**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**

# An Overview of ANSA

**Rob van der Linden**

**Abstract**

This document provides a brief overview of ANSA to date. It explains what an architecture is and why it is useful to have one. It highlights the principles on which the Architecture is based and introduces the Architecture Reports and Technical Reports which describe the Architecture and its application in more detail.

| APM.1000.01 | **Approved** | 15 July 1993 |
|---|---|---|
| | Architecture Report | |

**Distribution:**
**Supersedes**:
**Superseded by**:

# An Overview of ANSA

**Architecture Report**

# An Overview of ANSA

Rob van der Linden

APM.1000.01

15 July 1993

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

| | |
|---|---|
| TELEPHONE UK | (0223) 323010 |
| INTERNATIONAL | +44 223 323010 |
| FAX | +44 223 359779 |
| E-MAIL | apm@ansa.co.uk |

# Contents

# 1 Introduction

## 1.1 The ANSA objective

ANSA is an architecture with a specific objective to enable application components to work together despite diversity of programming languages, operating systems, computer hardware, networks, communications protocols and management and security policies. The Architecture is relevant to telecommunications, manufacturing, sales, cooperative working, banking health service, research, and other applications.

The Architecture provides a framework within which the design and implementation of distributed computer systems can take place. The framework ensures that different design choices which are made for particular applications can be related to one another. This in turn reduces the effort required to make two or more ANSA systems interwork, even after they have been designed and built. Existing (legacy) systems are easily captured in the ANSA framework and then become part of the distributed system in the same way as other ANSA systems.

## 1.2 The aim of this document

The aim of this document is to present a short overview of the Architecture:

- Chapter 2 explains what an architecture is and why it is useful to have one
- Chapter 3 summarises the architectural principles which underpin ANSA
- Chapter 4 introduces a set of Architecture and Technical Reports and relates these to one another and to the Architecture.

This document should help you select aspects of the Architecture about which you want to know more. It should also help you pinpoint what documents are most relevant to you and your colleagues.

## 1.3 Audience

This document is intended as a technical introduction. It is suitable for technical managers who want to understand the relevance of ANSA with respect to distributed systems design, implementation, management and maintenance.

## 1.4 Is it real?

It is not possible to offer proof of concept through the pages of any document. Many of the ideas expressed in this document have been taken to a fine level of design and implementation [AIM 93]. Prototypes are continually being developed to illustrate the practical application of the principles which underpin the Architecture. Several large scale distributed systems have been built using ANSA technology [MURRAY 92]. The technology and concepts are also used in many universities as a basis for teaching distributed computer system fundamentals.

## 1.5 Is it a standard?

The Architecture is not a standard by itself, but has been a significant input to the joint ISO/IEC and CCITT development of a Reference Model for Open Distributed Processing [ISO 92a], [ISO 92b]. Technology based on ANSA principles have been submitted by sponsors of the ANSA work programme to industry fora and standardisation groups, including the Object Management Group (OMG), the Open Software Foundation (OSF), and UNIX International (UI).

# 2 ANSA: The Architecture

## 2.1 Benefits

ANSA, the Architecture delivers benefits in three specific areas:

1. *designers and programmers* benefit because the focus is on application requirements instead of technological heterogeneity;
2. *vendors and end users* benefit because the focus is on portability and interworking, allowing a wide range of distributed systems to coexist;
3. *standards communities* benefit because the provision of a framework backed by proven technology aids development of timely standards.

### 2.1.1 Tools based approach

The ANSA Architecture applies to a class of computing systems in which different parts of a distributed application work together in a unified and natural way . The goal of ANSA is to allow application interworking and portability to be achieved with a relatively small effort; application designers do not need to concern themselves with the diversity or physical distribution of different computers, operating systems and network protocols which comprise the technological basis of the underlying computing environment.

The Architecture is tool oriented: application programmers use a programming language to state the application properties they desire. The tools automatically insert functionality required to achieve the desired application properties (such as atomicity , fault tolerance, quality of service).The tools so hide the technical details of the underlying mechanisms.

By removing the burden of technological detail from application designers, they can concentrate on meeting the requirements of end users and on issues concerning the provision of applications that span organisational, political, and geographical boundaries.

### 2.1.2 Distributed systems characteristics

The existence of genuine portability and general interworking allows great flexibility for systems configuration: within certain limits [1], hardware and systems software can be configured independently from application software. Increased configuration flexibility will mean more satisfied customers and lead to a larger market for systems and applications.

Once the technological and economical boundaries that hinder application interworking and portability are removed, there will be few limits on the extent to which users require their computer systems, applications and data to be combined. This will lead to very large open distributed processing systems.

1. The required performance or dependability of a system may place limits on configuration freedom.

For such systems to become and remain effective, they must satisfy several demanding requirements. The Architecture makes specific provision for systems which:

1. allow many different kinds of applications to coexist and interwork (including office, factory, telecommunications, and general data processing applications);
2. incorporate many different technologies, such as hardware platforms, operating systems, communications protocols, and programming languages;
3. scale from small (a few nodes) to very large (the size of the existing telephone network and larger);
4. allow applications to span organisational boundaries, whilst satisfying the policies and constraints of all organisations they span;
5. are economical to develop, extend and maintain;
6. allow graceful evolution from the current installed base of computing hardware and software to new innovative products (thus protecting existing investments, and offering opportunities for new products to fit with the installed base).

