A System Designer's Introduction to the Architecture

ANSA: A Systems Designer's Introduction to the Architecture

Release RC.253.00

April 1991

This document provides an introduction to ANSA. It is specifically oriented towards those with a software and systems background. It describes the underlying assumptions and principles of the architecture; it does not describe how the architecture is applied to specific application domains.

Architecture Projects Management Limited

The material in this technical report has been developed as part of the ANSA Architecture by the ANSA Core Team in Cambridge, UK. ANSA is currently under development in the Integrated Systems Architecture (ISA) Project, jointly funded by the Commission of the European Communities, DG XIII (ESPRIT 2267) and a consortium of European and North American IT and Telecommunications Companies. The ISA Project builds on the Advanced Networked Systems Architecture (ANSA) Project which started in 1985 under UK Alvey funding.

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
UUCP	...ukc!ansa!apm
ARPA Internet	apm@ansa.co.uk

CONTENTS

1 Introduction

ANSA is an architecture for building distributed systems which can individually or collectively operate as a unified whole so that the fact of distribution can be made completely transparent to application programmers and users. This is an entirely different approach to that which is typically assumed when networking single systems together. It allows full advantage to be taken of the inherent concurrency and separation of distributed systems for the provision of increased performance, decentralisation and reliability, while better masking the disadvantages that arise from communication errors and partial failures. It produces systems that can be managed as coordinated sets of sub-systems appropriate to the enterprise they serve rather than as random combinations of boxes.

2 The problem space

Many of today's computer systems are designed to work within "closed" localised contexts, either within limited physical areas, or within limited logical boundaries, possibly shared by other similar systems. When systems are required to cooperate freely in "open" distributed contexts with other dissimilar systems, both physical and logical separation can cause major difficulties.

Problems frequently arise from trying to separate and distribute closed systems in ad hoc ways, instead of applying the discipline of sound engineering practice to create open distributed systems founded on a clear understanding of carefully considered design assumptions, principles, and structuring rules.

The purpose of this paper is to examine the design approach taken by ANSA in addressing the technical problem space of building "integrated" open distributed computing systems. It is recognised from the outset that such systems may inevitably need to function in environments incorporating heterogeneous computing elements, and that such elements may be subject to different administrative authorities.

The paper concentrates on different aspects of the ANSA computational model (application writers viewpoint) and the ANSA engineering model (system builders viewpoint). Throughout the discussion reference is made to the concept of "service" from two perspectives: (1) as an abstract computational specification of some set of system/application functions; and (2) as an engineering entity that animates the same functions and makes them accessible as a service to other parts of the system.

3 ANSA design assumptions

In designing systems to be implemented in a single host environment, it is commonly the case that a number of assumptions are made which simplify the modelling of those systems. In the presence of distribution, however, those features from which the simplifying assumptions abstract cease to be negligible, and must be explicitly catered for in the design models. It is important, therefore, to identify explicitly the assumptions which are made for non-distributed systems, and to ensure that they are absent from the models for distributed systems.

Among the most important assumptions to be *avoided* are:

- *Single global name space*: in distributed systems, which may arise from federation of pre-existing systems, context-relative naming schemes are required in order to interpret names unambiguously across different administrative boundaries
- *Global shared memory*: in distributed systems, global shared memory would form a performance bottle-neck, and so is replaced by multiple, disjoint memories.
- *Global consistency*: in distributed systems, consistency of state and data may converge more slowly than state and data in non-distributed systems.
- *Sequential execution*: in distributed systems, execution may occur in any and all possible orders including sequentially, concurrently, and independently.
- *Total failure*: in distributed systems, the failure of one component can lead to partial failure of other operational components participating in services affected by the failure. Redundancy of components is thus essential to detect and mask failures in order to allow operational components to continue dependably. Moreover, there is a class of partial failure modes which can only be prevented by distributed cooperation; in particular, neither forward nor backward error recovery performed on behalf of a particular failed component may on its own be sufficient.
- *Synchronous interaction*: in distributed systems, both asynchronous and synchronous interactions are necessary in order to reduce communication delay.
- *Locality of interaction*: in distributed systems, interactions may be either local or remote, with consequent implications for communication delay and reliability.
- *Fixed location*: in distributed systems, which may have multiple locations, it is possible for components to move during the system lifetime. Advantage may be taken of this to improve performance by

co-locating some interacting services, but the processes for finding services must be extended to cater for this.

- *Direct binding*: in distributed systems, indirect binding is necessary to cater for the potential remoteness of interactions and the mobility of services.
- *Homogeneous environment*: in distributed systems, there can be no guarantee of homogeneity of components and so interacting parties must agree on abstract rather than physical data representations.

Summaries of the design principles observed by ANSA and the way in which they avoid these assumptions are presented below.

4 ANSA design principles

It is convenient to discuss the design principles of ANSA in the context of five key issues of distribution: *separation, heterogeneity, federation, concurrency,* and *scaling.*

4.1 Approach to separation

Physical separation of interacting computational entities brings the need for a general computational model for interworking.

