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

Boundaries and domains

Yigal Hoffner & Gray Girling

Abstract

Surmounting the technical and administrative boundaries that separate domains are key requirements when expanding or rationalising businesses. The ability to create such boundaries may also be vital to the preservation of a businesses self interest.

The elimination of unwanted boundaries and creation of required ones can be achieved once the nature of domains and boundaries is understood, the properties that may separate them are categorized and mechanisms to integrate and separate domains are identified.

After developing the notions of boundaries, domains and services this document describes a classification of boundaries applicable to large scale distributed systems. The problems in creating or overcoming each of these types of boundary are then considered and means to deal with those problems are proposed. Finally some of the elements of an infrastructure that would support those solutions are given.

APM.1139.00.03 **Draft** 17 August 1995

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

Distribution:
Supersedes:
Superseded by:

Boundaries and domains

Boundaries and domains

Yigal Hoffner & Gray Girling

APM.1139.00.03

17 August 1995

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	(01223) 515010
INTERNATIONAL	+44 1223 515010
FAX	+44 1223 359779
E-MAIL	apm@ansa.co.uk

Contents

1 Introduction

Note: The text in this document was assembled from Yigal's notes, Yigal's latest version of "Boundaries and domains" ("RFA.001"), Gray's latest version of "Dealing with Boundaries" and the chapter "Boundaries in federated systems" in "The Federation Manifesto" APM.1100.00.06.

Note: There was enough text to address some issues which may subsequently be considered to be beyond the scope of this document. Hopefully these have been isolated and partitioned in the latter chapters of the document.

Note: Editorial unification of this text has involved some changes: removal of semantically duplicated text; replacing system "aspects" by system "properties"; amalgamating the "peer-to-peer" (stream based) case in the term "object"; more uniform use of the phrase "large scale distributed system"; settling on a standard set of names for the different system aspects; removal of use of the first person;

Note: Further text has been added to: define a "service", some alignment with APM.1164 (TC slides on this topic) has also taken place. Quite a lot of work has gone in to the separation of issues associated with a service and those associated with the way in which it is accessed. Some alignment with ODP has been added.

Note: This is an incomplete document. There are various editorial notes in this draft document. Many of the references are not provided.

Surmounting the technical and administrative boundaries that separate domains are key requirements when expanding or rationalising businesses. The ability to create such boundaries may also be vital to the preservation of a businesses self interest.

The elimination of unwanted boundaries and creation of required ones can be achieved once the nature of domains and boundaries is understood.

This document develops an overview of the types of infrastructure that are required to overcome the principal boundary-related problems in large scale distributed systems. It does this by considering:

- what a boundary is;
- what problems boundaries pose;
- what types of boundaries should be analysed; and,
- an initial analysis of the problems that each kind of boundary presents.

1.1 Boundary Problems

The boundaries of interest are those that particularly focus on distributed service provision and consumption[1] during the development, implementation, deployment, provision, use and maintenance of large scale distributed systems. The primary goal is to control the co-operation that leads to service

1. Note that this implicitly includes the computer-based subset of business operation and that the structure of a business organization will normally be reflected in the structure of such services.

consumption and provision between and within such systems. The most important boundary problems considered in this document are:

1. *federation*, there is a wish to co-operate between sets of objects but a boundary prevents this;
2. *separation*, there is a wish not to co-operate with other sets of objects but no boundary prevents this; and,
3. *monitoring*, there is a need to discover what co-operation is taking place with other sets of objects.

The first is the problem of creating one domain out of many and the second is the problem of creating many domains out of one but these are the two extremes of a spectrum of likely problems. In practice the requirement for federation is likely to have some element of separation. For example a service designer may require a large element of federation to increase its accessibility but also insist that those who have not paid for it are unable to use it (a degree of separation).

In terms of common business processes these problems occur as follows.

1. Federation

   As businesses expand and coalesce federation is commonly required to overcome the technological and administrative boundaries that may have existed between its pre-united parts. Federation may enable a service user to make use of a service in a different environment, or enable two services separated by a boundary to appear as a single service.

Note: Should the latter scenario be called something other than federation? (e.g. integration) should it be included in this document at all?

   The principles required to safeguard the autonomy of federated components are provided in [APM1005.1 93].

2. Separation

   As businesses contract or are restructured separation is often required to exercise separate control over parts of a domain that previously interoperated more freely. This control may be required to enforce security, safety, quality, or charging policies or to establish common monitoring, auditing or administration procedures. In some of these cases (security in particular) the threat of malicious (intelligent) attempts to defeat separation mechanisms must be considered during their design, whereas this additional level of complexity need not be associated with federation mechanisms.

3. Monitoring

   Monitoring the interaction between domains can enable the current health, status and efficiency of a business to be derived. It may also enable the assessment of any mechanisms put in place to achieve federation or separation, or reveal a requirement for as yet unaddressed areas of federation, separation or monitoring.

1.2 Organization of the document

A number of relevant general concepts (such as those of service, boundary, domain) are established initially so that they can be used with a minimum of ambiguity.

The way the document addresses the above boundary problems is:

- provide a classification of properties that potentially are different;
- then, for each property:
  - define the types of co-operation that are required between systems with respect to that property and,
  - for each type of co-operation:

    say what the difference problems are.

Further work in subsequent documents may address specific types of co-operation, saying what mechanisms can be used to solve the relevant difference problems, describing them in terms of which elements of the mechanism can be supported by infrastructure/engineering components and what parts of their information requirements should be provided by a central repository (e.g. a trader).

2 Basic Concepts

2.1 Introduction

This chapter provides qualification of the terms “service”, “domain”, “boundary” and “object”.

2.2 Services

“Boundary Problems” in the chapter above are described in terms of “co-operation” . This co-operation may take place at any development phase, including during system operation. In the latter case this document speaks in terms of service provision and use. When federating “service” characterizes something provided in an unfamiliar domain, so its definition should preferably not be expressed in terms of the way it is provided in any particular kind of domain.

- A *service* is a set of capabilities available to a population of clients, but is conceptually distinct from whatever mechanism or configuration of parts that is used to provide it.

In practice a service must be related to some *mechanism* in order to be realized and it will be related to that mechanism by one (or more) *points of access* at each of which some *access method* allows all (or some) of its capabilities to be exercised.

Typically federation occurs when a service selected by a client's designer is required of a server providing environment over which the client's designer has had no influence. Because it is likely not to have been designed with the client's requirements in mind the alien environment may not provide individual server interfaces that provide the functions that the client requires. Nonetheless this environment may support those functions differently packaged. For example, the client's notion of a “file service” may include functions to return the current time, whereas the alien environment may provide file operations and a time service on separate servers. even though no individual server interface does.

