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

Application of the Architecture: a scenario

The ANSA Team

Abstract

The scenario described in this document is intended to act as a focus in the application of the Architecture.

APM.1041.00.05 **Draft** 2 August 1993

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

Distribution:
Supersedes:
Superseded by:

Application of the Architecture

August 1993

- **Generic scenario**
- **Prototype**
- **Performance and Timeliness**
- **Dependability**
- **Federation and Tools**
- **Technology options**

Need for a Scenario

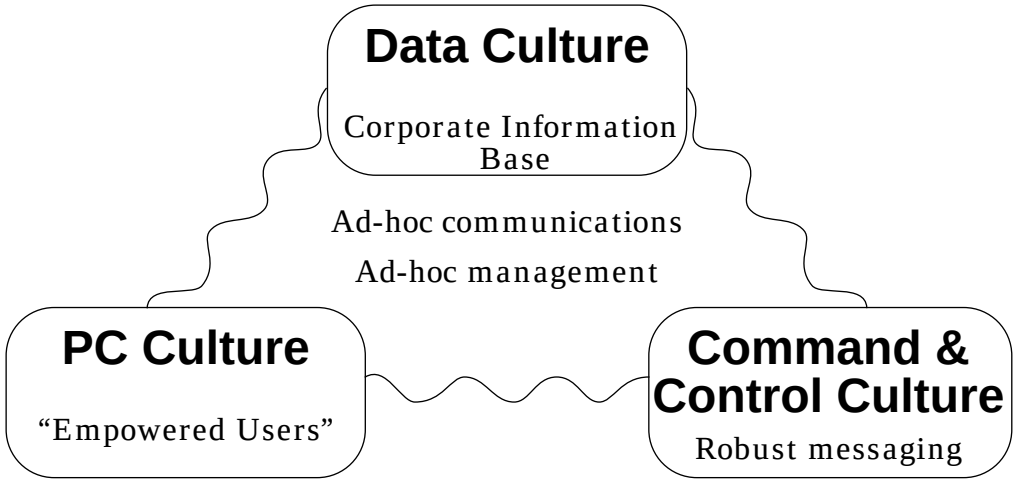
- **There is a need to deliver tangible, practical benefits to sponsors**
  - **required in workplan**
  - **scenario**
- **Apply the Architecture to a real, widely occuring problem**
- **Scenario will:**
  - **apply the Architecture**
  - **animate the architectural approach**
  - **demonstrate the soundness of the Architecture**
  - **feed back experiences**
- **We will not build a product or a complete distributed system**

Generic scenario

- **Benefits**
  - **tackles common relevant problems for each sponsor**
  - **enable/accelerate individual sponsor's product developments**
  - **allows incremental contribution from sponsors**
  - **cuts time to market for sponsors' products**
  - **identifies Microsoft weaknesses**
- **Focus**
  - **build on emerging distributed systems technology**
  - **integrate existing (legacy) systems**
  - **bring architectural and theoretical work into the real world**

Scenario Background – IT Culture Integration

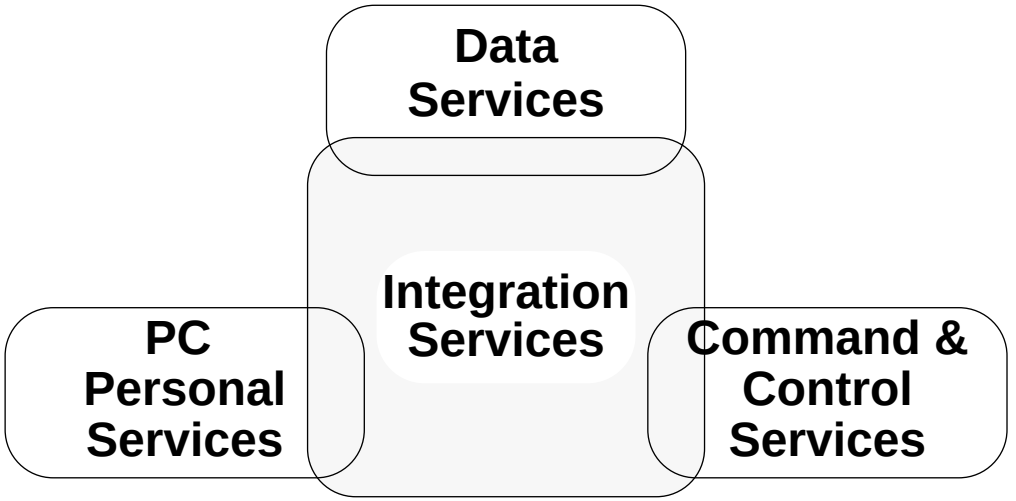
- **Three important, and quite separate Information Technology "user cultures":**



