



**Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom**

TELEPHONE:
INTERNATIONAL:
FAX:
E-MAIL:

**Cambridge (01223) 515010
+44 1223 515010
+44 1223 359779
apm@ansa.co.uk**

ANSA Phase III

A Guided Tour of RM-ODP Part 3 (Prescriptive Model)

Andrew Herbert

Abstract

This document is a slide presentation to accompany the second CD for ISO 10746-3 Basic Reference Model for Open Distributed Processing - Part 3: Prescriptive Model (July 1993).

The presentation reviews the major concepts in the same order as they appear in the CD with additional remarks giving rationale for the organization and content of the reference model.

APM.1052.02

Approved
Standards Contribution

13th January 1995

Distribution:

Supersedes: APM.1040.00.03

Superseded by:

Copyright © 1995 Architecture Projects Management Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.



A Guided Tour of the Basic Reference Model for Open Distributed Processing (Prescriptive Model)

**Andrew Herbert
(Editor)
(Technical Director, APM)**



Context

- **Part 1: Overview and guide to use**
- **Part 2: Descriptive Model**
- **Part 3: Prescriptive Model**
- **Part 4: Architectural Semantics**
- **Trader**



Scope

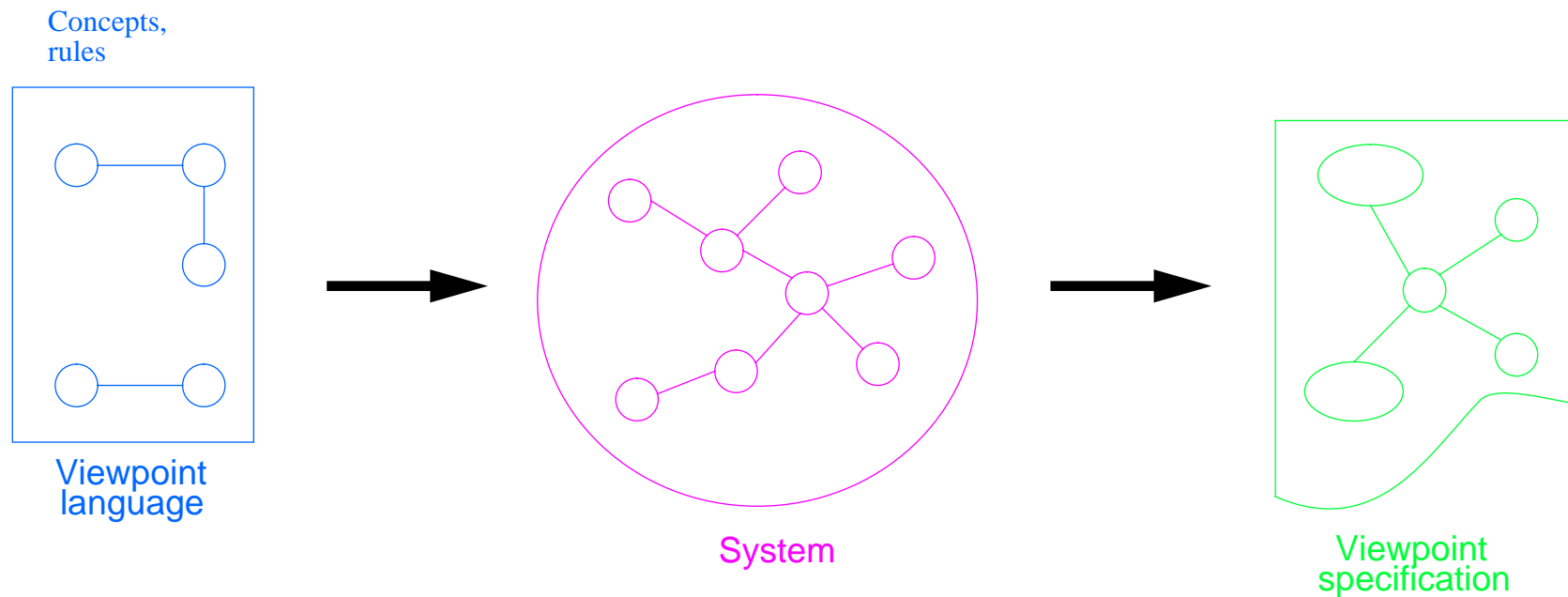
- **concepts and rules for specifying ODP systems**
- **framework for development of ODP standards**
 - **standards for *specification, modelling and programming* languages**
 - ***language bindings (APIs)* for ODP systems**
 - **functional components of ODP systems**



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)

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** - kinds on information handled by the system and constraints on the use and interpretation of that information
- **Computational** - functional decomposition into objects suitable for distribution
- **Engineering** - the infrastructure required to support distribution
- **Technology** - the choice of technology to support distribution
- **Don't equate viewpoints to design process refinement steps!**



Conformance

- Reference point - a potential conformance point
- **Positioned** by computational and engineering languages
- **Specified** in (some combination of) enterprise, information, computational, engineering and technology language languages
 - enterprise specification determines **roles and policies** with respect to the reference point
 - information specification determines **universe of discourse** for the reference point
 - computational specification determines the **dialogue structure**
 - engineering specification determines the **infrastructure**
 - technology language determines how to insert **tester and interpret results**
- An ODP system is one which conforms to ODP standards at all reference points asserted to be conformance points