When using ANSA, designs can be anticipated in which services of relevance are provided entirely by single interfaces to servers (although not universally). More broadly however, in terms of a client-server model, a service could be provided (for example):

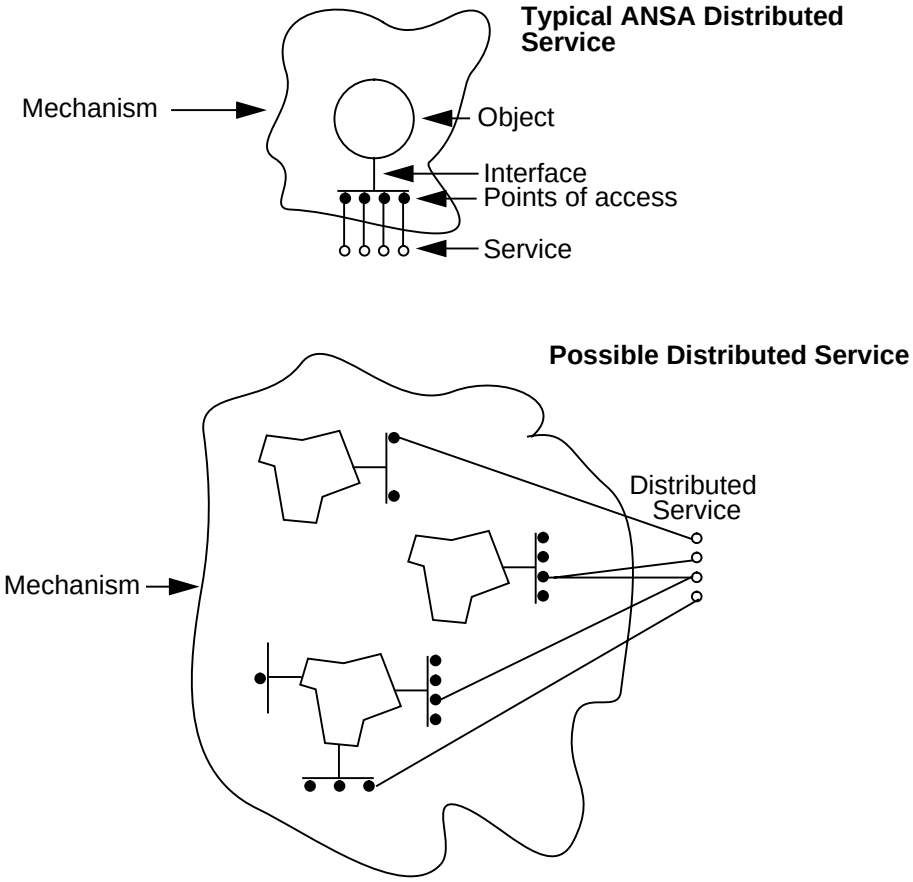
- by all of its capabilities being available only at a single interface to a single server;
- by partitions of its capabilities each being available only at single interfaces to single servers;[1]

1. This replication may not be able to be rendered transparent in the domain.

- by all of its capabilities being available (replicated) at each of a number of single interfaces to servers; or,
- (maximum complexity?) by overlapping subsets of its capabilities each being available at each of a number of interfaces to a number of servers in which interfaces may also provide additional capabilities outside the service.

Figure 2.1 illustrates some of these variations.

Figure 2.1: Example distributed service



In general, however (particularly in the case of the federation of “legacy” systems) the existing expression of a service may not be in terms of a client-server model.

Table 2.1 provides some examples of services, plausible points of access and required access methods.

Table 2.1: Example services

Service	Points of access	Access method
Trading service	trader interface	RPC
OSI[a] managed object	CMIS application service object	CMIP
POTS[b]	telephones	physical manipulation of telephone “dial”, voice and hearing

Table 2.1: Example services

Service	Points of access	Access method
OSI Transport	local Transport SAP	endsystem requests, indications, responses and confirms,
POSIX[c] operating system	libraries	external procedure call
WIMP[d] program	keyboard, mouse, screen	WIMP interactions

a. Open Systems Interconnection
b. Plain Old Telephone Service.
c. Portable Operating System Interface for Computer environmentS
d. Windows, Icons, Menus and Pointers

2.2.1 Service Access

In an infrastructure-based client-server model the information required to define an access method (for an arbitrary service) can be partitioned into:

1. (service level) identification of and server interfaces the function names that provide the service;
2. (object level) for each server, identification of the server-specific platform-supplied access options selected; and,
3. (infrastructure level) identification of the infrastructure implementation-dependent access options used.

A similar service access profile can be developed for services whose implementation does not follow an infrastructure-based client-server model by performing an analysis of the service to identify functions and interfaces, and some lower level of shared service analogous to that provided by a shared object infrastructure.

The remainder of the document assumes that such an analysis has been performed and talks separately of, for example, co-operation between objects and between infrastructures.

2.2.2 Objects

Computational objects are described in [APM 1001], the ANSA computational model, and [ISO ODP-3]. They support one or more interfaces (the number of which may vary over time). They are implemented by a number of different kinds of *engineering object* (e.g. as described in [ISO ODP-3]) one or more of which encapsulate the functional aspects of a computational object (these are called *basic engineering objects*) and others of which implement an infrastructure for the basic engineering object to use. Alternative arrangements of the latter result in qualitative (non-functional) implementation differences that are transparently available to the basic engineering objects.

Two styles of interface that computational objects can support are referred to in this document (and in [ISO ODP]):

1. operational interface

   in which functions are made available in a manner analogous to procedure calls; and,

2. stream interface

Note: (Gray) I think this is what Yigal is referring to when he says peer-peer. “Peer-to-peer”, however, would literally describe the relationship between a client and a server too.

in which the functions supported must involve one or more parallel information-flows.

At an engineering level compatibility to these interfaces must be considered in terms of:

- (object level) the basic engineering object interface that parallels the computational object's interface;
- (infrastructure level) the transparency mechanisms that are used; and,
- (infrastructure level) the end-to-end protocols that engineering protocol objects support.

During construction, or dynamic reconfiguration the interfaces between the engineering objects implementing a computational object are also of concern. The processes involved in co-operation between computational objects (including advertising services, trading and binding) can be applied (perhaps on only a local basis) among engineering object to achieve this. The reconfiguration implied by migration transparency requires non-local support of these processes, potentially treating the service supplied by infrastructure engineering objects to be treated as a separate computational object (to which basic engineering objects are the clients).

