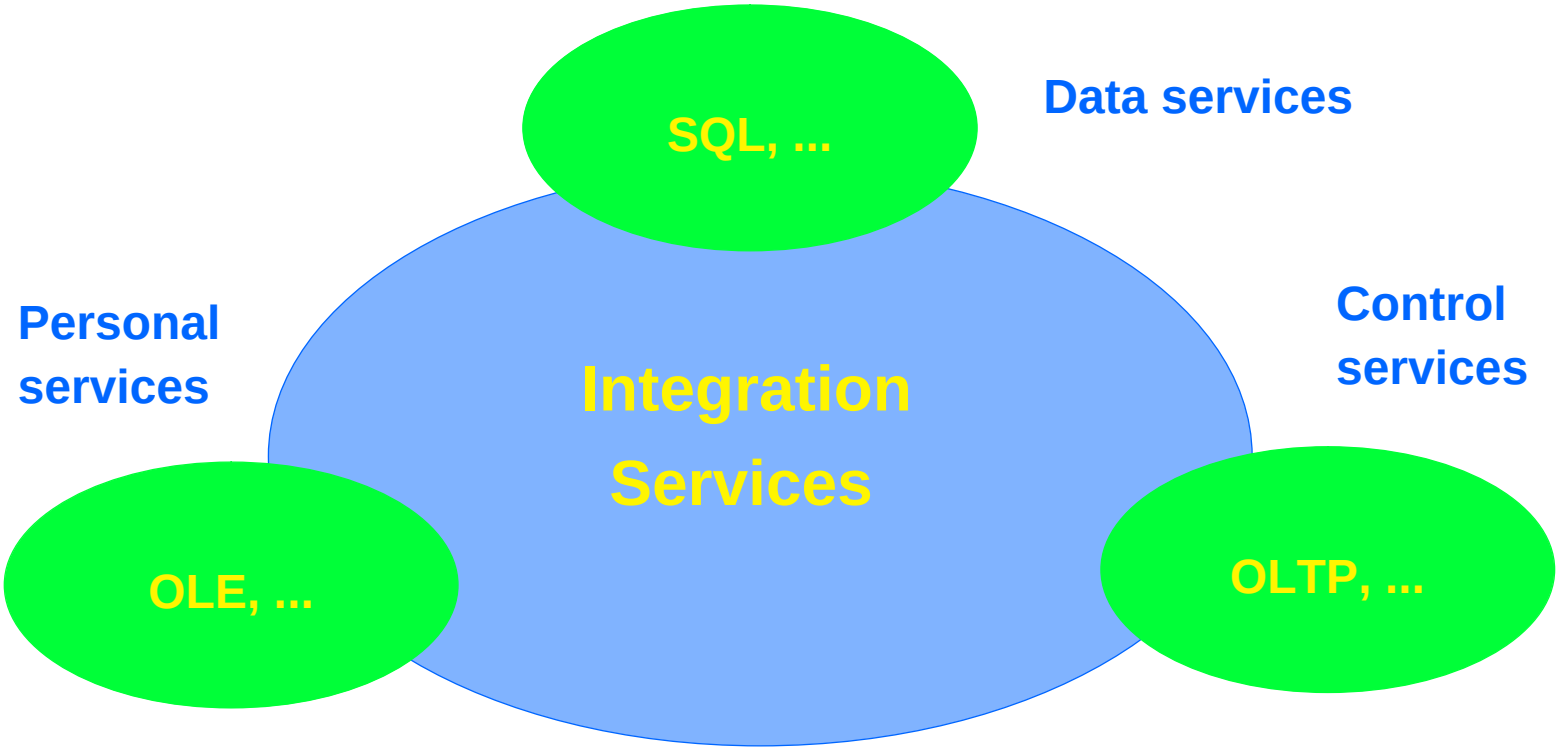


- Each has its own means of distribution and application integration
- They meet different needs - no single one will absorb the others
- Legacy of existing technology choices and applications will persist