ENTERPRISE AND INFORMATION LANGUAGES

- for writing requirements in ODP standards



Enterprise language

- **Role:** e.g. service provider, service user, resource manager, agent, artefact
- **Policy:** security, safety, objectives, contracts, codes of practice
- **Resource:** accounting
- **Community:** set of objects with an objective (*contract*)
- **Enterprise actions**
 - incur an obligation
 - fulfil an obligation
 - waive an obligation
 - acquire permission
 - be forbidden
- **OO flavour for alignment with computational viewpoint**



Federation

- Federation: a community of domains
- The need to support federations that maintain domain *autonomy* underpin colours many of the technical aspects of the Reference Model.
- Autonomy principles
 - freedom to join
 - freedom to leave
 - subject to agreed obligations
 - retain local policies, including technology choices
 - no single administrator
- “Open Systems” is all about maximizing domain autonomy without compromising scope for federation



Information Language

- **Concepts, relations**
 - define complex concepts as relations (“is-a”, “contained in”, “owned-by”) between simpler ones
- **Integrity rules**
 - static schema (rules about state and structure at some point in time)
 - invariant schema (independent of behaviour)
 - dynamic schema (possible behaviour)
- **OO flavour for alignment with computational viewpoint**
- **Compatible with methodologies such as Rumbaugh’s OMT**



COMPUTATIONAL LANGUAGE

- for specifying interfaces and distributed algorithms in ODP standards



Distributed Programming is Different

- **TRADITIONAL**
 - Local
 - Sequential
 - Single Environment
 - Fixed Location
 - Single Copy
 - Synchronous
 - Direct
 - Shared
 - Global
 - Complete failures
 - Early Binding
- **REVERSED**
 - Remote
 - Concurrent
 - Diverse Environment
 - Mobile
 - Multiple Copies
 - Asynchronous
 - Indirect
 - Separate
 - Context Relative
 - Partial Failures
 - Late Binding
- Trying to wedge this into a traditional view won't work, hence we need a model for distributed computation



Computational language

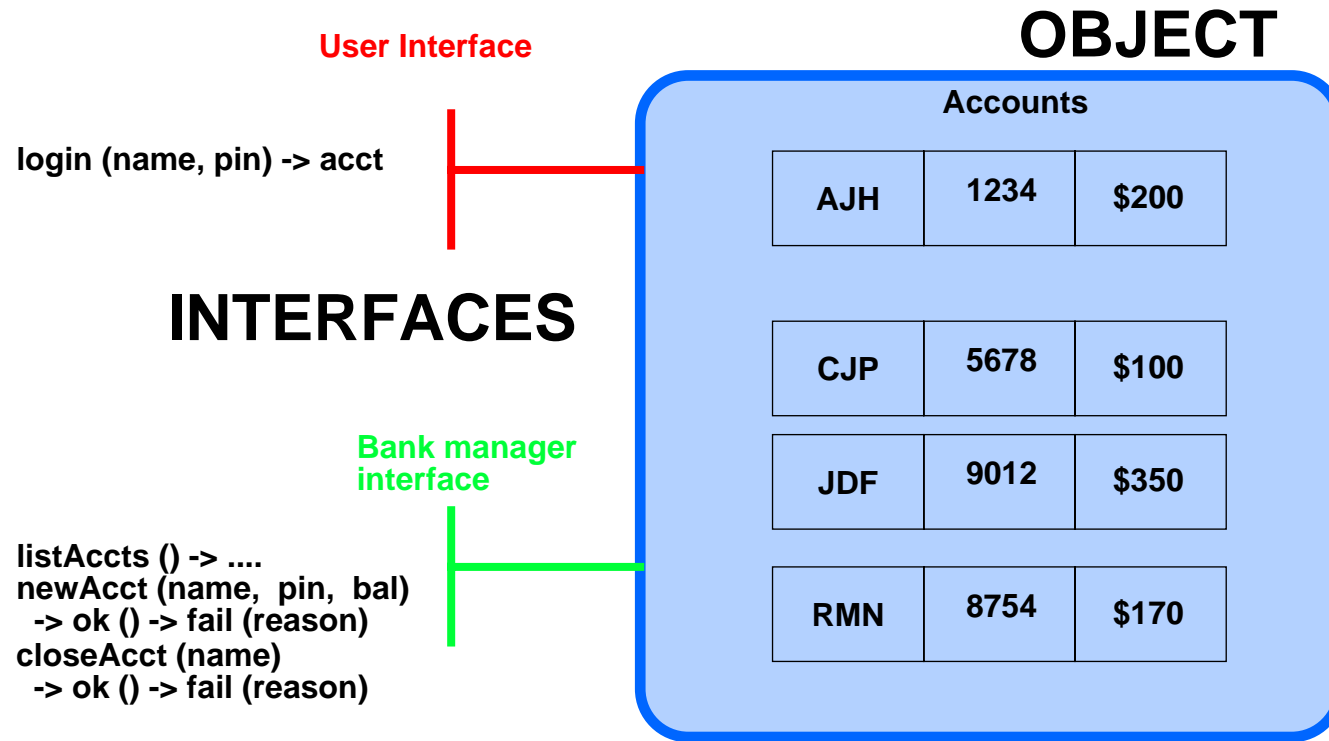
- **Significant level of prescription**
 - dialogue (interaction) structures for *interoperability*
 - activity (i.e. program) structures for *portability*
- **Abstracts over a wide choice of implementation mechanisms**
- **Object-based because encapsulation and separation of service from implementation inherent in object models is well matched to the needs of distributed systems**
- **Strong typing rules for maximum confidence**
- **Good alignment with OMG CORBA “standard”**