2.3 Domains of Homogeneity

Boundaries delineate differences and thus define the limits of areas which are uniform (homogeneous) in whatever properties differ at the boundary. These areas are called domains of homogeneity, and they are characterized by the properties in which they are uniform. Similarly boundaries can be characterized by these properties (but they differ at a boundary). There are many examples of such properties (e.g. as will be defined in the categorization in Chapter 3).

In addition to a property the type of entity to which the property applies is also an important element in the specification of a domain of homogeneity (the property may be meaningless without this information).

It is now possible to provide more formal definitions.

- A *domain of homogeneity* is a set of entities of that share common properties. It is characterized by the properties and the type of entities involved.
- A *boundary* is a difference in the properties of domains of homogeneity. It is also characterized by the properties and the type of entities involved.

Table 2.2 provides some example domains (or boundary) characterizations.

Table 2.2: Example properties and types defining domains

Domain property	Entity type
Author	Software specification
Owner	Local area network
Export control applicable	Cryptographic technology
OSI Transport-layer protocol class	OSI Transport-entity
Colour	Computer enclosure

Table 2.2: Example properties and types defining domains

Domain property	Entity type
Supplier	Application
Authentication mechanism	Operating system

This definition of a domain is less prescriptive than some and thus does not have some otherwise common implications of the term as discussed below.

- Domains (of homogeneity) need not be geographical areas

  Because an entity type and a property do not necessarily relate to geographical location (or, indeed, to realized instances of objects at all) domains of homogeneity do not necessarily occupy a geographical “area”, and nor do the boundaries between them. However, when the type of entity associated with a domain requires it to be located in a specific computer network, or when the property is shared only by objects in such a network then the domain is likely to cover an “area” defined by the network.

- Domains (of homogeneity) need not be defined by an authority

  Similarly the type and property do not necessarily define a common authority (e.g. one that dictates behaviour relative to the property). An authority is not a fundamental property of a domain. Boundaries do not exist only where the scope of different authorities meet.

  Authority domains are described in 3.3.

One set of objects may define a single domain under one property and many under another (for example the same set may define a single domain under an “owner” property but many domains under the “supplier” property. There is no reason that the domains formed under one property should represent the same division as the many domains formed under another (e.g. Figure 2.2). The existence of one type of boundary between two objects does not therefore necessarily imply the existence of an other. Boundaries of different types may not be coincident.

2.3.1 Independent decomposition

It simplifies the analysis of boundary problems if the problems posed by one kind of boundary can be considered in isolation from those posed by others. It is important to establish the principle by which a set of entities can be decomposed independently into domains according to different properties.

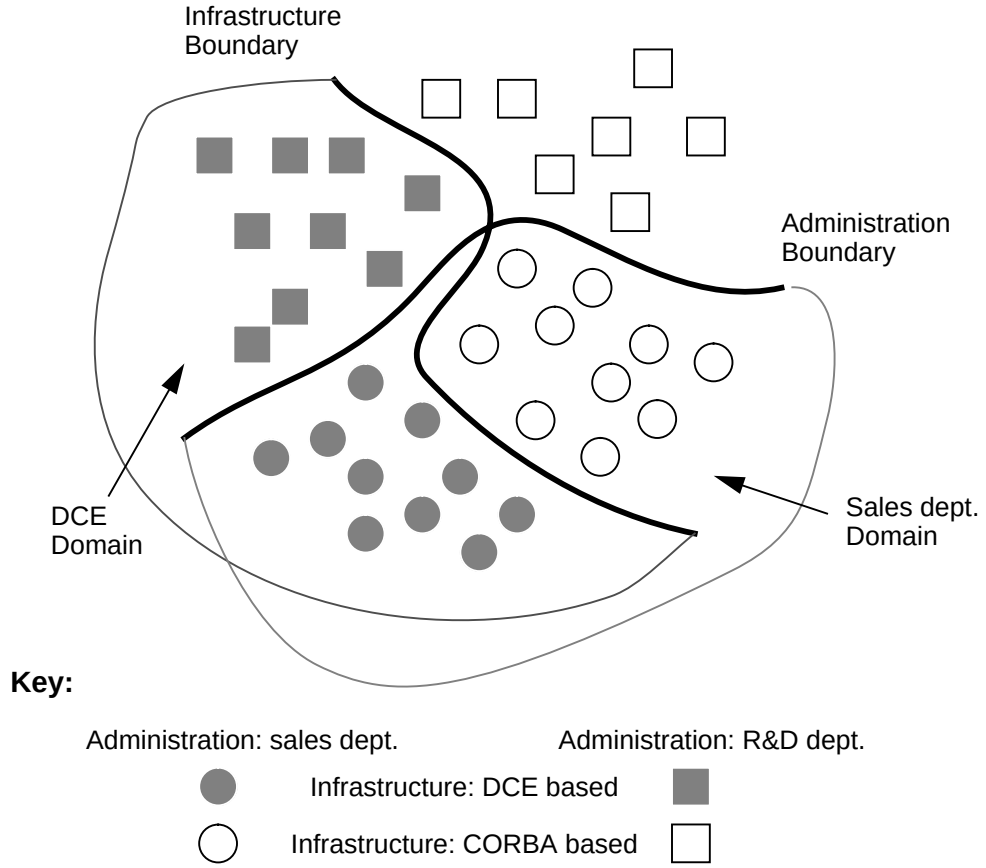
If the boundaries defined by one property are dependent on those defined by other properties then (for example) mechanisms providing for federation over one boundary may interfere with those providing federation over another and it will not be possible to consider each type of boundary independently.

Without an independent decomposition principle it will be necessary to assume that all boundaries potentially interfere with all others.

Note: (Gray) Further work is necessary here to describe this principle and to define the circumstances in which the boundaries defined by one property can be considered in isolation from those defined by other properties.

Note: (Gray, Jane) The relation of this to “feature interaction” should be investigated too.