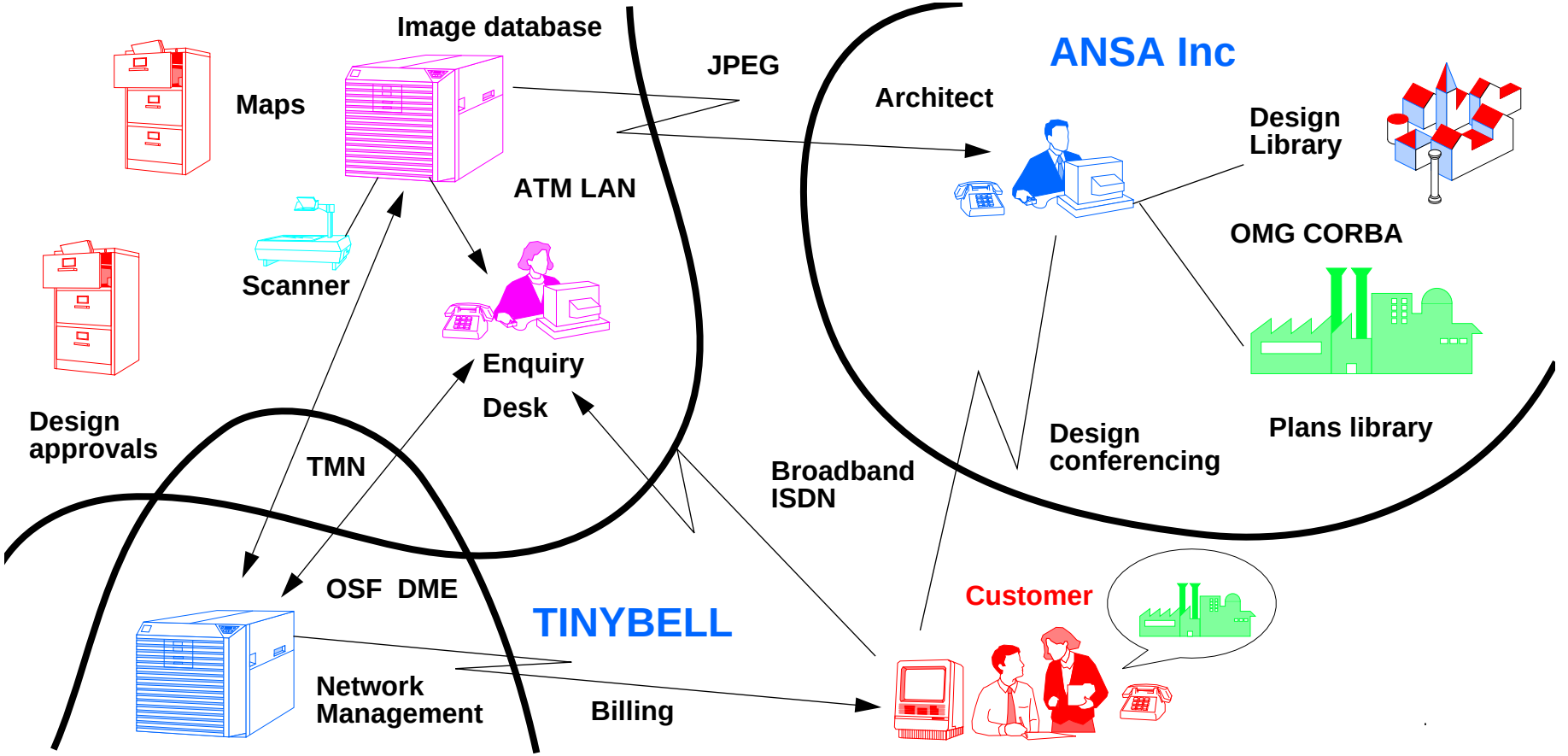
Strengths, Weaknesses

- **Data Culture**
  - Remote data access - mostly proprietary remote login SQL access
  - Federated databases - typically read-only access especially if heterogeneous
  - Stored procedures - for high throughput critical business transactions
  - Object repositories - for more flexible structuring, fine grained locking
- **PC Culture**
  - Single user productivity services (based on OLE, DDE etc.) - scaling issues
  - File and printer sharing - scaling, security and management issues
  - Group productivity services - very proprietary, not very robust, workflow based
  - Mobile computers, universal personal digital communication
- **Control Culture**
  - On-line transaction processing - reliable message routing and server activation
  - Strong on throughput, failover and security
  - Systems programmer oriented API
  - Workflow - high level description of transactions and data types - preconceived business models

Distributed Computing = Integration Services = Interoperability and Management



TINAVILLE - An Example Scenario



TINA Service Management - The Technical Questions

- How can we analyse business requirements to identify potential services?
- How can we ensure service engineering covers all options and meets all constraints
- How can services be developed rapidly to meet market windows?
- How can existing services interwork with new services?
- How can deployment and management of services be automated?
  - to reduce people costs
  - for faster, accurate response
- How can services be evolved to accomodate new requirements and adapt to new technology?

