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

Federation manifesto

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

Abstract

Organizations have to respond quickly and flexibly to change, and 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 report outlines the planned work for 1994 by the ANSA Federation group. The plan of work, to tackle these concerns, includes work on boundaries and domains, mechanisms to deal with boundaries, trading and data management. The development of a set of scenarios and prototypes with which to measure the progress of the work is also proposed.

APM.1100.00.12 **Draft** 22 April 1994

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

Distribution:
Supersedes:
Superseded by:

Federation manifesto

Federation manifesto

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

APM.1100.00.12

22 April 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

1 Introduction

1.1 Purpose

The purpose of this report is to outline the planned work by the ANSA project team on the Federation task during the remainder of 1994. It discusses the nature of the problems addressed and the approaches taken to solving them.

1.2 Audience

This report is directed primarily toward the ANSA Technical Committee and the reviewers for the Phase III Federation Task.

1.3 Related documents

- ANSA Phase III Operational Plan: 1/3/93 - 28/2/95 [APM1031.1 93]
- An Overview of ANSA [APM1000.1 93]
- The ANSA Model for Trading and Federation [APM1005.1 93]
- ANSAware Version 4.1 Manual Set [ARM 93]

2 The Need for Federation

2.1 The trend toward large scale, federated systems

The need for faster response to ever changing business situations is leading organizations to move from monolithic uniform organizational structures to networks of cooperating sub-organizations. Organizations typically require computing systems which reflect their organizational structure. This creates an increasing demand for flexible distributed computing.

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. This provides an increasing technology base for distributed computing.

The combined effect is that distributed computing systems are becoming:

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

In short, the trend is moving beyond centrally organized, homogeneous distributed systems, toward large scale, federated distributed systems.

As a result, 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 Characteristics of large scale, federated systems

A large scale, federated system typically spans organizational 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 everything that can be different will be different.

The system is in a constant state of flux to meet the constantly changing needs of its constantly changing user base.

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

System components are designed and developed by different organizations. 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.

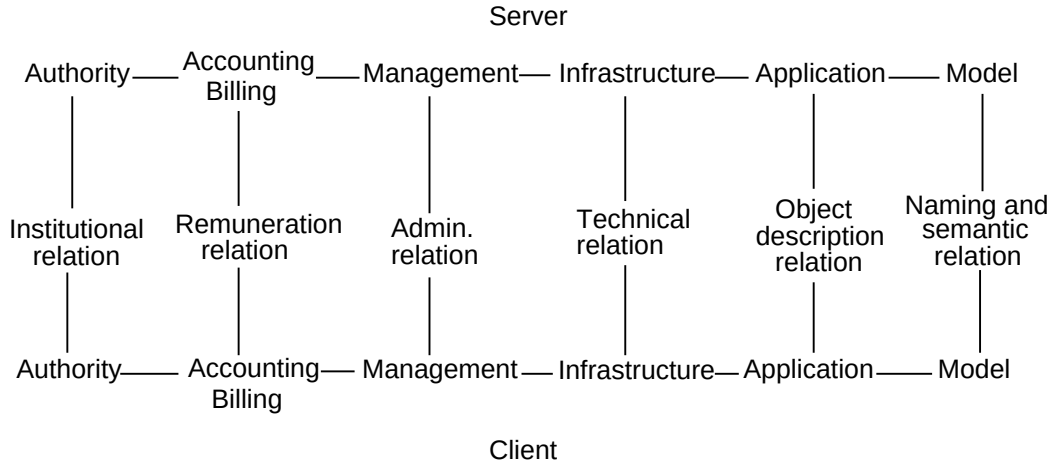
Different mechanisms are in place in different parts of the system to enable interoperation, to meet quality of service guarantees, and to manage the creation, deletion and reconfiguration of objects and resources. As with any complex technology, it is important to mask these mechanisms selectively from end users and/or application developers, and present them with a high level, application oriented interface. The ability to mask the fact that different mechanisms are being used in different parts of the system (hiding irrelevant details from the user) is especially important.