Figure 2.2: Different domains formed under infrastructure and administration properties



2.3.2 Federation and separation of domains

These problems are relative not only to the property defining the boundary in question but also by the type of co-operation addressed. (That is, each problem is defined by a property making a difference and a type of co-operation requiring a similarity). Some examples are shown in table 2.3. Not all forms of

Table 2.3: Example types of co-operation

Type of domains co-operating		Type of co-operation
Common property in domain	Element type	
Administrator	ODP computational object	Inter-management
Owner	Object	Payment
Function	Object	Use
Designer	ODP Engineering platform	Joint transparency provision
Author	Software specification	Mutual use of specifications
Author	Software specification	Conformance to common standard
Owner	Local area network	Permission to access LAN
Export controls applicable	Cryptographic technology	Specification of controls applicable to joint products
OSI Transport-layer protocol class	OSI Transport-entity	Transport-layer communication

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).[1]

1. The involvement of more than one type of domain when specifying a type of co-operation is not illustrated in the table of examples.

3 Classification of Properties

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

There will be different types of domains and boundaries depending on the properties that may differ between systems. This chapter outlines the main properties of large scale distributed systems in which differences are likely. Such properties form the basis for studying:

- the information necessary for the description and comparison of the systems (relevant to trading); and,
- the information and processes necessary to enable, disable or detect the crossing of these boundaries (relevant to federation, separation and monitoring).

Note: (Jane) what about the processing part of trading?

Systems will have to incorporate mechanisms

- for crossing these boundaries, where this is feasible and permissible, and
- to prevent boundary crossing, where it would lead to failure or breach of security.

3.1 Approach to using the distributed system properties

Two questions arise with regard to the differences between distributed systems:

1. What is the operational impact of each type of difference when planning, designing, implementing, installing or invoking services that span different systems?
2. When is it possible to bridge these differences, and, where it is, what mechanisms must be put in place between differing distributed systems?

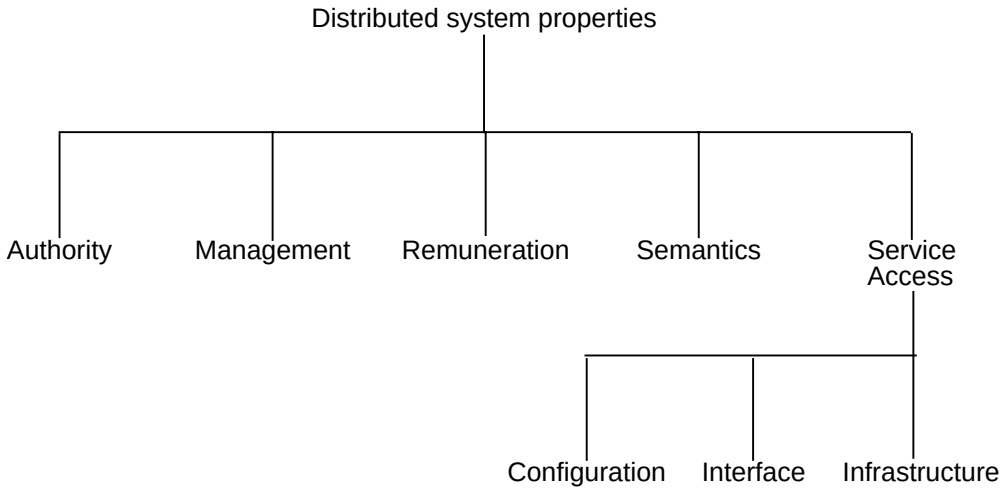
3.2 Large scale distributed system properties

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 (Figure 3.1):

- authority;
- management (administration);
- remuneration (i.e. accounting, billing, payment and receipt generation);
- service semantics (modelling and naming); and,
- service access (configuration, interface and infrastructure access).

Differences are anticipated in the information, procedures and mechanisms used to support each of these properties.

Figure 3.1: Properties of distributed systems



The distributed system properties shown in Figure 3.1 are neither exhaustive nor complete. Other classifications are possible. Nonetheless generating a list of issues associated with these properties will enable not only the specification of a number of relevant differences but also the identification of homogeneity.

Note: (Gray) I have placed remuneration and object interfaces on a par with the other properties (a) for simplicity of presentation and (b) because the opportunity to address these two areas with common text has not actually been taken.

Note: (Gray) I have also added “configuration” to interface and infrastructure and placed them all under the same heading “service access”, as described in 2.2.1.

The object interfaces property encompasses both operational [Birrell?] and stream-based [ISO ODP-3 94] interfaces. Increasingly, distributed systems are the result of development processes that are themselves carried out in a distributed fashion (i.e.distributed development processes are used). Such processes may differ in terms of:

- the decisions made concerning all of the properties listed above;
- the way the decisions are made; and/or,
- the way the decisions are recorded.

The result of the possible divergence among systems is a list of the following distributed system properties:

Table 3.1: Relationship between entities

Entity	Relationship
Authority	institutional framework – agreement
Management	administration
Remuneration	accounting, billing, payment, receipt generation
Semantics	models and naming
Configuration	distributed service mapping on to the object interfaces providing it

Table 3.1: Relationship between entities

Entity	Relationship
Interfaces	object service provision and consumption
Infrastructure	engineering and technical

Table 3.2: Process Relationship between processes

Process	Relationship
Development	object description agreement

Note: (Gray) I don't think we need the list of bullets in 3.2, and table 3.1 and figure3.1 – they all say the same thing. Suggestions for removal?

As an example of the use of this classification: if it is known that two systems have a single authority and management structure/policy, it is reasonable to concentrate on the other properties, where differences may lie. It is often inferred that if the differences between the infrastructures could be eliminated by using standardization, most of the problems associated with distributed systems would disappear. Figure 3.1 indicates other properties, such as authority and management for example, are likely to continue to present obstacles to federation, as they reflect organizational and not technical variety.

3.3 Authority

An *authority* is a decision making body that authorizes activities (such as creation and control). Authorities of a distributed system will determine the approach taken concerning all the issues where options are available. A software entity, person, organization, legislative body, or country are all examples of potential authorities.

An *authority domain* encompasses the range of objects and activities about which a specific authority may validly make decisions.

An authority may employ a policy or philosophy as a systematic basis on which to make decisions. These policies might address allocation and de-allocation of resources, provision and availability of services, trading, identification, authentication and authorization, accounting, billing and payment, quality notions and quality control, etc. The properties of this chapter represents one way to classify some of these issues.

Authorities will vary on the nature of the agreement with another authority that they will regard as acceptable.

3.4 Management and administration

An *administration* is a body responsible for the enactment of an authority's decisions. A software entity, person, organization, legislative body, or country are all examples of potential administrations. In a federated system there are likely to be many administrations, responsible to many authorities.

Each administration has local autonomy, subject only to its authority and agreements reached with other administrations. If object services are provided

and consumed across administrative domain boundaries, the domain administrations are responsible for ensuring that any guarantees given by the clients and servers are kept.

