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

Establishing Co-operation in Federated Systems

Mike Beasley, Jane Cameron, Gray Girling,
Yigal Hoffner, Rob van der Linden, Gomer Thomas

ANSA Project, Architecture Projects Management Limited, Cambridge

Abstract

Organisations have to respond quickly and flexibly to change, and large organisations are evolving into networks of co-operating small organisations. Mergers and de-mergers are common.

The above changes in organizations will have a corresponding impact on their computer systems. Distributed systems will be larger in scale than now and will cross more organisational boundaries. They will evolve as the demands of their users change, and components will be upgraded while the system is running. The software development for such systems will be distributed.

This paper addresses these concerns, considering boundaries and domains, mechanisms to deal with boundaries, trading and data management.

APM.1128.01 **Approved** 25 July 1994

External Paper

Distribution:
Supersedes:
Superseded by:

Establishing Co-operation in Federated Systems[1]

1 Introduction

1.1 Purpose and audience

With the ever growing scale and diversity of distributed systems, organisations will benefit from the ability to extend and interconnect systems in order to respond quickly to business opportunities and challenges.

Federation is concerned with facilitating co-operation between autonomous organisations for the purpose of sharing services and resources. Barriers to such co-operative computing may be technical, organisational, administrative or business-process related.

There is a need to tackle the problems associated with federation in a comprehensive and consistent fashion as they are strongly inter-related:

- trading of services across boundaries
- interception and adaptation at boundaries
- management of Quality of Service (QoS) aimed at delivering guarantees (dependability, security and performance) across boundaries.

These problems are in addition to the current industry focus on platforms for application integration (CORBA, DCE, OLE2). Solutions to federation can help solve interworking problems between these platforms.

This paper identifies the problems associated with federation and outlines the current ANSA work aimed at finding solutions to them. The ANSA architecture (both directly and via ISO ODP) is part of the OPEN*framework* reference architecture for Distributed Application Services [Brenner 93].

The issues discussed in this paper will be of interest to the following people:

- domains and boundaries: system designers and managers
- data management: systems integrators and database users
- interception: systems integrators.

1. Mike Beasley is seconded to the ANSA project from ICL, and Jane Cameron and Gomer Thomas from Bellcore.

2 The Need for Federation

2.1 The trend toward large scale, federated systems

The need for faster response to ever changing business situations has led organisations to move from monolithic organisational structures to networks of co-operating sub-organisations. Organisations typically require computing systems which reflect their organisational structure. The demand for more flexible distributed computing is high.

At the same time advances in technology are fuelling an ongoing trend of increasing computing power, storage capacity, communication availability and bandwidth, all at decreasing cost. The increasing technology base for distributed computing enables computing systems which are:

- more widely used
- larger (both geographically and in terms of computing power)
- more diverse
- more interconnected.

In short, the trend is towards large scale, interorganisational distributed systems. Increasing organisational demands, fuelled by improvements in the technology base, result in a number of interoperability and federation problems which have to be solved by software.

Computer and telecommunications vendors must be prepared to meet the growing demand for distributed computing components and services which can interoperate in large scale, federated systems.

2.2 Overview of areas addressed

The work described in this document has the ultimate aim of enabling large scale, federated, distributed systems to be built.

Four main objectives are defined:

- heterogeneous interaction: supporting the technological, naming/semantic and object description aspects of federated interoperation
- federated object management: supporting the overall management of objects and resources in a federated system.
- federated development: supporting design and development of interoperating clients and servers by different organisations

- federated enterprise negotiation: supporting the institutional, remuneration and administrative aspects of federated interoperation

Support for these objectives requires analysis of the boundaries which arise, and techniques for dealing with them (see sections 3 and 4).

The requirement for interactions to cross boundaries leads to the development of a model of interception (see section 5). This model must support the removal of barriers when interoperation is to be facilitated, and the imposition of barriers when interoperation is to be restricted.

To meet the requirements described here, it is necessary to provide access to the diverse kinds of information needed to support the four objectives listed above. This leads to the development of an extended model of trading (see section 6) and a "proof of concept" prototype trader implementation. The extended trading model must support an extensive and extensible collection of data types for information about services and offers, and must also support highly flexible retrieval of this information.

The design and implementation of the extended trader prototype itself needs query language access to data in open distributed systems (see section 7).

2.3 Characteristics of large scale, federated systems

A large scale, federated system typically spans organisational boundaries. Hence different authorities are in charge of different parts of the system.

Multiple authorities responsible for the creation, ownership and control of different parts of the system are subject to different needs, constraints and opportunities. Hence there are likely to be many differences between the parts.

System components must interoperate to meet the objectives of the various organisations involved. Generally these organisations have to negotiate terms and conditions for interoperation.

Systems are no longer static - they are in a constant state of flux to meet the constantly changing needs of their constantly changing user bases, and to accommodate new hardware and software.

System components are designed and developed by different organisations. Designers and developers need ready access to specifications of all parts of the system, at every level of abstraction, at the level of detail which the 'owner' permits. This could include contact details to tell them who to ask about parts of the system that they are not given access to.

Different mechanisms are in place in different parts of the system to enable interoperation, to meet quality of service guarantees, and to manage the creation, deletion and reconfiguration of objects and resources. As with any complex technology, it is important to be able to mask these mechanisms selectively from end users and/or application developers, and present them with a high level, application oriented interface, otherwise the complexity will be a barrier to rapid application development, will make maintenance more difficult and may restrict the reuse of applications to environments with a specific configuration.