End users and vendors can expect reduced cost associated with development, maintenance enhancement and integration of systems if they are built according to an architecture (a technically consistent set of components and rules).

### 2.1.3 Building consensus

To enable the design and construction of systems with the above characteristics, consensus between the computing and telecommunications industries is desirable. The ANSA work programme has provided a focus for such consensus by being directly supported by more than 20 companies in these industries.

Computer related standards communities often work on standards before products are available. This inevitably means that expertise in the area is less well founded than elsewhere. The ANSA Architecture and its prototypes help promote the confidence that standards for distributed computing, once implemented, will lead to systems with desirable characteristics.

The ANSA Architecture is being developed in close association with a number of industrial and consensual standards-making bodies. On the industrial side there are strong links with OSF, UI and OMG initiatives; on the consensual side with work on the Open Distributed Processing standards which are being developed jointly by the ISO/IEC JTC1 and the CCITT. Several companies which support ANSA are active in both types of standards making and advantage is taken of both formal and informal links to obtain a high profile for the consortium's views.

## 2.2 What is an architecture?

ANSA, the Architecture, is not a design for a distributed computing product, in the same way that the gothic architecture is not the same as the Notre Dame. Instead, ANSA specifies:

1. a set of components, which form the basic building blocks and tools of the architecture;
2. a set of rules, which constrain the way in which such components can be combined in designs that conform to the architecture;
3. a set of recipes, which provide advice on how to combine basic components using the tools to obtain subsystems with certain properties;
4. a set of guidelines, which help designers make design decisions if their own preferences do not offer sufficient basis for such decisions.

Using the components, rules, recipes and guidelines (i.e. the Architecture) helps ensure that the differences between the resulting systems remain such that applications can be ported and their interworking can be achieved more easily .

The Architecture allows many different designs. Each may be suited for a particular application or application area. For each area different optimisations will be appropriate, and individual designs must reflect these. The Architecture can be used as a framework to compare the different design optimisations and to help designers reason about combining dissimilar systems.

### 2.2.1 Properties of the Architecture

#### 2.2.1.1 *Evolution*

The Architecture is designed to allow evolution of the Architecture itself. As new technologies emerge, the assumptions upon which the Architecture is based are re-examined and adjusted where necessary . As a starting point, the Architecture accommodates today's legacy systems in distributed object based environments based on ANSA.

The "future proofing" of the Architecture is achieved by carefully selecting a small set of re-usable and expressive principles, which can be applied in most practical situations. Chapter 3 outlines these principles.

#### 2.2.1.2 *Design process*

The process of designing and building systems within the Architecture is affected only by the nature of the components and the composition rules. This is because architecture applies to components and their configurations and is principally structural. The way in which a conforming structure is designed and built is of secondary interest. The Architecture therefore does not prescribe any particular design process or methodology . A variety of tools and methodologies will inevitably be used and are permitted.

#### 2.2.1.3 *Abstraction*

The only way to tackle the complexities that result from the requirements expressed in §2.1.2 is to provide for separation of concerns within an architectural framework that provides overall integrity . The Architecture thus contains a set of models.

As the Architecture evolved, five of these were identified as necessary and sufficient to structure specifications of systems:

1. an *enterprise model* for expressing system boundaries, policies and purpose;

2. an *information model* for expressing the meaning of distributing information and information processing tasks;
3. a *computational model* for expressing functional decomposition into distributable units;
4. an *engineering model* for describing components and structures needed in support of distribution;
5. a *technology model* for describing the make up of a system in terms of components with conformance criteria.

Other models are more functional in nature (e.g. the Atomic Activity Model, the Naming Model, the Model for Interface Groups, the Storage Model, etc.) and equally serve to provide abstractions in aid of separation of concerns. The “specification” models are used to structure the “functional” models, such that the “functional” models can be integrated with specific system specifications. All models can be expressed in terms of rules, recipes and guidelines.

#### 2.2.1.4 Composition

The rules, recipes and guidelines of the ANSA Architecture describe the various ways in which components may be combined into compositions, and ways in which such compositions can be combined together.

Most other “architectures” known in the computer and telecommunications industry have a fixed structure similar to a systems design. Such “architectures” can often be represented by a block diagram.

The ANSA Architecture is characterised by the properties of its components and the nature of the composition rules. These cannot be expressed in a simple diagram.

# 3 Principles of the Architecture

The principles of the Architecture are placed in seven categories[1]:

- separation
- diversity
- scaling
- federation
- transparency
- concurrency
- configuration.

For each category, the principles adopted in the ANSA Architecture have been summarised below. Chapter 4 provides details of the Reports in which these principles are set out in greater detail and in which they are applied.

## 3.1 Separation

Most systems are designed to work within a fixed context, either within a single computer, or in some static relationship with other systems. Major difficulties are often encountered when such systems are required to cooperate with systems outside this context. This is because built in assumptions and optimisations prevent the extensions required for effective interworking with other systems.

### 3.1.1 Design for separation

Systems should be designed such that separation amongst its parts can be achieved. This increases the life of the design as the resulting system can be more flexibly configured. Because there are more components, such designs can be harder to understand; the systems more difficult to maintain. Appropriate tool support alleviates many of these problems [WILDE 93].