Scenarios illustrating aspects of federation are in §9.

2.3 Interoperation in large scale, federated systems

Figure 2.1 summarizes key aspects of the interoperation between clients and servers in large scale, federated systems.

Figure 2.1: Key Aspects of Interoperation



One can position each of these aspects in terms of the five viewpoints for viewing distributed systems: the enterprise, information, computational, engineering, and technical viewpoints [APM1000.1 93], [ODP 93].

The “authority” aspect concerns creation, ownership and control of resources of all kinds. The “accounting/billing” aspect concerns measuring resource usage, billing for usage, and handling payment. The “management” aspect concerns policies for resource allocation, monitoring, quality control, etc. Issues related to authority, accounting/billing and management are all part of an enterprise view of a system.

The “infrastructure” aspect concerns the mechanisms and protocols required for interoperation between clients and servers. Issues related to infrastructure are part of engineering and technology views of a system.

The “application” aspect concerns features specific to particular instances of applications or services, such as:

- interface types (signature)
- quality of service (security, performance, dependability, etc.).

Interface types are part of the computational view of the system; quality of service relates to enterprise requirements, and is realised through engineering mechanisms.

The “model” aspect concerns the semantics and naming of objects and services. This falls under an information view of the system.

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

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

where “compatibility” must take account of all the aspects above, not just the infrastructure and application aspects.

It is also necessary to be able to impose suitable restrictions on interaction at times, to ensure security, to provide performance guarantees, to manage resource utilization, etc.

2.4 Federation work plan

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

The foregoing discussion suggests four main objectives:

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

Support for these objectives requires further refinement of the model of interoperation presented above, and an analysis of the different kinds of boundaries that must be crossed during federated interoperation (see Chapters 3 and 4).

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

The requirement for access to the diverse kinds of information needed to support enterprise negotiation, heterogeneous interaction, federated development, and federated object management leads to the development of an extended model of trading and a “proof of concept” prototype trader implementation (see Chapters 6 and 7). The extended trading model must support highly flexible retrieval of a wide range of information from traders.

The design and implementation of the extended trader prototype leads to a need for a model for query language access to data in open distributed systems (see Chapter 8).

All of this work needs to be checked against a set of federation scenarios that describe a range of practical problems (see Chapter 9).

A summary of the federation work plan, together with a mapping of the planned work items to the deliverables of the ANSA Phase III Operational Plan [APM1031.1 93] appears in Chapter 10.

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 chapter 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 [APM1005.1 93]. Chapter 4 discusses in general terms the kinds of problems which arise at boundaries. Chapter 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:

- authority
- management and administration
- accounting, billing and paying (i.e., remuneration)
- infrastructure: engineering and technology
- service interfaces and properties
- modelling: semantics and naming.

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 domain managements are responsible for ensuring that any guarantees given by the clients and servers are kept.

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

•

- 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
- access control: deciding who should have access to what services when
- 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.)
- 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. From the trading point of view it is expedient to reach agreement on remuneration issues before consuming and providing services. For negotiation to succeed, it is necessary to have the relevant information concerning remuneration.

3.5 Service interfaces and properties

The 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:

- behaviour
- interface type (signature): description of interfaces in terms of operations and data types. Type systems involve notions of equivalence and substitutability; types are represented by interface definition languages (IDL) and abstract data types [APM1042.1 93].
  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 semantics of services, which is included under modelling, below.)

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.

The differences between distributed system infrastructures which affect interoperability can be broken down according to the categories of information necessary for setting up the cooperation (binding) between them:

- transparency mechanisms: declarative QoS requirements translate into the information, procedures and mechanisms necessary to provide the service with the required guarantees. Different systems may use different and incompatible mechanisms.
- communications protocols: different 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.

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 adapters, or “interceptors”. This topic is discussed at more length in Chapter 5.

3.7 Modelling: semantics and naming

Modelling concerns the differences in the way in which 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 [APM1003.1 93] and the information model [APM1017.3 93] illustrate some of the limitations with respect to the representation of concepts and the subsequent interpretation of representations, e.g., service specifications.