2.4 Interoperation in large scale, federated systems

To reach successful interoperation between objects in a distributed system it is necessary to either:

- find compatible objects (trading [ISO 93]), or
- detect incompatibility and take appropriate action where possible to bridge it (interception [ISO 93]).

where “compatibility” must take account of all the following aspects, summarised in figure 1:

Note: Figure 1 goes here.

- *authority*: creation, ownership and control of resources of all kinds.
- *accounting/billing*: measuring resource usage, billing for usage, and handling payment.
- *management*: policies for resource allocation, monitoring, quality control, etc.
- *infrastructure*: mechanisms and protocols required for interoperation between clients and servers using different technologies.
- *application*: features specific to particular instances of applications or services, such as:
  - — interface specification (e.g. its signature)
  - — quality of service (QoS) specification (security, performance, dependability, etc.).

  The specifications of interface signature and of QoS are part of the application programmer’s view of the system. The complex task of relating QoS requirements to appropriate engineering mechanisms can be automated, at least in part.[1]

- *model*: semantics and naming of objects and services. This falls under an information view of the system.

Successful co-operation requires the imposition of barriers as well as their removal, for example to ensure security, to provide performance guarantees, or to manage resource utilisation.

3 Boundaries in Federated Systems

3.1 Introduction

To support and manage interoperation in federated systems, the first step is to have a good understanding of the different kinds of boundaries which arise in such systems. The next step is to analyse the different kinds of problems they pose and develop solutions to the problems.

This section presents a high level view of the different kinds of boundaries arising in federated systems. It is a revised and expanded version of the classification of boundaries which appears in Chapter 6 of [Deschrevel 93].

Section 4 discusses in general terms the kinds of problems which arise at boundaries. Section 5 discusses interception, a technique which can be used to handle many types of problems at boundaries.

3.2 High level classification

A number of areas where differences between systems are likely to occur have emerged as a result of studying the relationships between components of distributed systems (see section 2.4).

Almost always there will be differences in the information, procedures and mechanisms used in each one of those areas to achieve their goals.

3.3 Authority, management and administration

An administration is a body of people and facilities which manage operations, subject to the constraints and policies laid out by an appropriate authority. In a federated system there are multiple administrations, responsible to multiple authorities. Each administration and authority has local autonomy, subject only to agreements reached with other administrations and authorities. If services are provided and consumed across domain boundaries, the

1. The extent to which this is possible is currently under investigation in ANSA work on Performance and Dependability.

management in each domain is responsible for ensuring that all guarantees given by the clients and servers are kept.

There are several different types of management which concern distributed systems. Examples include:

- *QoS management*: providing end-to-end QoS guarantees, both within individual domains and where service provision crosses boundaries (by QoS we mean non-functional aspects of services such as security, dependability and performance guarantees.)
- *access control*: deciding who should have access to what services when
- *object management*: monitoring and control of objects ('objects' can include parts of the infrastructure):
  - — object creation, destruction and relocation
  - — resource allocation and reclamation
  - — configuration management
- miscellaneous management functions such as monitoring and debugging which are needed occasionally.

3.4 Remuneration

Remuneration includes three different processes:

- *accounting*: measuring the amount of work carried out by a server and calculating the price of the service
- *billing*: charging the user for services provided
- *paying*: transferring some currency from the user to the provider of the service. This process may include sending a receipt from the provider accounting agency to the user remitting agency.

Different systems may have different views and implementations of each one of these processes. In particular, conflicts may arise between systems which have different accounting and billing strategies. Remuneration is closely related to issues of trust and security. It is expedient to reach agreement on remuneration issues before consuming and providing services; for this negotiation to succeed, it is necessary for the relevant information concerning remuneration to be available at this point.

Remuneration is a topic in which there is currently a great deal of interest, especially concerning the Internet and the World Wide Web.

3.5 Service interfaces and properties

Functional and non-functional specifications of services have to be available in a form which allows the trading process to compare them and to find compatible objects. Service descriptions can be broken down into the following major parts:

- *interface type (signature)*: description of interfaces in terms of operations and data types. Type systems involve notions of conformance and substitutability; types are represented by interface definition languages (IDLs) and abstract data types.
  Signatures are a part of behaviour that we know how to formalise and subtype automatically; doing the checks has the benefit of preventing interaction errors.
- *quality of service (QoS)*: relates to issues such as security, performance, dependability and how to specify requirements in these areas in a declarative manner which is translatable to a variety of platforms and mechanisms.

(This excludes the issue of the behaviour of services, which is included under modelling, below.)

Different systems will hold service descriptions in different formats - for example, there are different IDLs in current use (DCE, OMG, ...), and there are also different specification languages for system management (GDMO, SNMP, ...).

3.6 Infrastructure: engineering and technology

Distributed system infrastructures provide generic facilities which are common to a wide range of distributed systems and are used by the distributed application programmer.

We can classify the differences between such infrastructures according to the categories of information necessary for setting up the co-operation (binding) between objects (this information is contained in a DCE binding handle, ANSA interface reference, or CORBA object reference):

- *transparency mechanisms*: declarative Quality of Service (QoS) requirements translate into the information, procedures and mechanisms necessary to provide the service with the required guarantees. Different systems will use different and incompatible mechanisms; only ANSA [Li 94] has any concrete proposals in this area.