### 3.1.2 Assume services are remote, combine for performance

It should be assumed from the outset that all components are physically or logically remote from one another. Co-location is an opportunity for optimisation. With appropriate tools the optimisation process can be automated. Careful choice of optimisations guarantees interworking remains possible even after optimisation.

Separation and remoteness produce a requirement for distribution transparent access.

1. The categories are not independent: some issues concern several categories. No categorisation has been found which cleanly partitions all issues.

### 3.1.3 Use indirection, encapsulate state

It then follows that since local data is viewed as the co-located special case of remote data, all manipulation of data (as seen by the programmer) must be indirect.

It should further be assumed that the remote object can only be requested (not instructed) to change the data it holds. It is not even possible to guarantee that the remote object will update its data in the way anticipated by its client.

### 3.1.4 Use abstract data types

Since data may be remote and on a system of dissimilar and unknown design, it is impossible to manipulate it other than through defined interfaces. Also, it is not generally possible to obtain data as a direct response to a service request, since there may be no known common data representation. Instead, a reference to another interface which provides further data transformation services is returned. The exception to this arises only in the case of immutable data types and where there is agreement about what the data represents.

### 3.1.5 Be prepared for (partial) failures

In general, it is not possible to know the reliability of a remote system, nor of the communication system to it. A failure can occur in any remote service request. Since any service request can potentially be remote, any request can fail. Any distributed systems architecture must assume and tolerate all types of failure, including partial ones, and cater for these normally being of remote origin. Mechanisms may be provided to hide and possibly correct failures from the application programmer (see: §3.5).

### 3.1.6 Work from local time; don’t rely on global clocks

Physical separation implies that there can be no common assumptions about time. Clocks can be synchronised between remote systems when expensive protocols are employed. A cheaper solution is a protocol which determines a logical ordering of events. This is sufficient for most purposes.

### 3.1.7 Aim for convergence not global synchronization

In a distributed system it is not possible to assume a particular ordering of events, nor assume simultaneity . If order is important, it has to be determined explicitly . In particular it must not be assumed that some inherent system characteristic will cause processes to synchronise.

## 3.2 Diversity

Large distributed systems will include many significantly different individual systems, connected using many different communications systems and protocols.

### 3.2.1 Decentralize data

It is not practical to assume the existence of a widely distributed data pool that can be accessed directly from everywhere. The number of databases, each serving different local communities, is increasing continuously . They are designed, built and commissioned without much regard for other such systems. Insisting on integration of these databases at the conceptual schema level is clearly impractical. At the internal schema level, it would also impose constraints of scale, since all requirements for cost, speed, accuracy , and consistency cannot be accommodated at once.

### 3.2.2 Permit inconsistent data

There is always the possibility of inconsistent data in systems which encompass many independent data sources. Such inconsistencies cannot be avoided and must therefore be accommodated. Distinguishing the source of data can assist in coping with inconsistencies.

Local data however can always be treated as consistent, unambiguous and definitive within itself because it is strictly under local control.

### 3.2.3 Acknowledge different data representations

From the perspective of data manipulation, the differences in data representation, however problematic, must be accepted. Different data representations imply that one system cannot manipulate data directly in another .

### 3.2.4 Assume performance differences between components

A heterogeneous network implies performance and quality of service differences, both in the connected systems, and in the communications. These differences cannot be made uniform without going to a lowest common denominator , so they must be accommodated.

### 3.2.5 Allow for incompatible mechanisms

It is inevitable in large systems, that different “standards” are adopted to perform the same or similar functions in different parts of the system. Designers should be ready for incompatibility which hinders interworking: two phase commit protocols cannot interwork with three phase commit protocols, service relocation mechanisms may not be compatible, etc.

## 3.3 Scaling

Systems will always change, grow , and merge. This introduces differences in scale: geographical, numerical, time etc. V ariations and increments from small to large, slow to fast, specific to general purpose etc., must not impose constraints on the extent to which such systems can be interconnected and their applications be made to interwork.

### 3.3.1 Carefully design mechanisms and protocols

The solution to incompatible mechanisms and protocols lies in the careful design of mechanism which scale, that is mechanisms which are efficient in small systems whilst sufficiently functional to suit very large configurations.

Where possible new mechanisms should be able to interwork with existing ones, even if the existing mechanisms do not interwork themselves.

## 3.4 Federation

Federation deals with heterogeneity of authority in large distributed systems. It is important to retain (local) control of local resources. Allowing other authorities to control ones resources implies risk. Remote control over resources cannot be achieved reliably either .

Organisational structures which surround large distributed systems are subject to continuous and unpredictable change: no merger was ever pre-planned. The challenge is to build systems which can accommodate such changes.

### 3.4.1 Provide for decentralised control

Policies for security , naming, addressing, routing etc., are maintained in a local context by treating the object as a unit of encapsulation of policy . The principle of locality requires that any migration of policy into or out of a local context is under local and therefore decentralised control.

The direct consequence of this is that objects are responsible for their own management of resources, internal state and security .

### 3.4.2 Replace global policy by agreement to convention

Negotiation on policies and mechanisms can take place on the interfaces between components and systems. Such negotiation takes place any time before actual interaction and can take many forms.

