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

Access Federation

Gray Girling

Abstract

Surmounting the technical boundaries that separate domains are key requirements when expanding businesses. The integration of another enterprise into a business may be ineffective if the IT resources of each part of the business are not mutually accessible. Furthermore the process of doing business itself is increasingly facilitated by the ability for customer and merchant IT systems to interoperate, for which the ability to access each other is an obvious prerequisite.

The elimination of unwanted boundaries to access can only be achieved once the nature of inter-domain access is understood and mechanisms to integrate domains are identified.

This document defines the principle barriers to inter-domain access and proposes a range of mechanisms through which access can be established together with an analysis of their benefits and disadvantages.

APM.1266.00.01 **Draft** 28 June 1994

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

Distribution:
Supersedes:
Superseded by:

Access Federation

Access Federation

Gray Girling

APM.1266.00.01

28 June 1994

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

Surmounting the technical boundaries that separate domains are key requirements when expanding businesses. The integration of another enterprise into a business may be ineffective if the IT resources of each part of the business are not mutually accessible. Furthermore the process of doing business itself is increasingly facilitated by the ability for customer and merchant IT systems to interoperate, for which the ability to access each other is an obvious prerequisite.

[APM 1139.0 94] described the environment in which federation is assumed to take place, identifying the relevance of services, objects, boundaries and domains. A number of properties including “service access” that potentially form separate boundaries and domains were isolated and the principle problems that need to be overcome at each such boundary were described.

1.1 Access Paradigms

There are many metaphores that can be used to provide an insight into what is assumed to be required by computing entities when they wish to access each other. Examples include “message passing”, “procedure call”, and “events“ each of which is, at best, informally defined in colloquial use. Although there is sometimes a systematic means of realising “access” in the sense of one metaphore using access mechanisms created explicitly to support “access” in the sense of another these translations are often unintuitive and sometimes impossible. In effect, metaphores express different models, or paradigms, for what “access” actually is.

The use of a different access paradigms can present not only serious technical difficulties when attempting to interoperate but also conceptual problems in the minds of designers who must specify the operation of systems in which many paradigms are used.

One of the most important aspects of [ISO ODP] is that it provides a precise definition of standard access paradigms. This document does not attempt to describe the “federation” that might be requried between domains in which different access paradigms are used, it assumes the standard ones defined in [ISO ODP]. Two principle access paradigms are provided, one of the invocation of functionality and another for the transport of information. In both cases accesses are relevant to interfaces and an “access” is achieved as a result of a two phase process:

1. bind to the required interface[1]; and,

1. This stage may be initiated by the interacting parties (in which case it is called explicit binding) or by the infrastructure that supports interactions (in which case it is called implicit binding).

2. invoke the function (operational interface) or transfer the information (stream interface).

The “remote procedure call” object interaction paradigm is advocated and/or used by a number of other significant international standards including [OMG CORBA 93], [ISO Management 91], [ITU-T MHS 88], [ITU-T Directory 88] and [ISO RDA 90].

This implies that for the purposes of this document, at a first level of decomposition, access methods can vary in terms of the way in which they achieve:

- interface binding (implicit and explicit),
- invocation, and
- stream use.

The remainder of this document includes a range of mechanisms for federating binding and invocation. It may be that many of the solutions developed can be extended to stream use, but streams also present some practical issues that remain to be investigated – principally those regarding the potential separation of binding and access interfaces as described in [ISO ODP].

1.2 Service Access Federation

As defined in [APM 1139.0 94], a *service* is a set of capabilities available to a population of clients, but is conceptually distinct from whatever mechanism or configuration of parts that is used to provide it. When using the above paradigm its capabilities are manifest in either an operational interface or a stream interface [ISO ODP-3 94].

In this context *service access* is the means to:

- bind an interface reference to an instance of an interface;
- transfer the information required to utilise an operation provided at an instance of an operational interface; or,
- transfer the information of a single flow [ISO ODP-3 94] in an instance of a stream interface.

An *interface reference* is a name that embodies an opportunity (but not the right) to use a service provided at an interface [APM 1003.1 93].

An *access method* [APM 1139.0 94] is a set of mechanisms designed to accomplish service access (see example in figure 1.1).

[APM 1139.0 94] identifies three main boundary related problems: federation, separation and monitoring. This document addresses only federation.

Service access federation masks the boundary [APM 1139.0 94] formed by the use of different access methods. An *access method domain* is a set of interface users and providers all of which use the same access method.

1.3 Scope

This document focuses on the problem of service access federation which is one of the properties of a distributed system shown in figure 1.2.

[APM 1139.0 94] decomposed service access federation into:

Figure 1.1: An Access Method

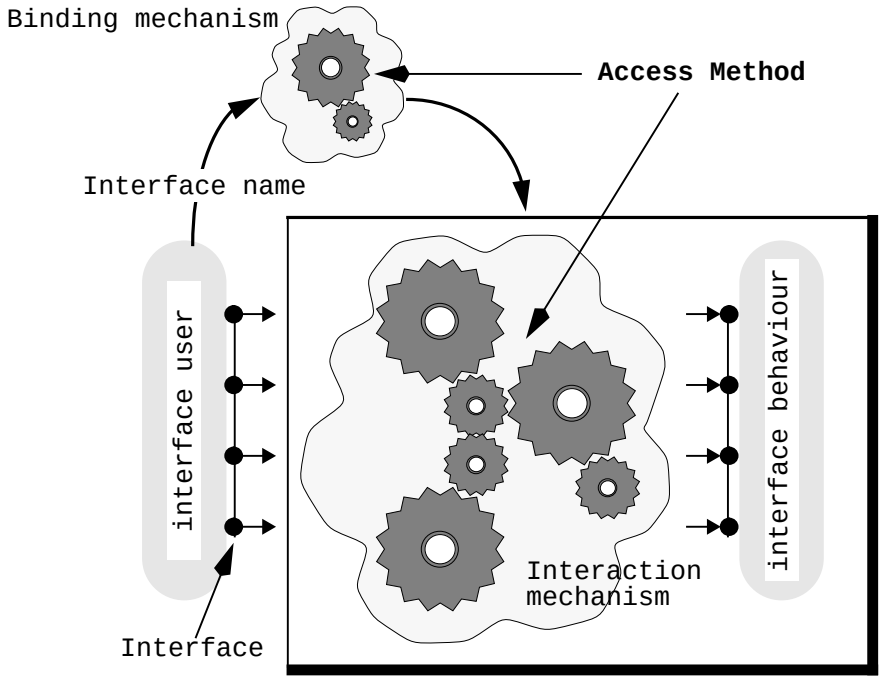
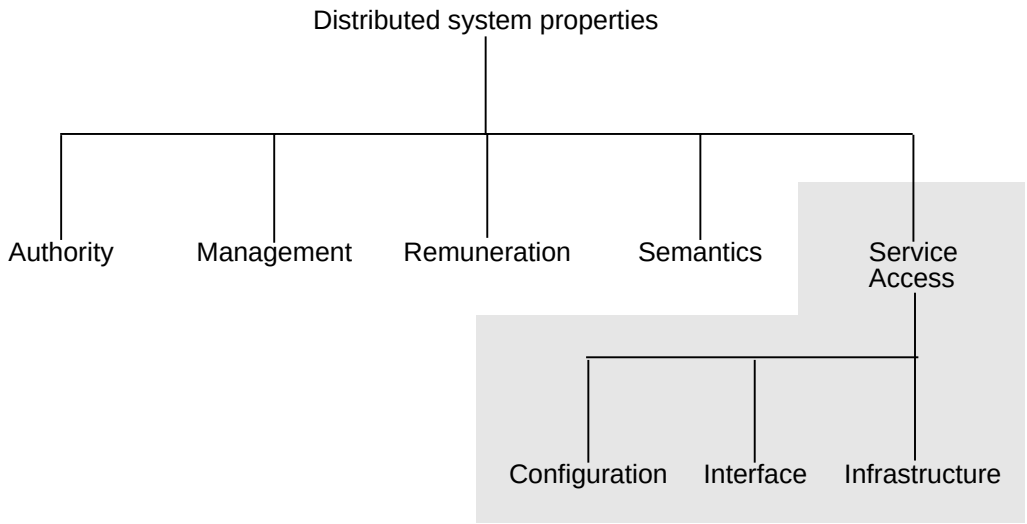


Figure 1.2: Properties of distributed systems



- infrastructure federation – access from a domain in which one infrastructure supporting interaction is used to another domain in which another is used;
- interface federation – access from a client requiring a operation with one specification to a service that provides an operation with another specification; and,
- configuration federation – access from a client expecting one configuration for the provision of a service's operations to a distributed service that provides a different configuration.

1.4 Document Structure

The next part of this document is structured into chapters addressing infrastructure federation, interface federation and configuration federation. Each of these chapters are divided into sections that consider the issues of interface binding and invocation separately.

2 Infrastructure access federation

2.1 Federating Binding

Binding is implemented in ANSAware [APM ANSAware 92], Orbix [Iona 93] (an implementation of the Object Management Group's Common Object Request Broker Architecture (CORBA) [OMG CORBA 93]), the Open Software Foundation's Distributed Computing Eenvironment (DCE) [OSF DCE 92], and elsewhere. In each case binding has the same semantics:

- an interface reference is mapped onto an interface instance;
- the integrity of this mapping is preserved over the binding's lifetime (e.g. despite the provision of migration transparenciy).

In general this behaviour establishes a context in which the interface can be accessed. The "interface reference" associated with this process has different names in different cases: for example it is referred to as a Universal Unique Identifier (UUID) in [OSF DCE 92], an interface identifier in [ISO ODP] and an object reference in [OMG CORBA 93]. "Interface reference" is the ANSA term defined in [APM 1003.1 93] which was also used in the [APM ANSAware 92] implementation.

However, dispite the similar semantics, there are engineering differences between the different implementations including the following:

- the format of binding information provided in interface references, both in terms of its representation and its semantics; and,
- the binding mechanism (e.g. protocols) used.

They also vary in terms of the form of the binding operations (c.f. a binding API). However this issue (essentially, the presentation of binding functionality) is independent of binding federation (it is a portability rather than an interoperability issue).

A *naming context* is a set of mappings from names onto entities in which the same name refers to at most one entity [APM 1003.1 93]. Interface references are names that are used to refer to interfaces. An access method will have an associated naming context which represents the interfaces that will be bound to interface references.

A *binding domain* is that part of an access method domain that employs the access method to implement a specific naming context. It will have an associated name space (the set of names that can be used as interface references) and an associated naming domain (the set of interfaces that can be named). Instances of DCE, Orbix and ANSAware implementations form examples of binding domains.

2.2 Federating binding – Binding mechanism differences

2.2.1 Universal binding mechanism

This solution is to adopt (universally) the same binding mechanism for every instance of binding. This does not solve the problem of interworking between different binding mechanisms, it merely makes it unnecessary.

Reliance on common or universal adoption of standards is contrary to basic ANSA architectural principles [APM 1000.1 93].

2.2.2 Engineering conversion in initiating infrastructure

This solution involves the binding initiator first discerning and then using whatever binding mechanism is appropriate for the type of interface reference being used.

This mechanism requires a means to derive the required binding mechanism from the interface reference. This is a task equivalent to the selection of the "next hop" binding domain, some options for which are discussed in 2.3.2.4.1.