- **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**

Use ANSA to integrate the cultures

- **Apply the Architecture: provide the integration services:**

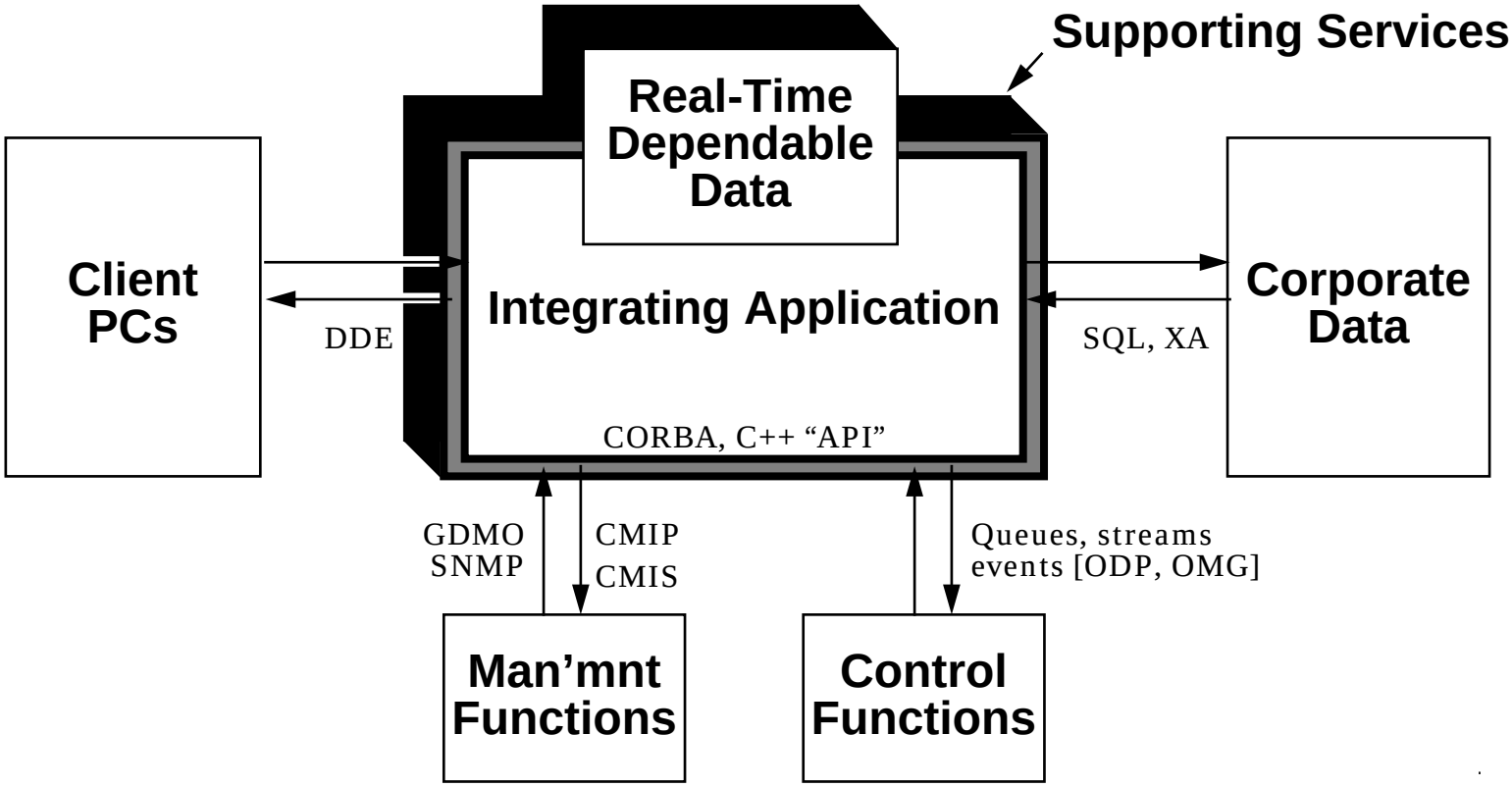


- **Make services in each culture available to services in other cultures**

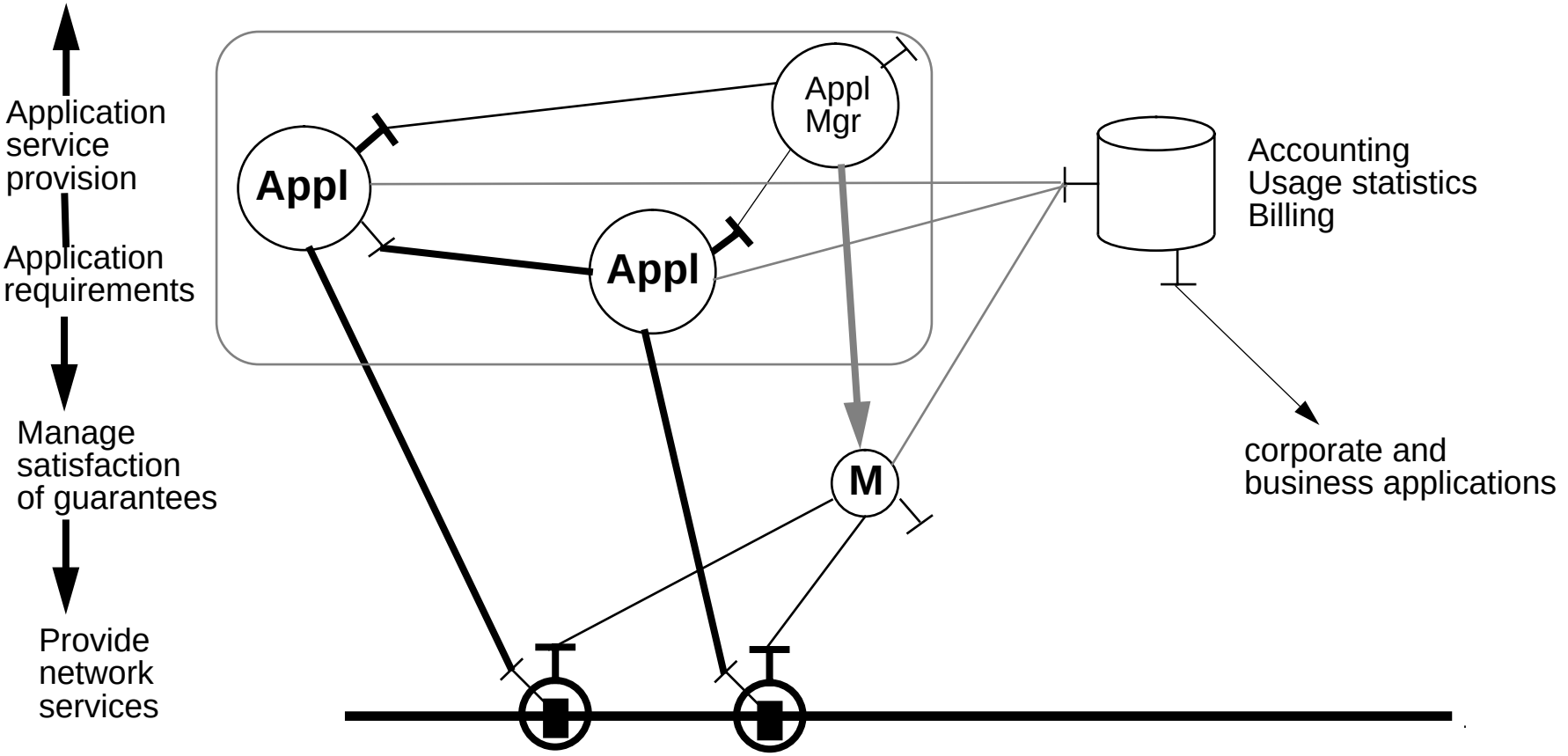
Integration Requirements

- **Any solution to the problem of integrating the IT cultures must use standards at culture boundaries**
- **It must allow evolution, i.e. be**
  - **highly reconfigurable**
  - **scalable**
  - **functionally extensible**
- **It must also be easily managed and be dependable**

Prototype

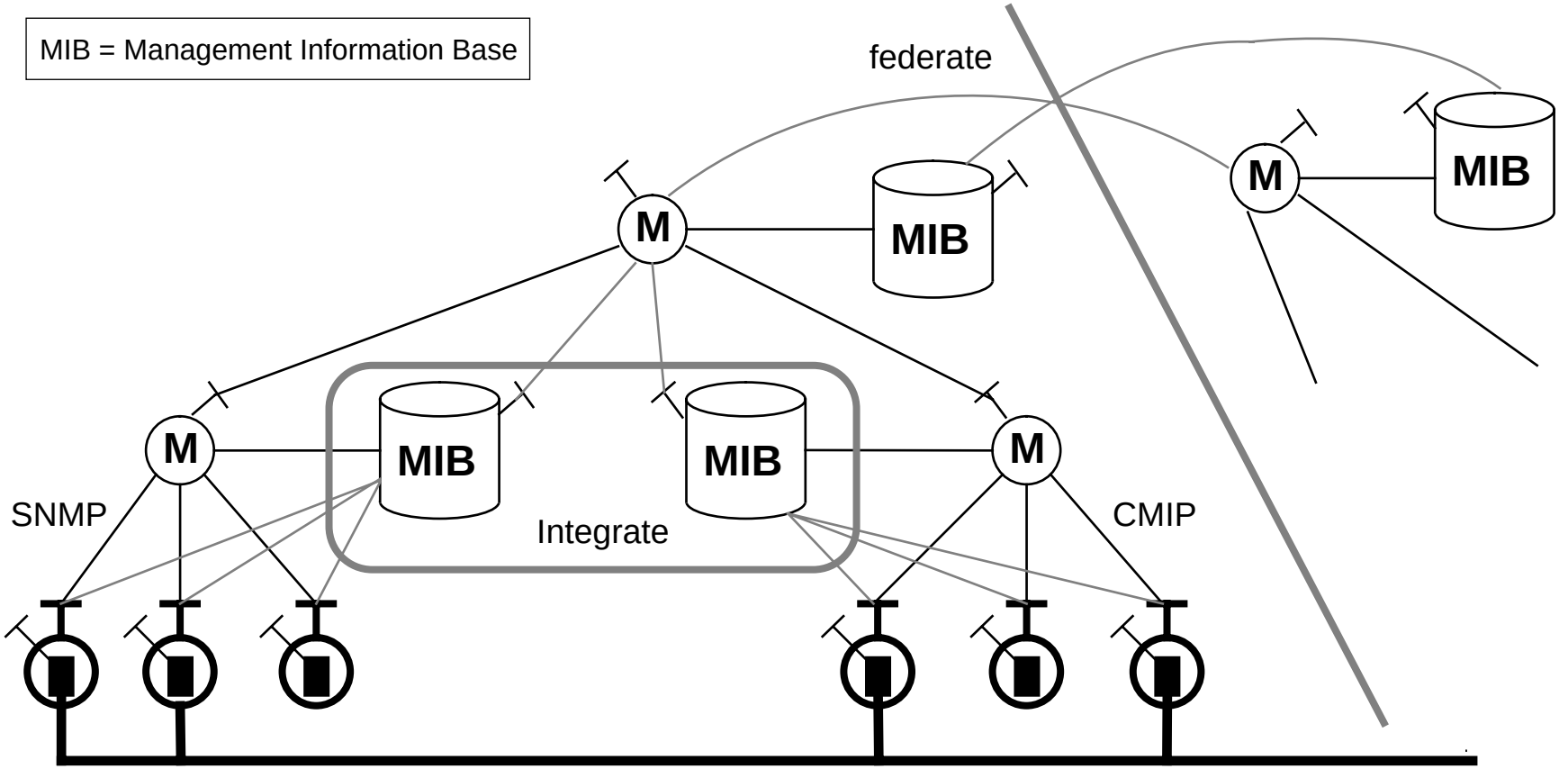

Supporting Services
Real-Time Dependable Data
Client PCs
Integrating Application
Corporate Data
DDE
SQL, XA
CORBA, C++ "API"
GDMO
SNMP
CMIP
CMIS
Queues, streams
events [ODP, OMG]
Man'mnt Functions
Control Functions