### 3.4.3 Permit cross-linking of autonomous systems

The federal model is one of interworking between clusters of closely coupled applications. Each cluster is independent of the others and is organised to best suit the constraints upon it. Each cluster chooses which facilities to make available (export) to the clusters with which it interconnects; these clusters in turn select which facilities to use (import). Federations are therefore peer to peer rather than hierarchical structures. The concept of selective export and import of services between applications provides autonomy for scaling. The ability to inter -link arbitrarily allows systems to share resources and build common context in an unplanned incremental way , much as people do.

### 3.4.4 Assume context relative naming

To allow smooth cooperation between members of different communities, there is a need to unambiguously refer to entities. This is often achieved by agreeing a name for an entity . To safeguard the autonomy of the cooperating parties, there is a conflicting need to remain in charge of one's own naming policies. As the number of cooperating parties and the number of entities which require names grows, it will become impossible to achieve the required agreements. The most extendable approach is to accept that every name is relative to a context and to extend it when the name passes into a wider context.

## 3.5 Transparency

A property of a system is said to be transparent if application programmers need not be concerned with it. For example, replication is transparent if it is not apparent to either clients or servers, i.e. the interaction model is exactly the same for replicas as it is for singletons.

### 3.5.1 Specify requirements as application properties

Distribution cannot be ignored: applications programmers will have to deal with the possibilities of concurrent access to shared resources, variable latency in accessing resources and failures disrupting access to resources. There are many different techniques for overcoming these problems, representing different design decisions with respect to for example consistency and availability , security and openness, strict management and flexibility , abstraction and fine-grained control. The aim is to minimize the extent to which applications are polluted by detailed knowledge of the individual techniques. This is achieved by allowing application writers to specify the requirements for application properties on the distributed systems infrastructure in a declarative manner .

### 3.5.2 Use tools to include transparency mechanisms

Transparency is achieved by employing a set of mechanisms in the infrastructure to compensate for an unwanted property of the system. It is generally not possible to associate a single mechanism with a particular transparency , rather a set of mechanisms are orchestrated to implement a particular transparency . A mechanism provides a service through a typed interface which can be defined in an interface definition language ( IDL ). This enables existing mechanisms to be substituted by new mechanisms conforming to the same type as the original. The statement of the required application properties and the interface type definitions of the mechanisms permit the use of tools when building the distributed application.

### 3.5.3 Make transparency selective

Sometimes application writers will want to exercise control over distribution or participate directly in its provision for instance by specifying a specific recovery procedure for certain failures. Transparency must therefore either be selective or allow application programmers to influence the mechanism by providing specific policies.

## 3.6 Concurrency

When writing application code, assumptions are often made about concurrency and synchronisation *mechanisms* . This can create difficulties when porting applications to different platforms, or requiring applications to work over heterogeneous domains. Assumptions for single node systems are invalid for distributed systems.

### 3.6.1 Separate concurrency specification from mechanism

A clear distinction is made between computational and engineering views of concurrency . Application writers are encouraged to indicate where computational parallelism is possible and where it is required, but without any reference to how or from where this may be obtained. Tools are then used to map the computation to available resources, exploiting parallelism where possible or necessary .

### 3.6.2 Separate concurrency from communications

From the perspective of the application writer interaction with a service can be either synchronous (execution continues when the (remote) activity is complete), or asynchronous (execution proceeds in parallel with the (remote) activity). The remoteness of the service must remain invisible to the application writer. Only when failures occur is it possible to deduce that a communication failure may have been the cause.

## 3.7 Configuration

The configuration of a large distributed system is never fixed. New parts are added, old parts removed. A maintenance program is carried out which continuously changes the configuration. Fault treatment and load balancing algorithms also influence the system configuration.

### 3.7.1 Do not fix binding times

To achieve flexible configuration of system components, late binding of clients to services is essential. For maximum flexibility, all components, both large and small, should be equally configurable and manageable.

### 3.7.2 Check early for safety

For maximum safety, all accesses must be type checked. To achieve this in a dynamic system, it must be possible to obtain the description of any component on-line; early type checking reduces the risks of unpredictable behaviour. Type checking should be an integral part of the configuration process if configurations are not to be fixed prematurely.

# 4 Reports on the Architecture

The ANSA Architecture is described in a series of Architecture Reports. These reports reflect a stable part of the Architecture and form the keystone of other work, described in a series of Technical Reports. Technical Reports consolidate results and give examples of the application of the concepts and rules of the Architecture.

Architecture and Technical Reports are reviewed and agreed by the ANSA Team and the Technical Committee (a committee on which each ANSA sponsor has a voting technical representative). Reports are subject to change control.

Architecture and Technical Reports are licensable documents, the copyright of which is held by APM on behalf of the sponsors for the time being of the ANSA work programme.

## 4.1 Architecture Reports

### 4.1.1 The ANSA Computational Model

The ANSA computational model [APM1001.1 93] is a framework for describing the structure, specification and execution of programs in the context of ANSA. It emphasises the principles of encapsulation and abstraction that are essential for the design of programs for use in an evolving distributed environment which is not necessarily subject to any central authority.