Differences in naming result in:

- different things having the same names in different contexts
- the same things having 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 the compatibility or incompatibility of objects or their properties denoted by those names.

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 have slightly different concepts in mind.

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 no sufficiently complete semantic description is 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 will accept more cases than manual negotiation allows, because manual negotiation will impose greater semantic knowledge. Automated checking will be used to check that asserted relationships are sensible.

4 Dealing with Boundaries

4.1 Introduction

Boundaries occur wherever a difference occurs. There are many possible kinds of differences between things, and their categorization is addressed in Chapter 3. 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

Boundaries define the limits of domains of homogeneity. These domains are defined by many different properties, corresponding to the different kinds of boundaries.

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 problems posed by one kind of boundary in isolation from those posed by other kinds of boundary. One

problem to be solved is to determine under what circumstances this is possible.

Another problem is that a mechanism which bridges one sort of boundary may create a new boundary of a different sort which then needs to be bridged. For example, a protocol gateway which bridges a protocol boundary may create 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 types of problems 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 the problem of creating one domain of interoperability out of many and the second is the problem of creating many domains out of one.

These are the two polarized 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.

These problems are defined not only by the property defining the boundary in question, but also by the type of co-operation desired. Not all forms of co-operation involve domains of the same type. E.g., in co-operation for the purpose of payment the payee may come from one type of domain and the payer from another.

4.5 System management across boundaries

Another type of problem which occurs in federated systems is system management across domain boundaries.

In general, the different kinds of boundaries in a federated system create different kinds of obstacles to solving system management problems. It is necessary to analyse the different kinds of obstacles created by the different kinds of boundaries and propose ways to overcome them.

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 or disable or reconfigure the use of 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 isolated 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 planned work on federation will attempt to:

- identify types of co-operation and address each individually
- categorize 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

In distributed systems there is a need to support:

- co-operation between clients and servers; such co-operation may require interaction across a boundary. The boundary marks a difference between two objects, which has to be bridged.
- prohibiting certain interactions; prevention may be necessary where crossing boundaries causes loss of integrity due to failure or breach of security, or may be undesirable for commercial reasons.
- observing events of interest; monitoring is often required where the recording of interactions is necessary for traceability, accountability, debugging, dependability, and garbage collection (for example). Monitoring takes place without necessarily interfering with the interactions themselves.

In all of these situations, 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. This general process is referred to as interception.

Thus, interception is about the information, mechanisms and processes necessary for:

- detecting attempts by entities to establish co-operation and actually co-operate
- determining the differences between the entities and whether any intervention is necessary
- inserting the necessary mechanism(s) for:
  - enabling the crossing of the integration boundaries,
  - disabling the crossing of boundaries, or
  - monitoring the crossing of 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.

A model of interception will be derived by refining the steps mentioned 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 will be used 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 will be 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 has experience in translating CMIS/CMIP (and other) specifications into CORBA IDL (and vice versa)
- our previous experience in the Harness project 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 [ARM 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 will therefore be put on interception performed at run-time resulting in interceptors being created, inserted and dismantled dynamically. The scope for automatic and dynamic instantiation of interception functions will therefore be of extreme importance and will be closely examined.

6 The Trading Model

6.1 Introduction

As noted in Chapter 2, the federation work has four main objectives: federated enterprise negotiation, heterogeneous interoperation, federated development and federated object management.

Trading can be viewed as encompassing all exchanges of information necessary to support interoperation. Thus, trading directly supports the first three objectives above 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) to locate information about existing or creatable objects
- select objects for interoperation (match make) 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.

