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

Engineering model for dependability (Nov 93 TC presentation)

List of author names goes here

Abstract

Need some instructions here.

APM.1087.00.01 **Draft** 2 November 1993

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

Distribution:
Supersedes:
Superseded by:

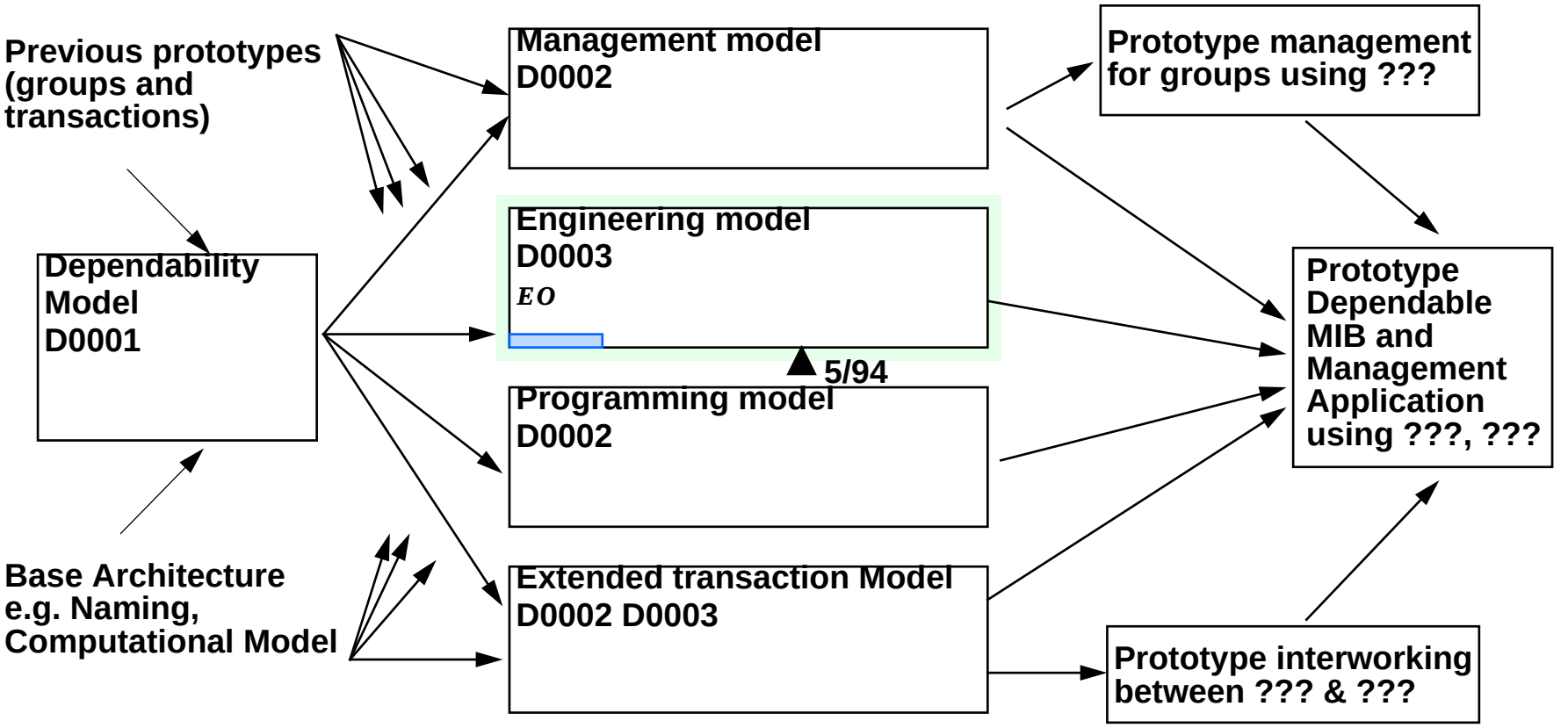
Engineering Model for Dependability

Work in progress

Ed Oskiewicz

Dependability Group

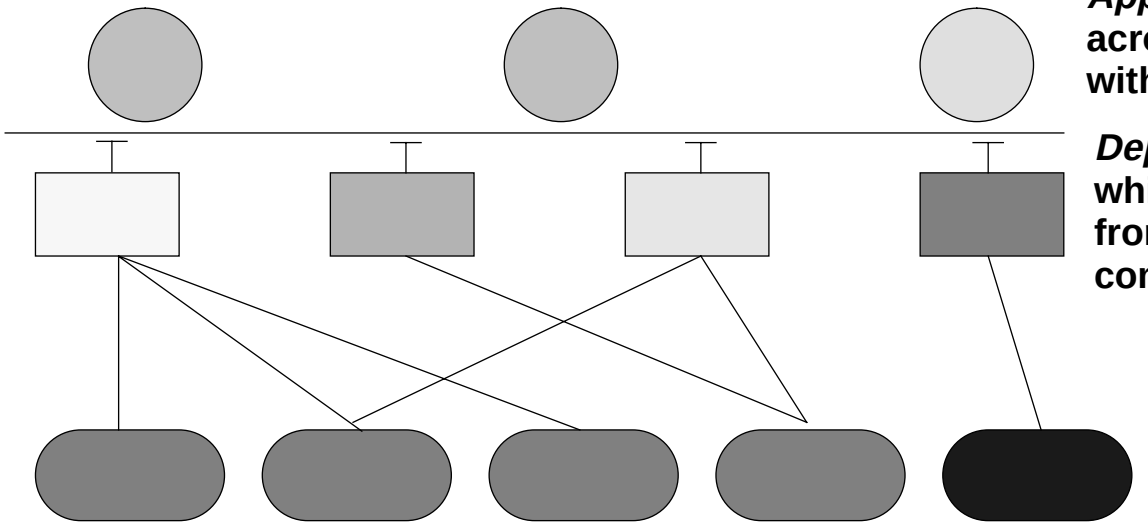
Dependability outline plan



Objectives of this talk

- **To motivate the need for an engineering model for dependability**
- **To describe some of the major issues and concerns**
- **To show why development of the model should be aligned with a credible application scenario**