- *communications protocols*: different protocols are used, with different guarantees and features (e.g. DEC RPC, ANSAware REX, both of which are implemented over TCP and UDP).
- *location*: different ways of addressing may exist, depending on the kind of network and communications protocols available.

To enable interoperation of clients and servers on opposite sides of an engineering/technology boundary, it is necessary to bridge the differences in some way so that bindings can be set up and messages can be meaningfully exchanged (one particularly important variety of messages is those that report that an exception has occurred, vital to enable troubleshooting). One promising approach is to use the information used by the trading process to support the automated generation of some sort of appropriate adapter, or "interceptor". This topic is discussed at more length in section 5.

3.7 Modelling: semantics and naming

Modelling concerns the differences in the way people describe the world around them. Mechanical systems such as computers can only deal with concrete representations, e.g., bit patterns. Previous work on naming models [Linden 93] and the information model [Iggulden 93] illustrate some of the limitations with respect to the representation of concepts and the subsequent interpretation of representations. These limitations apply also to service specifications, for instance:

- different services can have the same name in different contexts
- the same service can have different names in different contexts.

The major problem with regard to naming and co-operation in large scale systems is that different naming schemes make it difficult to compare names to discover whether the objects, or properties, denoted by those names are compatible.

Differences in semantics result in identical or similar concepts being represented in fundamentally different ways.

For example, two time services may provide different resolutions (1 second for one and 1 minute for another) and may vary in whether they include the date. In one sense they both mean the same thing by time, but in another sense they are different.

Moreover, production of a representation of something in one context (e.g., a specification of a service in project A) and the transfer of the representation to another context (e.g., project B) does not necessarily convey the meaning

attached to the specification. The two contexts may have different understandings of the representation used.

The major problem with regard to semantics and co-operation in large scale systems is that it may be difficult to discover whether the semantics of an operation offered by a server is compatible with the semantics required by a client. This may be because a sufficiently complete semantic description is not available to the client, or it may be because the semantic description provided by the server is not understandable by the client.

One way to make semantic information available is to make it part of the service specification maintained by traders. A difficult issue is the form in which such information should be maintained so as to make it understandable to people and/or computer systems. An even harder problem is determining whether or not the parties have conceptual models that are compatible enough for there to be any way to compare the nature of the offered service with the client's requirements.

It will always be necessary to integrate automated checking and manual negotiation. Automated checking of interface signatures will accept more cases than manual negotiation allows, because manual negotiation will take into account greater semantic knowledge. Automated checking will be used to check that asserted relationships are sensible. Some current telecommunications research is concerned with negotiating agents and their associated knowledge bases [Griffeth 94].

4 Dealing with Boundaries

4.1 Introduction

Boundaries define the limits of domains of homogeneity. These domains are defined by many different properties, corresponding to the different kinds of boundaries. This chapter considers the problems that the presence or absence of boundaries imply and outlines further work to be done in addressing these problems.

4.2 Multiple, overlapping domains

Two sets of objects may share some properties and not others. Thus, depending on the property under consideration, there may or may not exist a boundary between them. A division of a set of objects into domains under one

property may or may not represent the same division as is formed under another property.

Thus, the different kinds of domains corresponding to different kinds of properties may have different kinds of relationships to one another. Some domains may coincide with others; for example, a security domain may coincide with a remuneration domain. Some domains may be subsets of others; for example, a single authority may have delegated administrative control of its networks to several administrative bodies, so that each administrative domain is a subset of the authority domain. Some domains may overlap others in essentially arbitrary ways; for example, a security domain and a communications protocol domain may overlap in such a way that neither is contained in the other.

Because the properties which define domains do not necessarily relate to geographical location, domains of homogeneity do not necessarily occupy a geographical “area”. However, certain properties may be characteristic of a particular computer network, and then the corresponding domains are likely to cover an “area” defined by the network.

Similarly properties defining domains do not necessarily relate to authority. Boundaries may exist in places other than where the scope of different authorities meet.

4.3 Independent decomposition principle

It simplifies the analysis if one can consider the concerns associated with one kind of boundary in isolation from those associated with other kinds of boundary. We need to determine under what circumstances this is possible.

To show how tricky such separation of concerns can be, consider a protocol gateway which bridges a protocol boundary. This creates an authentication boundary if the gateway is in a different authentication domain from a client and server using the gateway.

4.4 Co-operation and independence

Two general issues at boundaries are:

- *co-operation*: there is a wish to co-operate with other sets of objects across a boundary, but there is a barrier at the boundary preventing it
- *independence*: there is a wish not to co-operate with other sets of objects across a boundary, but there is no barrier at the boundary preventing it.

The first is creating one domain of interoperability out of many and the second is creating many domains out of one.

These are the two polarised examples of likely problems. In practice the requirement for absolute co-operation, in which access is universally and transparently available, is likely to be tempered by some desire to ensure a quid pro quo arrangement (such as charging for a service) that is likely to require some level of independence to be asserted.

4.5 System management across boundaries

Another area of interest in federated systems is system management across domain boundaries.