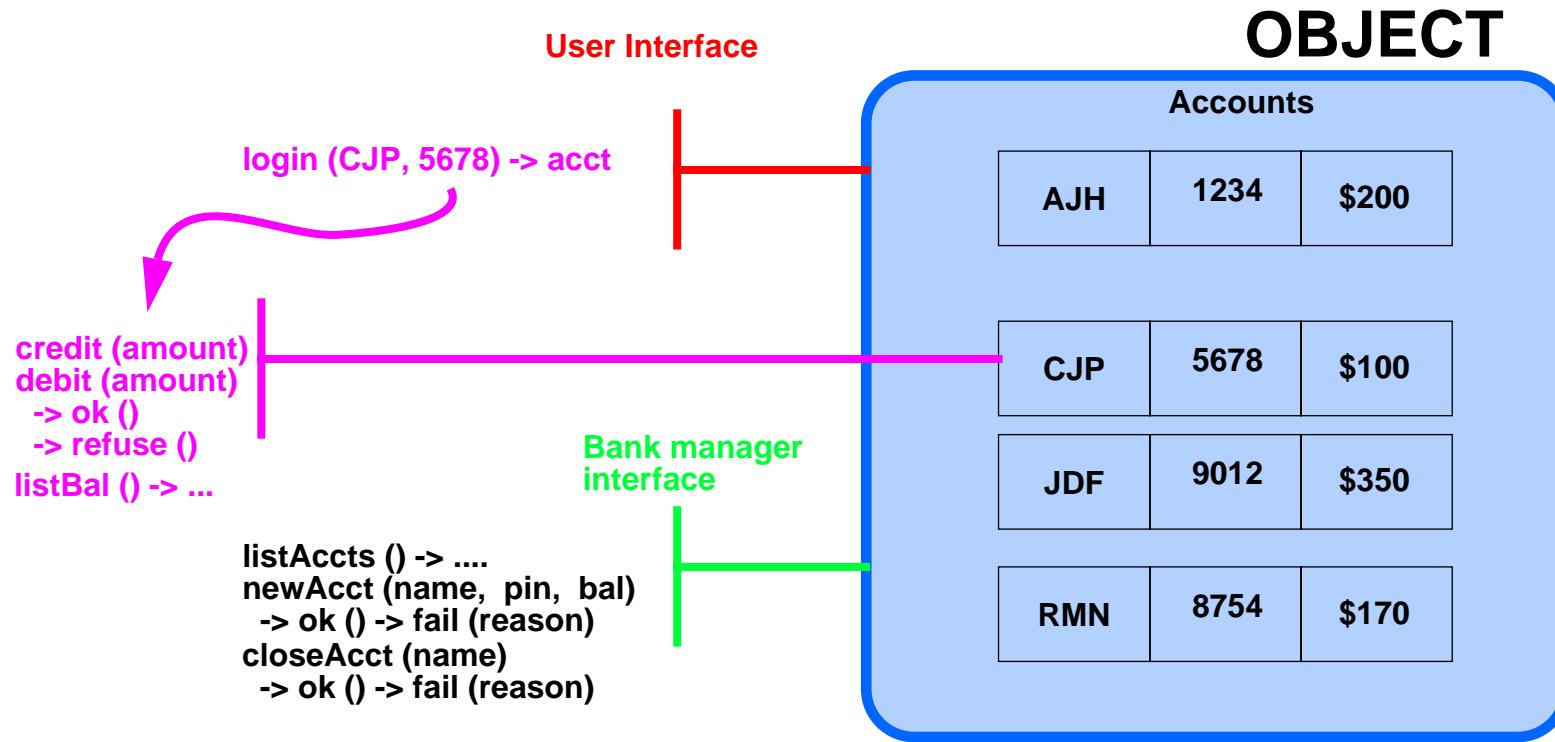
Computational Concepts

- **Object - unit of encapsulation and hence configuration**
 - an object can have several interfaces
 - an object can create interfaces dynamically
 - an object is persistent
 - an object can migrate, fail, checkpoint, replicate, ...
- **Interface - provides a service in a context**
 - an operational interface has operations (methods)
 - operations have a set of terminations
 - a stream interface has flows
 - signal interfaces for event handling view of flows or operations
- **call by sharing - operations, terminations and signals accept interfaces as parameters**
- **Activities - fork / join vs spawn, call, bind, handle termination**

Computational Object Model (1)



Computational Object Model (2)