The model is in two parts: an interaction model and a construction model. The interaction model defines the relationships between service providers and service users; this includes the invocation and parameter passing scheme and a corresponding type system based upon conformance [1]. The construction model defines the relationship between an object and an idealised supporting environment, making extensive use of the ANSA Naming Model concepts [APM1003.1 93].

The extension of the model to atomicity is described in [APM1004.1 93], which also explains activities in more detail. Issues of concurrency control are explored in more detail in [APM1010.1 93].

### 4.1.2 The ANSA Model of Interface Groups

Distributed systems have a wide variety of communication, synchronization and error recovery mechanisms. Interface Groups [APM1002.1 93] are an abstraction for these mechanisms. In addition to resolving distribution issues, interface groups provide an easy-to-understand programming structure for replicating services as the basis for reliable systems.

1. Informally, this means that it is possible to determine the likelihood of interaction errors (i.e. server does not understand client request, or client does not understand server response) before interaction actually takes place.

The basic group abstraction is to treat a number of interfaces as though they were one: they provide a single service. This enables clients to invoke operations on collections of interfaces without needing to know the exact membership of the collection or the location of the members. This capability provides the basis for distributing the implementation of a service over a set of interfaces.

The Model of Interface Groups supplements the ANSA Computational Model [APM1001.1 93] by looking at the computational issues for interface groups. It shows how a transparency for active replica groups can be built out of a number of mechanisms and policies and suggests how these mechanisms and policies might be manipulated to build other kinds of groups. A number of issues are identified which need to be addressed so that groups can be managed to deliver a stated quality of service. Finally the engineering model requirements for groups are discussed; this requires the relocation service described in [APM1003.1 93] and [APM1021.1 93] .

### 4.1.3 The ANSA Naming Model

Many naming systems and standards for naming exist. The ANSA Naming Model [APM1003.1 93] provides a framework which accommodates the existing heterogeneity and allows the interconnection of diverse naming systems. It leads to distributed systems in which many different naming policies and mechanisms can be effectively managed and combined.

The ANSA Naming Model is appropriate to all projections, without exception. The most important applications of the model are in the construction model of the ANSA Computational Model [APM1001.1 93] , as the basis for Federation and Trading [APM1005.1 93] and interoperability [APM1021.1 93] , and as the basis for much of the work in the Information Projection [APM1017.1 93] . Naming is also central in security [APM1008.1 93] , [APM1009.1 93] , where naming of entities is linked to mechanisms for identification. A formal description of the Naming Model appears in [TR.021] .

### 4.1.4 The ANSA Atomic Activity Model and Infrastructure

The ANSA atomic activity model and infrastructure [APM1004.1 93] presents the motivation, principles, structure and techniques for a class of fault-tolerant computations which maintain and preserve the integrity of long-lived distributed object state, despite the occurrence of different kinds of system failures. The model defines an integrated, modular control structure for supporting closed nested atomic computational activities (atomic transactions) based on the "all or nothing execution semantics" of the ACID properties. The associated infrastructure provides facilities for concurrency control, deadlock detection and resolution, and recovery from processor , memory and communication failures.

Validation of the atomic activity model and infrastructure has been carried out by constructing a tool for automatically transforming the behaviour of non-atomic computational activities and objects to their atomic counterparts.

The computational structures presented in this architectural report are based on the ANSA computational model [APM1001.1 93] . Related work on concurrency control using path expressions is covered in [APM1010.1 93] .

### 4.1.5 The ANSA Model for Trading and Federation

The trading service, specified in [APM1005.1 93], is a distinguished application which specialises in passing information about services from service providers to potential service consumers. The trading service stores this information in a structure which is amenable to queries by potential service consumers.

Federation is concerned with interconnection of distributed environments motivated primarily by organisational considerations. Such interconnection affects the trading services in these environments. Many kinds of boundaries between dissimilar systems can be distinguished. Mechanisms which allow the trading service to operate smoothly across these boundaries are identified. Strong links exist with the ANSA Naming Model [APM1003.1 93] and the ANSA Information Model [APM1017.1 93].

### 4.1.6 A Framework for Federating Secure Systems

The ANSA security framework [APM1008.1 93] assists system designers to understand the issues of security in federations of distributed systems, and suggests how mechanisms which support both distribution and security may be combined. It introduces concepts and terminology which may be used to describe security and, in particular, secure computer systems.

The framework does not prescribe any particular security policy; it allows a wide range of policies. It helps administrators understand the implications of their choice of policy without making that choice for them. Similarly, it does not prescribe any particular security mechanisms, but it does allow for the introduction of new security mechanisms with minimal disturbance.

### 4.1.7 Security Services in ANSA

The ANSA Security Services are described in [APM1009.1 93], which proposes an approach to open system security that aligns closely with the way human society operates. In society, groups of individuals set their own security standards and ultimately defend themselves within the context provided by the authority to which they belong. This observation suggests that computer security could benefit from a change from the traditional emphasis of security imposed through an infrastructure, to the service provider as the focus for security policy and self defence. Developing this theme leads to a different view of the primary security services, mechanisms and protocols needed for large scale secure distributed processing.