With respect to separation, ANSA observes the following principles:

- *Assume all services are remote, allowing co-location as an optimisation*
- *Require each service to be entirely responsible for transforming its encapsulated data*
- *Perform all interactions with services via instances of interfaces*
- *Allow propagation of interface references as the means of acquiring access to services*
- *Name and report all detected interaction faults and failures*

Part of the solution to separation is to assume from inception that all services are physically or logically remote from each other, leaving the possibility of co-location, with the potential for optimisation it yields, as an engineering concern. This view leads to the requirement for each service to encapsulate its data.

Another part of the solution is to ensure that the state and data of each remote service can only be manipulated indirectly via interaction with one or more interfaces supported and made available by the service. This approach is similar to the programming view of manipulating data through abstract data types.

This picture mirrors the essential properties of the ANSA computational model in which remote services are shared by propagating interface references between interacting parties. The components between which service interactions occur are formulated as "computational objects", each of which encapsulates data and the service operations for manipulating that data. Consequently, each object and its data are wholly contained within a private memory space, which is disjoint to the private memory spaces of other objects.

Specification of what each object does is primarily in terms of the services it provides, and thus the types of its interfaces. Likewise, specification of how an object achieves its required services may be modularised in terms of the services they use. The technique of one service using other services can be applied recursively to yield computational objects of extremely fine granularity.

This object model is synergistic with, but not dependent on, specific object-oriented programming models.

A client-server relationship applies to service interactions. One object (the server) provides one or more services to other objects (the clients). There is prior agreement about service specifications: services conform to particular interface types, and the interactions occur by mutual consent, and at the initiative of the clients. The same object may participate in interactions with many different services of the same or different interface type (concurrently and/or consecutively, as client and/or server, and with many different partners).

In distributed systems the physical separation of services (and their containing objects) is unavoidable, introducing the possibility both of failures occurring during communication and of partial failures of the services. In order to have equivalent failure semantics to non-distributed services, the service interaction model must allow such failures to be reported and processed.

Service interactions therefore require multiple outcomes (each of which may comprise multiple results). In ANSA, this approach has been integrated into a general model for reporting different kinds of outcome. Such outcomes are distinguished by name and are known as *terminations*. Connotation of "failure", as well as which terminations represent failure, is defined as part of the service semantics rather than as part of the interaction model.

4.2 Approach to heterogeneity

There are variations in the design of different hardware and communications systems which arise for a variety of reasons. Not all systems are designed to be the same, nor is it always desirable that they should be, since there can be benefits in diversity and specialisation. The challenge is to make such diversity work harmoniously and to derive positive benefit from the specialisations it provides.

To provide the flexibility to cope with inevitable variation in distributed systems, ANSA observes the following principles:

- *Assume heterogeneity and identify unnecessary diversity*
- *Abstract away from unnecessary diversity, while still retaining the benefit of specialisations*
- *Request remote services to manipulate their encapsulated data through interface instances*
- *Pass interface references rather than data presentation syntax*

Consideration of *heterogeneity* and the discovery of *unnecessary diversity*, and the impedance that these *phenomena* present to interworking compatibly across different systems, leads to the identification of the root problems: the incompatibility of different operating system interfaces, the incompatibility of different physical (hardware) and logical (software) data representations, and the incompatibility of different communications protocols.

Standards for logical data presentations and communications protocols exist, but these in themselves produce neither sufficient nor complete solutions to the problems of heterogeneous interworking.

These difficulties can be addressed by appealing to the principle of abstraction.

ANSA specifies an integrated platform which can be built onto existing heterogeneous operating system environments to form the basis for compatible interworking across dissimilar technologies. The definition of this platform does not require or force a particular implementation, but does require adherence to conformance on all matters of interaction between distributed services (§7).

This architectural platform supports an object based computational model for simplifying the way in which remote applications are structured and are permitted to interact (§4.1); and an engineering model which specifies the components and structuring rules for building practical realisations of the platform (§5).

Even with this platform in place, there are still other principles that must be observed if successful interworking is to be achieved. Different data representations imply that one system cannot manipulate data directly in another system. Remote information must be represented abstractly. It is important to characterise the services available, without knowing how encapsulated data is to be transformed. The approach of read/modify/write styles using primitive operations, as in stand alone systems, or homogeneous networks, are inappropriate in a heterogeneous distributed system. They result in an attempt to create a global database as a vehicle for transforming a heterogeneous environment into a homogeneous one.

These arguments suggest that it is dangerous to present the application writer with a data encoding scheme such as ASN.1. Use of a presentation syntax for transporting data from one environment to another perpetuates the view of the read/modify/write styles of data access. Moreover, there are further problems. Cooperating applications must still agree on a presentation syntax, and agree on the semantics of the data as information. The ASN.1 approach moves data to the processing, whereas an object based approach moves processing to the data, requesting the responsible service to process and transform the data wherever it happens to be.