The binding mechanisms built in to the binding initiator's are liable to be limited in number and this will place limits on the range of binding domains with which federation is possible. Any additional engineering mechanism required in order to federate to a new binding domain will be required in each binding initiator. Its installation is thus apt to represent a major management task.

2.2.3 Engineering conversion in an interceptors

This solution involves the use of interceptors each acting as a proxy for a responding binding peer. Interceptors map the binding mechanism the binding's initiator uses either directly onto the one its responder users or on to one accepted by another interceptor in a chain linking the initiator and the responder.

As above this mechanism requires a means to derive the next required binding mechanism from the interface reference. This is a task equivalent to the selection of the "next hop" binding domain, some options for which are discussed in 2.3.2.4.1.

In principle a single interceptor could perform the function to select whether or not the binding mechanism of the home domain is to be used. However, in practice this is liable to be inefficient. At least this degree of selection should therefore preferably be undertaken by the binding initiator.

Again the binding mechanisms built in to the binding interceptor are liable to be limited in number. However, the installation of a new interceptors is liable to be a less pervasive management task and potentially offers access to a greater diversity of binding domains once it has been accomplished.

2.3 Federating binding – Interface reference differences

2.3.1 Normalization of Interface Reference Mobility

Referential mobility is a desirable property of a binding domain. Three levels of mobility are discriminated on the basis of restrictions placed on the means that can be used to move an interface reference from one entity to another.

1. location independent – any means of communication available can be used to transfer the interface reference from one entity in the binding domain to another;
2. access method dependent – only the associated access method may be used to transfer the interface reference; and,
3. location dependent – no method is available that will guarantee to be able to transfer an interface reference from one entity to any other in the binding domain.

An example of the latter might be provided through the use as an interface reference of a context relative communication address associated with the interaction mechanism (e.g. an X.25 address). ANSA deprecates location dependent interface references – relocation of interface references is necessary to achieve trading and is fundemental to the realization of interfaces based on the ANSA computational model [APM 1001.1 93].

Access method dependent interface references place requirements on the access method to provide a prescribed way to transfer an interface reference from one location to another.

The latter means that access method dependent interface references represented according to an application dependent encoding can not generally be incorporated into data structures used as arguments or results in operations or elements of an information flow in a stream. For example, they could not be enciphered to achieve confidentiality.

In federating binding domains it is assumed that some degree of referential mobility must be provided. This can be achieved by mechanisms that implement an “enhanced” binding domain given an underlying one.

The following sections describe and compare different mechanisms that may be used.

2.3.1.1 Indirection through domain-wide interface references

Note: To be done

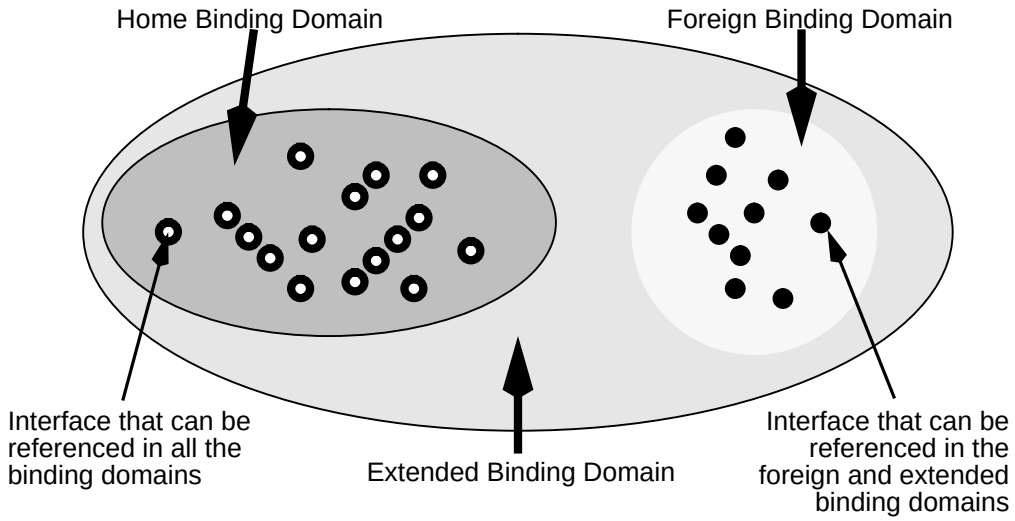
2.3.2 Widening of Interface Reference Naming Domains

The most important task of federating binding domains is to increase the range of interfaces that can be referred to (and bound to) with an interface reference, thus creating a new (larger) binding domain. Note that such extension can be asymmetrical – entities in the binding domain that can be bound to may not themselves be able to bind (to others in that same binding domain).

Widening the interface reference naming domain may thus be able to be carried out in an “autonomous” fashion – one binding domain gaining access to the entities in another without necessarily enabling reciprocal access to its own entities. If mutual federation is required it may be necessary to undertake a reciprocal widening of the interface reference naming domain associated with both binding domains.

When describing the widening of a naming domain it is useful to be able to distinguish the initial, unwidened domain from the resulting widened one. The terms “home binding domain”, “foreign binding domain” and “extended binding domain” are used when making this distinction and these are illustrated in figure 2.1

Figure 2.1: Binding Domains when Widening Interface Reference Naming Domain



The following sections describe and compare different mechanisms that may be used to widen an interface reference naming domain.

2.3.2.1 Broadening the interface reference name space

When considering the extended binding domain's interface reference name space there are two fundemental approaches to increasing the home domains's naming domain (i.e. the set of interfaces that can be bound to):

1. create a new name space for the extended binding domain (e.g. capable of incorporating the names from other federated name spaces) or,
2. use the same name space as that used in the home binding domain.

Approach 1. has the disadvantage that, if the home binding domain has previously been in use, interface references conforming to the home binding domain are liable to be embedded into the system using it.