Structure of the engineering model for dependability



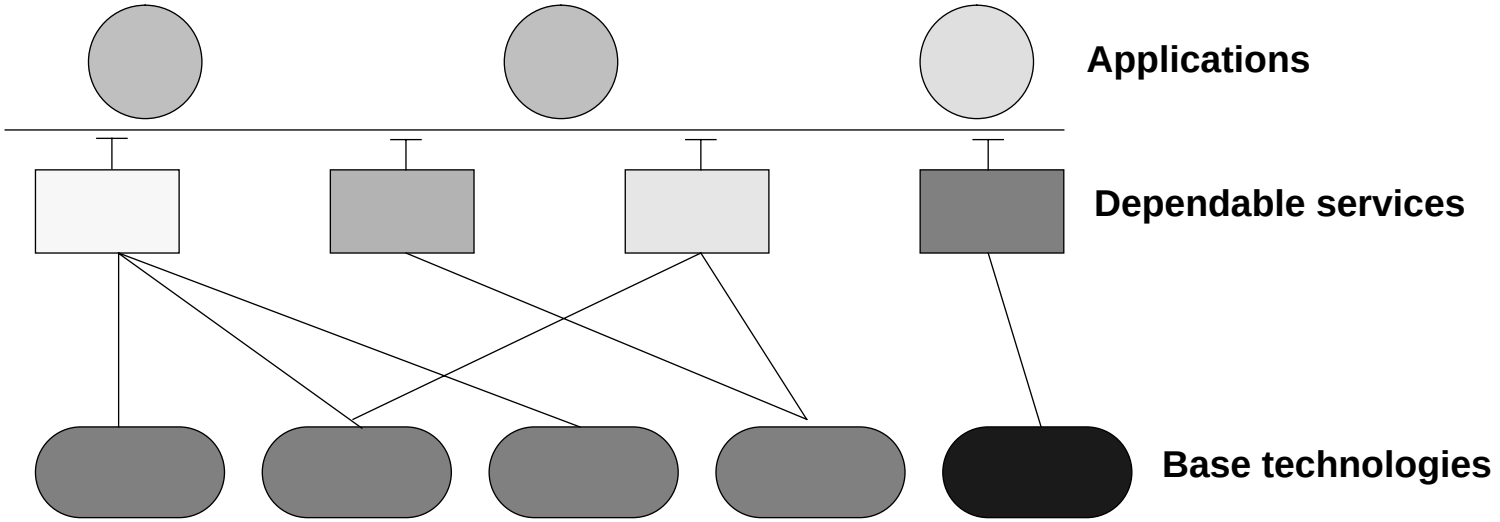
Applications **interact across an infrastructure with:**

Dependable services **which are composed from varying combinations of:**

Base technologies **such as CORBA, DCE**

- **Applications have expectations of other applications and of the infrastructure**
- **Dependable infrastructure services isolate applications from details of the underlying technology**

Satisfying application requirements



- **The engineering model must provide a vocabulary and taxonomy to be able to choose and compare technologies and services**
- **Do applications need sight of the details of base technologies?**