Types, Trading and Binding

- **Trading** - discovering an interface that provides a service
 - exporters make service offers
 - importers make service requests
 - trading marries imports to exports
 - an ODP function
- **Binding** - knitting together a set of matching interfaces to enable interaction
 - typically **implicit** for RPCs
 - necessarily **explicit** for streams and for RPCs where QoS is to be controlled
 - built into the computational model
- **Type safety** - match signatures to ensure valid interaction
 - “no surprises” rule
 - information types (called “properties”) checked when trading

Binding

- **BIND <binding contract> {set of interfaces}**
 - instantiates a new binding between the set of interfaces
 - binding contract determines pattern of interconnection and choice of infrastructure
 - binding contract specifies types for the participating interfaces
- **Interfaces have roles relative to the binding**
 - client, server
 - producer, consumer
 - binding controller
- **Binding controllers can**
 - add/remove interfaces
 - stop/start information flows
 - monitor events



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
- each operation has **one or more** *terminations*
- 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 interfaces
- Rules for signal and stream interfaces are similar



ENGINEERING LANGUAGE

- for specification of infrastructure needs in ODP systems (viz. block diagrams)

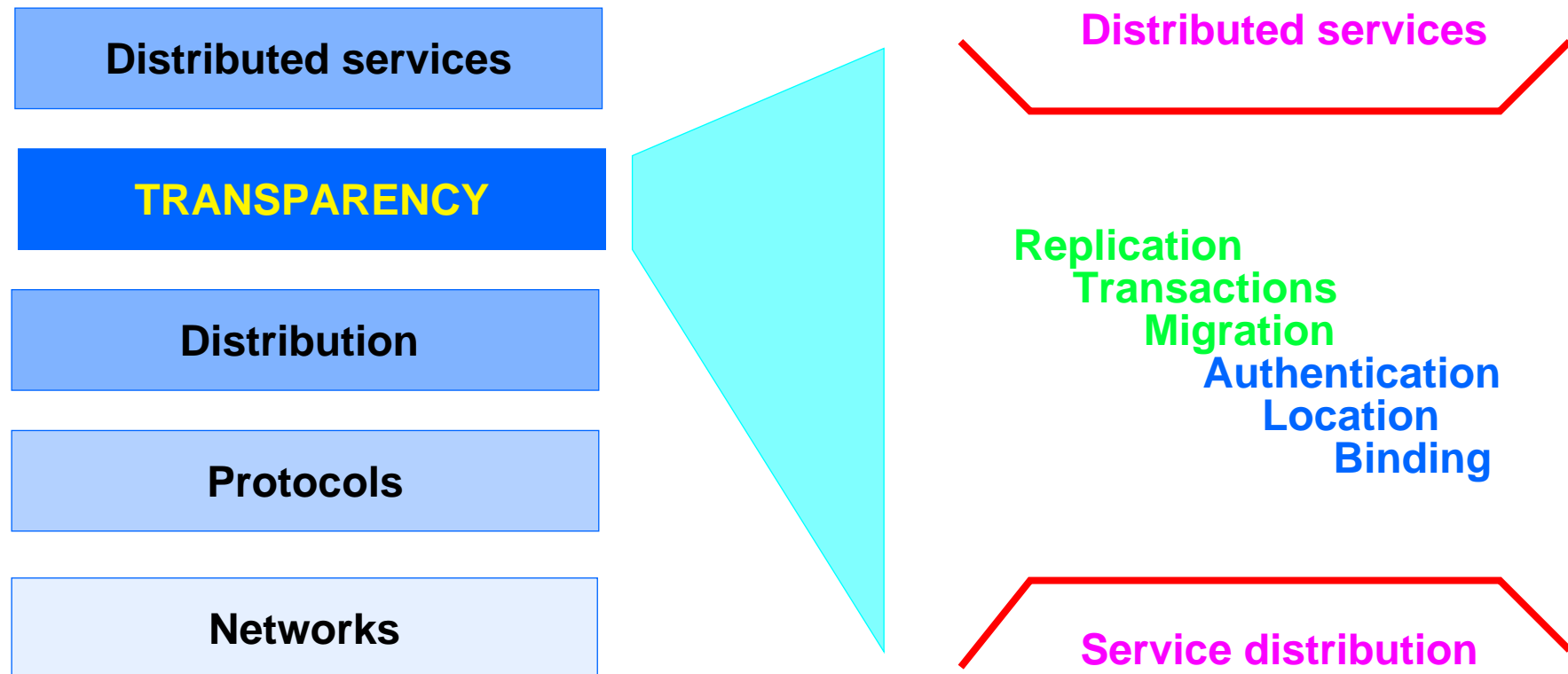


Trade-offs in Distributed Systems

- **Distributed systems engineering is all about trade-offs**
 - **ABSTRACTION** versus **SPECIALIZATION** -the more you hide, the less control you have
 - **CONSISTENCY** versus **AVAILABILITY** - availability implies 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 framework into which they slot**
- **Moreover the framework must accomodate coexistence of alternative parts for the same job**