- The representation of the home interface references is liable to be restricted by existing software that uses it – a change may require it to be rebuilt.
- Interface representations are liable to be held in traders and repositories, and possibly also in individual programs in the system – should they become invalid it may be very difficult to restore the system to its former behaviour.
- The format of the home domain's interface reference is liable to be built in to the federated binding access mechanisms of other domains – a change may mean that these other binding domains can no longer access the system using the extended binding domain.

If the home binding domain interface references have been designed with extension in mind, for example, if they incorporate information whose format is not prescribed[1] and they routinely incorporate a version identification or a relative naming context, then these disadvantages may not occur.

1. Incorporation of this type of information could be taken to indicate that the existing name space is, in fact, large enough to incorporate the names of an extended naming domain, and that broadening it is not therefore necessary.

Nonetheless, as a general proposition, broadening the interface reference name space should not be considered lightly.

Approach 2. has the disadvantage that, if the home naming domain's name space has a limited size there may not be enough interface references to name individually each of the interfaces in the extended binding domain's naming domain. This situation can be alleviated by recognising the normally finite life-time of an interface reference and to reallocate previously used ones so that the name space is used to refer only to interfaces that are currently required. The difficulty in supporting this approach is to determine when an interface reference is no longer required. Interaction with a reallocated interface reference may give unexpected results (any interface signature or environmental contract associated with the previous interface may now have been replaced by a different one). With a sufficiently large set of federated binding domains some small name spaces may still, ultimately, run out of names.

The disadvantages of 1. and 2. provide a strong argument for encouraging the design of binding domains that do not have finite name spaces for interface references. Such features might be available if all interface reference representations followed standard guidelines – however reliance on common or universal adoption of standards is contrary to basic ANSA architectural principles [APM 1000.1 93].

2.3.2.2 Universal single interface reference name space

This solution is to adopt, in every instance of binding, a single global name space. This does not solve the problem of widening of interface reference naming domains, it merely makes it unnecessary.

The change to the name space used by the system formerly using the home binding domain has the disadvantages listed above.

The arguments against this solution, or any solution that involves a global name space, can be found in [APM 1003.1 93].

2.3.2.3 Context relative naming in interface references

This mechanisms relies upon the incorporation of the representation of foreign interface reference into a home one (e.g. a DCE UUID in an ANSAware interface reference – this was the approach taken in integrating DCE and ANSAware in the HARNESS project [APM TR.037.00 93], although federation was not provided there). It does not involve the extension of the interface reference name space.

The extended binding domain interface references must identify not only an interface name but also carry some indication of the binding domain (e.g. "ANSAware" or "DCE") to which they are relevant. These extended interface references conform to the ANSA naming model, insofar as they are context relative [APM 1003.1 93].

The use of this type of context relative naming is proposed as a means to allow the interoperation of CORBA Object Request Brokers (ORBs) in [ICL 94].

A feature of this solution is that, when foreign domains have themselves adopted this approach, the information held by an interface reference may identify an arbitrarily long sequence of binding domains (each identifying the context of a name from the next). The representation chosen for them must, therefore, be capable of holding an extendable quantity of information.

The binding mechanism of the extended binding domain uses the indication of which underlying binding domain the interface reference refers to to select the correct binding mechanism (local or foreign).

Unfortunately the representation of an interface references in the home binding domain does not allways allow extension in this way (see the disadvantages of using the same name space for extended and local binding domains described above).

2.3.2.4 Binding interceptor interface reference translation mechanism

In this mechanism the home domain's name space is used in the most obvious way – interface references continue to refer to interfaces in the home domain, however a special purpose interceptor acts as proxy for interfaces in the foreign binding domain. Such an interceptor takes interface references presented in the home binding context under its binding mechanism (e.g. using the local protocol) and maps the interface reference onto a representation appropriate to the foreign binding context (which it presents using the foreign binding mechanism).

2.3.2.4.1 Variants

Variants of this approach concern the relation between home and foreign interface references and thus ways in which interface reference translation are achived. These relations include the following.

- Identity – no translation is required (although the interceptor may still have to map between different mechanisms).

  This is effectively equivalent to the approach of 2.3.2.2 except that the mapping between binding mechanisms is exported to the interceptor.

- Interpretation – the foreign interface reference can be derived using only information carried by the represntation of the home interface reference. Information from other sources is not required.

  For example: both the home and foreign addreses may be different encodings of the same name; or, the foreign address may be encoded within the part of the home address.

  The approach of (2.3.2.2 and) 2.3.2.3 is liable to conform to this case.

- Indirection – a directory function is used to look-up (perhaps only part of) the home interface reference to give the foreign interface reference.

  This approach requires a directory of home to foreign interface references to be available.

- Next-hop mapping – a directory function is used to look up part of the home interface reference to give either a foreign interface reference directly or the interface reference of a further binding interceptor to be used.

  The information provided as “home address” to the subsequent interface reference translation interceptor may be determined based on the home interface reference in a number of ways (e.g. using the above identity, interpretation or indirection approaches).

  The above “indirection” case is a simple example of this approach (in which the “next-hop” option is never identified).

This approach requires a directory of home to either foreign interface references or next-hop interface reference interceptor (whichever is appropriate for each name) to be available.

Further variants combine these methods used on different parts of the home interface reference and combine the results to form the foreign interface reference.

2.3.2.4.2 Scaling

The identity and interpretation methods have the advantage that, at any given time, the extended binding domain's naming domain is not restricted in size by a directory function. Outside the home binding domain interfaces can not be referred to unless they can be looked-up (possibly indirectly in the case of the next-hop mapping) by such a function in the indiraction and next-hop mapping approaches.