Scope of the engineering model for dependability

- **Concentrate on computationally significant features**
- **Is largely about the engineering of selective transparency**
- **Dependable infrastructure services can transparently enhance dependability**
  - **communications**
  - **applications**
- **Must be able to enhance pre-existing (*legacy*) applications**

Engineering approaches

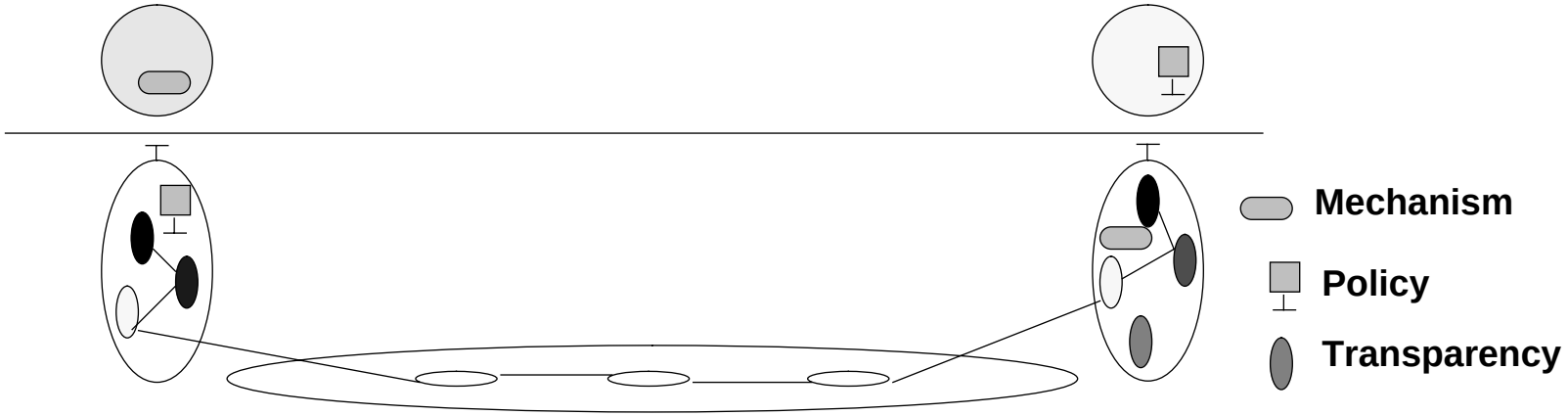
- **Two extreme approaches**
  - ***fault avoidance*, e.g. type checking, formal design methods**
  - ***fault tolerance*, e.g. replication and transactions**
- **These are complementary approaches *not* alternatives**
  - **fault avoidance may someday be able to produce fault-free components**
  - **fault tolerance is necessary to survive unpredicted or external failures**
  - **many techniques are a mixture of fault avoidance and tolerance**
- **Fault tolerance is the initial emphasis of the engineering model for dependability**

Phases of fault tolerance

- **Fault detection: *observing and testing expectations***
  - **can observe the passage of time**
    make inferences about the non-occurrence of events
  - **can exploit redundancy**
    implict redundancy requires semantic knowledge
    explicit redundancy exploits extra storage/processing/communications
    - **this gives a basis for a taxonomy**
    - **can be provided in space and/or time**
- **Fault recovery: *re-establishing expectations***
  - **about management and reconfiguration - will be covered in later tasks**
  - **engineering model for dependability must encompass interaction with management**

Transparency

- **Compensating for, or masking, unwanted aspects of an interaction**



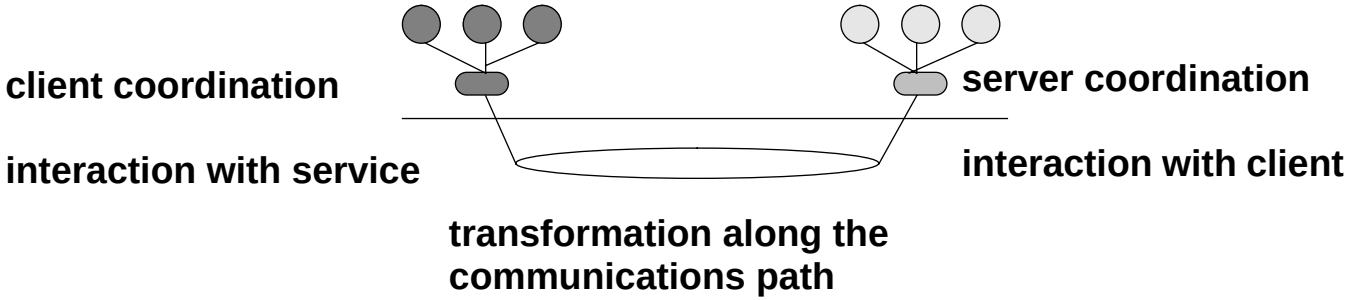
- **The engineering model for dependability must define standard interfaces, composition rules and permit standard configurations**
- **There are also equivalence rules and compatibility problems**