This principle of avoiding the mechanistic view of data encoding schemes in the computational model does not of course preclude their practical application in the engineering model. Ultimately, systems have to be built with agreement on the syntax and semantics of data presentations for passing requests, parameters, and results in interactions. A standard such as ASN.1 may well be the choice for specific implementations.

The guiding principles of *performing remote service interactions through interface instances*, and of *propagating interface references to share services* provide a sound basis for dealing with problems of heterogeneity. All useful diversity and specialisations can be defined as service objects and accessed through instances of interfaces.

4.3 Approach to federation

In large-scale distributed computing systems, the existence of centralised ownership and universal and technical control cannot be assumed. There will inevitably be separate sources of authority (e.g. separate enterprises, autonomous departments, different technical policies, dissimilar technologies, and separate administrations). In such cases, interworking can only be achieved via cooperation in "federal" style, and not by "dictat".

To accommodate federation of separate systems, ANSA observes the following principles:

- *Allow each system to control its own policies and services locally*
- *Allow cooperating systems to negotiate the sharing of services*
- *Require cooperating systems to identify all available services via a context-relative naming scheme*
- *Provide a trading facility through which federated cooperating systems can organise and control the sharing of services*

A local system cannot reliably or effectively control a remote one for all the reasons underpinning the issues of separation and heterogeneity discussed earlier. Furthermore, stand-alone systems are designed to meet individual requirements, and are not deliberately built to assume non-local administrations.

A need for interoperation between individual systems arises when it is realised that some mutual benefit can be most effectively met by federating them. Since the systems were not designed to fit within some agreed overall structure, they must combine forces as cooperating peers, ideally without impairing their individual functionality or performance.

The federation of separate systems directly affects the architectural views of "naming" and "trading".

4.3.1 Naming

Names are the general means of referring to entities within a system. In information systems there are many different entities to be named, and many different ways of naming them. Any large-scale distributed computing system will inevitably encounter such diversity.

ANSA provides a naming model to address this issue:

- *Separation of naming domains*

  Separate naming domains are formulated for the different kinds of entities that can be named.

- *Separation of naming conventions*

  Different ways of naming entities are distinguished as different naming conventions. For each naming convention there is a defined syntax and semantics. In an ideal world, there might be exactly one naming convention per naming domain; but in the real world there is usually more than one (if only for historical reasons).

- *Separation of naming contexts*

  There should be considerable freedom in the way in which particular names can be associated with particular entities. Each set of bindings between the entities in a naming domain and the names in a name set is known as a naming context. Different naming contexts arise for reasons of scaling and management for instance. For consistency, each naming context must adhere to a single naming convention. The validity of names is tested with respect to the naming context; the name must be constructed using the naming conventions, and a binding with an entity in the naming domain must be defined.

- *Naming networks*

  Some of the entities that can be named are themselves naming contexts. Thus, it may be possible to name one naming context from another naming context. The structure that is formed by the manner in which naming contexts can be so linked is called a naming network. ANSA imposes no constraints on the structure or size of the naming network in a particular system and so allows arbitrary administrative structures (hierarchies as well as federations) to be reflected in the naming network.

- *Path names*

  To name a particular entity in some domain, it may be necessary to refer first to (i.e. name) the naming context in which the name is valid. The name for a particular entity is thus extended by the name of the naming context in which it is known. A path name is such an extended name - it traces a path through the naming network.

- *Name transparency*

  Each naming context that is named in a path name is logically independent of all other naming contexts in the path name. Therefore, name resolution involves successive logically independent interpreters. For each interpreter, all other elements in the path name are transparent, leaving unresolved names for successor interpreters. Each name interpreter may be modelled (and implemented) as a service object which internalises the naming context and the naming convention concerned.

The above naming model provides an orderly basis for cooperation between disjoint naming domains and contexts which can be separately administered under different authorities. This arrangement is referred to as *federated naming.*

4.3.2 Trading

It was stated earlier that interface references may be obtained by clients in response to interactions with any accessible server; and that this is the basic method by which distributed computations naturally acquire access to different services dynamically. However, it is also important to provide a means by which separate clients and servers can rendezvous for the very first time in order to allow subsequent interaction between them. In ANSA, this process is called *trading*, and is available through a special service provided to clients and servers.

Trading gives access to a graph structure that can be searched by clients via *import* requests and updated by servers via *export* requests, qualified by *typename* and optionally by *property name/values pairs*.

- *Typename*

  A *typename* denotes the set of permissible interactions that a service instance can engage in. It identifies a set of common service interface instances.

- *Property name/value pairs*

  A set of *property name/value pairs* is used to help make a choice from a set of interface instances with the same typename.

  (For example, there may be several Fourier transform services, that all calculate the same transform. The computational cost associated with each service may vary according to the algorithm used. A client of the service will get charged for each transformation. It must then be possible for the client to state how much it is prepared to pay on the basis of a choice of the most suitable Fourier transform service.)

- *Typed imports and exports*

  Servers can export instances of interface types by *typename* and *property name/values* to the trading service to make these instances accessible to clients. An import operation is provided to clients so that they can retrieve references to interface instances of the required type.