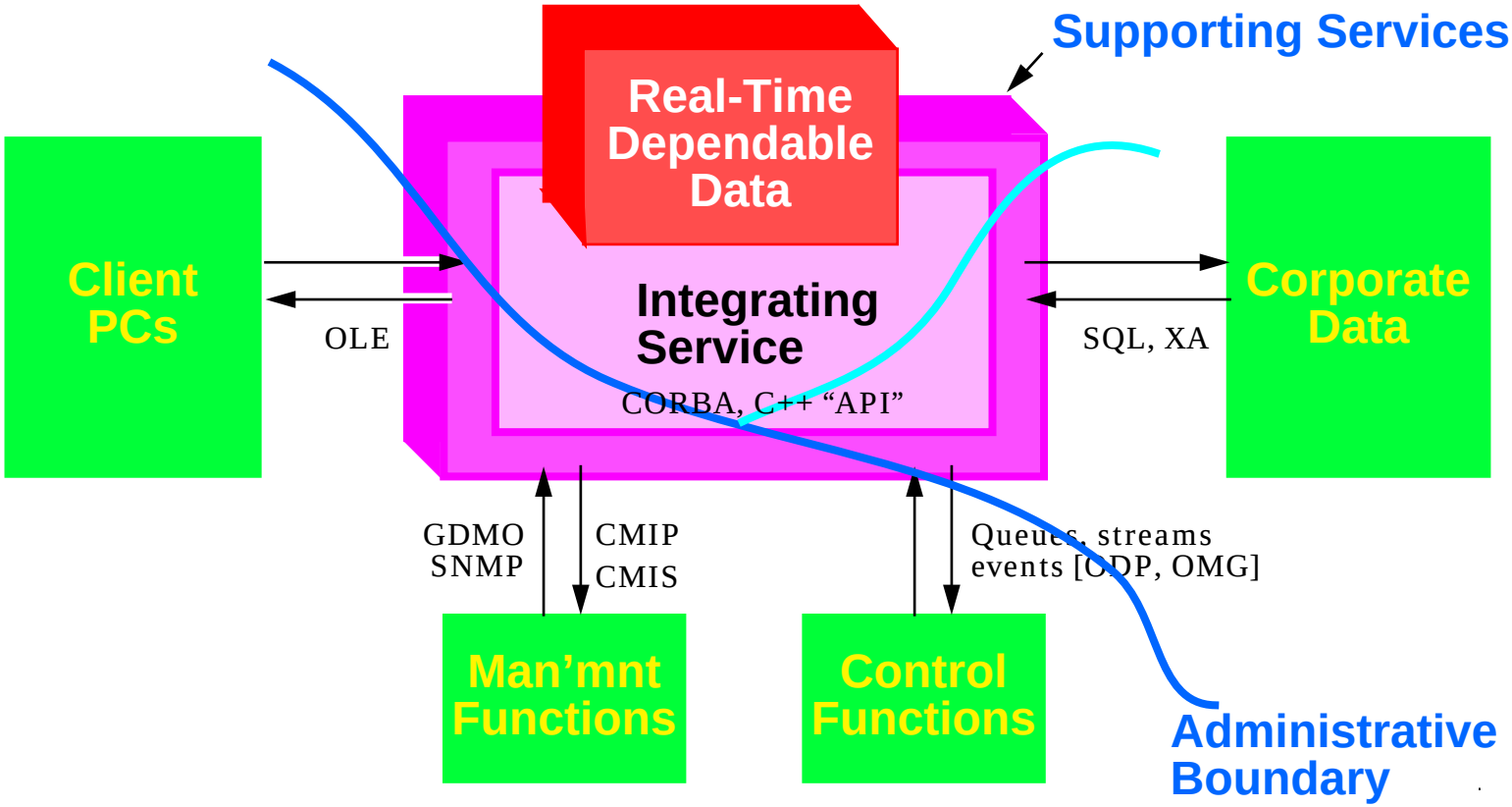
TINA Service Management - The Business Questions

- How will cooperative services between organizations be set up?
- What are the obligations, liabilities and sanctions (sensitivity, charging, regulation)?
- How will service contracts be negotiated?
- How will feature interactions (policy conflicts) be identified?
- How will the worth of information and information services be assessed (copyright, licensing)?
- How much of this will be supported by service engineering tools?
  - service validation
- How much automation will be available?
  - service generation

This isn't Just a Telcommunications Problem - The Generic Integration Wish List

- **Integrate products from many vendors** - exploit innovation, price/ performance trade-offs
- **Span application domains**- enable information to flow between departments easily
- **Hide system boundaries** - size sould only affect perormance, not functionality
- **Protect enterprise boundaries** - preserve autonomy and enable flexibility
- **Enable inter-organisation computing** - controlled federation - doing business doesn't require a merger
- **Preserve existing investments** - enable evolutionary change
- **Match IT style to Enterprise requirements** - make the business drive the system, not the other way round
- **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,....