An administration exercises its responsibilities by managing objects in an authority domain.

There are several different types of management of concern in large scale distributed systems (Figure 3.2) including the following.

- System management:

  this is largely the monitoring and control of the infrastructure resources that provide the basic as well as the extended distribution facilities. System management includes:

  — resource allocation and garbage collection;

  — configuration management; and,

  — object creation and destruction.

- QoS management:

  this concerns the provision of end-to-end QoS guarantees when service provision crosses between different domains. This requires the appropriate policies, mechanisms and procedures addressing:

  — security;

  — dependability; and,

  — performance.

- Application management:

  this concerns the provision of application specific functionality:

  — trading; and,

  — application specific management.

- Development process management:

  this concerns the monitoring and control of the distributed development process.

- Specification configuration management

  — conformance testing

  — refinement checking

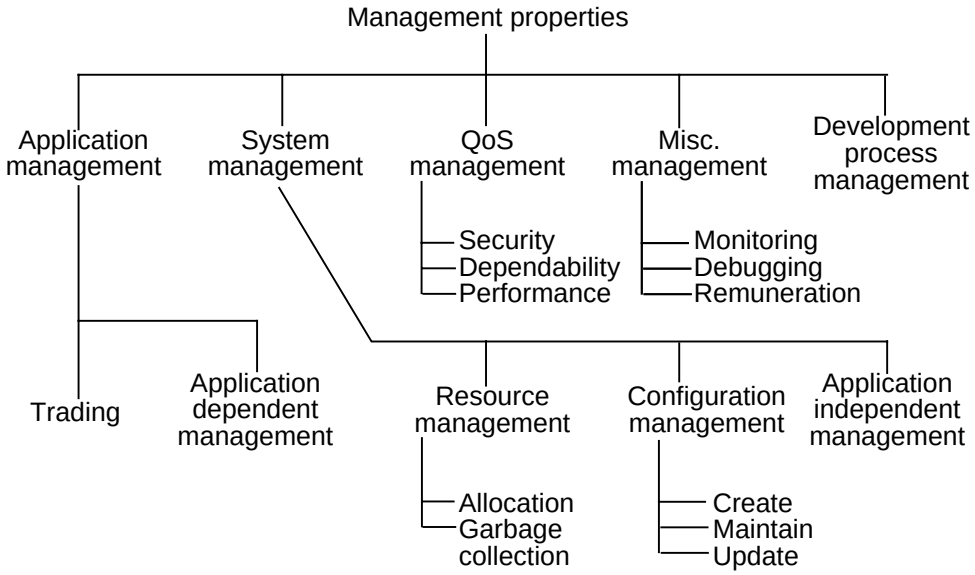
  — specification access control

- Miscellaneous management functions:

  — monitoring; and

  — debugging.

Figure 3.2: Management properties

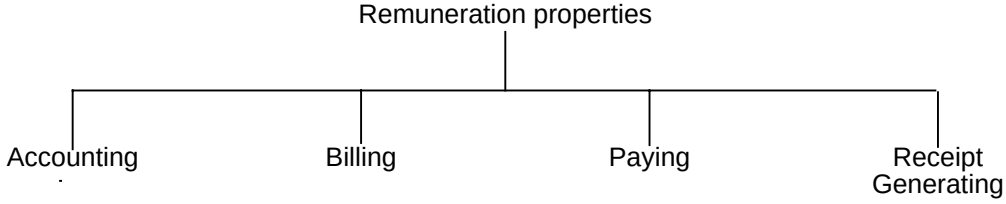


3.5 Remuneration

Remuneration includes four different processes (Figure 3.3):

- 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; and,
- receipt generation – (optionally) producing evidence that payment and the corresponding service provision have occurred.;

Figure 3.3: Remuneration properties



Remuneration is closely related to issues of trust and security.

3.6 Semantics

Semantic modelling concerns the way in which people and systems organize the meanings they associate with the world around them. This includes the way they represent the necessary information, its organization and the names used to refer to it.

Mechanical systems such as computers can only deal with concrete representations, such as bit patterns. Previous work on naming models [APM1001.1 93] and the information model [APM1005.1 93] illustrate some of the limitations with respect to the representation of concepts and subsequent interpretation of representations (e.g. of service specifications).

A semantic description may be expressed at many different levels of abstraction and may be either descriptive, intending to describe only what the behaviour is (e.g. a behavioural specification provided in a formal language, or as a set of semantic property names) or prescriptive, intending to describe how the behaviour is to be achieved (e.g. an object's behavioural representation in a programming language).

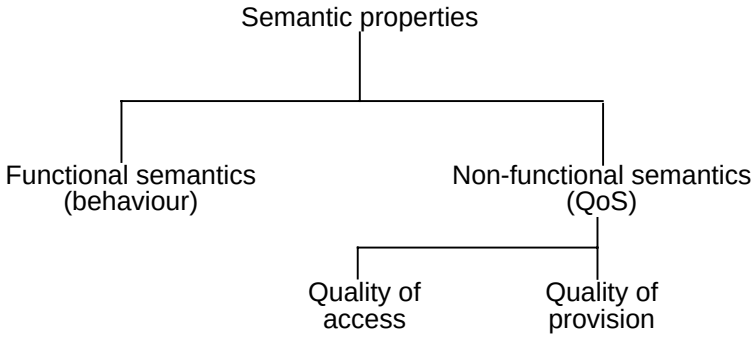
Two areas of semantic description can be discerned: the semantics associated with the functional aspect of a service; and the semantics associated with its non-functional aspects (for example its security, performance, or dependability).

- A functional semantic description of a service will describe the meaning (e.g. in terms of cause and effect) of the capabilities it provides.
- A non-functional semantic description of a service (its Quality of Service, QoS) will describe qualitative aspects of the way in which the service is provided/delivered in terms of (for example) administrative, temporal, confidentiality or accuracy guarantees.

The QoS associated with an interface perceived by a client is composed of the QoS of the access method used (the Quality of Access, e.g. as provided by ANSA transparencies) and the QoS associated with the object itself (the Quality of Provision).

The functional and non-functional specifications of objects have to be available in a form that allows the trading process to compare them and to find compatible objects. The potential categories of information about semantics are shown in figure 3.4.