Application and network management

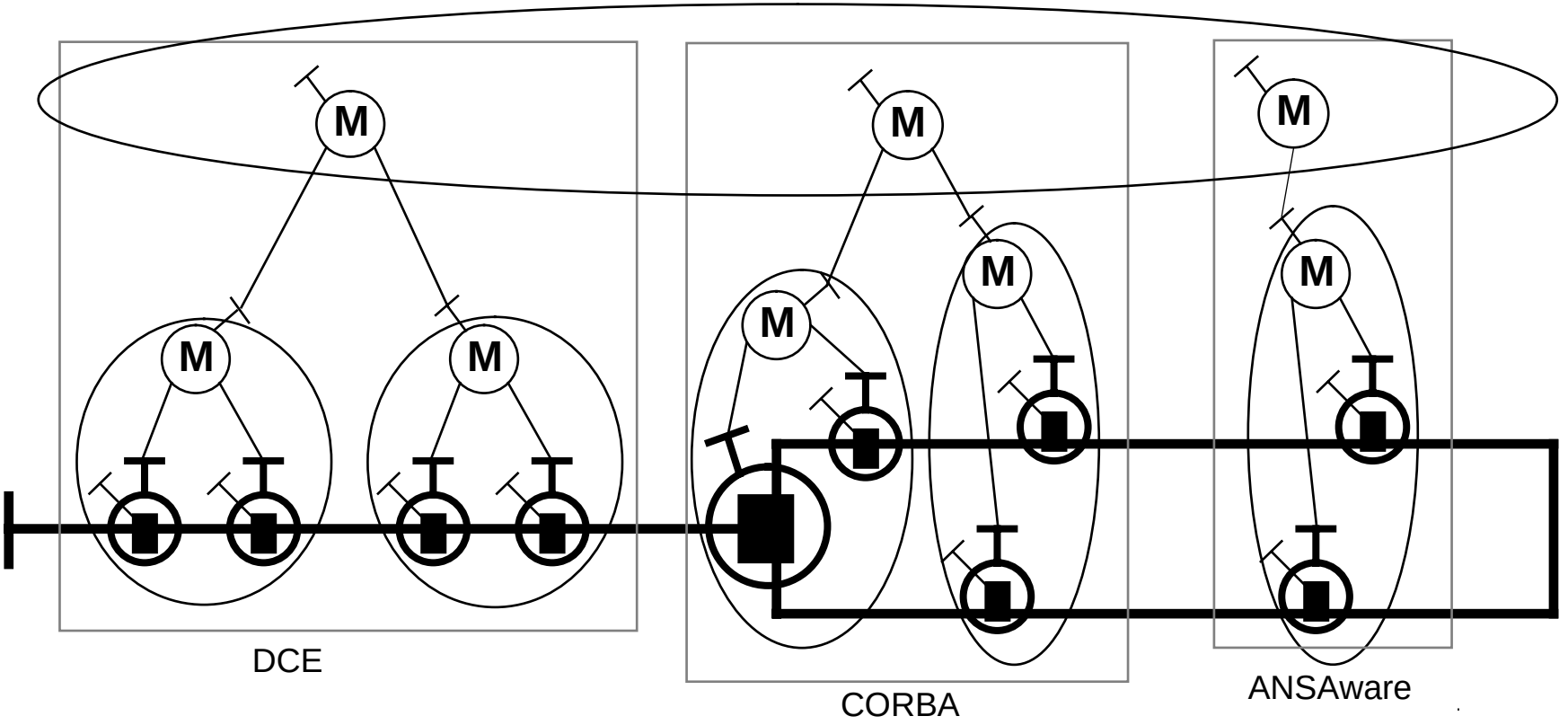


Prototype: Management components

MIB = Management Information Base



Prototype: Physical Structure



What Services are we going to manage

?

Offers please!

Prototype characteristics

- **Based on service provision and use**
  - **service contracts are negotiated and agreed**
  - **manage infrastructure and application services**
  - **services are managed to achieve agreed quality of service**
  - **services are replicated for performance and dependability QoS**
- **Real time services managed by non real time services**
- **Failure resilience built in**
- **Integration of existing transaction processing services**
  - **data integrity and consistency in the face of concurrency**
- **Include heterogeneous existing systems (e.g. legacy databases)**
- **Provide monitoring and debugging support**
- **configuration management**

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 the prototype application**
  - **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:**
  - **domains, resource pools, scheduling points, choice of scheduling policies**
  - **QoS guarantees (deadlines, criticality, etc.)**