Support for Integration Services - Management Engines



Reaching the Solution - using ANSA Principles

- **Specify systems using application concepts**
  - good abstractions
  - minimise programmed book-keeping
- **Define a robust, high level model for distributed programming**
  - use objects for encapsulation / separation
- **Use tools to automatically generate the engineering detail**
  - generalize the idea of a *stub generator*
- **Define structures for integration**
  - cradle to grave object management and monitoring
  - as much checking as possible, as early as possible
  - trading and federation
  - transparency

The ANSA Architecture

So lets look at some of the details

The Programming Model

- **Guidelines for adding distribution features to programming languages**
  - PREPC for C, Arjuna class library, DPL “wrapping language”, CUCL Modula 3
- **An object orientated model refined to suit distribution**
  - Successfully anticipated OMG CORBA
- **Interface - point of provision or use of a service**
  - Operational - RPC style “method invocation” [interrogation]
  - Transactional - Operation plus atomicty and concurrency control
  - Stream - flow with synchronization events - e.g. voice, video, TCP byte stream
- **Operations have set of possible outcomes [terminations]**
- **Arguments and results are references to interfaces - i.e. call-by-sharing**
  - reinforces view of an object as a service
  - migrate and copy available as management operations on objects
- **Object - encapsulated state for a set of interfaces**
  - an object can have several interfaces, an interface is a closure

Types, Trading and Binding

- **Type - description of a potential "service requirement "or "service utilization"**
  - signature - to ensure interaction is valid - i.e. C function prototype
  - behaviour - to ensure nature of service is understood - e.g. pre and post conditions
  - environment - to ensure correct infrastructure is in place - e.g. list of transparency requirements
  - polarity - client, server, producer, consumer
  - **N.B. a type is a predicate, not a class or template**
- **Type descriptions are objects in the runtime environment**
  - enables dynamic type checking during trading
  - because we cannot formalize all of a type we need a type repository to hold user assertions about type compatibility
- **Trading - discovering an interface that provides a service**
  - exporters make service offers
  - importers make service requests
  - trading marries imports to exports
- **Binding - knitting together a set of matching interfaces to enable interaction**

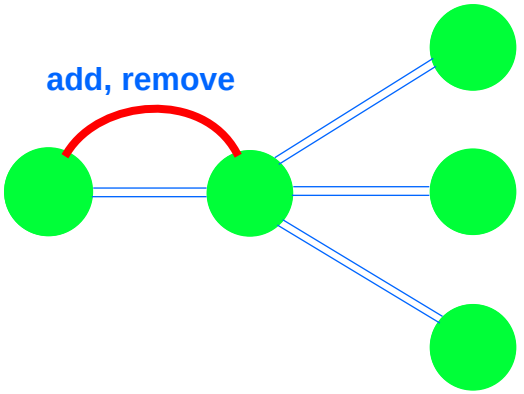
Trading

- 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

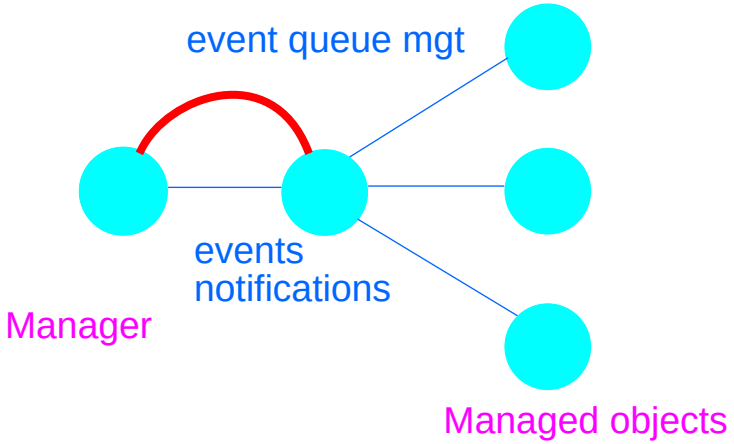
Binding

- ***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 (e.g. conferences, domains)**
  - **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