One example is maintaining acceptable system performance, which requires managing the work load in relation to the available computing and communications resources, perhaps reconfiguring resources as necessary. In order to do this it is necessary to monitor and analyse work loads and performance measurements in different parts of the system, and to enable, disable or reconfigure the use of specific resources. In some situations this is largely a question of object interoperability, where client applications performing system management functions need to interoperate with server objects which have collected the relevant data and/or control the relevant resources. In other situations it may require interoperability among infrastructure components which are not implemented as objects, i.e., which do not communicate via object interfaces.

Another example is trouble shooting. When something fails, it is necessary to pinpoint the failure. Is it at a node or a link? Which component? Is it software or hardware? Here again both object interoperability and infrastructure interoperability may be involved. The situation is complicated by the fact that different domains may use different exception reporting mechanisms, so it may be difficult to propagate exception codes across boundaries.

4.6 Scope of work on boundaries

The boundary problems that need to be addressed are those occurring during the design, implementation and operational use of distributed computer-based services. The primary goal is to control (selectively enable and disable) interoperation between and within such services, and to facilitate system management.

Interoperation between objects (potentially from different domains) typically requires a number of different types of co-operation, including sharing responsibility for joint operation, sharing authority over it, sharing the management of the objects and infrastructure involved, and enabling the interactions that are required to carry it out (including both the provision of functionality and the transport of remuneration). Different types of co-operation may also be needed between designers and implementers that enable, encourage, discourage or prevent interoperation in later development phases.

Independence is often required when control over a domain needs to be established, perhaps to enforce security, safety, minimum quality, or charging controls or to establish common monitoring, auditing or administration procedures. In some of these cases the threat of malicious (intelligent) attempts to defeat mechanisms inserted to provide independence must be considered.

The ANSA work on federation is intended to:

- identify types of co-operation and address each individually
- categorise suitable mechanisms for independence, consider their automated production, and establish mechanisms which are not overly susceptible to malicious attack
- identify the most important aspects of system management and the problems posed by the different kinds of federation boundaries, and propose solutions.

5 Interception

Interception is concerned with the information, mechanisms and processes necessary to carry out the following activities:

- detecting attempts by entities to interact across boundaries
- determining the differences between the entities and whether any intervention is necessary
- inserting the necessary mechanism(s) for enabling, disabling or monitoring the crossing of the integration boundaries
- acting on attempts to interact across the boundary.

The mechanisms to be used will in general depend on the kind of boundary being crossed, as well as the purpose of the interception.

There may be a need to modify objects or insert adaptors of some sort between them in order to make them compatible with each other, or to prevent certain undesirable interactions between them from taking place, or to monitor interactions.

A model of interception is being derived by refining the activities listed above, and by relating the model to the different kinds of boundaries which will have to be bridged in distributed systems.

The general model of interception is being tested to derive the interceptors, encapsulators, translators, adaptors, or whatever, each suitable for a particular purpose and boundary. The feasibility of combining interception strategies which deal with different boundaries is also being examined.

To prove the concepts and the generality of the model it will be necessary to develop prototypes of particular interception strategies, and test them in the context of different scenarios.

There is some experience in the area of interception which we will build upon. For example:

- the object management community (the X/Open Joint Inter-Domain Management Group) has experience in translating system management specifications into CORBA IDL (and vice versa) [X/Open 94a, X/Open 94b].
- our previous experience in other ESPRIT projects showed one way to generate application level gateways between applications in different distributed systems environments (ANSAware and DCE)
- stub compilers have been used to generate stubs from IDL in DCE [OSF 92], CORBA-based products [OMG 92] and ANSAware [ANSA 93].

The model and techniques of interception are applicable at different stages of the development process. However, distributed systems are characterised by the dynamics of the applications and the changes which will inevitably take place in the configuration of the system. Particular emphasis must therefore be put on interception performed at run-time, resulting in interceptors being created, inserted and dismantled dynamically.

6 The Trading Model

6.1 Introduction

Trading can be viewed as encompassing all exchanges of information necessary to support interoperation. Thus, trading directly supports the first

three objectives listed in section 2 and indirectly supports the fourth, by facilitating the interoperation of object management applications.

To see the implications of these objectives for trading and traders, it is necessary first to look at both the process model and the information model for interoperation in a federated system.

6.2 Processes required for interoperation

The following processes are required for successful interoperation in a federated system:

- search in appropriate repositories (e.g. name services, traders, X.500 directories) for information about existing or creatable objects
- select objects for interoperation by comparing the offers made by service providers with the requests made by prospective users
- resolve any barriers to interoperation. This may involve the use, or dynamic creation, of adaptors of some sort. It may also involve the dynamic creation of servers.
- carry out any negotiations and agreements required for co-operation
- bind the co-operating objects, by configuring the engineering infrastructure
- set up a connection between the objects
- carry out the interaction itself, i.e. the invocation of operations which implement the services.

These processes need not be co-located in time or space. Part of the matching process may take place at design time, and part at run time. Binding may be dynamic at instantiation time or static at compile time. Some of the steps may be carried out by people outside the computing environment. Thus, trading needs to be viewed as a process which extends throughout the application life cycle, rather than something which happens only at run time.

6.3 Information required for interoperation

6.3.1 Symmetry of the information model

The basic information model for trading is symmetric, in that the client has to have information about the server in order to interact with it successfully, and similarly the server has to have information about the client in order to interact with it successfully (see Figure 2). Clients and servers have to trust each other - the trader is just an intermediary.