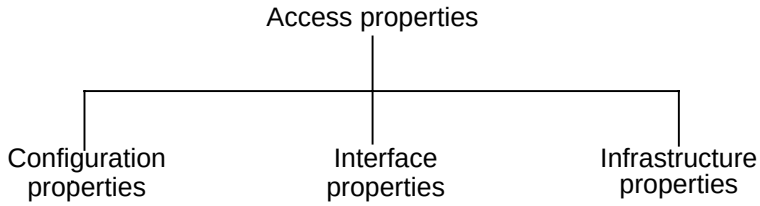
Figure 3.4: Semantic properties



3.7 Service Access

As indicated in 2.2.1 service access can be split into three main areas, as shown in figure 3.5.

Figure 3.5: Access properties



3.7.1 Configuration

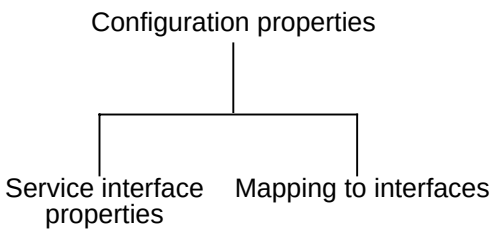
A service may be implemented by a configuration of objects interacting in a predetermined fashion and perhaps constructed according to a specific template. The user of such a service need not be aware of this when the service is manifest at a single interface to a single service. However, factors that may prevent this transparency include:

1. the capabilities of a service required by the client may be provided at no single interface, only at a combination of them;
2. the nature of the service may require access points supporting different access methods[1] (such as the separate control and call interfaces to a telecommunication service) and thus division in to separate interfaces; and,
3. the unit of service availability may be regarded as that available from a set of interfaces for the purposes of marketing.

Thus information describing this configuration (at least in terms of the interfaces involved) is required for the general case of service.

Some of the same factors may also require the identification of a configuration associated with the user of a service. For example a telecommunications service may require a configuration of two objects, one the client of a control (operational) interface and another the client of a call (stream) interface.

Figure 3.6: Configuration properties



1. For example, [ISO ODP]] currently requires operational interfaces to be regarded separately from stream interfaces.

3.7.2 Interfaces

ANSA advocates a Remote Procedure Call (RPC) style of access to interfaces. This object interaction paradigm is also advocated and used by a number of relevant international standards (such as [ISO ODP], [ISO Management 91],[ITU-T MHS 88],[ITU-T Directory 88] and [ISO RDA 90]). [ISO ODP] effectively promotes this style of interaction (associated with operational interfaces) whilst associating all other forms with “streams”. [ISO ODP] unifies stream interfaces and operational interfaces as examples of signal interfaces.

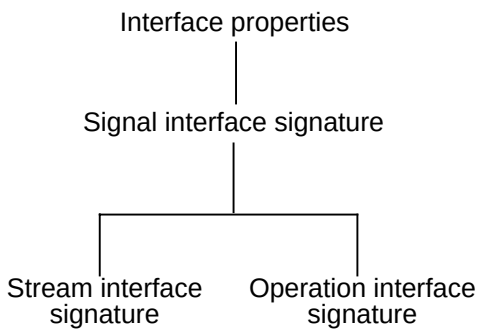
The use of different object interaction paradigms by client and server is a boundary issue. Whilst there is a preference for ANSA interfaces to be operational interfaces, a federated object must be modelled as providing or requiring the more inclusive range of signal interfaces.

Interface descriptions can be broken down, according to the interaction paradigm associated with the interface, into a number of categories as follows (see figure 3.7):

- operational interface signature: e.g. in terms of operations and data types;
- stream interface signature: e.g. in terms of information flows and data types; and,
- signal interface signature: e.g. in terms of signals (e.g. client or server events) and data types.

Type systems involve notions of equivalence and substitutability; types can be represented by interface definition languages (IDL) and abstract data types [APM1042.1 93].

Figure 3.7: Interface properties



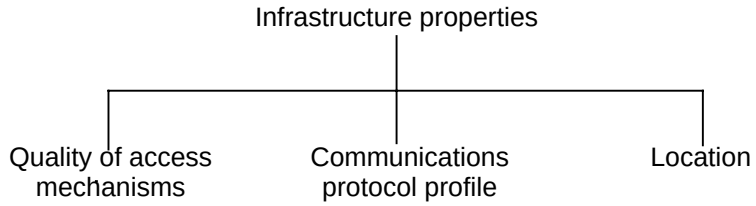
3.7.3 Infrastructure

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

The properties associated with distributed system infrastructures can be broken down according to the categories of information necessary for setting up the co-operation between them (Figure 3.8):

- transparency mechanisms that provide a required quality of access;
- communications protocols; and,
- location.

Figure 3.8: Infrastructure properties



Note: (Jane) these appear arbitrary, is this list complete? I think there should be more, is it representative or illustrative?

Transparency mechanisms and communications protocols are closely related to the quality of service, insofar as they represent the principle means through which a given quality of service is provided.

4 Relationships across Different Types of Boundary

The following sections describe the boundary problems that each of the properties of distributed systems (as shown in Figures 3.1-3.8) give rise to when considered independently.

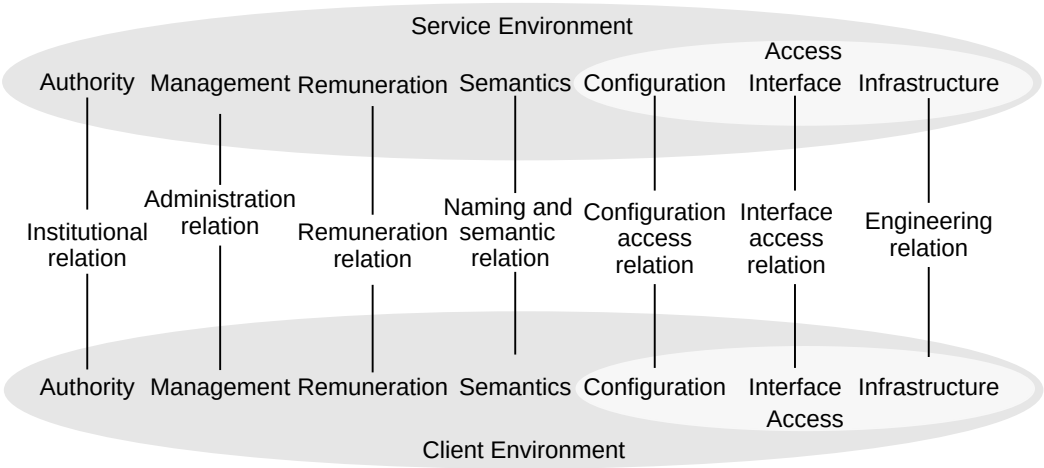
Note: (Gray) this chapter should describe *problems* not *solutions.*

Note: (Yigal) the following sections should be expanded.