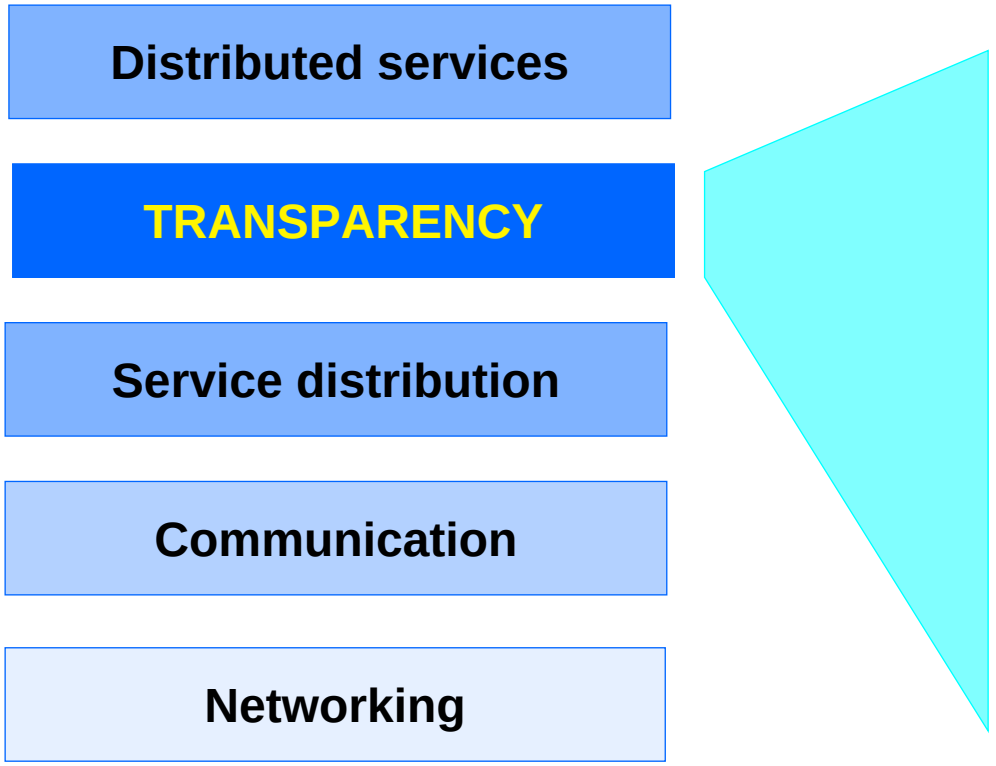
Type Safety in Evolving Systems

- old clients must be able to use upgraded servers without recompilation
- basic principle - no surprises
- 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
- We have built a checker for abstract data types, needs extending
  - to handle dynamic type checking
  - to handle generic functions (e.g. ListOf [BankAccount])

System Structuring Rules

- **Distributed systems engineering is all about trade-offs**
  - **ABSTRACTION versus SPECIALIZATION - the more you hide, the less control you have**
  - **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**
- **Therefore we need a kit of parts and an open "nucleus" into which they slot**

Structure of a Distributed System



Distributed services

Replication
Transactions
Migration
Authentication
Location
Binding

Service distribution

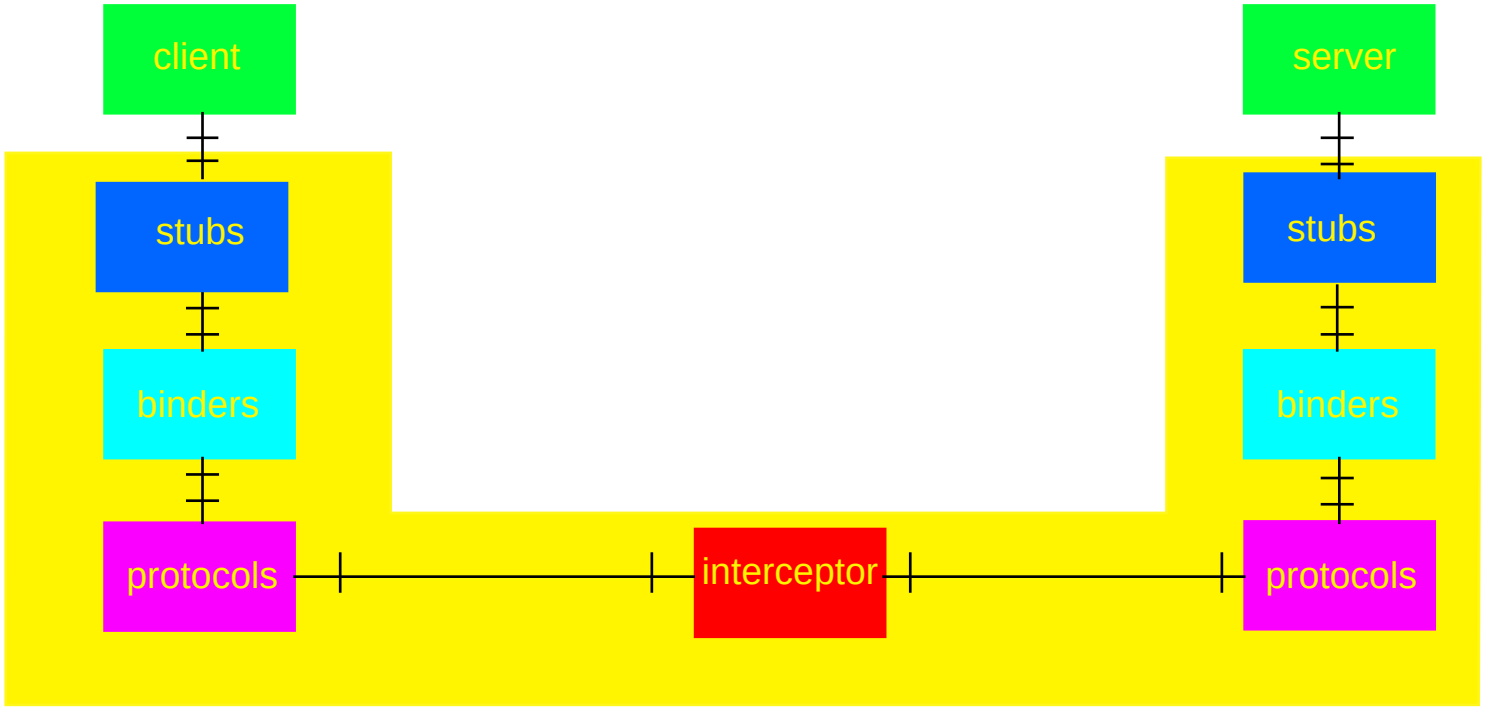
Selective Transparency

- **Transparency is about hiding irrelevant complexity**
  - **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**
  - **Resource - 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 transparency layer supports the functionality of the service distribution layer, and adds additional guarantees**