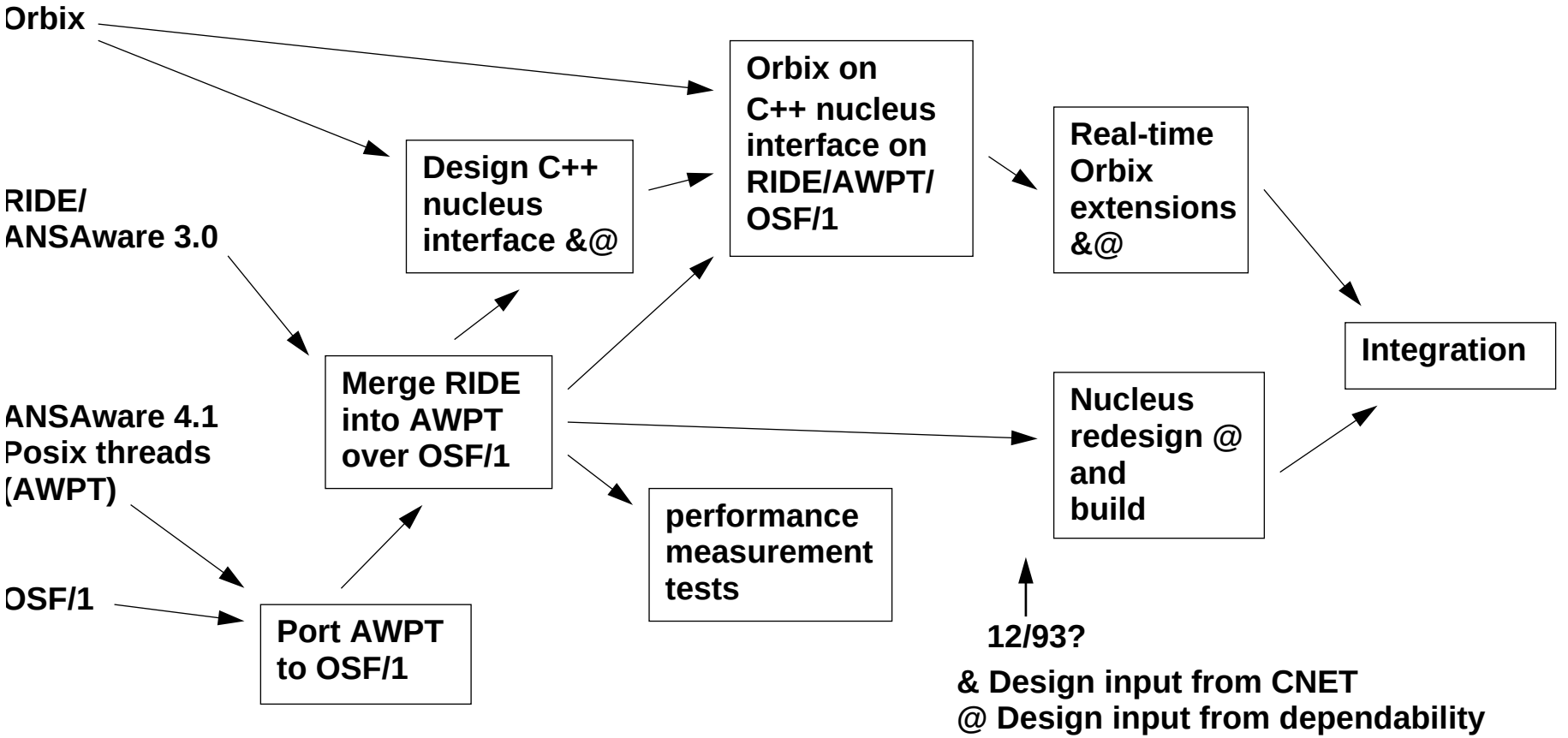
Performance Technology Choices

- **Phase III policy is to base prototypes on next generation products**
- **Real-Time Operating Systems:**
  - **OSF/1, NT, HP-RT, Chorus, Wanda, Solaris II [we have OSF/1]**
- **Distributed Systems Environment (DSE):**
  - **ANSAware/RIDE, Orbix (CORBA-C++ intercept) [we have Orbix]**
- **Proposal:**
  - **Enhance Orbix with real-time capabilities and map onto OSF/1**
  - **Use ANSAware and RIDE tools and engineering to fill in the gaps**
  - **Use Posix real-time threads for portability**
- **Implementation choices for language abstractions**
  - **Pre-processor**
  - **Class hierarchy**
  - **Language extensions**

Performance Design

- **Computational - RT model for application programs**
  - **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

- **It is the SERVICES which the MIB provides which must be dependable**
  - **you needn't make the database itself dependable**
  - **access must be efficient**
  - **integrity must be maintained**

- **Many implementation possibilities for a distributed MIB**
  - **heterogeneous legacy systems (= heterogeneous TP models)**
  - **newly implemented systems**
  - **distributed across the objects themselves**

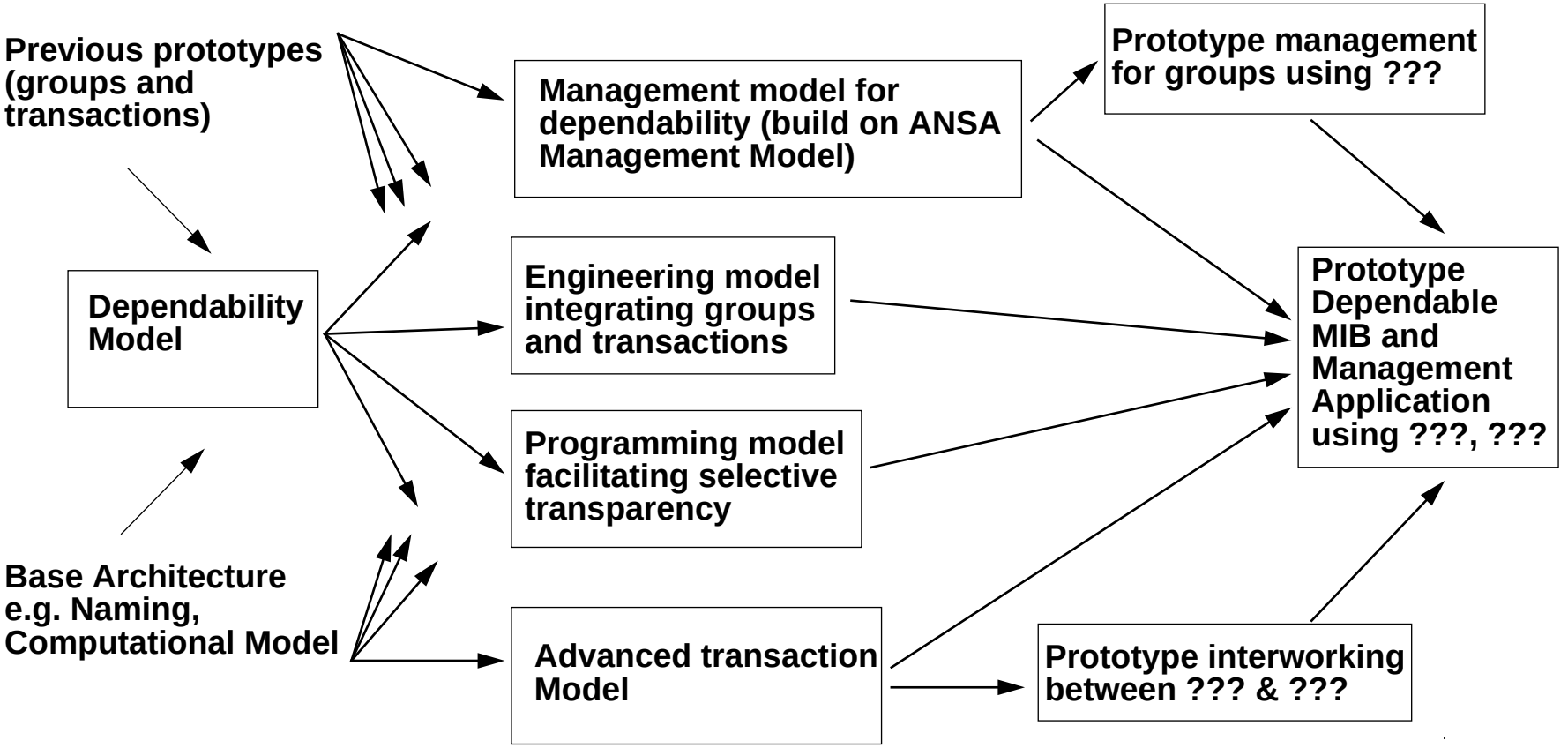
Management Application

- **Accesses/manipulates the MIB, with consistency across multiple MIBs**
- **Interaction with transparency engineering**
- **How to make CMIP, SNMP based applications dependable**
- **The system must be managed for dependability**
  - **Initialisation and configuration of applications**
  - **failure detection/treatment, error passivation and reconfiguration**
  - **data logging for statistical analysis**
- **System management must itself be dependable**
- **What does it mean to cross boundaries?**
  - **who is responsible/accountable for dependability?**
  - **To what extent may management applications and MIB's cross boundaries?**

Transparency engineering

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

Dependability outline plan



Dependability Technologies

- **The architecture must be general but prototype makes pragmatic choice from available technologies**
- **Sponsors have specific requirements which will guide technology selection**
- **Your help (suggestions and offers) are welcome**
- **More detailed implementation strategy will be derived once technologies are decided**

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**
- **Tools to generate interworking code from service contracts**