Note: Figure 2 goes here.

Note that there may be an asymmetry in the implementation and in the use of agents, e.g. the agent which conveys a certain kind of information from the server to the client is not necessarily the same as the agent which conveys the same kind of information from the client to the server. For example, a client may get information from a trader about allowable billing arrangements for a service, and then the client may be required to send billing information about itself directly to the server (e.g. fill out a direct debit authorisation), rather than going through a trader.

Moreover, information is not necessarily conveyed in the two directions at the same time. In fact, sometimes it is not even conveyed during the same phase of application development, deployment and/or operation. For example, a client may get information about the signature of a service from a trader during the design phase of the client, so that the client can be designed to match that signature, and then the client may provide information about the interface it needs at bind time. (Some information about the interface the service offers would have to be available again at this time, of course, to enable binding.)

6.3.2 Scope of the information model

As noted, the successful co-operation of service providers and consumers in a federated system involves finding suitable objects to interact with, and then resolving any barriers which arise due to incompatibility. The compatibility checks necessary to ensure successful co-operation are concerned with a wide range of functional and non-functional aspects in which systems may differ.

For example, authorisation policies and procedures, accounting and billing, recourse if quality of service guarantees are not met, etc. need to be included in the negotiation if required. Heterogeneous interaction may require a great deal of information about the specific mechanisms being used for transparency and quality of service, as well as about communications protocols and addresses. Federated development requires detailed information about the semantics of services, as well as about the signatures of operations.[1] Federated object management may require extensive information about resource allocation and recovery policies being followed, etc.

1. An increasingly important problem in large, federated systems, such as the public telephone network, is adverse feature interactions. A necessary step toward detecting and avoiding feature interaction problems is to have available a sufficiently complete specification of the services involved.

Thus, the information model behind object specification has to be generalised. Object specification must be considerably more encompassing and incorporate information on a wide range of properties where differences between systems are expected to arise.

6.4 Trading as information service

The process of trading should be viewed not just as providing a match making facility but more as an information service which will be used by different agents at different stages of the development process. A developer interested in minimising development effort may want to browse through available services to identify candidates for re-use. A prospective customer may want to look at accounting, billing and quality of service information on a number of similar services to decide which one to use.

In these situations the trader needs to be able to support what might be called a “shopping” model, as well as the usual match making model. In the match making model a client gives the trader a complete specification of the service desired, including the interface description, and the trader returns to the client interface references of one or more matching services, together with perhaps some additional predefined information on properties of the services to assist the client in selecting one. This model is primarily intended to support dynamic binding. In the shopping model a client gives the trader an incomplete specification of the service desired. The specification may or may not include the full interface description. The client also tells the trader what kinds of information to return about the qualifying services. The client then takes this information and makes a decision on what service to use, based on some algorithm known only to the client. The client may not be interested initially in obtaining an interface reference. That may be requested later after a service selection is made. In fact, the client may first have to adapt to the interface of the selected service (perhaps, for example, using the CORBA Dynamic Invocation Interface), so there may be some time lag between the initial query to the trader and the actual invocation of the service.

The shopping model is analogous to a person shopping for something, say a fridge. The person looks at what is available from various stores, comparing features, prices, credit and payment terms, warranties, exterior appearance, energy usage, etc., finally making a decision based on some combination of these factors which would be impossible to explain in advance. The purchase is then made (analogous to binding). Certain adaptations may be made, such as

determining precisely where to put the unit, based on the properties of the unit finally purchased, rather than specifying these properties in advance.

In the shopping model we have four phases rather than the current three:

1. *shopping*: trading for specifications (or for types)
2. *ordering*: trading for an object of the chosen type
3. *delivery*: binding
4. *using*: invocation.

The shopping model is often required for negotiating a service (e.g., “electronic video rental” service, or on-line database service) from multiple commercial offerings, for facilitating re-use, and for federated development.

This will require a modification of the model of trading to include a variety of queries and on-going interactions aimed at supporting a dialogue with the trader. This points towards an implementation of the trading function as an information repository supporting query language access by clients, as well as the usual lookup functions.

Moreover, the object specification information should be available throughout the life cycle of an object, to assist in systems management or reconfiguration, for example.

6.5 Dynamic Creation of Servers and Interceptors

Dynamic service instantiation (i.e. delaying the creation of a server until a client wishes to make use of the service) is described in [Deschrevel 93]. A related capability in a federated system is the ability to instantiate interceptors dynamically. Traders play a role in both of these processes. They can provide the information necessary to determine the specifics of the server or interceptor to be instantiated. They can also invoke an appropriate operation to initiate the instantiation when a client requests a binding to a service which requires it.

There are several issues to be addressed in this connection:

- The provision of appropriate mechanisms to enable such dynamic instantiation
  For example, the user can be given explicit control by such means as ‘trading for factories’, or the service creation could take place transparently to the client, as in ANSAware. There could be an agent function in the client to hide the factory.

- The provision of the necessary service specification information to support dynamic instantiation
- The extent of the trader's involvement in the whole process of dynamic instantiation.

6.6 Requirements on Trader

The above analysis leads to the following requirements on traders. A trader must:

- maintain many more categories of information about services
- allow much more flexible retrieval of information
- be able to participate in the automated creation or instantiation of servers and interceptors (by providing information and perhaps invoking the operation).