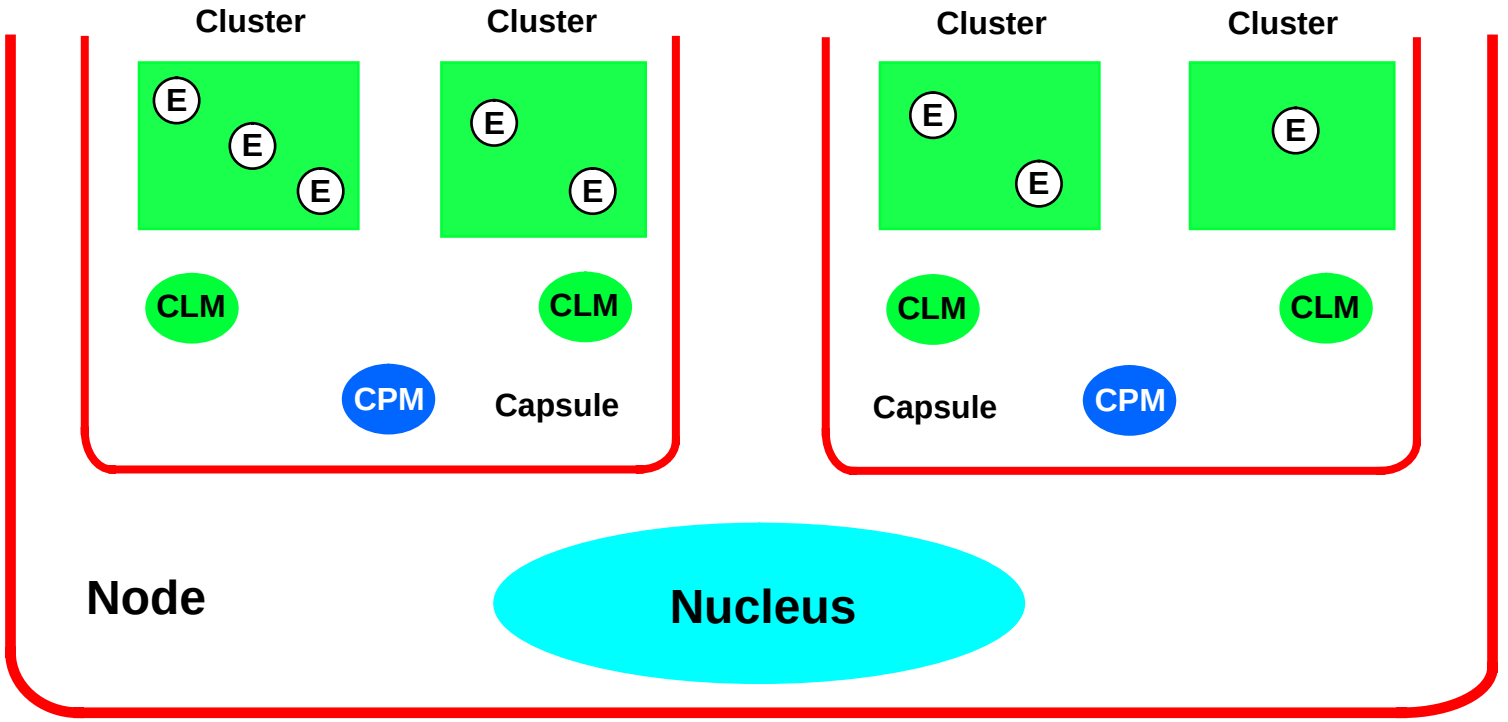
Engineering Model

- **Structures - for object failure and security management**
  - **objects** for program modularity
  - **clusters** for activation, deactivation, migration
  - **capsules** for resource allocation, protection
  - **nodes** for network addressing
- **Template for an ORB supporting *selective transparency***
  - stubs, binders, protocols, interceptor configured into channels
  - simple client server channels
  - multipoint (groups)
  - stream channels

Client-Server Channel



Engineering Structures



Functions to Support Service Integration

- **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 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”?

What Are We Working On Now?