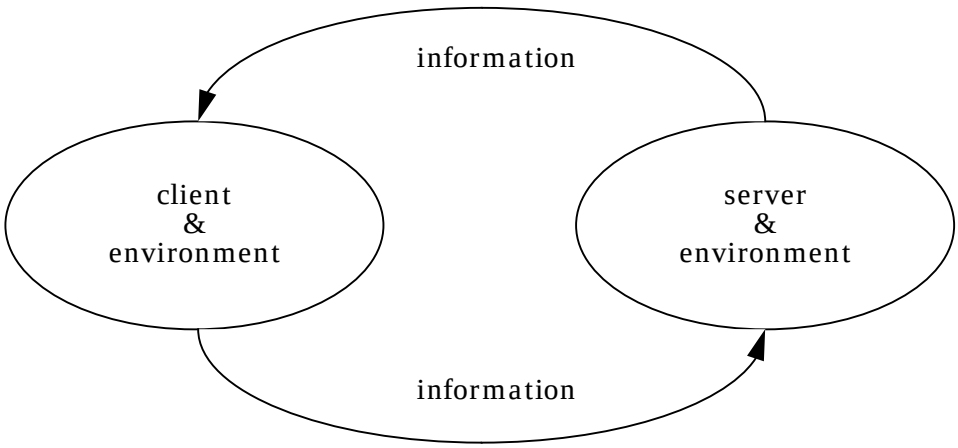
These processes need not be co-located in time or space. Part of the matching process may take place at design time, and part at run time. Binding may be dynamic at instantiation time or static at compile time. Some of the steps may be carried out by people outside the computing environment. Thus, trading 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 6.1).

Figure 6.1: Symmetric Information Model for Trading



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, 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, enterprise negotiation requires information about enterprise aspects of services -- authorization policies and procedures, accounting and billing, recourse if quality of service guarantees are not met, etc. 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.

Thus, the information model behind object specification has to be generalized. 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. Thus, for example, a designer may want to look up the specification of some service with a view to designing a client capable of using the specified service. A developer interested in minimizing 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 is needed 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, 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.

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.

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

1. *shopping*: trading for specifications (or for types) e.g. by browsing an OMG interface repository
2. *ordering*: trading for an object of the chosen type
3. *using*: binding. There will also be some means of payment, which can take various forms.

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.

6.5 Dynamic Creation of Servers and Interceptors

Dynamic service instantiation is described in [APM1005.1 93]. A related capability in a federated system is the ability to dynamically instantiate interceptors. 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:

- What mechanism(s) should be used to implement such dynamic instantiations?
  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.
- Precisely what service specification information is needed to support such dynamic instantiation?
- Precisely what role should a trader play, and how?

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. The requirement to support many different trading policies suggests a trader 'kit of parts' which can be used to build multiple traders with an ability to call out to policy controllers on a per-offer basis.

These requirements are analysed at more length in the next chapter.

7 Prototype Trader Design

7.1 Introduction

Chapter 6 presented a model for trading which incorporates extensions to the current ANSA model of trading. This chapter describes plans to develop a new prototype trader. The main objective for this prototype is to incorporate the extensions to the model of trading. A secondary objective is to improve certain known implementation deficiencies of the current ANSAware trader.

Major features needed to support the extended model of trading are:

- Ability for a trader to manage more categories of information on service offers, including:
  - — management information needed to support enterprise aspects of interoperation, such as contractual relationships between client and server, and charging for the use of a service.
  - — service specification information needed to support federated development tasks, such as browsing of service specifications by planners and designers, and searching for re-usable code by developers.
  - — other information which may be needed to support automated dynamic instantiation of servers and interceptors
- Ability for a trader to return specified categories of information on all services which meet a partial service specification, to support the “shopping” paradigm and to support federated development.
- Ability to support dynamic instantiation of servers and interceptors.

Of course, a trading service must also continue to support dynamic binding between clients and servers.

The primary implementation deficiencies to be addressed are:

- The current ANSAware trader matches offers only on the basis of asserted type conformance; there is no mechanism for checking type conformance based on the Interface Definition Language (IDL) definitions of types. Such checking would be desirable because it would trap errors earlier.
- The current ANSAware trader supports only a limited form of federation, in which there is a single “master trader” and some number of “local traders.” A local trader can access information in other local traders only by going through the master trader. More versatile forms of trader federation would be desirable [APM1003.1 93].

7.2 Requirements for the new trader prototype

This section describes in more detail the requirements for the extended trading service.