The above suggests a single trader, but the information and the functionality may be partitioned among multiple traders, and implemented by means of different technologies. [ISO 93] suggests that traders will be governed by various policies concerning security, searching, etc., and such policies could be supported by a trader 'kit of parts' which can be used to build multiple traders with an ability to call out to agents which will enforce those policies.

These requirements are analysed at more length in section 7.

7 Data Management

As pointed out in section 6, a trading capability which will be needed increasingly in the future is very flexible access to information about services.

The working draft of the ISO ODP Trading Function specification [ISO 93] provides a "Search" operation which goes some distance toward meeting this need. For this operation, the client must provide as input parameters:

- service interface description (an interface type)
- matching criteria (an arbitrary boolean condition on the properties of the service)
- list of property names designating the properties for which the client wants values returned.

The trader returns a list of all the service offers which are compatible with the given interface type and which satisfy the matching criteria. For each offer in the list, it includes:

- the interface reference
- values of the requested properties

There are several serious limitations to this “Search” operation as currently specified:

- The client may not always know the interface type of the service it is looking for. It may be looking for a service which performs a particular task and be prepared to adapt to the required interface type (perhaps using the CORBA Dynamic Invocation Interface).
- The client may not need the interface reference when the Search operation is invoked, as it may not intend to use the service right away. If dynamic service instantiation is involved, it is undesirable to provide the interface reference long before the service is needed, since the service must be instantiated before the reference can be provided.
- A “Search” operation may return a very large list of offers. There should be some specification for retrieving the list piece-wise, rather than all at once.
- There seems to be an implicit assumption that properties have simple data types. However, to support the needs of federated systems, some properties may have very complex data types. The client may want to use only certain components of certain complex properties in the matching criteria, and may want to retrieve only certain components of certain complex properties, not the entire properties.

For example, if the service is an information retrieval service, one of the “properties” of interest is the information model describing the structure and semantics of the information which can be retrieved. This is an enormously complex property. Clients may want to retrieve from a trader descriptions of certain entity sets in the information model, but not the entire information model.

In principle, one would like to simply generalise the “Search” operation to remove these limitations. However, the part about the complex data types presents significant problems, both with specifying the operation property and with implementing it.

In practice, one can achieve equivalent functionality much more easily by storing the property information in a standard database, e.g., a relational database, and allowing clients to access it via a standard query language, e.g., SQL. Such remote query language access will allow clients to select and

combine interrelated information about services in precisely the combination needed.

Moreover, trading is just one of many distributed applications which need remote query language access. The power and flexibility of this paradigm have led to its increasing popularity in client-server computing, as evidenced by the very large market today for client-server systems using SQL-based remote database access.

Therefore, an important problem to be solved is how best to support remote query language access to databases in an open distributed processing environment. Potential problems arise from the following characteristics:

- the numbers and types of input arguments and output results of a query language operation are determined by the query statement itself, often generated dynamically at run-time, in contrast to other types of operations with statically defined signatures.
- in order to provide the ability to return results of a query piece-wise, and in order to provide transactional properties across multiple operations, it is necessary to have a dialogue interaction paradigm between client and server, in contrast to other types of operations where the server need not maintain any information on the state of the interaction between operations.

These characteristics are accommodated by the basic ANSA/ODP computational model [Rees 93, ISO 93], but they are often not well supported by existing distributed processing environments.

The ANSA federation task group has developed a conceptual model for how to view databases and query language access to databases in an ANSA/ODP framework [Thomas 94]. Current work on an experimental implementation of an enhanced trader prototype is providing a test bed for a practical implementation of this model.

The current ANSA work on Data Management, as described above, is complementary to that done within ICL for RIBA/DAIS, and described in [Crockford 92]. The ANSA work is concerned with fitting query language access into the ANSA computational model, i.e. the model of client/server interaction; the ICL work is concerned with providing the client with a conceptual view of the DBMS that is easier to interact with.

8 Conclusions and recommendations

This paper has analysed the requirements for establishing co-operation in large scale systems in terms of a set of boundary types that may need to be crossed (federation) or established (independence). A number of consequent areas of investigation were isolated relevant to trading, interception and federated software development in such a system.

Interoperation between domains will require mechanisms addressing at least their administrative and remunerative differences in addition to difference in their interface descriptions, semantics, naming models and in the infrastructure supporting them. Similarly, isolation of domains may require differences to be created in any of these areas.

A greater wealth of information about clients and servers is required in order to achieve the above and this results in a requirement for a more advanced model of trading, including greater parity between the role of a client and a server and the ability to describe a greater range of objects that come into existence dynamically. In addition, the use of traditional data management technology to support this information requires the extension of existing ANSA RPC technology (e.g. dynamic typing) to conform to the existing computational model.

Boundaries need to be bridged by a variety of mechanisms, not all of which operate solely at run time. Specific instances of interception mechanisms can be derived from a generic model which is under development. Federation during the development of systems, for instance, requires the bridging of differences between tools used in the process of specification, design and implementation. This differs from, for example, a protocol gateway.

Establishing co-operation between systems is a fundamental pre-requisite for the wide scale deployment of an electronic services market place and the availability of truly open distributed processing. This paper suggests that further advances in a broad range of areas are necessary before this goal can be achieved.