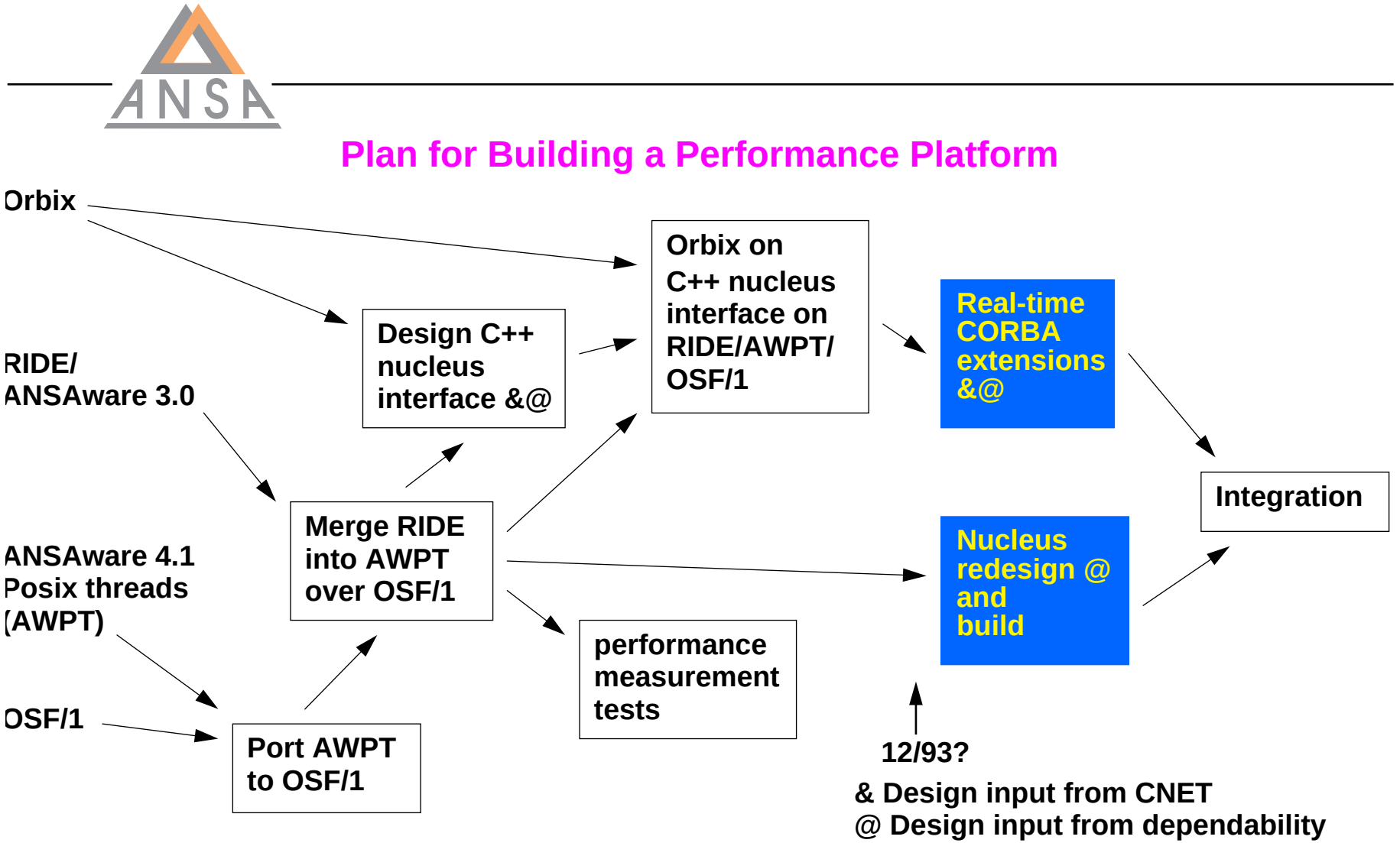
The Work in Progress for Phase III

Performance / Real-time

- **Context**
  - open, federated, scalable and heterogeneous
  - control performance in an 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
  - negotiating and meetingQoS guarantees (deadlines, criticality, etc.)
  - low latency, high throughput enginering solutions

Performance Design

- **Programming - additional 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
  - signals and synchronous programming (e.g. ESTEREL)
- **Engineering - 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 explicit binding
  - bounded RPC protocol


Plan for Building a Performance Platform
Orbix
RIDE/
ANSAware 3.0
ANSAware 4.1
Posix threads
(AWPT)
OSF/1
Design C++
nucleus
interface &@
Port AWPT
to OSF/1
Merge RIDE
into AWPT
over OSF/1
Orbix on
C++ nucleus
interface on
RIDE/AWPT/
OSF/1
performance
measurement
tests
Real-time
CORBA
extensions
&@
Nucleus
redesign @
and
build
Integration
12/93?
& Design input from CNET
@ Design input from dependability