4.1 The relationships between distributed systems

By using the list of properties described in Figure 3.1, it is possible to show the relationship between the two distributed systems environments associated with a client and a service (figure 4.1).

Figure 4.1: The relations between properties of two distributed systems

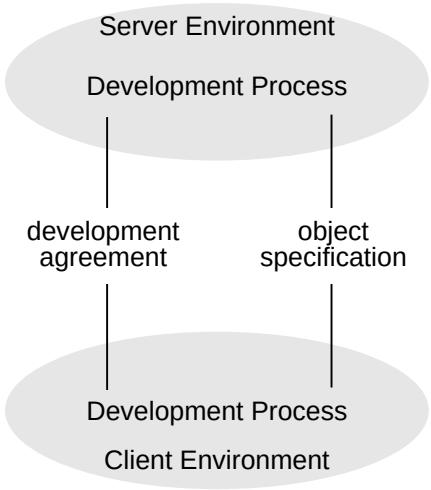


Note: (Yigal) Feature interaction??? – (Gray: i.e. issue of section 2.3.1?)

In order to ensure the co-operation between two objects it is necessary to have agreement and compatibility regarding (at least) all the properties listed in figure 3.1. This would require the specification of an object in terms of these properties.

A distributed development process will have to communicate at some epoch(s) and exchange these specifications in order to ensure co-operation between objects. Thus the relationship between two development processes consists of agreement and object description (Figure 4.2). The relationship between development processes is based on the description of the objects being developed.

Figure 4.2: The relations between two development processes



4.2 The institutional relation (authority)

Differences between authorities reflect organizational, judicial/legal, economic, social, political, cultural as well as technical issues. For co-operation to be possible between objects of different authorities, it is first necessary for the authorities to reach an agreement concerning co-operation. Such an agreement would entail:

- a specification of what is offered;
- a specification of what is required;
- an undertaking to provide the guarantees within an institutional framework; and,
- a statement about the consequences of not being able to fulfil the guarantees.

4.3 The administration relation (management)

There are different types of management which concern distributed systems (as shown in figure 3.2). Differences between administrations reflect the many ways of doing management

In each one of these areas the differences between administrations may manifest itself in different:

- management policies;

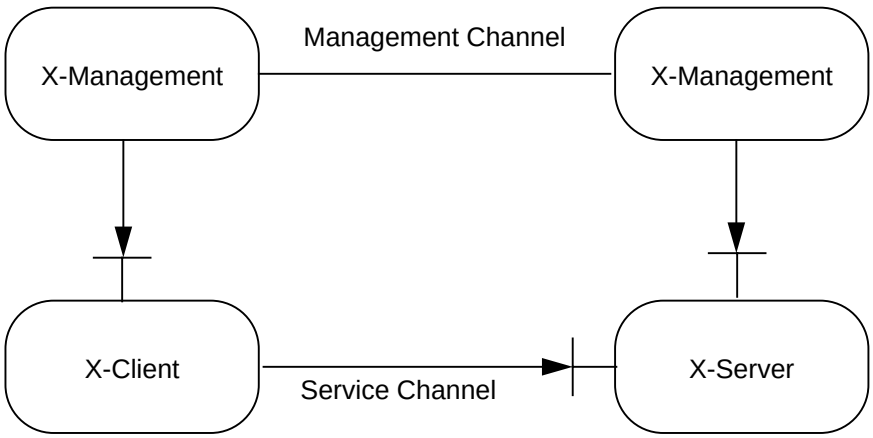
Note: (Jane) examples here?

- management procedures (e.g. accounting, billing, monitoring or quality control); or,
- management agents and interfaces.

4.3.1 Service management

Models of all the different properties of large scale distributed systems are needed to compare these systems so that the differences can be dealt with in a coherent way.

Figure 4.3: A generic model of X-service provision and consumption and its management



Differences in these properties will make it difficult to cross boundaries. The following examples demonstrate this.

- Monitoring across a management boundary:

Note: (Gray) further text needed here:

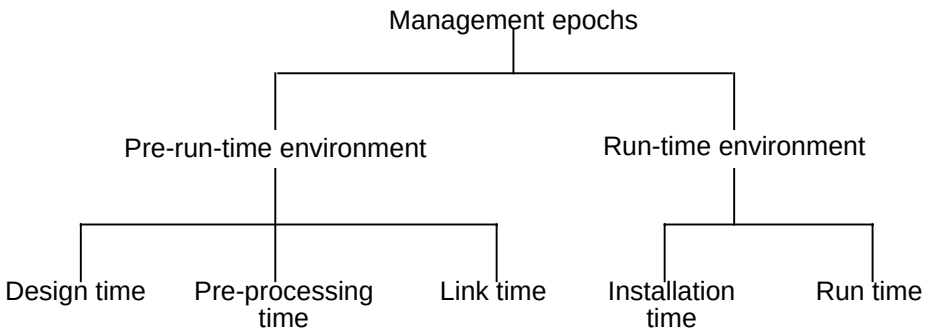
- Failure management across boundaries: what happens if a client and server from two different systems co-operate whilst a failure occurs in one of them. What if the idea of dealing with failure is contradictory?

Note: (Yigal) roll back versus transactions - (Jane) only two ways, not complete

Note: (Gray, Jane) Text needed to explain relevance of Figures 4.3 and 4.4.

In addition to the model of the actual X-service being performed, there is also the way in which the management of X is done: for example, monitoring.

Figure 4.4: Management epochs



4.3.2 Integration of different management agents

Note: (Gray) Text required

4.4 The remuneration relation

The remuneration relationship can be extremely complex as it reflects the variety of existing remuneration models, agents and procedures. Differences in remuneration models may cause problems when an invocation crosses one or more of these boundaries, as is likely in the case of large scale distributed systems (such as telecommunication systems).

Different systems may have different views and implementations of each of the processes identified in section 3.5. In particular, conflicts may arise between systems that have different accounting and billing strategies. 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.

4.5 The naming and semantic relation (semantics)

Part of the information needed to be able to access a service is the syntax in which things of different types are to be represented. This includes the way in which references to objects (names) are represented. These issues are discussed in 4.6.2.

Differences in semantics, where two domains do not share the same concept, result in fundamental barriers to interaction, that are quite distinct from the issue of how a concept is represented for service access purposes.