Engineering Framework





Selective Transparency

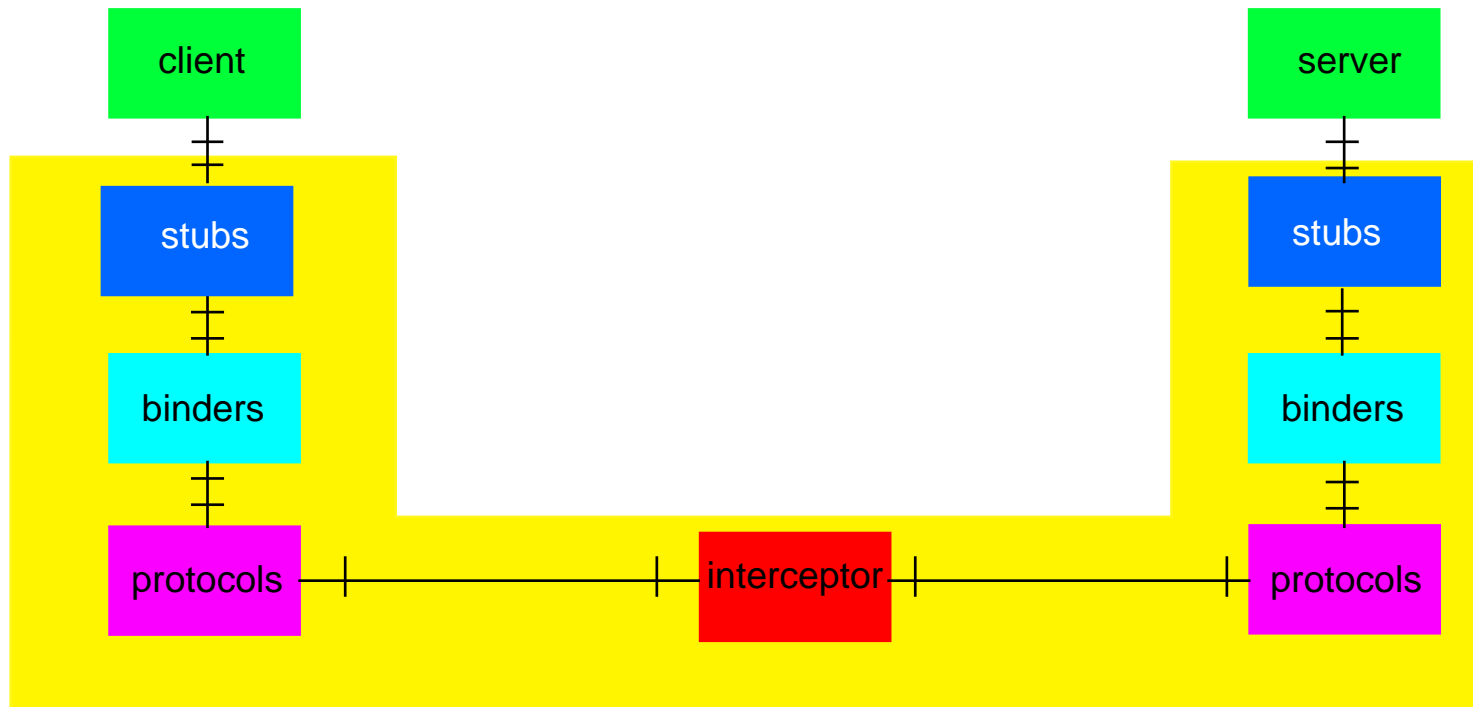
- **Transparency is about hiding irrelevant complexity**
 - **Location** don't need to know where it is to use it
 - **Access** don't need to know how it works to use it
 - **Migration** it can move while you're using it to balance loads or reduce latency
 - **Replication** there may be copies for reliability and/or availability
 - **Persistence** it only gets resources when it needs them
 - **Partial Failure** it always gets to a consistent state
- **The transparency layer supports the functionality of the basic service distribution layer, and adds additional guarantees**
- **extra management functions for controlling transparency**