7.2.1 Extended and Extensible Information Model

To accommodate the new categories of information about service offers, the data model for the information managed by the trader must be extended. The data model must also be further extensible dynamically, since it is impossible to predict in advance all the kinds of information which will be relevant to all the various kinds of services in a large-scale, federated system.

7.2.2 Query Language Access to Trading Information

To accommodate shopping for services and browsing through service specifications, it is necessary to give clients great flexibility in retrieving information from a trader. The proposed way to do this is to support query language access to the information in a trader.

Given currently available technology, the most reasonable query language to support is SQL. At some point in the future the language OQL proposed by the Object Database Management Group [ODMG 93] may emerge as an alternative, but it will not be supported in the current prototyping effort - see Chapter 8.

7.2.3 Federation of Traders

A client (including a trader) should have access (either direct or transparent indirect) to multiple traders.

7.2.4 Dynamic Service Instantiation

Dynamic service instantiation is described in [APM1005.1 93]. Two implementations of dynamic service creation based on ANSAware (or derivatives) are available -- one from BNR (ITOM) and one in ANSAware 4.1 [ARM 93]. In the BNR implementation intelligent proxy code in the client will cause a server to be started if the trader fails to find an active server. In the ANSAware implementation the relevant Node Manager posts a ‘proxy offer’ (or a ‘monitored offer’) with the trader. When the trader gets a client request for the service, it turns the request over to the Node Manager. The Node Manager instantiates a server, and then returns the interface reference for the server to the client.

Some emerging CORBA-compliant products (e.g. Orbix [Iona 93a] [Iona 93b] and ACAS [DEC 93a] [DEC 93b]) also support automatic dynamic instantiation of servers.

Although dynamic service creation is not considered part of the trader, it is intimately related to it. Some mechanism using proxy offers is the recommended way of providing the facility, though in a more architectural way than the existing facilities in ANSAware.

7.2.5 Dynamic Interceptor Instantiation

Just as it may be desirable at times to instantiate servers dynamically, it may also be desirable in a federated system to instantiate interceptors dynamically. The involvement of a trader for instantiating interceptors is similar to that for instantiating servers, but an interceptor factory may have to get a substantial amount of information from a trader to properly configure the interceptors.

7.2.6 Type Checking

It is desirable to perform conformance-based type checking (i.e. checking of operation signatures), e.g. to support the 'type' interface of ODP [ODP 93], which has two operations:

1. is this type conformant to that one ?
2. return the IDL (or a representation thereof) for a type. (This is mainly a private operation, but also useful to specialised clients).

A type library, consisting of pre-computed and asserted type relationships, would be a useful optimization, but assertions still ought to be checked.

Conformance-based type checking only exists at present in the DPL (Distributed Programming Language) system [APM1014.1 93], produced for research purposes at APM. We would like to implement type checking using today's technology - i.e. the CORBA Interface Repository [OMG 92] which gives us, in effect, access to the IDL at run time.

7.3 Re-engineering the existing trader

Our plan is to:

- make use of existing technology as much as possible
- gain experience in the use of C++ and CORBA-compliant products, in particular the Interface Repository
- gain experience in supporting remote SQL queries to relational databases in an ODP environment

Therefore we plan to build an enhanced trader prototype which

- re-uses as much as possible of the general design of the existing ANSAware trader
- is written in C++
- is based on a CORBA-compliant product (including use of the Interface Repository for type checking)
- uses a relational database for the service offer repository

The primary candidates for the CORBA-compliant platform are Orbix or DAIS (if a CORBA-compliant version of DAIS is available soon enough).

If we re-engineer the existing ANSAware trader in a modular fashion using C++, we can build different versions of the trader, for example a trader with identical functionality to the existing one, or one that uses Orbix and an SQL database; these two traders would have a number of components in common.