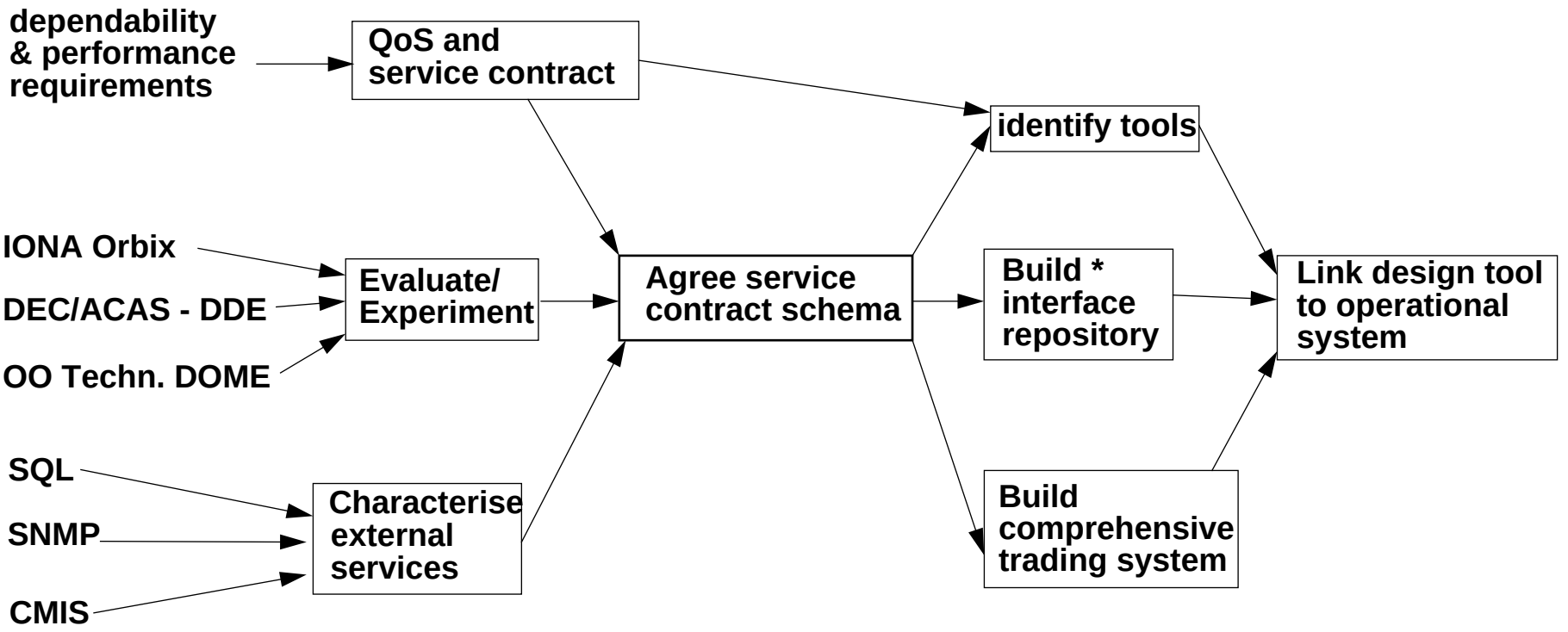
Federation and Tools
- service contracts (1) -

- **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 in any epoch to help 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

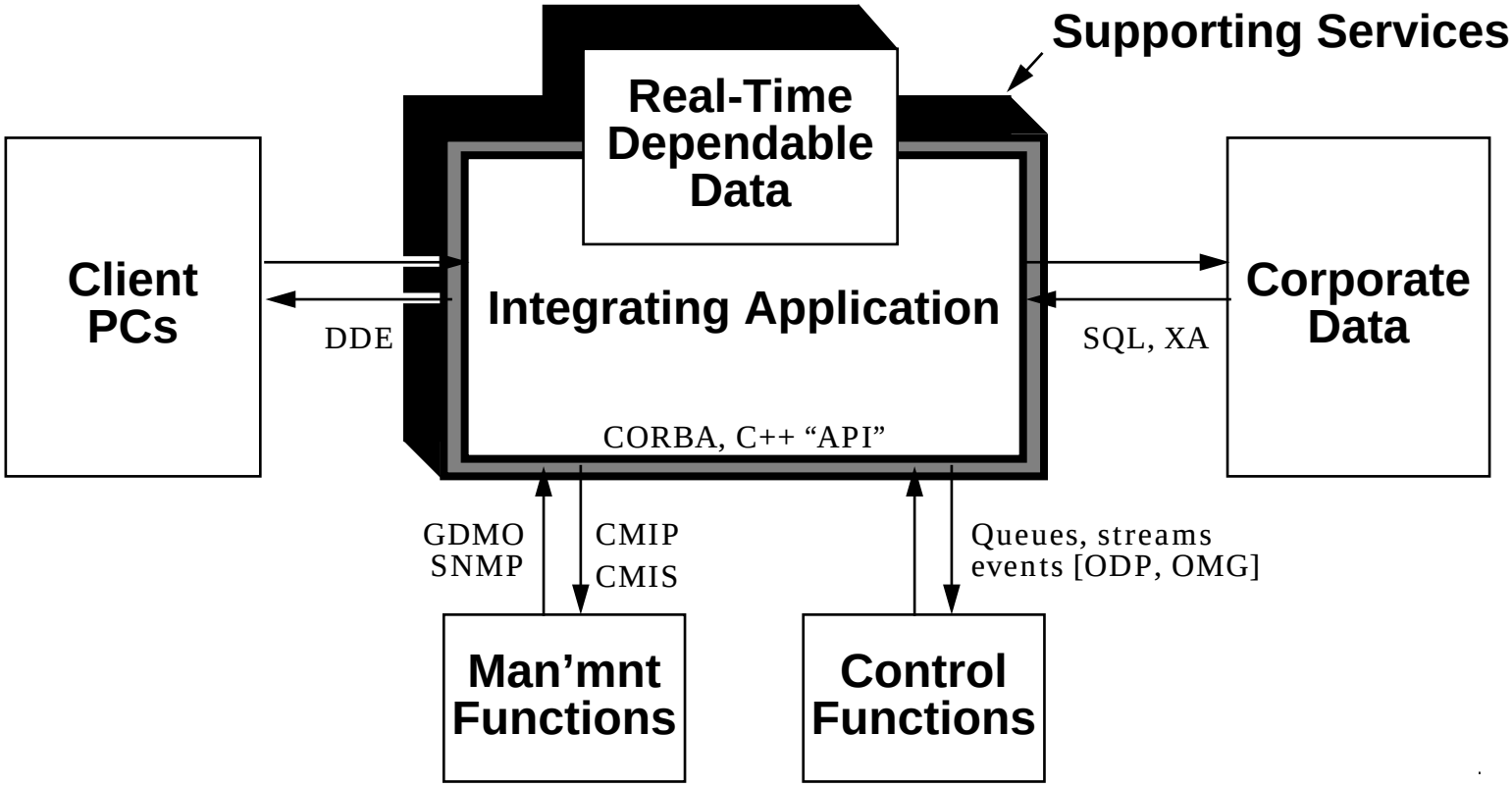


We are not changing the workplan

This is the scenario required by the workplan

How do you want to be involved?

What we will build for you



1 Introduction

This paper describes how the Architecture can be applied to a real world situation. The application of the Architecture increases the confidence in the Architecture and acts as a focus for the work in the reporting groups.

The text in this document accompanies the set of slides which are appended.

1.1 Audience

The audience for this paper is the Phase III Technical Committee.

1.2 Contents

This paper consists of 8 chapters

Chapter 2 describes the need and the way in which a scenario meets this need.

Chapter 3 gives an overview of the generic scenario, setting out its necessary properties.

Chapter 4 focuses the scenario by proposing a specific network and application management scenario. This specific scenario is not the only possible one; others can be suggested. To retain sufficient focus, the number of specific scenarios which are worked on at once should be limited (to one?).

Chapter 5 describes the way in which the work in the area of *performance and timeliness* relates to the scenario. It sets out how the scenario assists the explanation of issues in that area of work. It also sets out what specific technologies are required in the scenario.

Chapters 6 and 7 describe the same for the work in the areas of *dependability*, and *federation and tools* respectively.

Chapter 8 sums up the technological requirements set out in the previous three chapters, in terms of technology options in the generic scenario.

2 A scenario

2.1 Need for a scenario

There is a need to

1. increase the credibility of the Architecture
2. focus the work in the areas of performance, dependability, federation and automated transparencies.

2.2 Meeting the needs

The needs above can be met by means of a scenario.