System objects compare with human individuals. Systems constructed from interacting objects behave like human corporations. In ANSA autonomous objects with secure boundaries are responsible for their own actions. Objects may control their own security policies and protect their own integrity, but with freedom to seek advice and guidance from other objects they trust. When they do so, security within and between systems appears much the same as security within and between human corporations.

The ANSA Security Services are aimed at architects concerned with inter-domain interactions, and the prerequisites needed to achieve inter-domain security. It will also be of interest to designers of nucleus interfaces, and to designers of individual nuclei for individual platforms. Sufficient background work has been done to be confident that current security crypto-mechanisms can be used to create system designs sympathetic with the proposed principles.

### 4.1.8 Monitoring in Distributed Systems

A model of monitoring and its management for object-based federated distributed systems is presented in [APM1010.1 93]. The information and structures necessary for conducting a monitoring session with multiple objects are presented. The management of a monitoring session, where the set of objects under observation changes dynamically, is addressed. Finally, management across federation boundaries is discussed. Parts of this report have been converted into a paper [HOFFNER 93].

This Architecture Report, together with [APM1018.1 93], [TR.040], and [APM1019.1 93], outline many of the facilities needed to manage, monitor and visualise the activities in distributed systems.

## 4.2 Technical Reports

### 4.2.1 A New User Interface Architecture

This paper [TR.011] proposes a new user interface (UI) architecture which is intended to help designers with the internal structuring of the UI - the mechanism which animates and facilitates the dialogue between the user and the application.

The architecture was used in the design of the Trader user interface tool-kit in ANSAware [AIM 93]. The paper was presented at the IFIP TC 2/WG 2.7 Working Conference [HOFFNER 89].

### 4.2.2 Integrating Multi-Media into the ANSA Architecture

The continuous nature of voice and video signals, coupled with the need to handle multiple streams of such signals simultaneously, poses several specific problems for multi-media applications. Such applications are inherently distributed. Many, such as computer supported cooperative working applications, also require traditional distributed services such as naming, filing, communications, security, etc.

This Technical Report [TR.028] proposes a uniform treatment of computing and multi-media resources. It covers the synchronisation between processes and multi-media streams and quality of service management. Design implications for addressing and naming [APM1003.1 93], protocol design and implementation and processing, thread scheduling and synchronisation are examined.

### 4.2.3 Distributing Objects

[APM1009.1 93] explains how the concepts of object-orientation can be applied to meet the requirements of distributed processing. It is in many ways an interpretation of the formal concepts of [APM1001.1 93] (Computational Model) against the backdrop of object-oriented approaches to programming and modelling systems. It is based on the structure of an HP Technical Report by Alan Snyder called “The essence of objects”, later published as [SNYDER 93].

### 4.2.4 A Formal Model for Naming

The ANSA Naming Model [APM1003.1 93] was developed alongside a formal treatment of naming [TR.021]. The formal specification is in support of the informal model. The process of deriving the formal specification had many benefits: it helped clarify the issues and sharpen the definitions. The presentation of the formal model in [TR.021] must be seen as secondary to the process of its derivation.

### 4.2.5 Using Path Expressions as Concurrency Guards

[APM1010.1 93] builds on the concepts introduced in the ANSA Computational Model [APM1001.1 93] to add a more explicit model of concurrency control. This is illustrated by a declarative notation - path expressions, and an explanation of how controls specified as path expressions could be implemented.

### 4.2.6 DPL Programmers' Manual

The DPL Programmers' Manual [APM1014.1 93] describes the ANSA view of distributed applications programming and how to use the ANSA distributed programming language DPL with embedded C code. It takes a tutorial approach. The intended audience are application programmers who are familiar with high level programming languages and commercial operation systems used in open systems.

DPL was designed as a concrete syntax for writing programs that conform to the abstract semantics of the computational model [APM1001.1 93] . The syntax and semantics of DPL are more succinctly described in [APM1015.1 93] and the rationale behind its design is covered in greater detail in [APM1020.1 93] .

### 4.2.7 DPL Reference Manual

The DPL Reference Manual [APM1015.1 93] defines the syntax of the ANSA Distributed Programming Language (DPL) and gives an informal definition of its semantics. It is meant to be a reference document for implementors and defines the language precisely and unambiguously . It is not written as a tutorial and is not designed to be read in any particular order . Each chapter completely covers a particular aspect of the language.

No attempt is made in this document to justify the design of the language or explain how to use it. The design of the language is based on the computational model [APM1001.1 93] and the rationale for it is discussed in [APM1020.1 93] . An explanation of how to write programs in DPL is contained in [APM1014.1 93] .

### 4.2.8 The Challenge of ODP

[APM1016.1 93] is a derivative of a paper written for the first ODP conference in Berlin 1991. It sets out a technical agenda for open distributed processing in response to the requirements of open distributed processing. It is an overview of the ODP Reference Model, explained in terms of the ANSA architecture. It justifies the need for a separation of computational and engineering models, the need for selective transparency and the requirement for federation. It is useful as an introductory tour of the issues that confront the designers of platforms to support open distributed processing.

### 4.2.9 Mapping ANSA Concepts to C++

The concepts introduced in the ANSA Computational Model [APM1001.1 93] can be mapped to different language environments. Mappings to PREPC [AIM 93] and to DPL [APM1014.1 93] , [APM1015.1 93] , exist and have been implemented. The mapping to C++ concepts is described in [TR.036] .