Implementation options

- **Dependability may be provided transparently or non-transparently**
  - **clients and servers need not have the same view**
- **Non-transparent users are able to influence the provision of dependability**
  - ***indirect*** **- can adjust management parameters**
  - ***direct*** **- can actively participate in dependability processing**
- **Options for the placement of dependability processing**
  - **in the application**
  - **in the local infrastructure**
  - **along the communications path**

Structure of a dependable interaction

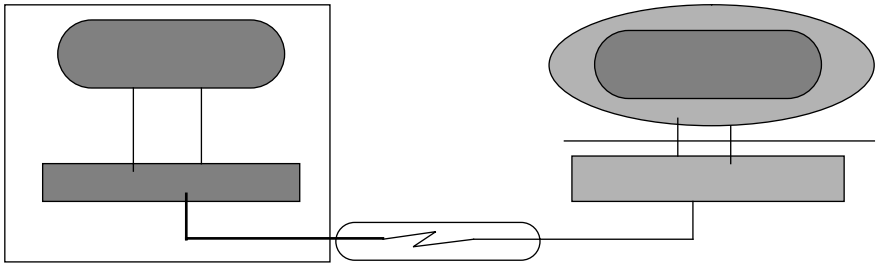
- **Five distinct activities**



- **The client view**
  - **able to pre-select most suitable service**
  - **can detect server failure and initiate corrective action**
- **The server view**
  - **has less control over invokers**
  - **must detect and protect itself against client failure**

Making legacy applications dependable

- There are fewer opportunities to engineer entirely from scratch
  - legacy applications *are* designed without regard to distribution
  - dependability ehancement must be transparent and externally applied
  - Message interception requires no platform changes
  - Encapsulation enables the application to migrate to a new platform



Application bonded to its platform

Application encapsulated above a nucleus

- Can only achieve so much - inherent limitations of original application
- Many naming issues

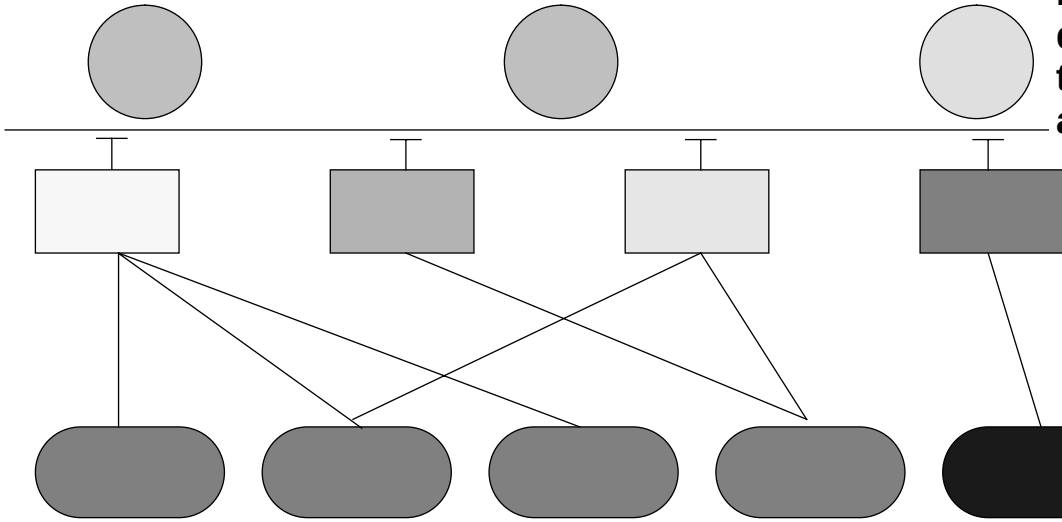
Interaction between legacy and non-legacy applications

- **Legacy clients**
  - **determine interface corresponding to client usage**
  - **perform trader interactions**
  - **cope with opening, interacting with, closing service**
  - **assign a name which the service can use for e.g. charging**
- **Legacy servers**
  - **determine interface corresponding to server functionality**
  - **detect/manage conflict between multiple clients**
  - **assign suitable client names**

Aligning the engineering model with a scenario

•

Summary



Enable *application requirements* for dependability to be satisfied transparently, accomodate legacy applications

Enable *dependable services* to be defined, compared and classified

Incorporate current and future *technologies*, show how to compose them into services

- **The scenario keeps this grounded in credible, realistic requirements**