7.3.1 What are the modules ?

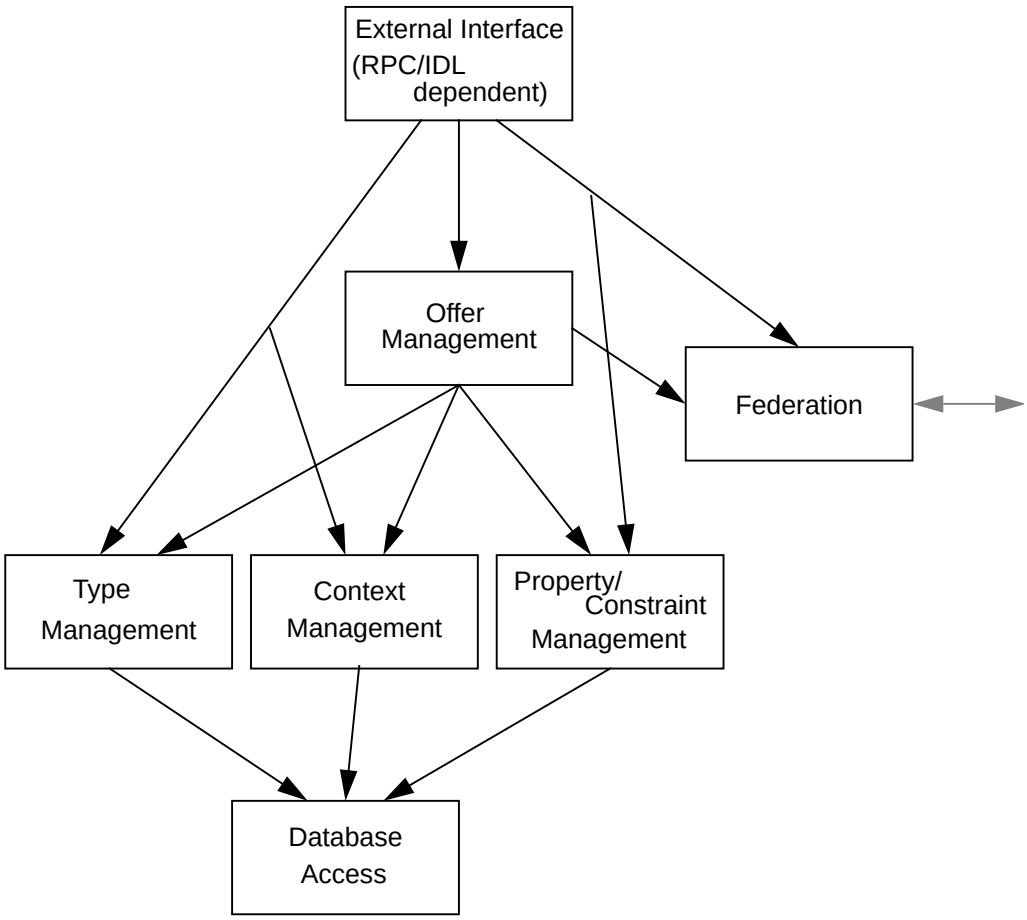
The modules of the trader would be as shown in Figure 7.1.

The External Interface module is where invocations of trader operations first arrive at the trader.

The Offer Management module implements the Register, Lookup and Delete operations of the trader.

The Type Management module is computationally separate from the other modules, but it will be co-located with the other modules from an engineering point of view. It supports the operations for adding type definitions, deleting

Figure 7.1: Modules of the Trader



type definitions, listing type definitions, and comparing types for compatibility.

The Context Management module supports operations for adding trader contexts, deleting trader contexts, listing trader contexts, and adding or deleting relationships between trader contexts. (The current ANSAware context graph is a tree, but a more general graph structure is envisioned for the enhanced trader.)

The Property/Constraint Management module supports definition of new properties and their allowable values, and comparison of property values.

The Federation module handles passing requests on to other traders.

The Database Access module handles access to the repositories of interface references and service offers.

Note that server factories (monitors), interceptor factories and resource management are being kept separate from an engineering point of view, as well as computationally.

The modules listed above will be implemented as C++ classes.