The indirection approach suffers a scaling problem if the whole of the extended binding domain is to be available simultaneously since this requires that every interface outside the home binding domain must, individually be mapped onto a different foreign interface reference by the directory function. The information storage requirements of such a function are liable to grow proportionately with the size of the foreign naming domain. The size of the foreign naming domain may grow as it becomes federated and as its foreign binding domains become federated etc. This scaling problem is limited in the case of the next-hop approach since the amount of information needs to be kept by the directory function to refer to the possible "next hops" does not grow as distant domains are federated.

The indirection approach does not suffer this problem if the whole of the extended binding domain is not requried to be available simultaneously. However, in this event, some effective policy that determines the interfaces that may be referred to at any given time must be found.

2.3.2.4.3 Access method dependency

One such policy is to provide access only to those foreign interface references that have been seen by the extended binding domain's interaction mechanism (see figure 1.1): in particular every such interface supplied in the arguments or results of operations. This policy means that interface references are access method dependent (see section 2.3.1).

However, this is also true of all of the variants referred to above (other than identity), since, assuming interface reference mobility is provided at all and given an interface reference is translated from one form to another as it traverses binding interceptors, it will have to be provided explicitly.

2.3.2.4.4 Binding interceptor engineering

A binding interceptor must appear to be a large number of different foreign interfaces and for each one must its home interface reference onto a foreign one. This can be achived by providing a number of objects each addresed by only one home interface reference or by a single object addressed by many.

The latter approach can be used only when the interaction mechanism associated with the home domain supports such objects. Not only must this object be invoked by a large number of different interface references, but the interface reference actually used must be available to the object.

2.3.2.5 *Explicit binding and rebinding*

Note: Need to consider application to results of performance work on explicit binding [APM 1239.0 94]. The impact of rebinding (e.g. for migration transparency) needs looking at too.

2.4 Federating Invocation

The data used to represent a value of a given type may vary in different parts of the infrastructure responsible for carrying out an interaction. The rules determining how values of the types in an abstract syntax are represented in a client or server (e.g. as determined by a computer language compiler) are termed a *local syntax* whereas those applying to data transferred by communication protocols are termed a *transfer syntax*. For example, an implementation of DCE or CORBA will normally define a separate transfer syntax domain.

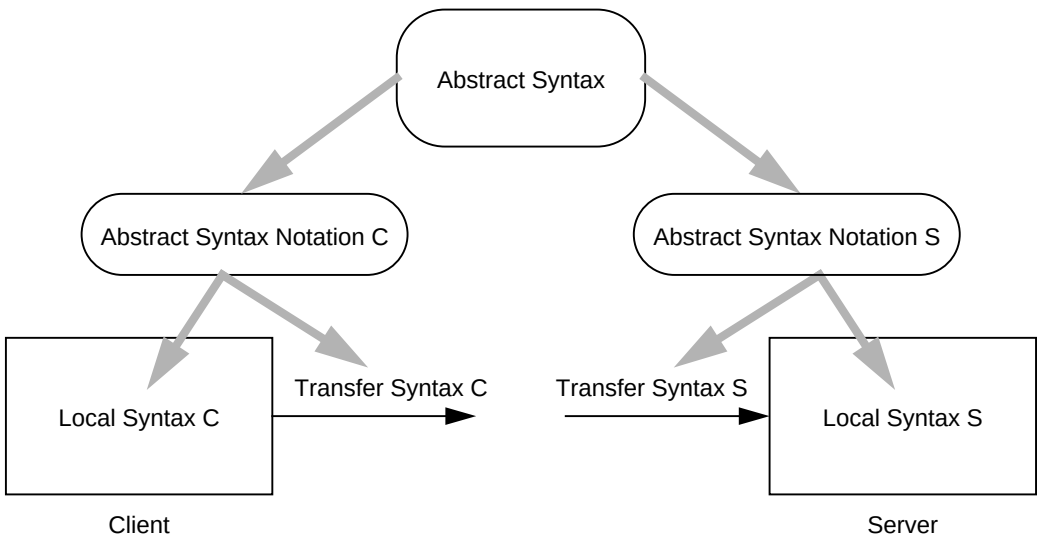
On a fine grain (e.g. when client, communication medium and server are considered as separate objects) the difference between local syntax and transfer syntax is a boundary problem that itself requires solution.[1]

The use of a transfer syntax (at all) can be regarded as a partial solution to the problem arising from a difference in local syntaxes used by a client and a server.

Any difference between transfer syntaxes associated with client and server must be overcome.

The representations of references to objects (names) needs particular care during translation to maintain the semantic distinction between the naming domain that created the reference, the naming context in which the reference is used and, potentially, the domain that created the object [APM 1003.1 93].

The following syntax differences may exist



- local syntax (in client and server)

1. Access transparency addresses this problem.

- transfer syntax (between client and server)
- abstract syntax notation (used to describe the above, e.g. IDL)

note: differences in abstract syntax are outside the scope of “Access Methods”

2.4.1 Federating Invocation - Local Syntax Differences

Epoch variations of the problem:

- known operation template & argument types (e.g. ANSAware)

known = “known before compile time”

- unknown operation template & argument types (e.g. SQL DBMS, CORBA)

the circumstances in which data bases require the last two will be described this afternon by Gomer Thomas

Possible mechanisms to bridge these differences

2.4.1.1 traditional access transparency mechanism (like ANSAware)

* specify interface in some abstract syntax notation (e.g. using IDL)

* share specification between client and server at design time

* at both client and server feed specification into stub generator

* client stub represents abstract invocation in transfer syntax

* server stub generates local invocation from transfer syntax and invokes it

- not obviously suitable for variable operation template, argument types

2.4.1.2 virtual invocation object mechanism (almost CORBA)

* communicate abstract syntax specification between server and client, **or** derive from context