  The trading service will only search through exports of the required type (and its subtypes) when trying to match on interface type conformance and required service properties.

The trading service of different systems may be structured as a federation of autonomous trading domains and managed by separate administrative authorities.

4.4 Approach to concurrency

Assumptions made about concurrency and synchronisation mechanisms in single host systems are frequently invalid for distributed systems. This can create difficulties when transporting single host applications to distributed, possibly heterogeneous, environments.

ANSA addresses the problems of distributed concurrency and synchronisation by observing the following principles:

- *Distinguish between the computational and engineering views of concurrency*
- *Require declarative expression of parallel execution and concurrency control in the computational model*
- *Provide programmers with suitable linguistic tools for building distributed applications*
- *Provide engineering tools to map computational specifications to engineering mechanisms*

An application writer generally assumes that his program executes serially; that is, with a concurrency of one. It may be possible for components to execute in parallel, but except where meeting an explicit functional requirement, parallelism is rarely accommodated. Mechanisms to apply parallel processing to assumed serial code, for example *pipelining*, have been adopted in many systems. However, such ingenuity may not be possible with the many processors of a distributed system. Furthermore, if specific serialisation and synchronisation mechanisms are built into an application, the opportunity for exploiting parallelism via distributed processors using different mechanisms will be lost.

Distribution introduces special problems. If a server has many clients, it will inevitably be faced with overlapped requests. If a server does not make provision for concurrency, the delay imposed upon clients will become excessive. Only by designing applications with the greatest scope for parallel execution will optimal scaling characteristics be obtained.

When interacting with a remote service, an application writer may assume two options:

(1) synchronous service request: *execution continues when the remote activity is complete.*

(2) asynchronous service request: *execution proceeds in parallel with the remote activity.*

With option (2), and assuming a dependency between the interacting parties, a synchronisation mechanism will be needed to suspend execution of the issuing activity until completion of the remote activity. This type of synchronisation becomes increasingly complex if several remote parallel activities are involved. For example, the issuing activity may desire to wait for the remote activity that finishes soonest - but which one is that? And how are the remaining remote activities subsequently handled?

There are many engineering techniques for the obtainment of concurrency control, but these are too numerous to mention here.

To tackle these issues, ANSA provides a Distributed Processing Language (DPL) which makes a clear distinction between the concurrency expressed in the computation, and that which is realised by engineering mechanism. The application writer is required to indicate declaratively where in a computation parallelism is possible, or where synchronisation is required, but without any preconceived ideas about their mechanisation, whether through local or remote resources. The engineering domain is accordingly given proper control over the choice of appropriate implementation strategy. In particular, the decision on whether to take advantage of the parallel options of a computation is controlled at the proper place.

DPL is supported by engineering tools that map abstract computational specifications to appropriate engineering mechanisms.

4.5 Approach to scaling

Systems will constantly change, grow, and merge. This introduces variations in scale: small to large, slow to fast, specific to general.

In response to these needs, ANSA observes the following principles:

- *Allow for scaling variability by building expansion capability into the architecture*
- *Provide extensible naming and trading facilities*
- *Federate through negotiable, cooperating, remote services*
- *Do not assume global mutable knowledge*

The principles of separation, federation, and heterogeneity enforce the view that it is not possible to assume the existence of a widely distributed global resource pool which can be accessed directly from anywhere. It is simply not realistic to encompass the entire universe of systems for all space and time.

The ANSA view is that scaling differences must be accommodated as needs arise in much the same way as the federation principles described allow naming and trading to expand in ever wider domains and contexts.

Scaling issues are also greatly eased if data is manipulated where it is held, and all requests for its manipulation are permitted only via references to instances of interfaces. Although this requires negotiation of service agreements, it makes no assumptions of global mutable knowledge.

5 Engineering structure of ANSA systems

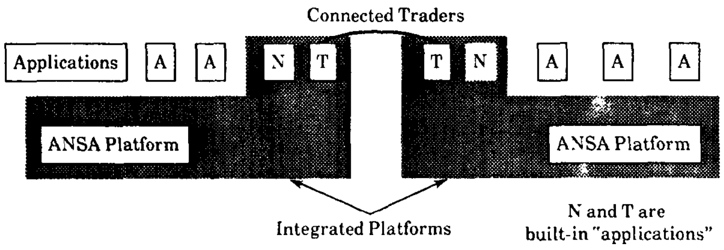
The following presents a sketch of the way in which the principal components of ANSA systems fit together.

Figure 1 shows two ANSA systems. Each system is running several applications (comprising clients and servers), together with a *node manager* (N) and a *trader* (T).

Each trader provides a trading space that can be searched by *typename* and by optional *property values*. Any server can export instances of interface types to the trader in order to make them accessible to clients. Any client can use import operations on the trader to acquire access to required interface instances.

The traders are connected (possibly federated) to permit the sharing of services across the systems. This federal arrangement, together with the distributed integrated platforms, gives the illusion to clients and servers in both systems that they exist in a single homogeneous system.