9 Acknowledgements

The authors of this paper would like to acknowledge the contribution of their colleagues in the ANSA team, particularly Andrew Herbert, Technical Director of APM Ltd. and Chief Architect of the ANSA project, who made

valuable comments. Our colleagues in the ANSA sponsor companies also made a valuable contribution with their comments.

10 References

ANSA project, “ANSAware Version 4.1 Manual Set”, Architecture Projects Management, Cambridge, 1993.

BRENNER, J.B., “OPEN*framework*: Distributed Application Services”, ISBN 0-13-630518-0, Prentice Hall, 1993.

CROCKFORD, L.E., DRAHOTA, A., “RIBA - A Support Environment for Distributed Processing”, ICL Technical Journal (Volume 8, Issue 2, pp 284-301), November 1992.

DESCHREVEL, J-P., “The ANSA Model for Trading and Federation”, Architecture Report APM.1005.01, Architecture Projects Management, Cambridge, 1993.

GRIFFETH, N.D., VELTHUIJSEN, H., “The negotiating agents approach to runtime feature interaction resolution”, in “Feature Interactions in Telecommunications Systems”, Bouwma and Velthuijsen (Eds), ISBN 90 5199 1657, IOS Press, Amsterdam, 1994.

IGGULDEN, D., REES, R.T.O., VAN DER LINDEN, R.J., “Architecture and Frameworks”, Technical Report APM.1017.03, Architecture Projects Management, Cambridge, 1993.

ISO, “Open Distributed Processing - ODP Trading Function”, ISO/IEC JTC1/SC21/WG7, Draft, 1993.

ISO, “Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model”, ISO/IEC 10746-3, ISO/IEC JTC1/SC21/WG7, 1994.

LI, G., “An Open Architecture for Real-Time Processing”, ICL Technical Journal, November 1994.

OMG, “The Common Object Request Broker: Architecture and Specification”, Document Number 91.12.1, Object Management Group and X/Open, 1992.

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

REES, R.T.O., “The ANSA Computational Model”, Architecture Report APM.1001.01, Architecture Projects Management, Cambridge, 1993.

THOMAS, C.G., BEASLEY, M.D.R., HOFFNER, Y., “Data Management for an Enhanced Trader”, External Paper APM.1162.01, Architecture Projects Management, Cambridge, 1994.

VAN DER LINDEN, R.J., “The ANSA Naming Model”, Architecture Report APM.1003.01, Architecture Projects Management, Cambridge, 1993.

X/OPEN, “Translation of GDMO/ASN.1 Specification into CORBA IDL”, X/Open, Apex Plaza, Forbury Road, Reading, Berks, RG1 1AX.

X/OPEN, “Translation of SNMPv2 MIB Specification into CORBA IDL”, X/Open, Apex Plaza, Forbury Road, Reading, Berks, RG1 1AX.

11 Biographies

Mike Beasley

Mike Beasley received a first-class honours degree in Mathematics at the University of Cambridge in 1977, and a Diploma in Computer Science a year later. He then joined ICL, working as a designer/implementor on a variety of software development projects on VME and UNIX in Kidsgrove, Manchester and Stevenage before being seconded to the ANSA project in Cambridge in 1993.

He is also a Member of the British Computer Society and a Chartered Engineer.

Jane Cameron

E. Jane Cameron is seconded by Bellcore to ANSA. Prior to joining ANSA she was Director of the Network Systems Specification Research Group at Bellcore. This group focused on problems of interoperability among Telephone service, in particular problems of adverse feature interactions. She has also worked in areas of language processing, formal specification, and graphical user interfaces.

She holds a Ph.D. in Mathematics from the University of Washington in Seattle, Washington, USA and is a member of the ACM and IEEE.

Gray Girling

Gray Girling received an honours degree in Computer Science at Imperial College of Science and Technology in 1978, and a PhD from Cambridge University in 1983 with a thesis on authentication in computer networks. Following a period at Acorn Computers he joined Topexpress Ltd. where he worked for six years in the research and development of a secure computer

network for the UK Government. He joined the ANSA team in 1992 after a year at Perihelion Software Ltd. He has been involved over much of this period in the production of National, European and International security standardisation.

He is also a Member of the British Computer Society and a Chartered Engineer.

Yigal Hoffner

Yigal Hoffner received his BSc (Hons) in Computer Science and Cybernetics from the University of Reading in 1980. He stayed to work as a research assistant in the Microprocessor Unit at the Computer Centre from 1980 to 1983. He received his PhD in 1986 while working as a research fellow in the Computational Sciences Research Group at the department of Computer Science, University of Reading. The PhD topic was “The design of a reconfigurable multiprocessor system and its use in the solution of a class of numerical problems”. He worked in the R&D department of ISTEL Ltd. before joining the ANSA project in 1986 as a full time employee; work here has included user interface, trading, management, monitoring and visualisation of distributed systems.

Rob van der Linden

Rob van der Linden received his degree in Electronics and Telecommunications in The Netherlands in 1976. He was awarded a Master's degree in electronics and computing at the University of Southampton in 1981. He has been active in communications and computing first at the University of Southampton, then at BSO in The Netherlands. There he developed various early networked and distributed computing applications including an office publishing system and a distributed database. Since 1986 he has been with the ANSA project in Cambridge, where he has worked on object modelling, storage, naming and trading. He now manages the ANSA team.