* virtual invocation object supports methods to create an invocation according to specification at client

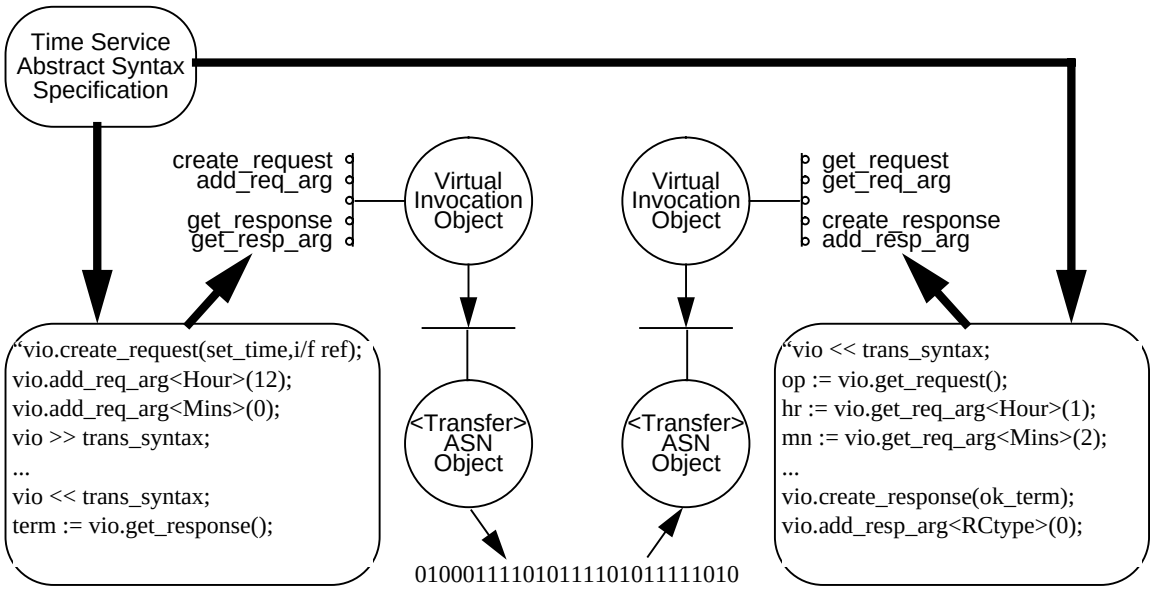
* represent virtual invocation object in transfer syntax

* instantiate virtual invocation object at server and uses specification to determine local invocation to be made

- invocation contructors provide inconvenient programmer-level interface

- interworking with traditional mechanism at transfer syntax level possible

- ATS specification can be carried embedded in transfer syntax invocation representation (e.g. type identifiers and type constructor identifiers can be incorporated), but interworking may then not be possible



2.4.2 Federating Invocation - Transfer Syntax Differences

Transfer syntax client uses is not the one the client uses

Possible mechanisms to bridge these differences

2.4.2.1 transfer syntax negotiation mechanism

* Give client and server sets of transfer syntaxes that they could use.

* Negotiate one in common

- there might not be one, requires universal adoption of one of a common set - against ANSA assumptions

2.4.2.2 common abstract syntax notation: transfer syntax interception mechanism

Assuming that client and server have invocation specifications in the same abstract syntax notation (ASN) it is possible to provide a transfer syntax translation interceptor that, given the invocation specification instantiates a virtual call object from one transfer syntax and represents it in another thus bridging this boundary.

- if the ASN specification is carried embedded in transfer syntax for invocation representation, it does not need to be provided to the interceptor via a separate route

2.4.3 Federating Invocation – Abstract Syntax Notation Differences

Abstract syntax notations (e.g. an IDL) are based on

- a selection of base types, and
- a selection of type construction mechanisms

What if the selections made in abstract syntax notation (ASN) are different?

Two abstract syntax notations may choose different base types and different type constructors (i.e. ones for which there is no direct equivalent in the other ASN)

Possible mechanisms to bridge these differences

2.4.3.1 *ad-hoc interface-specific interception mechanism*

* ignore abstract syntax specifications and generate a special purpose interceptor that acts as proxy for server based on experience and pragmatism

- not amenable to automation

- human error more likely

2.4.3.2 *ASN x ASN-specific interception mechanism*

* use greatest common subset of base types and constructors and agree an unambiguous ASN representation for others in terms of these

* an ASN interceptor, given the abstract syntax specification in one ASN makes a specifications in the other and a replacement specification in the original

- must happen before abstract syntax information is shared by client & server

- clients and servers must know and support the agreed ASN x ASN mapping

- situation in which different agreements give different mappings between same two ASNs is possible

- approach becomes geometrically more complex with additional ASNs

2.4.3.3 *generic ASN interception mechanism*

* complexity can be reduced if a generic ASN can be identified to which it is guaranteed that no extension will be necessary

* define canonical mappings between each ASN and generic ASN

* translation maps first ASN to generic ASN and generic ASN to second ASN

- may not exist - if it does more than one might exist => standardization

- clients and servers still need to support this scheme in advance

2.4.3.4 *call by reference mechanism*

* instead of transferring a representation for an argument object for which no analogous expression in both ASNs exist transfer a representation of a reference to the object

* map what would otherwise have been local uses of the object to invocations of the referenced object.

- needs both ASNs to support reference data type

- may be inefficient

- garbage collection problems

- conceptually used by ANSA computational model

Note that the infrastructure can sometimes be regarded as an object in its own right, supplying a service to e.g. a migratable basic engineering object. Most of the boundary related problems described in this document apply to this pairing of objects including problems that arise when:

- the functional and/or non-functional semantics of different infrastructure domains differ; and,
- access to the infrastructure functionality varies.

Note: (Yigal) link to Abstract and Automate [APM 1020.1 ?]