Figure 1: ANSA systems



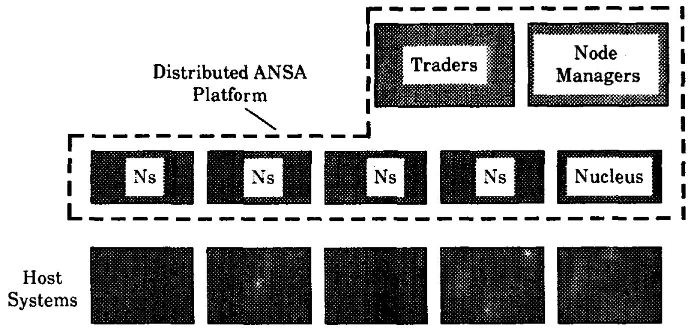
Each node manager maintains a database of configuration details pertaining to its node in the distributed system. A *node* is the engineering abstraction of a host machine in the system.

As shown in Figure 2, a node supports one or more nucleus components (Ns), each of which takes the basic resources of its local host's infrastructure (operating system and hardware) and builds upon it to provide a basic distributed computing environment common to all hosts in the distributed system. The nucleus components are then able to work together, along with the traders and the node managers, to provide an integrated support platform for distributed computing.

Node managers work in conjunction with a distributed *factory service* (not shown) to instantiate application objects above the platforms. The factory service creates capsules for the containment of instantiated application objects.

Figure 3 reveals the structure of an ANSA *capsule*. Each capsule's address space will be logically partitioned to provide a private memory space for each

Figure 2: Distributed nucleus components



contained application object. The distributed system will comprise one or many objects per capsule, one or many capsules per node, and many different nodes.

Transparency services are the components that enable the various aspects of distribution to be hidden from application clients and servers (see §6).

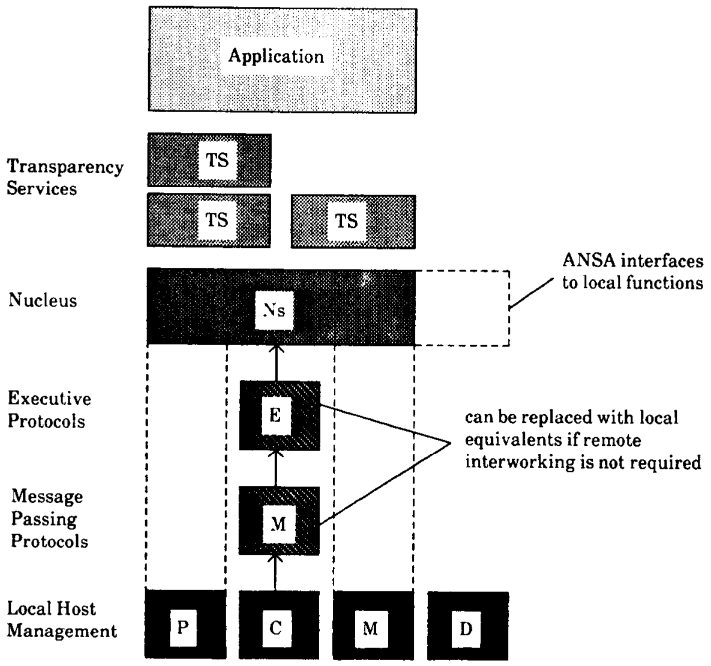
At a level below the nucleus are the components that provide executive (O/S) and message passing protocols. (If interworking between heterogeneous systems is not required, either or both of these can be replaced by local equivalents.) The lowest level contains the physical host's processor (P), communications (C), memory (M), and device (D) management functions.

The ANSA engineering model specifies the mechanisms needed to provide the various kinds of transparency and the protocols for interaction between nucleus components on different node/hosts. Application components are structured according to the ANSA computational model, and the distributed computing aspects of the application are compiled into calls on the interfaces to the appropriate transparency and platform components.

The engineering model can also be taken as a template for the implementation of the nucleus and the transparency components, although this is not mandatory for either application portability across implementations, or for interworking between them. The conformance criteria for portability are the interfaces to the transparency and platform components. Once conformance to the computational model has been established, it is possible to conceive of multiple implementations of the architecture which make different engineering trade-offs (see §7).

Many hosts will provide a range of functions and resources beyond those needed by the platform and may wish to contribute them to the distributed computing environment as potential application components. This can be achieved by extending the nucleus with additional interfaces that map onto the locally available functions. Thus the nucleus acts as an architectural

Figure 3: An ANSA capsule


Application
Transparency Services
TS
TS
TS
Nucleus
Ns
ANSA interfaces to local functions
Executive Protocols
E
can be replaced with local equivalents if remote interworking is not required
Message Passing Protocols
M
Local Host Management
P
C
M
D

switch, transparently linking application components to both local and remote resources in a uniform way.