Gomer Thomas

Gomer Thomas received a BA from Pomona College in 1962, a BA (Hons.) from the University of Cambridge in 1964, and a PhD from the University of Illinois in 1968, all in mathematics. He has spent 11 years on university faculties in mathematics or computer science, and over 14 years in industrial positions. For the past 7 years he has worked at Bellcore (Bell Communications Research), where he has been focusing on distributed data management. He is

currently a secondee to the ANSA project from Bellcore. He is a member of the ACM and the IEEE Computer Society.

Diagrams

Figure 1: Key Aspects of Interoperation

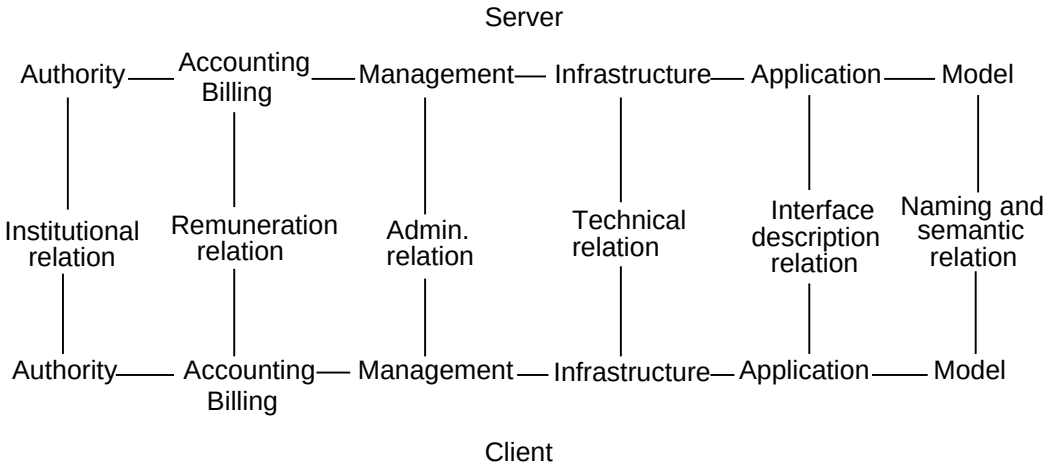
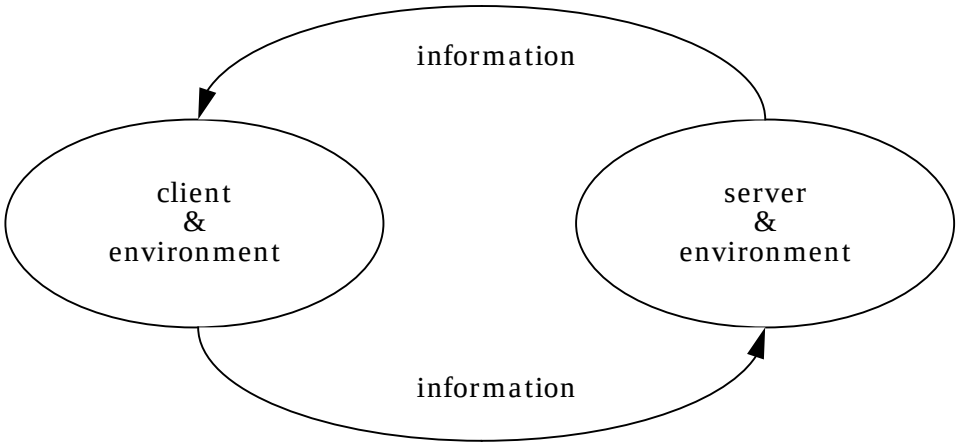


Figure 2: Symmetric Information Model for Trading



Text for editorial introduction

ICL has been involved in the ANSA project from its beginning in 1985.

The project has had three phases. The first was concerned with scoping the architecture and was supported by the Alvey programme, involving 12 UK and US companies. The second (ISA) was concerned with development and demonstration of the architecture; it was supported by the ESPRIT programme, and involved over 20 European and US companies. The third phase is fully industrially funded by 10 major sponsors and extends the architecture in the direction of Distributed Service Management.

In the second phase, ANSAware was produced as a prototype to demonstrate the soundness of the architecture. ANSAware has been used to build several large-scale distributed systems including NASA's Astrophysics Data System; ICL has developed it (in collaboration with BNR and others) in the RIBA platform, and now the commercially available DAIS platform.

The research in the third phase is taking place in three areas: dependability, federation (and tools) and performance/real-time. A paper has been submitted by each of the three work groups.

Contact Information

ANSA papers can be obtained within ICL by contacting John Brenner at BRA05 (J.B.Brenner@bra0511 or jbb@oasis.icl.co.uk, tel: 7285 1063 or 0344 711063).

The ANSA project has a World Wide Web server (URL is http://www.ansa.co.uk) and an FTP server, though access to the latter is restricted to sponsoring companies (e.g. ICL).

The authors can be contacted at:

Architecture Projects Management Ltd, Poseidon House, Castle Park, Cambridge CB3 0RD

Tel: 0223 323010

Fax: 0223 359779

Email: mdrb@ansa.co.uk, nje@ansa.co.uk, fw@ansa.co.uk