The purpose of the scenario is to

1. apply the architectural principles set out in APM.1000
2. increase our own and the sponsors confidence in the soundness of the architecture
3. feed back experiences and so refine the architectural principles

The purpose is NOT to:

1. build a distributed system
2. bring a product into a market
3. integrate beyond a reasonable demonstration of principles

2.3 Benefits

A well chosen scenario can address issues which are relevant to all (most) of the sponsors. It will be up to the sponsor companies to determine the extent to which this particular proposal meets their needs and to comment on the areas which the scenario addresses.

Since the scenario is generic, it will continue to protect the IPR's of the sponsoring companies. To ensure focus, the scenario is specialised for network and application management. A different specific scenario (e.g. ATM and video on demand) can be made to fit with the generic structure if required, thus catering for different needs.

Sponsors will be able to incrementally contribute to the scenario, either by placing new demands on it and/or by providing new technology which requires to be integrated with the scenario.

The scenario can act as an advanced development which helps scout a terrain unknown to sponsor development teams. The experiences can be used by sponsors to cut lead time to advanced products.

2.4 The real world connection

The aim is to base the scenario on emerging technologies, many of which will be brought in via sponsors or bought. Often, a decision to use a particular technology will be based on what is available or provided first.

As a consequence many engineering trade-offs will be made. The intention is to label all parts of the prototype within the scenario with a degree of architectural soundness. The way in which this is to be done is under investigation. The aim is that it should be possible to understand what parts of a prototype are true reflections of the architecture and which parts are “quick fixes” put into place because a working system is expected within a certain deadline. We seek to avoid confusion between the Architecture and a system built using the architectural principles.

The scenario and the Architecture are however expected to be kept roughly in step. Thus, the experience from the scenario must be fed back into the Architecture. Sufficient effort will be reserved to ensure this is possible.

2.4.1 How much implementation?

The scenario is not intended solely as a focus for the development and integration of software prototypes. Where appropriate, the scenario will also be used to focus the more abstract architectural and theoretical work.

3 Generic scenario overview

3.1 Background

The background behind the scenario is the realisation that there exist three principal information technology user cultures:

- a data culture
- a messaging culture
- a personal computing culture

Each culture has its own techniques for distribution and application integration. They meet different needs – no single one can replace the others and each existing technology base has many vested interests. The resulting heterogeneity is here to stay.

3.2 Integrating computing cultures

Rather than attempting to replace these cultures, *the architecture should be applied* to integrate the services offered by each culture.

Systems in each of the cultures are currently interworking on an ad-hoc basis but essentially remain separate systems. The aim is to provide the integrating facilities needed to achieve “IT culture integration”.

3.3 Generic requirements

Any solution to the problem of integrating the IT cultures must use de facto standards at culture boundaries.

It must allow evolution, i.e. be

- highly reconfigurable
- scalable
- functionally extensible.

It must also be easily managed, and be dependable.

4 Network Management Application Scenario

To give the generic scenario more direction, we have developed a specific scenario, which addresses all the issues set out in the generic scenario. The scenario concerns the management of a network and a set of distributed application services.

This specific scenario is not the only one that can fit the generic scenario. We believe that it can easily be replaced by another scenario (ATM networks and video on demand for instance). To retain the focus of the work, a single scenario should be worked on at any one time.

4.1 The physical structure

The chosen example is one of two networks which are connected via a gateway. The nature of the networks is not relevant. What is important is that on each network there are a series of devices (denoted by black boxes on the slide), which offer a service to their users. In the following we shall talk about devices in the full knowledge that if these were application services, the same basic principles could be applied.

To ensure that the service which is provided remains according to the users expectations, each device can be managed through a separate management interface. The management interfaces may in some cases require extra software, hence the circles round the devices.

To manage the services on the network, a manager object is introduced (the circles marked M). Each manager object manages a set of devices. The set of devices, together with their manager are enclosed in a management domain[1].

Several domains may be running in a single distributed systems infrastructure (e.g. two domains in DCE to the left). A management entity may control several manager objects and a partially hierarchical structure can be built up.

The management structure in the ACAS based distributed infrastructure includes the management of the bridge between the two networks. If any requirements to influence the operation of this bridge exist in the DCE based system, then there has to be a link between the management applications in both systems to effect this. It is not obvious what the relationship between the two management structures ought to be. Introducing a top level management entity may raise unanswerable questions with regards the ownership of this entity.

1. There can be a lot of discussion about domains and boundaries and it is not clear what it buys you. For the time being the notion is introduced informally.

4.2 The logical structure

The devices and their management software and manager are recognisable at the bottom of the picture. Each manager has access to Management Information Base (MIB). At the very least this is a list of the devices which are being managed. Some protocol is in use so that the manager can interrogate the devices and send it information which will control the operation of the device in some way. Logically speaking, the management information resides in the MIB. In some cases it may remain resident in the managed device. The important issue is that the MIB exists in some form and can be arbitrarily distributed. When managers cooperate with one another, possibly via a third party (another management entity), the management information is partially shared. Any sharing requires an explicit or implicit agreement and may need explicit negotiation. The sharing also imposes constraints on the information schema and can lead to integration of the MIBs or federation amongst MIBs, leading to the usual issues of ownership and autonomy.

The collection of management information and the distribution of controlling data must happen in a consistent manner. For instance, if new routing tables are to be distributed, then that operation needs to be atomic. It also needs to be perceived as near instantaneous. In a system that knows no downtime or quiet hours, some interesting dependability and timeliness questions arise.

4.3 Application and network management

The devices are not just managed, they are used as well. There is a sense in which applications rely on a service provided by the network. The network manager controls the resources and enables service provision.

The application itself is also providing a service. It is equally important that service provision at this level is guaranteed. There may thus be manager objects at the application level as well.

When the needs of the application change dynamically, there will be a need to communicate the requirements to the underlying services (to their management) so that the appropriate resources can be marshalled or released.

4.4 Characteristics of the scenario

4.4.1 Service provision and use

An important aspect of the scenario is to test how the model of service provision and use stands up in a real environment in which existing systems have to be integrated.

Service contracts have to be negotiated and agreed. This happens in the design stage as well as at run time.

Service providers reside both in the distributed systems infrastructure and at the application level.

Services are managed, so that they may continue to provide the service at the quality levels agreed in the service contract. This includes the management of replication for performance and dependability.

4.4.2 Other characteristics

Real time services will be managed by entities for which no particular real time constraints apply.

The aim is to consider failure resilience of certain failure classes: the dependability model has more details.

In the specific scenario, there will be a need to achieve data integrity and consistency in the face of real concurrency. Some of the IT cultures have well developed transaction technology to deal with this. The technologies are not always compatible (even within a single IT culture). Transaction processing service integration (not the building of yet another TP system) is an important aspect of the scenario.

The scenario integrates services from diverse IT user cultures. Heterogeneity is unavoidable. Access to existing systems (e.g. database management systems) will be required to test this aspect of the scenario.

Monitoring and debugging support is to be included from the start. Monitoring is an essential part of any management scenario.

Configuration management is another important aspect: we envisage a modular world in which services are the modules. Motivation for configuration changes are (amongst others) rooted in the desire to continue to meet a contractual obligation for service provision. As stated earlier, management plays an important role in this process.

5 Performance and timeliness issues

5.1 Context

Much work has already been done on performance issues in academic and research establishments world-wide. While drawing on this experience, it is important that emphasis is given here to the control of performance in an integrated open architecture, as opposed to a closed real-time system.

The four principal characteristics of an ODP system are:

- *Heterogeneity*: Components of compatible functionalities always exist in a system for a variety of cost reasons. Their existence frees the system from being locked into a particular technology.
- *Federation*: Autonomy facilitates de-centralisation and is intrinsic in a distributed system. Federation respects autonomy and is a viable basis on which common goals can be achieved.
- *Scaling:* Distributed Systems can in theory span the world, and can certainly be very large. Because of their scale they cannot be restarted as a single entity, but must evolve while still in use (eg. telephone system).
- *Evolution:* The first two characteristics above afford the system the ability to evolve by accommodating technological and organisational changes, integrating new systems or amalgamating existing systems.