6 Transparency services

The question of whether it is practicable to distribute a computation may depend on many things. Where communication costs are high it may be prudent to minimise the distribution of those parts that are expected to interact heavily. Where parts of a computation are processor intensive, the extra concurrency introduced by distribution may lead to improved performance. Where replication is used to increase reliability and availability, it is essential that software replicas are located on distinct hardware replicas.

The extent to which an application writer needs to be concerned with the integration of distributed system components can be controlled by the selective application of *transparency services.*

In an application with complete distribution transparency, the application writer has delegated all responsibility for distribution to the underlying support environment. Without such support, the writer must assume full responsibility for all aspects of distribution.

In practice, the application writer may require control over *selected* aspects of distribution. For example, a configuration management application would obviously require control over the location of system components. By allowing the selection of transparency services, each application need only deal with those distributions aspects that are pertinent to the application.

ANSA supports the following transparency services:

- *Access transparency* hides the difference between local and remote provision of services. The overriding criterion is to remove the concept of co-located clients and servers. (Local optimisations can be effected by engineering decisions where appropriate.) With this transparency service in place, all invocations are considered to be remote.
- *Location transparency* hides the location of servers from the clients that interact with them, and vice versa; thus enabling interacting parties to be located anywhere in the distributed system.
- *Migration transparency* hides the effect of servers moving from one location to another while clients are interacting with them.
- *Concurrency transparency* hides the existence of concurrent users of servers. If a server is supported by concurrency transparency, then each of its clients is unable to observe any effects due to other clients that make simultaneous use of that service
- *Failure transparency* hides the effects of partially completed interactions that fail for what ever reason. This transparency service is built upon mechanisms which

  (a) make interactions *atomic* so that they either complete entirely, or fail with complete removal of partial effect;

(*b*) make interactions completely impervious to single point failures in client and server configurations comprising *replicas*

- *Replication transparency* hides the difference between replicated and non-replicated clients and servers

The technique for providing any transparency service is based on the single principle of replacing an original service by a new service which combines the original service with the transparency service, and which permits clients to interact with it as if it were the original service. The clients need not be aware of how these combined services are achieved.

7 Conformance

A *conformance point* is a place where a test can be made of a system component (a platform component or an application object) to see if it meets a set of *conformance criteria.* A *conformance statement* for a component must identify where the conformance point is, and what criteria are satisfied at that point.

In ANSA, all architectural conformance points are described abstractly rather than by reference to concrete data formats and protocols. Thus architectural conformance does not automatically guarantee interworking or portability. Practical interworking and portability guarantees require systematic choice of actual formats and protocols, or the use of translators between alternative formats. These are *system conformance* choices and fall outside of the architecture's rules and recipes.

Note however that conformance to the architecture does not always guarantee compatibility of interworking, as the following example makes clear.

Imagine two airline reservation systems built using the same hardware, protocols and programming languages, conforming to ANSA throughout. The information structures for flight reservations and cancellations are the same. Since both systems serve the same purpose it might be hoped that they will work together. Suppose however they have different ways of treating cancellations. One may have an exchange policy: "make the customer a booking on another airline"; the other a refund policy: "give the customer's money back". When both systems are interconnected, the clash of policies could cause problems to reservation staff as well as to passengers, since the composite system will not exhibit a consistent cancellation policy.

To overcome this class of problem, service specifications must be cross-checked for compatibility on all points of policy between the application components, and between all supporting ANSA components.

The following provides some guidelines on system conformance in the context of the ANSA computational model and the ANSA engineering model.

7.1 Conformance in the ANSA computational model

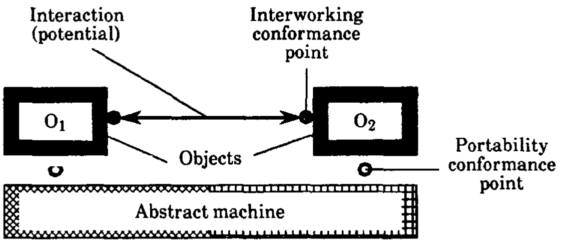
The ANSA computational model is in two parts:

- the *interaction model* defines permitted forms of interaction and a type scheme within which potential interactions are to be classified.
- the *construction model* defines elements from which the interacting objects may be constructed.

The structure of the model and the organisation of the description of the model are derived from the relationships that exist between computational objects and the relationship between a computational object and its supporting environment. The model establishes conformance requirements that must be satisfied if the pieces of a distributed system are to fit together.

There are two computational conformance points; the *interworking conformance point* and the *portability conformance point.* Figure 4 shows two objects and the positions of the conformance points with respect to the objects and the environment that animates them.

Figure 4: Computational conformance points



It is possible to conform to the interaction model without conforming to the construction model. Conforming to the construction model guarantees conformance to the interaction model since there are no interaction facilities other than those corresponding to the interaction model.

7.1.1 Computational interworking conformance

At the interworking conformance point there are two kinds of conformance. The first is conformance to the interaction model. The second is interface type conformance for potential interactions.