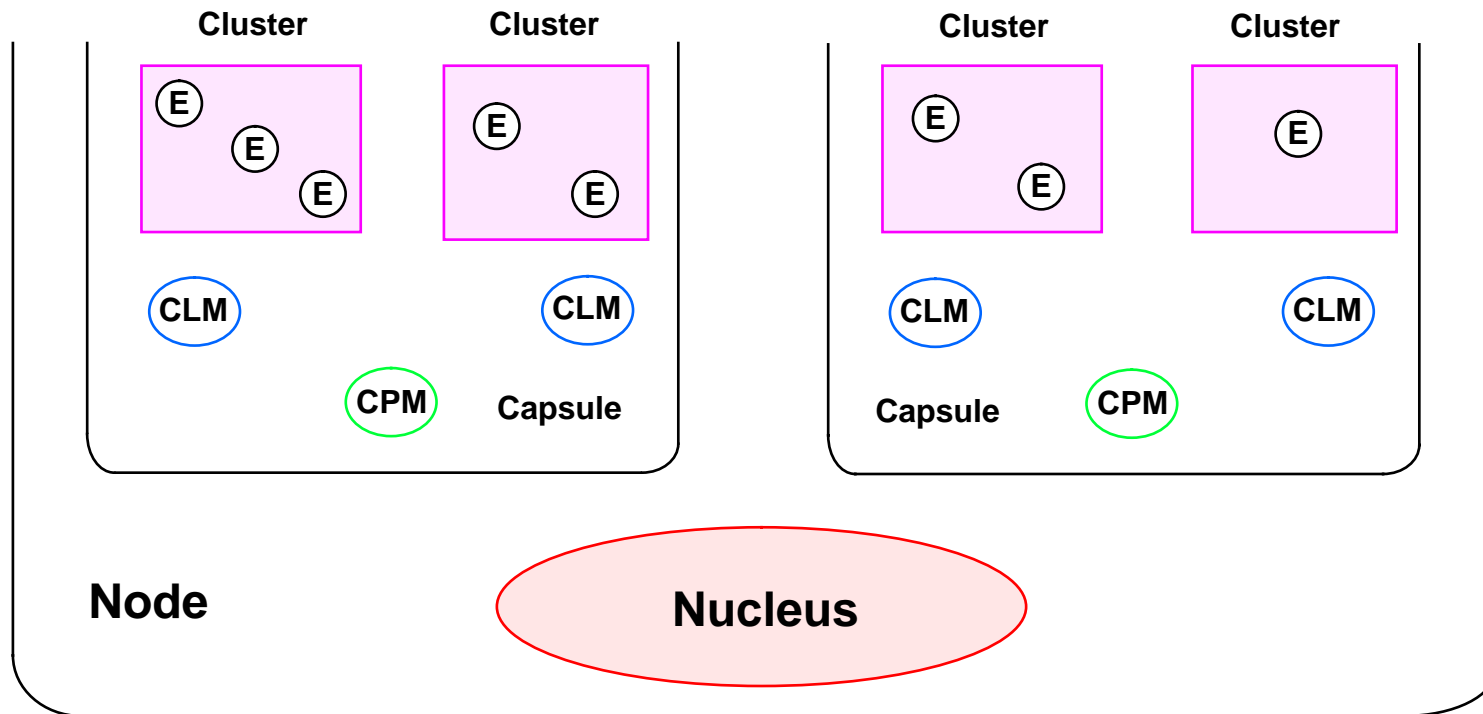
Engineering language

- structures for implementing infrastructures supporting *selective transparency*
- *channels* for communication
 - simple client server
 - multipoint channels
 - stream channels
- *clusters* for activation, deactivation, migration
- *capsules* for resource allocation, protection
- *nodes* for network addressing

Client-Server Channel



Engineering structures





Engineering bindings

- **Stubs marshall interfaces into and from interface references**
- **Marshalling stub calls the nucleus stating any binding constraints**
 - e.g. use of specific protocols
 - e.g. use of non-local identifiers (such as DCE UUIDs)
- **Nucleus creates local supporting infrastructure and invents interface reference**
 - interface reference tells other objects how to bind to the interface
 - potentially complex data structure, especially if federation is involved
- **Stub send interface reference in message to stand for interface parameter**
- **Recipient stub asks nucleus to bind to received interface reference**
 - nucleus creates supporting infrastructure and completes the binding



Technology language

- **Implementable standard: template for a technology object**
- **Implementation Extra Information for Testing**
 - **technology specification in a standard defines a proforma IXIT**
 - **IXIT defines templates and names of interfaces required for testing**



Consistency constraints

- Rules for avoiding inconsistency between specifications
- Structural relationships between computational and engineering specifications



Functions

- **Management (object, cluster, capsule, node, interface reference)**
- **Coordination (checkpointing and recovery, deactivation and recovery, migration, transaction, group, replication, events)**
- **Repository (storage, relationships, types, trading)**
- **Security (access control, security audit, authentication, integrity, confidentiality, non-repudiation, key management)**



Management Functions (1)

- **node management (viz., the nucleus)**
 - threads
 - binding - channel setup
 - capsule manufacture
- **object management**
 - objects manage themselves
 - cluster managers ask objects to terminate, snapshot etc.
- **cluster management**
 - deactivate, checkpoint, recover, migrate, replicate, terminate



Management Functions (2)

- **capsule management**
 - **cluster instantiation, reactivation**
 - **termination**
- **interface reference**
 - **tracking interface references**
 - **enables distributed garbage collection**
 - **initiating use of coordination functions to repair broken bindings transparently (e.g. after migration)**



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**



Groups

- **general model, specific forms of cooperative computation defined by policies for**
 - **distribution of interactions among participations**
 - **collation - correlating requests (e.g. voting)**
 - **ordering - visibility of requests to participants**
 - **fault detection and recovery**



Repository functions (1)

- **Separate out components of repository to enable distribution**
- **Storage function**
 - data storage
 - deactivated cluster repository
 - interface reference fixing done on reactivation
- **Relationship repository**
 - enables structured queries over sets of objects
- **Types**
 - abstract data types for signatures to enable dynamic type checking



Trading

- **Each object has access to a trader**
- **A trader contains**
 - **services offers - type, properties, interface reference**
 - **links to other traders - name, properties**
- **A service offer can be tied to an export controller object**
 - **to monitor imports, to allocate resources**
- **A trader may be optimized for**
 - **speed of look up vs volume of offers stored**
 - **accuracy of offer**
 - **dependability**
- **No structure is forced on traders, to enable federation**
 - **context relative naming is the key**



Interception

- **The trader instantiates an interceptor when a service is imported across the boundary**
 - **this makes interceptors less painful than traditional gateways**
- **An interceptor can**
 - **make a boundary transparent**
 - **impose an administrative boundary**
- **Interception can include**
 - **protocol conversion**
 - **name mapping**
 - **adding administration payload**
- **Intercept at**
 - **node**
 - **LAN**
 - **WAN bridge / gateway**



Security

- **Every object is responsible for its own security**
- **Capsule concept defines the smallest protection boundary**
- **Functions**
 - **Access control**
 - **Audit**
 - **Authentication**
 - **Integrity**
 - **Confidentiality**
 - **Non-repudiation**



Transparency Schemas: Let Compilers and Tools Take The Strain!