The differences between distributed system infrastructures that concern interoperability can be broken down according to the following.

- Transparency

  declarative quality of access requirements translate into the information, procedures and mechanisms necessary to provide the service with the required quality guarantees. Different systems may use different and incompatible mechanisms to achieve the same quality of access.

- Communications protocol profiles

  Different hierarchies of protocols may be used, with different guarantees and features.

- Location

  Different ways of addressing may exist, depending on the kind of network and communications protocols available.

One often neglected but particularly important form of interoperation concerns exception reporting mechanisms to support, monitoring, auditing and troubleshooting. If these mechanisms are incompatible, exception generating interaction may incorrectly appear compatible

3 Interface access federation

Differences in the interaction paradigms assumed and used by objects must be resolved.

The signature of an interface is described in terms of data types and the structure of an operation [APM 1001.1 93], flows or signal data [ISO ODP] in the interface. A number of *type systems* that differ in the way data types are defined and interrelated are likely. Each style of signature requires a means of determining access compatibility (between what is provided and what is required). Such interface access compatibility checks will normally be based on a substitutability relation inherent in whatever type system is chosen.

The combination of information describing the protocol associated with the interaction and the information about the types to be used, constitutes the *abstract syntax* of the interaction. An interface signature effectively defines the abstract syntaxes of each of the interactions that may be chosen in an interface. The language used to specify the abstract syntax in is called an *abstract syntax notation* (for example, an interface definition language). It must include constructs capable of describing the protocol implicit in the chosen interaction paradigm and constructs capable of describing types in the type system chosen.[1]

Since the main purpose of an abstract syntax notation is to prevent interactions taking place in which the abstract syntax required is not compatible with that supplied it is important that a substitutability check can be made between two abstract syntax specifications. Differences in the abstract syntax notations used to specify an interaction required and the one provided must be overcome in such a way as to enable this substitutability check.

1. If the type system is defined in terms of interactions supported on the data typed (as is the case in the type system described in [APM 1001.1 93], [APM 1014.1 93] and [APM 1015.1 93]) these constructs may coincide.

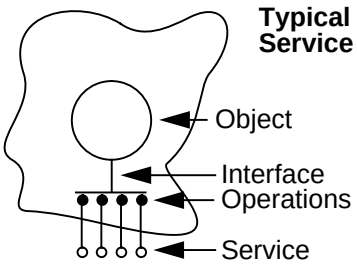
4 Configuration access federation

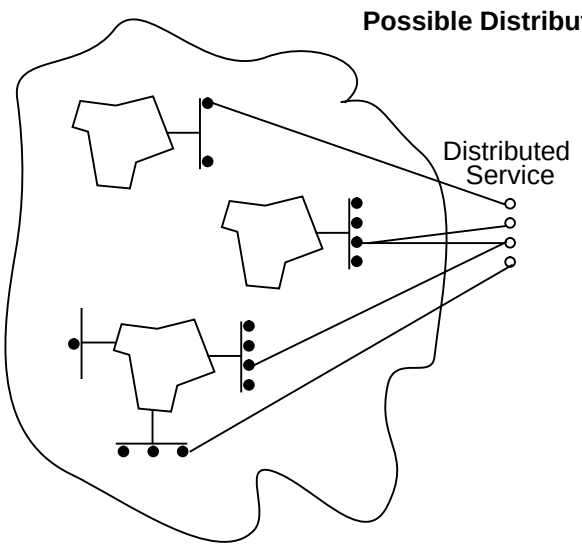
Differences in the distribution of a service's functionality over a set of interfaces assumed in the client and the server environment must be overcome by access mechanisms that map what is provided onto what is expected.

The circumstances in which such mechanism can be provided vary. The ability to use only a subset of the operations in an operational interface, for example can be determined based on substitutability rules associated with the operational interface's definition. The use of many interfaces to provide the functionality that a client associates with a single interface may depend, in part, on the level of co-operation that exists between the objects that provide the separate interfaces.

4.1 Distributed Services - absent slide

In ANSAware each useful service is expected to be designed and be provided by all the operations on one interface on one object. (Reversed assumptions)





4.1.1 Distributed services may not:

- have been designed to provide the service
- account for all of the operations in interfaces involved
- be provided by one interface
- be provided by one object
- be based on the use of the same platform/infrastructure
- have a natural/enforced distinction between platform/infrastructure and application

Notwithstanding the latter point it is assumed that a relevant distinction between computation/engineering in an implementation can be found at an early stage of federation analysis.

Whilst not a reversed assumption it is also true that the nature of a service does not always lend itself to provision at a single access point (i.e. a single object), e.g. a telecoms service.

Example: date and time service exists in one network but in another date and time are available at a time of day server and the date is available from a file server.

4.1.2 So a configuration specification would need to address:

- distributed service configuration – the configuration of objects needed to provide the “service” (we call this a distributed service)
- interface configuration – the interfaces and operations that an object provides

infrastructure – the infrastructure that supports binding, invocation and streams

5 Engineering Infrastructure - future work

- Purpose: infrastructure to support mechanisms such as those above
- Differences in semantics, remuneration, managment and authority – federation, separation and monitoring also need consideration
- Some work has already been done, e.g. in HARNESS e.g. TR 37 appendix, transfer syntax interceptors between ANSAware and DCE
- All needs to be brought together as part of a coherent architecture
- Anticipated components in architecture

We don't know exactly what will be involved at this stage but the involvement of the following kinds of component is anticipated:

- binding interceptors
- transfer syntax translating interceptors
- abstract syntax notation mapping interceptors
- abstract syntax repository
- engineering encapsulators (for separation)
- wrappers (co-located interceptors?) for federating legacy systems
-

References

[APM 1000.1 93]

van der Linden, R J, *An Overview of ANSA*, APM Ltd, Cambridge, UK, July 1993.