8 Data Management

As pointed out in chapter 5, a trading capability which will be needed increasingly in the future is very flexible access to information about services. This kind of flexibility can best be provided by supporting remote query language (e.g., SQL) access to the information contained in traders. Query language access will allow applications to select and combine interrelated information about services in precisely the combination needed, and then select from the combined information precisely the attributes needed.

Moreover, trading is just one of many distributed applications which need remote query language access. Its power and flexibility have led to its increasing popularity in client-server computing. For evidence of this popularity, one need only look at the very large market today for client-server systems using SQL-based remote database access.

The question, then, 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. Thus, it may not be possible to provide an IDL (Interface Definition Language) definition of the signature of a query language operation at compile time, since the query statement may be generated dynamically at run time.
- the quantity of data to be returned by a query language operation is not known in advance. It depends not only on the nature of the query, but also on the state of the database, and it may be very large. Thus, there needs to be some way for the requesting application to “gate” the returned results.
- it is often necessary for the server to maintain interaction context between operation invocations from the same client. For example, a client application may engage in a conversational dialogue consisting of a whole sequence of operations and require that they all be part of a single transaction, or a client application may invoke one operation to initiate a retrieval operation and then invoke a sequence of operations to retrieve the results of the retrieval operation.

To address these problems, the ANSA federation task group is in the process of developing a conceptual model for how to view databases and query language access to databases in the ANSA framework, together with a corresponding engineering model for how one could implement such access to current and future generations of database products. The proposed experimental implementation of an enhanced trader prototype will provide a test bed for this work.

The deliverables from this work will be a report on “The ANSA Model of Remote Database Access” and an experimental implementation, as part of the enhanced trader.

9 Micro-scenarios

A “micro-scenario” is a description of a single, limited aspect of an interaction in a federated system. The federation task group has been accumulating a collection of micro-scenarios as an aid to understanding the different types of domains and boundaries in a federated system and the nature of the problems which they pose.

The document containing the accumulated collection of micro-scenarios is primarily intended as a living, internal document of the project team, to be augmented on a continuous basis. However, it will also be made available to sponsors as a released report, updated periodically.

The following are some examples of micro-scenarios:

- A resource manager grants selected rights to individuals or groups within the administrative domain containing the resource. The manager is also willing to grant limited rights to users from other domains, but does not want the administrative overhead of dealing with individual users or groups of users. She just wants to grant certain limited privileges to ALL members of certain other domains.
- Service X has a response time guarantee within a particular LAN. Client Y invokes the service from another LAN, which is connected to the first LAN via a slow long haul network.
- A client is accessing a service through a gateway. The client, server and gateway are all in different administrative domains. Thus, an end user must have a userid for each domain and be authenticated separately in each domain. The client software may have been written under the assumption that the server is in the same domain, or at least that there is no gateway between client and server. Of course, it is not acceptable for the gateway system administrator (or anyone else) to be able to see the user’s userid or password for the server, nor should this information be stored on disk. The scenario can be further complicated by postulating two or more gateways in series.
- A server keeps track of resource utilization and issues bills to users at regular intervals (e.g., monthly), where each client node is automatically identified with a particular user. (This is sometimes used by computer centres with hard-wired terminals, also used by telephone companies for many types of services -- e.g., long distance calls from a home telephone.)
- A client in name context A invokes a service in name context B. The arguments and the results involve names. (In the computational model the parameters are interface references, but the recursion ends with named terminations. In the engineering model they probably come down to names well before that.)
- The transaction model of the MVS/IMS system is that every message received by the system initiates a new transaction. Thus, there can be no conversational interaction within a transaction. An application in a

system based on a conversational (dialogue) interaction model must interoperate with an application in an MVS/IMS system.

- Given the great diversity of communications protocols, with their associated diverse addressing formats, and given the amount of other information which must be part of an interface reference, in the engineering model the interface reference will typically be a pointer to some infrastructure data structure where all of the associated information can be found. In a federated system, different domains may have different data structure, and in fact the data structures may be totally incompatible in format. During an interaction interface references must be passed across domain boundaries.