The emphasis of the work will be on the high level supervisory control of the system, rather than the low level interrupt handing, which will be carried out by the underlying system.

As a general architectural principle the policies involved will be kept separate from the mechanisms used to implement them. By use of selective transparencies it should be possible to avoid the involvement of application programmers in the policies and mechanisms used any more than is strictly necessary.

The scenario is perceived as a requirement to manage real-time services. The management applications themselves may also have real-time requirements, but they will be less stringent than those of the services which they are managing.

5.1.1 Quality of Service

The managed services will exhibit different performance characteristics. Applications will demand from these service providers certain Qualities of Service. The following attributes are identified as those where negotiations can be made:

- *Time*: eg. what sort of jitter bounds can a computation tolerate? what sort of deadline does it require?
- *Space*: eg. what is the bandwidth requirement of a communication? what are its buffering requirements?

- *Cost*: eg. what is the cost for not carrying out a computation on time? What is the benefit of carrying out a computation on time?
- *Criticality*: eg. the consequence of failure of some services is more severe than for others.
- *Dependability*: eg. what is the probability of failure? What is the probability of survival against failures?
- *Precision*: eg. how many video frames need to be transmitted in a certain time frame in order to maintain a coherent picture at the receiving end?

It is not immediately obvious from the above list that any of these items, other than the first, are related to real-need scheduling over those tasks which require their use. Since several tasks may need an identical resource which is only sufficient for a single task, then the tasks may not acquire access to the resource simultaneously. A real-time element is thus inherent in the use of all resources, whether inherently of a timely nature or not.

5.1.2 Protection of Guarantees

The protection of guarantees is fundamental to the control of performance in an open system, as opposed to a closed system. In a closed system the demands made on that system may be predictable, and as such may be scheduled in advance, with no threat to the guarantee of meeting QoS requirements. In an open system this is never the case.

An open system is susceptible to demands from outside, and is never predictable. No control can be guaranteed over external demands upon the system. To protect guarantees which have already been made, boundaries must be placed around resources and their schedulers. It then becomes a federation concern as to what may cross these boundaries, and equally importantly, when. Such changes to pre-scheduled use of resources must be negotiated to ensure that previous guarantees are not violated.

5.1.3 Domains, Resource Pools, Scheduling Points

Scheduling points are the mechanisms used by which guarantees can be protected. Distinct scheduling points can be identified which then may have access to pools of resources which they may either own or share with other known scheduling points. So each scheduling point can access more than one resource pool, and a resource pool may be accessed by more than one scheduling point. In the latter case negotiation must take place between the scheduling points concerned.

It is at the scheduling points that the scheduling policies are determined. The scheduling policy is an algorithm defining how scheduling is carried out, and is determined by scheduling attributes (see below). Different scheduling points may support the same, or different, scheduling policies. Interfaces, operations and objects will attach to a scheduling point in order to access resources. A resource pool may consist of tasks, channels, cpu's, segments, files, devices, or any other resource which may need to be scheduled. Scheduling domains are groups of scheduling points and resource pools under the same overall authority.

5.1.4 Choice of Scheduling Attributes

There are two primary management performance activities: resource allocation and resource scheduling. Resources are allocated to satisfy

demands. Resource scheduling organises the availability of resources in keeping with the current scheduling policy.

The term *binding* is used to refer to the commitment of the system to schedule system services to meet QoS requests. The negotiation of service guarantee is a separate concern. The semantics of this type of binding can take on one of serveral forms:

- *Absolute:* The guarantee is delivered as demanded.
- *Pre-emptive:* The guarantee is scheduled to be delivered as requested but subject to withdrawal.
- *Time-variant:* The QoS is guaranteed to a certain level on average. However the level of access may fluctuate.
- *Binding set:* Where there are several QoS requirements, these are delivered in total, or not at all.

There is a spectrum of strategies for resource allocation. At one end of the spectrum is the "demand driven" strategy where resource allocations are open to demand. Time-shared, multiple user systems are an example. At the other end is the "completely predictive" strategy where resources are pre-allocated and are closed to additional demands. Embedded mission-oriented systems fall into this category.

Within this spectrum of strategies are degrees of "statistical predictive", where resource allocations are open to demands according to urgency or criticality of the tasks in hand.

5.1.5 Computational Model

The computational model needs to provide QoS attributes, access to resource pools and scheduling points, and specification of scheduling attributes. Suitable performance abstractions need to be developed, their semantics specified and implementations defined in terms of application programming languages.

5.2 Technology Choices

The policy of Phase III is to base prototypes on next generation products. In addition, rather than generating new code, existing software should be used wherever possible, and a variety of existing software exhibiting appropriate characteristics should be linked together to form a suitable platform for exploratory work.

Of the various real-time operating systems currently on the market, OSF/1 is available at APM for immediate use.

Of the Distributed System Environments (DSE) available, ANSAware is an obvious choice. RIDE is a real-time environment based on ANSAware 3.0 and is available to us. It has a bounded RPC protocol. We also have Orbix, which is a C++ based CORBA implementation.

Using Posix threads for portability, an obvious suggestion is to enhance Orbix with real-time capabilities and use this with RIDE implemented over OSF/1 to provide a platform for further real-time work.

There are three main choices for computational model implementations: pre-processor, class hierarchy, and language extensions.

6 Dependability issues

6.1 Dependability and the MIB

The service provided by the MIB must be dependable.This does not mean that all the components of the MIB must be dependable: the infrastructure can support the construction of a dependable service from less dependable components. By the end-to-end argument, it is the point at which the service is consumed which needs to be dependable. Access to the data contained in the MIB must be efficient; this may require replication and partitioning to achieve the performance needed. It is very important to maintain the integrity of the data in the MIB, transactions can be used for this.

The are at least three possible manifestations of the MIB: it may be composed of heterogeneous legacy databases; it may be a purpose built database; the data may be distributed amongst the objects which are being managed. (A combination of these three options may be used.)

The first case will require infrastructure support for interworking between different transaction models. The infrastructure may also provide facilities which encapsulate the legacy database to make it more dependable (e.g. the interceptor between the data services and the integration services). This should not involve attempting to re-engineer the database.

Sometimes it may be possible to construct the MIB database from scratch. The infrastructure should provide facilities such as replication and transactions to facilitate building a dependable MIB.

If the MIB is distributed across the objects which are themselves being managed it is likely that yet another approach will be needed to make the MIB dependable. It is unlikely to be possible to replicate all the managed objects (since in many cases they will correspond to physical devices).

In summary the dependability requirements for the MIB will require interworking between a variety of transaction schemes, a spectrum of mechanisms will be needed to support dependability across the possible implementations of MIB, in turn, this will require a range of replication and communication options.

6.2 Dependability and management applications

Management applications access and manipulate the MIB. An application may use more than one MIB, so consistency will need to be maintained across multiple MIB's (as well as ensuring each individual MIB is consistent). This will require interaction with the transparency engineering provided by the infrastructure.

Management applications need to be dependable, but will have to interact with managed objects using standard protocols such as CMIP and SNMP. The

facilities which are used to provide dependability need to be compatible with these protocols (e.g. replication and transactions using CMIP/SNMP).

One class of management application is the management of the dependability of the system. This involves configuring and initialising services so that they have the required level of dependability. In addition the application needs to provide fault detection, fault treatment, fault passivation and reconfiguration (see the document describing the Dependability Model) so that this level of dependability can be maintained in the presence of faults. It may also necessary to log data and state for later analysis. This management application itself needs to be dependable.

Management applications may well cross boundaries (technological or organisational) to access services in different federations. The dependability of these services needs to be stated in a service contract. Somebody or something needs to be responsible and somebody or something needs to be accountable for dependability. MIB's may also span federations.

6.3 Engineering transparencies for dependability

Two important principles are:

- The application/service should only use the mechanisms it needs;
- The application/service should only be provided with the mechanisms it needs.