- **Exploit abstraction, program in application oriented concepts**
 - most aspects of OO really help, some hinder
- **Simple (pre-processor) extensions go a long way**
 - especially if leveraging an OO language
- **orthogonality - e.g. “dot” and “bar” vs. threads and RPC API**
 - languages minimize complexity without losing scope for optimization
- **declarative - state requirements and policies not mechanisms**
 - point already proven by IDLs and stub generators
 - decouple applications from engineering - ANSA PREPC experience
- **strong type checking for safety and confidence**



Abstraction and Automation in Arjuna

- **Distributed transactional, persistent C++ objects**
- **Application objects inherit from Arjuna class for**
 - **transaction coordinator**
 - **locking**
 - **persistence**
 - **RPC**
- **Stubs generated from class header files**
 - **could also do this from class definition...**
- **Application programmer only has to**
 - **define application objects**
 - **defines “saveState” and “restoreState” methods**
 - **brackets transactions and decide ‘success’ or ‘fail’**
 - **sets locks**
- **Stubs generated from class header files**
- **All the mechanisms are transparent c.f. traditional TP programming**



Federation

- **Large systems are made up of autonomous islands interconnected incrementally**
 - no central authority
 - legacy of old technology
 - conflicting choices of new technology
- **Administrative boundaries**
 - where checks and accounting are to occur at the boundary
- **Technology boundaries**
 - where protocol conversion and data translation are to occur at the boundary
 - **Set up interceptors (gateways / bridges) on demand, when trading**