An *interaction conformance* statement for an object asserts that all interactions at the conformance point follow the rules of the interaction part of the ANSA computational model.

Interface type conformance applies to the potential interactions between objects rather to the objects themselves. An interface type conformance statement can be made only about a potential interaction in which the participant objects are interaction conformant. An interface type conformance statement for a potential interaction asserts that neither party to the interaction will attempt to interact in a way that the other does not expect.

Objects cannot interact if their models of interaction are different. Interaction conformance is mandatory for an object that is to participate in an ANSA system.

7.1.2 *Computational portability conformance*

The *portability conformance* point is between an object and the abstract machine which animates it.

A *portability conformance* statement for an object asserts that the object is defined in terms of the elements of the ANSA construction model.

A statement of *portability conformance* for an abstract machine asserts that it can animate objects that conform to the ANSA construction model.

Each object must match the animation environment that supports it. The animation environments in a system need not conform to the ANSA construction model. If a system has animation environments based upon more than one model then there will be restrictions upon where each object may be placed which will limit the way in which the system resources can be exploited.

The ANSA construction model has been designed to be well matched to the interaction model and also to permit the development of mechanisms and techniques that allow the resources of a distributed system to be exploited effectively.

7.2 Conformance in the ANSA engineering model

Figure 5 shows an engineering structure of an ANSA system that illustrates application objects, transparency services, nucleus components, operating systems, and the underlying communication networks.

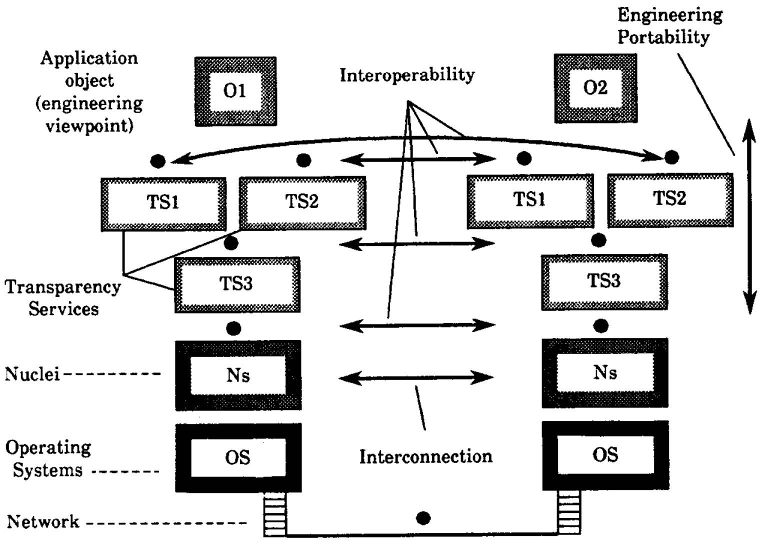
7.2.1 *Engineering interworking conformance*

In the engineering viewpoint there is an *interworking conformance* point between interacting engineering objects. Two kinds of conformance statement can be made at this point.

A statement of *interoperability conformance* for an object asserts that a stated layering of transparency protocols will be applied above the nucleus-to-nucleus protocol to all interactions through the conformance points. Interoperability conformance guarantees that the required transparency can be maintained with other nodes asserting the same interoperability conformance. System interoperability conformance can be tested relative to a specified test service, test interface, and stack of interconnection protocols.

A statement of *interconnection conformance* for a node asserts that identified nucleus-to-nucleus communications services will be used to exchange data and synchronisation messages. Interconnection conformance guarantees remote interaction between nodes. System interconnection conformance can be tested relative to a specified test service, test interface, interconnection protocols and data formats.

Figure 5: Engineering conformance points



7.2.2 *Engineering portability conformance*

In the engineering viewpoint there is an *engineering portability conformance* point between an engineering object, and the transparency services and nucleus upon which it depends.

A statement of *engineering portability conformance* for an object asserts that the procedures and data structures comprising the object conform to the definition of a given engineering object specification, and that the object depends upon a specified selection of particular transparency service interface types.

A statement of *engineering portability conformance* for a node asserts that it provides a nucleus and a given set of transparency services for the execution of engineering objects.

Engineering portability guarantees the ability to exchange engineering objects, including transparency services, with other conforming nodes. Systems conformance at this point asserts that the node will accept one or more concrete representations of objects conforming to the nucleus and the interface types of the transparency services.

Engineering portability conformance can be omitted when exchange of engineering objects between nodes is not a requirement.

8 Summary

This paper has presented a a brief picture of the technical design philosophy of the ANSA architecture from the perspectives of the ANSA computational model and the ANSA engineering model. These two different but complementary viewpoint models do not, however, tell the whole story. ANSA also defines other models with specific focus on enterprise, information, and technology viewpoints. Moreover, many technical issues and/or details have not been discussed, e.g. security, atomic transactions, interface groups, fault management and recovery, concurrency control methods and event ordering techniques, distributed programming language facilities and interface type systems, and system installation management. The architecture covers these aspects, and much more, but the interested reader will need to consult specific ANSA technical reports and manuals. This technical literature is available through Architecture Projects Management Ltd, Cambridge, England.