The first means understanding how to achieve a given dependability with a given set of mechanisms, so that the application does not try to use more than it needs. The second means that the application should not be forced to include or use mechanims which it does not require. Taken together these two principles will ensure that nucleus will be small and that the runtime infrastructure required by applications will be minimal.

Transparencies can be selected by stating an appropriate QoS. This needs to be translated into requirements for various mechanisms which requires understanding how to describe, engineer and reuse transparencies.This means architecting a kit of parts and understanding how the components interact to provide the transparencies. Some of the mechanisms providing a transparency may be embedded in the application (e.g. via a library) or provided by the infrastructure: the trade-offs need to be investigated.

Some transparencies can be selected during trading others may be selected during binding. The options and trade-offs between these choices need to be understood; the first is static selection of transparency the latter is dynamic selection.

An interface needs to be provided to allow programmers to select the appropriate QoS and to translate this into an appropriate configuration of transparency mechanisms. It is desirable to automate as much as possible.

6.4 Outline plan

The dependability group already have extensive experience in prototyping infrastructures supporting groups, replication and transactions. This experience will be used in conjuction with the scenario to develop the architecture further before any prototyping activity takes place.

The dependability model is undergoing a major revision. Once this revision is complete it needs to be applied to MIB's and management applications to check that it can model their dependability requirements adequately.

Transactions play an important role in the scenario. Legacy databases will support a variety of transaction styles; any management application which needs to access different databases will have to support interworking between the different models. Transactions can also be used to ensure that changes made to MIB's are consistent (e.g. it is desirable to make atomic updates to routing tables).

The engineering model needs to be documented. Transactions and replication will be used to drive this work: a structured kit of parts integrating both will be architected (later the technique could be applied to other transparencies). This work will also involve exploring how to make legacy systems dependable. (Note that this is a completely new work item which ANSA has not addressed before.)

The trade-offs need to be understood in the current and possible programming strategies for dependable MIB's and management applications. We need to understand what concepts a programmer needs to use to build dependable MIB's and management applications, what tools can be used in this development and how these tools affect the concepts.

It is important to understand how to configure management application and MIB's to deliver a given level of dependability and maintain this dependability. It is desirable to use tools to automate as much of this as possible: these tools will themselves be examples of a management applications.

It is not clear what choices should be made for the prototyping technology, hence the question marks on the slides.

6.5 Technology Choices

The plan in §6.4 says nothing about technology choices — these have yet to be made. Althought the architecture should be general, encompassing a range of possible options and choices, any prototyping activity needs to make pragmatic choices from available technology.

These choices need to be guided by the technology the sponsors regard as important and which can be made available to the core team (e.g. what kind of databases can be obtained).

Once a decision about the technology has been taken, a more detailed implementation strategy can be derived for prototyping.

7 Federation and tools issues

7.1 Context

Distributed applications will increasingly be designed and built in the context of an existing environment which consists of many services. The emphasis will be on the design and construction of services, not systems. New services must fit in a world of service provision and use. The use of existing services is encouraged by the need to keep development costs down. Re-use is already becoming an important trend: we know how to achieve it in the context of a small or medium scale project, using object oriented languages and their libraries (SmallTalk, C++, Objective C, Eiffel).

7.1.1 Distributing the design process

The challenge is to scale up the tools which successfully help achieve re-use in small to medium scale projects, to programming in the very large.

The first step is to characterise all system components as service providers and users. This places us in a well known paradigm. The key question is whether this characterisation is too restrictive and if so what extensions are required.

The next step is to determine what information is needed to characterise a service such that a service user can make use of that service. For billing purposes for instance, the service provider also needs to know about the service users. There is thus a need to negotiate the use of a service. The negotiated use is represented by a *service contract*.

It is also necessary to consider when the service contract is exchanged and what this means in contractual terms. There are many analogies with contracts in daily life: many are implied by the conventions to which we adhere. When crossing frontiers, such conventions need examining. It is expected that as we cross the boundaries of a small scale project, the conventions implied in a contract need to be made explicit in the contract itself.

Furthermore, we need to consider what tools can be devised to automatically generate interworking code from service contracts, and in what epoch such tools are appropriate.

7.2 Service contracts

A service contract is a representation of

1. an undertaking to provide a particular service
2. an undertaking to use a service under certain conditions (e.g. the payment of money)

It may include a statement of penalties in case the contract is broken. The information contained in an ANSAware interface reference and an interface definition in the context of a type system are components of such a contract. Quality of service parameters and information about transparency mechanisms also belong in the contract, as does information about the conditions under which a service may be used (meaning & purpose).

7.2.1 Design decisions

Design decisions which affect interworking should be reflected in the service contract. Specific design decisions will lead to the creation of technology domains. Ideally, a service contract which reflects such design decisions could be used to generate interceptor code which can be used to bridge technology boundaries.

7.2.2 Relation to system management scenario

The proper management of a service provides the safeguards for a service to continue to meet its contractual obligations.

7.2.3 Epochs

Service contracts are used in any epoch. Traditionally ANSA has proposed late binding. Only part of the service contract is required for the binding process. Type checking, using another part of the contract, is typically done early in ANSA. The use of the service contract in different epochs in ANSA and elsewhere (e.g. CORBA interface repositories which are accessible at runtime) must be examined and placed within the Architecture.

7.2.4 Representation

Service contract components are traditionally represented in many different ways. There exist several IDLs and there is as yet no clear de facto standard.

Many tools operate internally on the basis of an Abstract Syntax Tree (AST). This is used as the internal representation of an interface definition for instance. Different tools use different ASTs. The use of a particular AST is sometimes said to give one tool a competitive advantage over another. ASTs will thus not easily be standardised.

The type of a service is defined in the context of a type system (system of classification). There exist many of such systems and although there is some agreement on the base types (boolean, integer, floating point, etc.) there is not enough agreement for it to be called standard. There is also little agreement on the constructed types.

There will be a need for tools which can translate from one representation to another. Until the tool manufacturers see clear benefit in a common standard, the NxN problem will not be reduced to a 2N+1 problem.

7.2.5 Exchanging and negotiating service contracts

As alluded to above, the exchange of service contracts between design authorities needs examining from the contractual perspective. The tool support for this exchange must also be considered. Advanced trading functions and the use of service specification (interface) repositories and implementation repositories (libraries) are involved.

7.3 Plan

7.3.1 Existing tools

The IONA/Orbix tools should be examined as they will most likely be used in the scenario as the CORBA IDL to C++ compiler. To link into personal services under MS/Windows, it is proposed to examine DEC/ACAS, which provides DDE[1] interfacing from a UNIX environment. There is one more candidate for evaluation in the short term: DOME from Object Oriented Technologies for which we are to receive a free evaluation copy.

7.3.2 Other systems

The systems in the various cultures described by the scenario will be connected to other systems. The system and application management example contains links to an accounting and billing database, which serves corporate and business applications. Links to personal services are also expected.

Such external systems can be made to look like services and their interfaces can be characterised by a service contract. Tools can then be used to generate the interworking code, bridges etc. to make the links between the management scenario and existing systems.

7.3.3 Repository

Once we understand the nature of a service contract, a service repository can be constructed. We could start from the proposals made for OMG interface repositories. An advanced trading system should be linked to this.

7.3.4 Tools

The nature of the service contract will also allow us to identify the kinds of design and construction tools which can generate code templates from service contracts.

7.3.5 Design tool and operational system

If design is taking place within the operational system, then the link between the design tool and the operational system may also be used for reconfiguration and in-service upgrades.

7.3.6 Qos

Together with the performance and dependability reporting groups, further work on the integration of Qos parameters in the service contract and the relationship with the tools which insert the correct transparency code will be undertaken.

1. Dynamic Data Exchange - DDE - is a Microsoft protocol to allow two concurrently running applications in a Windows environment to exchange data and instructions. It is based on the messaging system built into Microsoft Windows.

8 Technology choices

The slide is to be extended and annotated with the required technologies for the scenario.