[APM 1001.1 93]

Rees, R T O R, *The ANSA Computational Model*, APM Ltd, Cambridge, UK, 1993.

[APM 1003.1 93]

van der Linden, R J, *The ANSA Naming Model*. APM Ltd, Cambridge, UK, February 1993

[APM 1014.1 93]

DPL Programmers' Manual, APM Ltd, Cambridge, UK, 1993.

[APM 1015.1 93]

DPL Reference Manual, APM Ltd, Cambridge, UK, 1993.

[APM 1139.0 94]

Hoffner Y and Girling C G, *Boundaries and Domains*, APM Ltd, Cambridge, UK, April 1994

[APM 1239.0 94]

Otway D J, *Explicit Binding*, APM Ltd, Cambridge, UK, 1994.

[APM ANSAware 92]

ANSAware Version 4.1 Manual Set: [APM ANSAware RM.99 92], [APM ANSAware RM.100 92], [APM ANSAware RM.101 92] and [APM ANSAware RM.102 92]

[APM ANSAware RM.99 92]

APM Ltd, *An Overview of ANSAware 4.1*, **RM.099.02**, APM Ltd, Cambridge, UK, May 1992

[APM ANSAware RM.100 92]

APM Ltd, *ANSAware 4.1 System Manager's Guide*, **RM.100.02**, APM Ltd, Cambridge, UK, May 1992

[APM ANSAware RM.101 92]

APM Ltd, *ANSAware 4.1 System Programmer's Guide*, **RM.101.02**, APM Ltd, Cambridge, UK, May 1992

[APM ANSAware RM.102 92]

APM Ltd, *ANSAware 4.1 Application Programmer's Guide*, **RM.102.02**, APM Ltd, Cambridge, UK, May 1992

[APM TR.037.00 93]

Nicolaou, C A, *ANSAware use of DCE/POSIX thread and RPC*, **TR.037.00**, APM Ltd, Cambridge, UK, April 1993

[ICL 94]

ICL plc (ed. Holt, N), *ORB Interoperability – ICL submission to the OMG Object Request Broker 2.0 Request for Proposals*, OMG document **94.3.3**, March 1994

[Iona 93]

Iona Technologies Ltd, *Orbix: Programmer's Guide*, Iona Technologies Ltd, Dublin, Republic of Ireland, 1993

Iona Technologies Ltd, *Orbix: Advanced Programmer's Guide*, Iona Technologies Ltd, Dublin, Republic of Ireland, 1993

[ISO Management 91]

ISO/IEC JTC1/SC21/WG4 and ITU-T SG VII, *Information Technology – Open Systems Interconnection – Common Management Information Service definition*, **ISO/IEC 9595** and **ITU-T Rec. X.710**, 1991

and,

ISO/IEC JTC1/SC21/WG4 and ITU-T SG VII, *Information Technology – Open Systems Interconnection – Common Management Information Protocol specification*, **ISO/IEC 9596** and **ITU-T Rec. X.711**, 1990

[ISO ODP]

[ISO ODP-1 93], [ISO ODP-2 94], [ISO ODP-3 94] and [ISO ODP-4 93].

[ISO ODP-1 93]

ISO/IEC JTC1/SC21/WG7 and ITU-T SG VII Q.ODP, *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – Basic Reference Model of Open Distributed Processing – Part 1: Overview and Guide to the use of the Reference Model*, **ISO/IEC CD 10746-1** and **Draft ITU-T Rec. X.901**, Nov 1993

[ISO ODP-2 94]

ISO/IEC JTC1/SC21/WG7 and ITU-T SG VII Q.ODP, *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – Basic Reference Model of Open Distributed Processing – Part 2: Descriptive model*, **ISO/IEC DIS 10746-2** and **Draft ITU-T X.902**, Feb 1994

[ISO ODP-3 94]

ISO/IEC JTC1/SC21/WG7 and ITU-T SG VII Q.ODP, *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – Basic Reference Model of Open Distributed Processing – Part 3: Prescriptive model*, **ISO/IEC DIS 10746-3** and **Draft ITU-T X.903**, Feb 1994

[ISO ODP-4 93]

ISO/IEC JTC1/SC21/WG7 and ITU-T SG VII Q.ODP, *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – Basic Reference Model of Open Distributed Processing – Part 4: Architectural semantics*, **ISO/IEC WD 10746-4** and **Draft ITU-T X.904**, 1993

[ISO RDA 90]

ISO/IEC JTC1/SC21/WG3, *Information Technology - Database languages - Remote Database Access - Part 1: Generic model, service and protocol,* **ISO/IEC 9579-1**, 1990

and,

SO/IEC JTC1/SC21/WG3, *Information Technology - Database languages - Remote Database Access - Part 2: SQL Specialization,* **ISO/IEC 9579-2**, 1990

[ITU-T MHS 88]

ITU-T SG VII Q.18 and ISO/IEC JTC1/SC18/WG4, *Message Handling Systems,* **ITU-T X.400** series Recommendations and **ISO/IEC 10021** , 1988

[ITU-T Directory 88]

ITU-T SG VII Q.20 and ISO/IEC JTC1/SC21/WG4, *Directory Systems*, **ITU-T X.500** series Recommendations and **ISO/IEC 9545**, 1988

[OMG CORBA 93]

Digital Equipment Corporation, Hewlett-Packard Company, HyperDesk Corporation, NCR Corporation, Object Design Inc, SunSoft Inc, *The Common Object Request Broker: Architecture and Specification,* **Revision 1.2**, December 1993

[OSF DCE 92]

Open Software Foundation, *OSF DCE Application Development Guide,* 11 Cambridge Center, Cambridge MA 02142, USA, 1992