Dependability

- **Build on the legacy from Phase II (replication, transactions, persistence, migration)**
  - only use what you need
  - only provide what is required
  - Select transparency engineering by stating an appropriate requirement at compile / link time
- **Architect engineering and reuse of mechanisms**
  - what is the kit of parts?
  - group view manager, failure detector, distributor, checkpointer, log, sequencer, lock manager, collator, ...
  - how do the mechanisms interact? what guarantees to they give?
  - 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 for selective transparency**
  - making application specific knowledge available to the mechanisms
  - generating the low-level calls from high level programs (e.g. workflow)

Dependability outline plan

Previous prototypes (groups and transactions)

Failure Model

Base Architecture e.g. Naming, Computational Model

Management model for dependability (build on ANSA Management Model)

Engineering model integrating groups and transactions

Programming model facilitating selective transparency

Advanced transaction Model

Prototype management for groups using ???

Prototype Dependable MIB and Management Application using ???, ???

Prototype interworking between ??? & ???

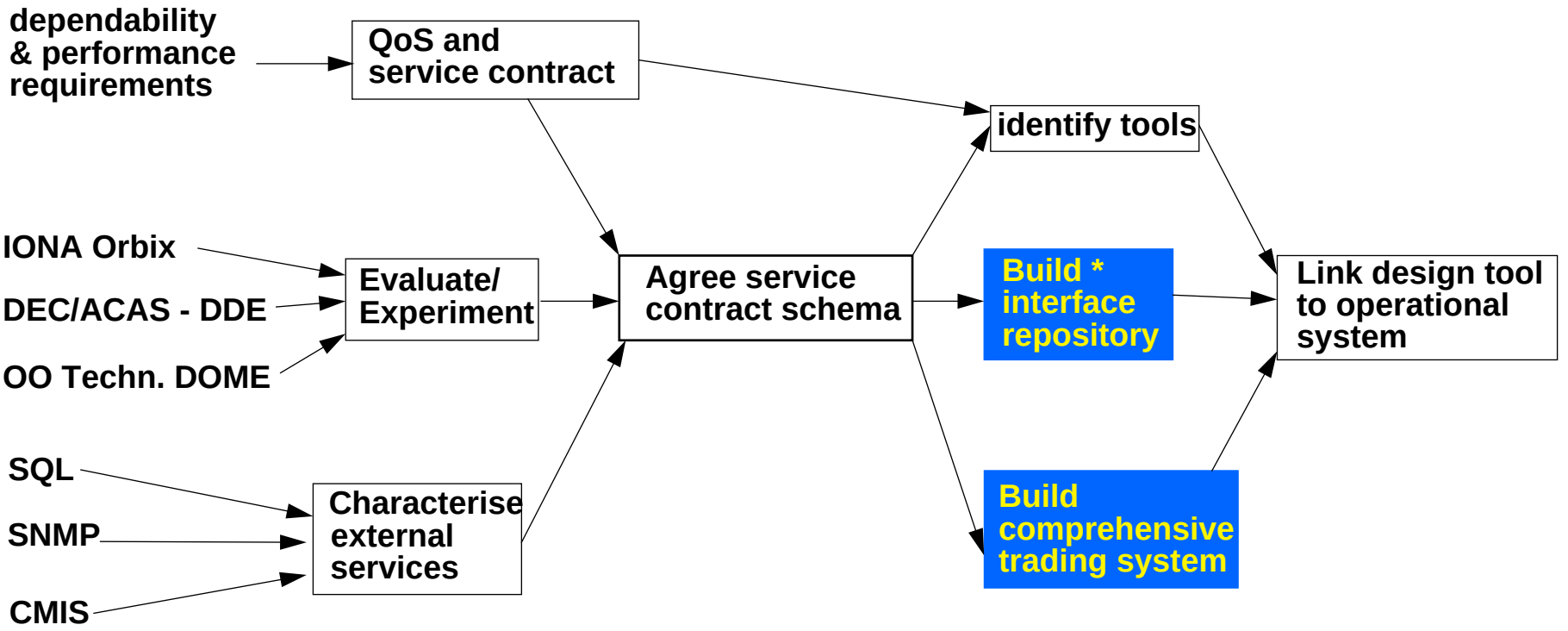
Federation and Tools

- 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 - look to wrapping and interception
- Characterise system components as services
- Service types to describe services
- Means for exchanging types and negotiating service contracts
- Maximize opportunities for checking
- Use repositories when available to store system meta data online
- Tools to generate interworking code (stubs, binders, interceptors) from service contracts

Federation and Tools

- **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
  - how can we use service contracts alongside code templates and business heuristics in programming productivity tools?
- **Service contract representation**
  - completeness: an IDL is not enough: what about QoS?
  - representation: DCE IDL, CORBA IDL, Abstract Syntax Trees
  - programming language 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



Why am I here?

To bring you up to date with ANSA

To make you aware of ANSA as a resource for HP to use

To find out where ANSA can help you