### 4.2.10 ANSAware Use of DCE/Posix Threads and RPC

To improve performance, ANSAware has integrated thread scheduling and communication. For flexibility, support for multiple message passing protocols is already provided. As a first step in experimenting with different thread and higher level communication packages in support of ANSA applications, it was decided to port ANSAware to run over DCE/Posix threads and to extend the communications facilities to allow use of DCE RPC as an alternative transport protocol. The resulting design is described in [TR.037]. The implementation is supplied with ANSAware 4.1.

### 4.2.11 Architecture and Frameworks

The ANSA Enterprise and Information Projections are described in [APM1017.1 93]. It is prefaced by a discussion which relates the genesis of the architecture and the idea of projections, and also relates the architecture and projections to a notion of frameworks. These latter provide the general scaffolding into which can be placed the architecture proper, a categorisation of those who use the architecture, and the design methods and tools they use to develop distributed applications. The architecture provides the basis for the development of infrastructures for distributed applications.

The Enterprise and Information projections are part of the architecture and are concerned with purposeful and meaningful interaction respectfully. The report describes the concepts underlying these two concerns. The concepts have been developed with an eye to being applied to Enterprise and Information Modelling as well as the architecture.

### 4.2.12 Management in Object-Based Federated Distributed Systems

The philosophy of management in object-based federated distributed systems is examined in [APM1018.1 93]. The problems introduced by distribution and which concern management are presented and the approach taken to deal with them is then outlined. The information and structures necessary for management are discussed. Finally, some conclusions and requirements concerning management within the ANSA architecture are drawn.

This Technical Report builds on the experience gained with the management of monitoring in distributed systems. Monitoring distributed systems is described in [APM1010.1 93].

### 4.2.13 Survey of Ordering Algorithms

When monitoring a distributed system [APM1010.1 93] it is necessary to collect monitoring messages from different parts of the system. Given the nature of communication in such systems, it is not usually possible to assume that the order in which the monitoring messages are collected will reflect the order in which they were sent. A survey of the ordering methods which can be used to order the monitoring messages produced in a distributed system is provided in [TR.040].

A taxonomy of the characteristics of re-ordering methods is presented as the starting point for the discussion of their relative advantages and disadvantages. Finally conclusions on which information should be available in distributed systems to facilitate ordering is given.

### 4.2.14 Visualization of Distributed Systems

When managing a distributed system [APM1018.1 93], it is necessary to monitor the activities [APM1010.1 93] and to help the system manager visualise what is going on.

A model of the visualization process is presented in [APM1019.1 93]. This report discusses the problems associated with the process of visualization of activities in distributed systems. The problems can be divided into those concerned with incomplete information, and those concerned with unexpected information arriving from the monitored system. Solutions to these problems are introduced and are used to obtain requirements for the design of distributed systems and visualization tools.

The visualisation model was accepted and presented at COMPUGRAPHICS 92 [HOFFNER 92].

### 4.2.15 Abstract and Automate

Standardized protocols and Application Programming Interfaces are necessary but not sufficient to solve the problems of programming distributed systems; language based approaches using increased abstraction and automation are required.

The abstractions needed to cope with the complexity of developing large scale, federated, heterogeneous distributed systems are identified in [APM1020.1 93]. This report also shows how these abstractions can increase the level of automation.

These arguments form the rationale for the development of the computational model [APM1001.1 93] and the DPL language wrapper for C [APM1014.1 93], [APM1015.1 93].

### 4.2.16 Addressing and Interception: ORB Interoperability

Addressing and interception [APM1021.1 93] are important concepts used to achieve location transparent addressing in distributed systems. This report identifies the requirements for interoperability between Object Request Brokers in the Object Management Group's Object Management Architecture, and how these requirements should be addressed in revising the specification of the Combined Object Request Broker (CORBA).

The report sets out the ANSA model of binding as seen from a computational perspective. It highlights the engineering view on how to link names for interfaces of distributed objects to names for network endpoints.

Often distributed systems will cross organizational and technological boundaries. At these boundaries names will have to be translated, checks made on validity of interactions and controls imposed. This is the function of an interceptor. It is important to link interceptors into the location transparent addressing scheme in such a way that technology boundaries remain invisible to applications.

Interface references contain addressing information. Traders are an important source of interface references and need to participate in interception. Trading and Federation are described in [APM1005.1 93]. Objects may change their addresses when they are temporary passivated (to storage) and when they migrate from one place to another. This relocation has implications for addressing and interception. Object storage and migration is described in

[TR.044] . General principles of naming and the application of these principles to binding and relocation are also described in [APM1003.1 93] .

### 4.2.17 The ANSA Storage Model

The ANSA Storage Model [TR.044] consists of a set of components and processes involved in providing the storage functions. Object passivation/ reactivation and object migration in distributed systems requires support for the self-management of the objects involved. The storage model explains these functions in the context of an engineering model for structuring storage resources and positioning object management interfaces.