10 Summary of Planned Work

The specific deliverables planned for the federation work are described below, together with their target dates and the Phase III work items to which they pertain.

- Boundaries and domains

Report on managing federation (F1) May 1994

This report will contain a taxonomy of the different types of boundaries in a federated system and an analysis of the problems which arise with each type and how to deal with them. It will consider problems of object management within and across domains, as well as problems of enabling and/or restricting interoperation across domain boundaries.

- Enhanced trading service
  - — Report on enhanced trading model (F1) March 1994
  - — Report on functional specification for trader (F3) March 1994
  - — Report on design of prototype trader (F3) March 1994
  - — Implementation of prototype trader (F3) June 1994

The first two reports, or perhaps all three reports, may end up combined into a single report.

The report on the enhanced trading service will include an information model, process model and responsibility model for trading, including the role of traders in dynamic server and interceptor instantiation and other aspects of object management.

The report on the prototype design will include a detailed description of the components and the interfaces between them, and for each component what product(s) will be used to implement it.

- Data management

Report on databases and remote queries in ANSA (F3) February 1994

This report will describe how to accommodate query language (e.g., SQL) access to databases in the context of the ANSA computational model. (This approach will be used and demonstrated in the enhanced trader prototype.)

- Interceptors
  - — Report on model of interception (F1) May 1994
  - — Report on design of prototype(s) (F4) October 1994
  - — Implementation(s) of prototype(s) (F4) February 1995

The report on the model of interception will give a taxonomy of interceptors, gateways, encapsulators, translators, converters, adaptors, or whatever, together with a description of the function(s) of each. It will include a model for automated dynamic instantiation of interceptors.

The intent is that the prototype will be able to generate automated dynamic instantiation of interceptors.

- Micro-scenarios

Micro-scenarios report (F1, F3) February 1994

This document was originally envisioned as a purely internal document, to be used as a tool in the analysis of how to deal with different types of boundaries. However, since it may be of interest to sponsors in its own right, it will be issued as a technical report. It should be understood that it is a working document. A new version will be released from time to time as we add to it.

Information analysis report (F1) June 1994

The micro-scenarios are a rich ground from which to extend the information model for trading. This report will update and extend the information model developed in the March report on the enhanced trading model. Information content and representational issues will be addressed.

References

[APM1000.1 93]

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

[APM1003.1 93]

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

[APM1005.1 93]

Deschrevel, J-P., **The ANSA Model for Trading and Federation**, APM Ltd., Cambridge, UK, 1993.

[APM1014.1 93]

Howarth, N.J., Watson, A.J., Rees, R.T.O., Otway, D.J., **DPL Programmers' Manual**, APM Ltd., Cambridge, UK, 1993.

[APM1017.3 93]

Iggulden, D., Rees, R.T.O., Van der Linden, R.J., **Architecture and Frameworks**, APM Ltd., Cambridge, UK, 1993.

[APM1031.1 93]

ANSA Phase III Operational Plan: 1/3/93 - 28/2/95, APM Ltd., Cambridge, UK, 1993.

[APM1042.1 93]

Hoffner, Y., Beasley, M.D.R., **Type Conformance and IDLs**, APM Ltd., Cambridge, UK, 1993.

[ARM 93]

ANSAware Version 4.1 Manual Set, APM Ltd., Cambridge, UK, 1993.

[DEC 93a]

DEC ACA Services: System Integrator and Programmer Guide, Digital Equipment Corporation, Maynard, Massachusetts, USA, 1992.

[DEC 93b]

DEC ACA Services: Reference Manual, Digital Equipment Corporation, Maynard, Massachusetts, USA, 1992.

[Iona 93a]

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

[Iona 93b]

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

[ODMG 93]

Cattell, R.G.G. (editor), **The Object Database Standard: ODMG-93**, Morgan Kaufmann, 1994.

[ODP 93]

Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model, ISO/IEC CD 10746-3 (Committee Draft), ISO/IEC JTC1/ SC21/WG7, June 1993.

[OMG 92]

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

[OSF 92]

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