9 Acknowledgements

The editor would like to take this opportunity of expressing gratitude for all written and verbal contributions to this paper given by technical members of the ISA project core team: John Bull (APM), Jane Dunlop (APM), Andrew Herbert (Chief Architect, APM), Yigal Hoffner (APM), Nicola Howarth (APM), David Iggulden (APM), Rob van der Linden (Research Manager, APM), Erling Lindholm (Ellemtel), Cosmos Nicolaou (APM), Dennis Nyong (CASE), Michael Olsen (HP), Ed Oskiewicz (BT), Dave Otway (GEC Marconi), Owen Rees (APM), Alastair Tocher (STL), John Warne (STL), and Andrew Watson (APM).

Additional thanks are due to John Dobson (Newcastle University) for contributing points of clarification to parts of the text.

Appreciation is also extended to all management, business, sales and secretarial staff of APM for their dedicated support of the ISA/ANSA pursuit, its ideas, and its promotion: Janice Crofton, Mike Eyre (Managing Director), Andrew Herbert (Technical Director), Chris Jones, David Learmonth (STL seconded to APM), Elaine Mills, Garth Shephard (Director), Bill Talbot (Company Chairman), Judy Tillotson (Company Secretary), and Hugh Tonks (Business Manager).

Finally, special thanks must be given to all collaborators and associates of the Esprit ISA project, without whom this opportunity would not have arisen.

10 Background

The Advanced Networked Systems Architecture (ANSA) originated in a project undertaken by BT, DEC, GEC/Marconi, GPT, HP, ICL, ITL, Olivetti, Plessey, Racal and STC within the UK Alvey Information Technology Programme . As the results of the project became more well known it became apparent that a more formal structure was needed to manage the development and exploitation of the architecture. To this end Architecture Projects Management Ltd (APM) was set up as a company in 1989. APM undertakes work on ANSA on behalf of the sponsors at a central laboratory in Cambridge, England. Much of the work is currently funded via the Commission of the European Communities (CEC) ESPRIT II Programme within a project called ISA - Integrated Systems Architecture - in which many of the sponsors of APM are joined by AEG, CASE, CTI-Patras, Ericsson Telecom, Televerket, Philips, France Telecom (SEPT), and Siemens. The architecture continues to be known as ANSA, and APM also trades under the name ANSA.

11 Standards

Standards are an essential part of the development of distributed processing systems. This was recognised early in the ANSA phase of the project, and strong efforts have been made to introduce the architecture into standards work. The main activity has centred on the ISO/IEC JTC1 WG7 Open Distributed Processing project where project members are active at the national and international level. In this particular forum the ideas of the ANSA Architecture have been accepted and incorporated into the working draft of a prescriptive model of Open Distributed Processing.

There are two other standards activities where the project is active through the participation its members. The first is ECMA whose technical reports are directed to the ISO ODP work and the second is CCITT whose work on a Distributed Applications Framework has a technical orientation based more on telecommunications but which nevertheless has a strong overlap with the ISO work. This overlap shows itself in a number of project members who contribute to both activities. An agreement has recently been made between ISO and CCITT for joint working which is expected to lead to joint text. Work has also started on specific standards related to the ODP framework, notably Remote Procedure Call, and on plans to generate new work items, for example, on trading, are emerging as the framework activity matures.

Other relevant standards activities, reflecting on the large scope of the topic, such as document architecture, dictionaries, application programming interfaces, user architectures, database reference models, upper layer architecture, etc are kept under review by the team. Contributions to the ECMA work on Support Environments for ODP, Remote Procedure Call, and Open Systems Architectural Framework are made either directly or by review.

12 Related documents

(a) The ANSA Reference Manual, Vol. A, B and C, *Architecture Projects Management Ltd., Cambridge*, 1989.

(b) ANSA: An Engineer's Introduction to the Architecture, *Architecture Projects Management Ltd., Cambridge*, 1989

(c) ISO/IEC JTC1/SC21/WG7: Topic 4, Doc. N309, *ODP*, October 1990.

(d) JTC1/SC21/WG7 & CCITT/SG VII, Doc. N314, *ODP*, December 1990.

(e) JTC1/SC21/WG7 & CCITT/SG VII, Doc. N314, *ODP*, December 1990.

(f) ISO/IEC JTC1/SC21/WG7, Doc. N315, *ODP*, December 1990.

(g) Bull, J. A., Object Management Group, Object Request Broker, CO.059, *Architecture Projects Management Ltd., Cambridge*, 1991.

(h) The ANSA Computational Model, AR.001.00, *Architecture Projects Management Ltd., Cambridge*, 1991.

(i) Linden, R.v.d., Trading in the Five Projections, RC.101, *Architecture Projects Management Ltd., Cambridge*, 1990.

(j) The ANSA Naming Model, AR.003.00, *Architecture Projects Management Ltd., Cambridge*, 1991.