# References


[AIM 93]

ANSAware Version 4.1 Manual Set:
RM.099.02 "An Overview of ANSAware 4.1"
RM.100.02 "ANSAware 4.1 System Manager's Guide"
RM.101.02 "ANSAware 4.1 System Programmer's Guide"
RM.102.02 "ANSAware 4.1 Application Programmer's Guide"
Architecture Projects Management Ltd., Cambridge (UK), March 1993

[APM1001.1 93]

R.T.O. Rees, "The ANSA Computational Model", Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1002.1 93]

E. Oskiewicz, N.J. Edwards, J.P. Warne, "A Model for Interface Groups", Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1003.1 93]

R.J. van der Linden, "The ANSA Naming Model", Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1004.1 93]

J.P. Warne, R.T.O. Rees, "ANSA Atomic Activity Model and Infrastructure", Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1005.1 93]

J.P. Deschrevel, A.J. Herbert, "The ANSA Model of Federation and Trading", Architecture Projects Management Ltd., Cambridge (UK), June 1993

[APM1008.1 93]

R.T.O. Rees, J.A. Bull, "A Framework for Federating Secure Systems", Architecture Projects Management Ltd., Cambridge (UK), June 1993

[APM1009.1 93]

J.A. Bull, A.J. Herbert, "Security Services in ANSA", Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1010.1 93]

Y. Hoffner, "Monitoring in Distributed Systems", Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[HOFFNER 89]

Hoffner, Y., Dobson, J. and Iggulden, D. "A New User Interface Architecture", In Engineering for Human-Computer Interaction, Ed. Cockton, G., Proceedings of the IFIP TC 2/WG 2.7 Working Conference, August 1989, North-Holland, pp. 113-136.

[HOFFNER 92]

Hoffner, Y. and Statham, A. “The Visualization of Distributed Systems”, Proceedings of the COMPUGRAPHICS 92: Second International Conference on Computational Graphics and Visualization Techniques conference, Lisbon, December 1992.

[HOFFNER 93]

Hoffner, Y. “The Management of Monitoring in Object-Based Federated Distributed Systems”, Third IFIP/IEEE International Symposium on Integrated Network Management, San Francisco, USA, 18-24 April 1993.

[ISO 92a]

Recommendation X.902: Basic Reference Model of Open Distributed Processing Part 2: Descriptive Model, ISO 10746-2, November 1991.

[ISO 92b]

Recommendation X.903: Basic Reference Model of Open Distributed Processing Part 3: Prescriptive Model, ISO 10746-3, November 1991.

[MURRAY 92]

Stephen Murray, “The NASA Astrophysics Data System, Smithsonian Institution, Astrophysical Observatory, published by Architecture Projects Management Ltd., Cambridge (UK), April 1992.

[SNYDER 93]

Alan Snyder, “The essence of objects: Concepts and Terms”, IEEE Software, Vol. 10, No. 1, January 1993, 31-42.

[TR.011]

Y. Hoffner, “A New User Interface Architecture”, Architecture Projects Management Ltd., Cambridge (UK), February 1991.

[APM1009.1 93]

A.J. Herbert, “Distributing Objects”, Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[TR.021]

A.J. Tocher, “A Formal Model of Naming”, Architecture Projects Management Ltd., Cambridge (UK)

[APM1010.1 93]

R.T.O. Rees, J.P. Warne, “Using Path Expressions as Concurrency Guards”, Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[TR.028]

C.A. Nicolaou, “Support for Multi-Media Operations”, Architecture Projects Management Ltd., Cambridge (UK), January 1993.

[APM1014.1 93]

D.J. Otway, “DPL Programmers’ Manual”, Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1015.1 93]

D.J. Otway, “DPL Reference Manual”, Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1016.1 93]

A.J. Herbert, “The Challenge of ODP”, Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[TR.036]

C.A. Nicolaou, “Mapping ANSA Concepts to C++”, Architecture Projects Management Ltd., Cambridge (UK), February 1993

[TR.037]

C.A. Nicolaou, “ANSAware Use of DCE/Posix Threads and RPC”, Architecture Projects Management Ltd., Cambridge (UK), February 1993

[APM1017.1 93]

D. Iggulden, R.T.O. Rees, R.J. van der Linden, “Architecture and Design Frameworks”, Architecture Projects Management Ltd., Cambridge (UK), June 1993.

[APM1018.1 93]

Y. Hoffner, “Management in Object-Based Federated Distributed Systems”, Architecture Projects Management Ltd., Cambridge (UK), June 1993

[TR.040]

Y. Hoffner, “Survey of ordering algorithms (Design & Implementation issues)”, Architecture Projects Management Ltd., Cambridge (UK), February 1993

[APM1019.1 93]

Y. Hoffner, “Visualization of Distributed Systems”, Architecture Projects Management Ltd., Cambridge (UK), June 1993

[APM1020.1 93]

D.J. Otway, “Abstract and Automate”, Architecture Projects Management Ltd., Cambridge (UK), June 1993

[APM1021.1 93]

A.J. Herbert, D.J. Otway, R.J. van der Linden, A.J. Watson, I. Domville, “ORB Interoperability”, Architecture Projects Management Ltd., Cambridge (UK), May 1993.

[TR.044]

D.N. Nyong, “The ANSA Storage Model”, Architecture Projects Management Ltd., Cambridge (UK), July 1992.

[WILDE 93]

N. Wilde, P. Matthews and R. Huitt, “Maintaining Object-Oriented Software”, IEEE Software, Vol. 10, No. 1, January 1993.