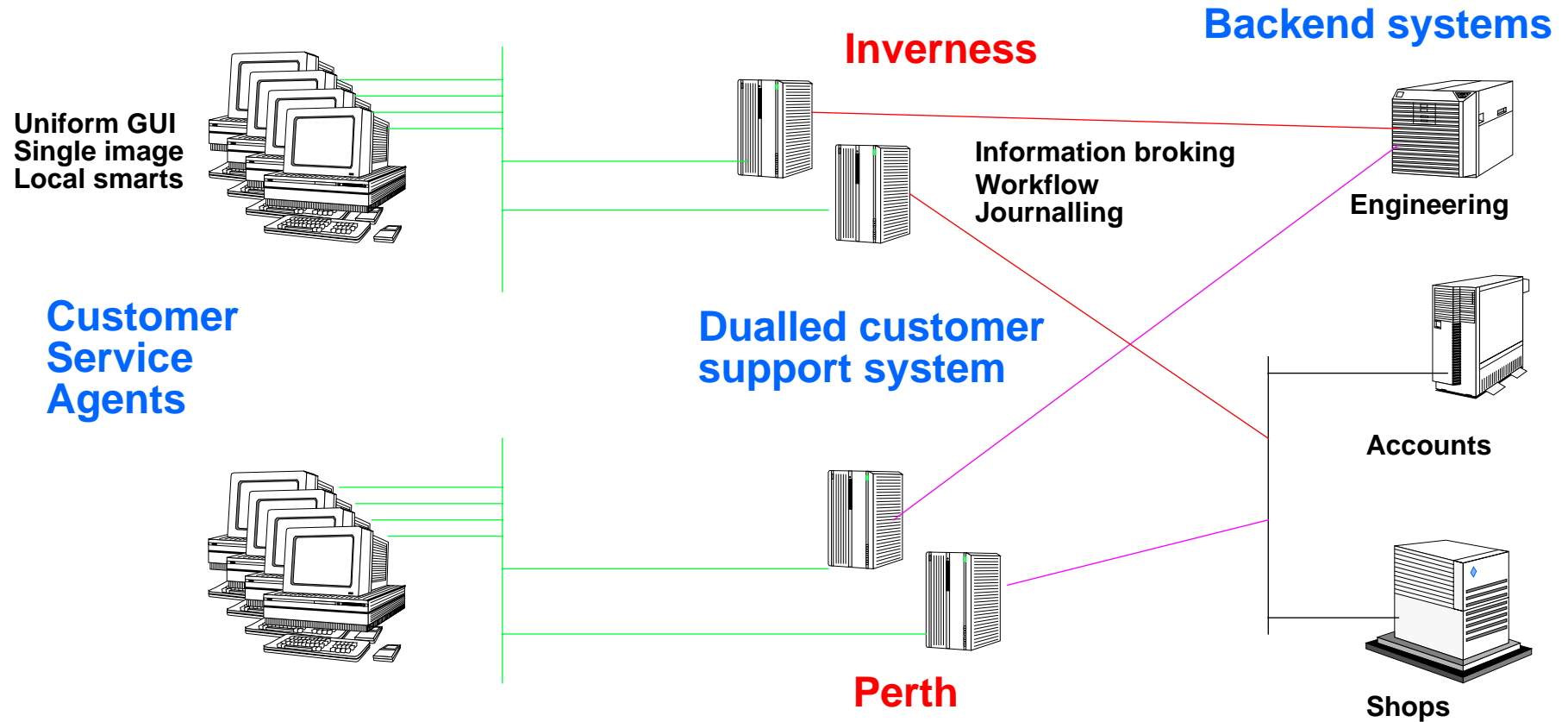


Interception

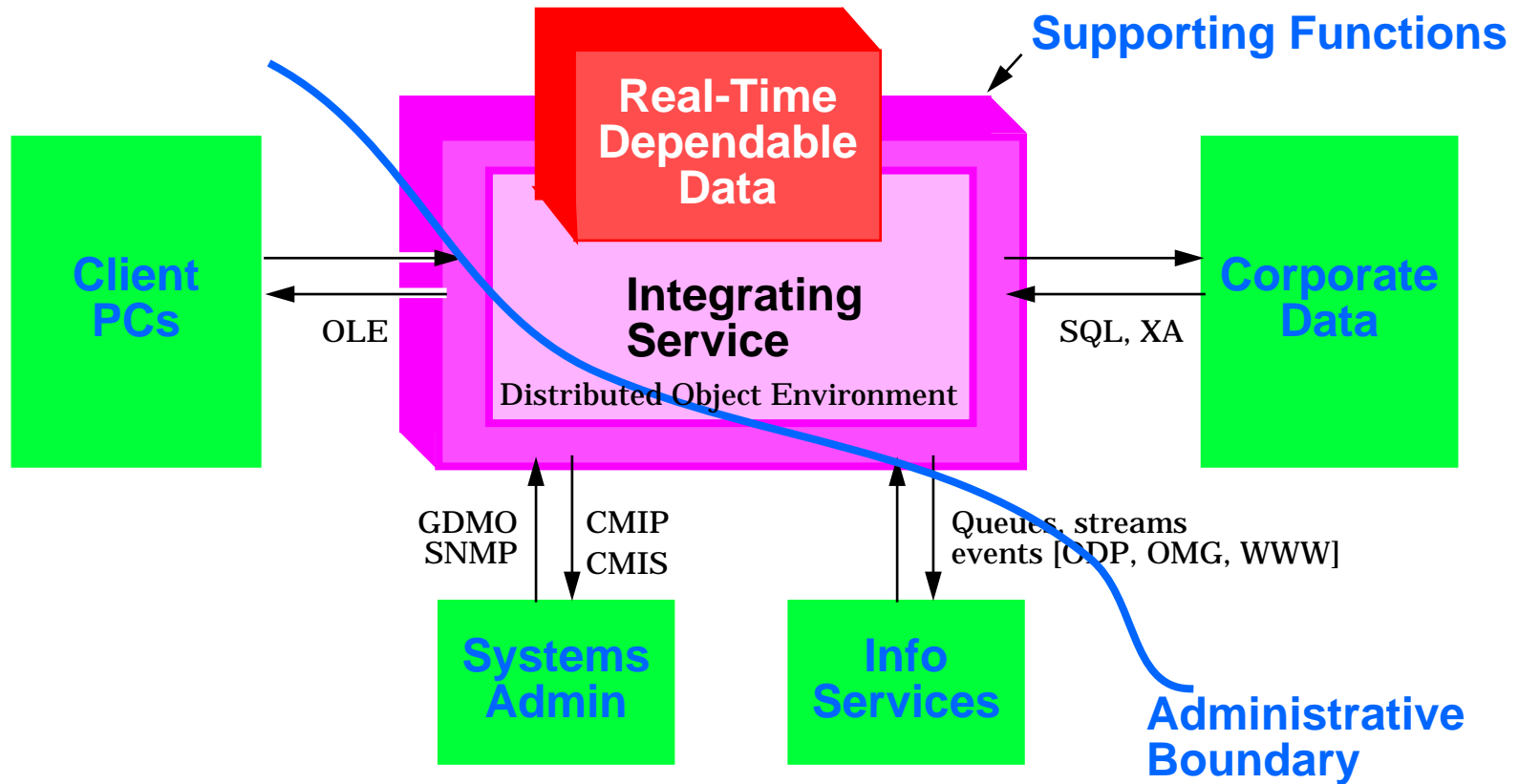
- **The trader instantiates an interceptor when a service is imported across the boundary**
 - **this makes interceptors less painful than traditional gateways**
- **An interceptor can**
 - **make a boundary transparent**
 - **impose an administrative boundary**
- **Interception can include**
 - **protocol conversion**
 - **name mapping**
 - **adding administration payload**
- **Intercept at**
 - **node**
 - **LAN**
 - **WAN bridge / gateway**



An Example ODP System - Scottish HYDRO



Template for an ODP System





Status

- **1988 First Working Documents**
- **November 1992 First Committee Draft**
- **June 1993 Second Committee Draft**
- **February 1994 Draft International Standard**
- **1995 International Standard**
- **Populated via OMG activities**
 - **liaison in place**
 - **function correspondences identified**