4.5.1 Service functional semantics

Semantic differences between services will range

1. from services with entirely different purposes
2. through ones that deal with the same problems but may have a slightly different model of the problem
3. to services that achieve an identical behaviour (but may do so in different ways).

Bridging the semantic difference in case 1 is not appropriate and in case 3 is not necessary. Consider two time services as an example of the remaining case. The time services may provide different resolutions (1 second for one and 1 minute for another) and may vary in whether they include the date. They have the same purpose: to report time, but they have slightly different concepts of time in mind.

In such cases it is possible to imagine “semantic interceptors” that converts invocations suitable to the semantics (and syntax) associated with one service to a form suitable to the semantics (and syntax) of another.

A difficulty in bridging these differences is finding the specification of the semantics of a service prior to evaluating its compatibility with the semantics required by a client. This may be: because no sufficiently complete semantic description is available to the client; or, because the semantic description provided by the server is incomprehensible to the client.

One way to make information about semantics available is to make it part of the service specification that traders maintain. A potentially difficult issue is the form in which such information should be maintained so as to make it understandable to people and/or computer systems.

4.5.2 Service non-functional semantics

Quality of service is a measure of the quality attributed to or required of a service or activity. Major areas concerned with quality are:

- security (including confidentiality, integrity and availability measures);
- dependability (including reliability and integrity measures); and,
- performance (including time and capacity measures).

Consideration must be given as to how to specify requirements in these area in a declarative manner that can be translated to a variety of platforms and mechanisms.

Quality of service specifications have to be translated into mechanisms which support its achievement. The same quality of service requirement may be provided by different mechanisms (i.e. there isn't a simple 1:1 mapping between them). Some of these mechanisms may be provided by a distributed infrastructure (e.g. transparency mechanisms), which are addressed in section 4.6.3. However, the translation between quality of service requirement and mechanism capable of providing it is part of the development process which is expected to be automated by tools as much as possible.

4.5.3 Naming

Many concepts are represented in large scale distributed systems and among them the representation of a reference (i.e. a name) is particularly important. Differences in naming result in:

- (different semantics, same syntax) different things having the same names in different contexts; or,
- (same semantics, different syntax) the same things having different names in different contexts.

The former is a problem of crossing a semantics boundary, the latter is not. The former can be avoided through the mapping of a name in naming domain to one in another. The latter can be solved through the mapping of a name's representation.

The major problem with regard to naming and co-operation in large scale systems is that the use of different naming schemes makes it difficult to compare names to discover the compatibility or incompatibility of objects or the properties denoted by those names.

4.5.4 Naming models

Note: (Mike) this section is verbose and unclear

The naming problem arises when referring to things. These might be:

- things external to the computer system; or,
- objects inside the computer system which have themselves to be named (for example, where activities relate to internal names such as management entities).

There are differences in the way naming models are set up in systems, and also in the way in which things are named.

The use of different naming models to describe systems makes it difficult to compare objects referred to by the names in these models and to discover the compatibility or incompatibility of objects that wish to interact. Naming

models reflect the way in which people view the world around them. This results in naming models differing in:

- what is being named;
- how it is named; and,
- in what context the naming take place.

Note: (Gray) “how it is named” is another example of a syntax or representation boundary, see note above.

Note: (Yigal) this is related to notions of boundary.

4.6 Service access relations

4.6.1 The configuration access relation (configuration)

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

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

4.6.2 The interface access relation (interface)

An interface description can be broken down as shown in Figure 3.7. Semantic differences are discussed in Section 4.5. Differences in the interaction paradigms assumed and used by objects must be resolved.

The signature of an interface is described (in section 3.7.2) in terms of data types and the structure of an operation [APM1001.1 93], flows or signal data [ISO ODP] in the interface. A number of *type systems* that differ in the way data types are defined and interrelated are likely. Each style of signature requires a means of determining access compatibility (between what is provided and what is required). Such interface access compatibility checks will normally be based on a substitutability relation inherent in whatever type system is chosen.

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

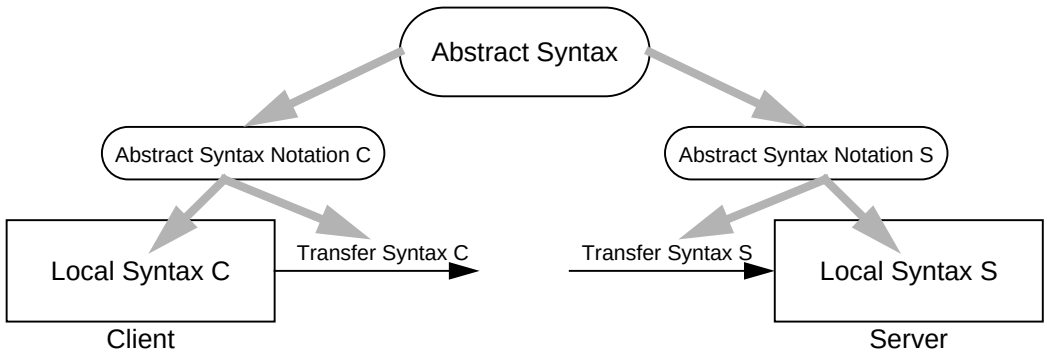
1. If the type system is defined in terms of interactions supported on the data typed (as is the case in the type system described in [APM1001.1 93], [APM 1014.1 93] and [APM 1015.1 93]) these constructs may coincide.

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

4.6.3 The engineering relation (infrastructure)

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

Figure 4.5: Local, transfer and abstract syntaxes



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

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

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

The representations of references to objects (names) needs particular care during translation to maintain the semantic distinction between the naming domain that created the reference, the naming context in which the reference is used and, potentially, the domain that created the object [APM1003.1 93].

Note that, as discussed in 2.2.2, the infrastructure can sometimes be regarded as an object in its own right, supplying a service to e.g. a migratable basic engineering object. Most of the boundary related problems described in this document apply to this pairing of objects including problems that arise when:

1. Access transparency addresses this problem.

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

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

The differences between distributed system infrastructures that concern interoperability can be broken down according to figure 3.8.

- Transparency

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

- Communications protocol profiles

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

- Location

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

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

5 Conclusion

This document has described the environment in which federation is assumed to take place, identifying the relevance of service, objects, boundaries and domains. A number of properties that potentially form separate boundaries and domains have been isolated and the principle problems that need to be overcome at each such boundary have been described.

6 Dealing with Different Classes of Boundary

Note: (Gray) My feeling is that the following material should be exported to documents focused on each of the boundaries identified in the previous chapter. This document should end here.

Note: (Gray) This material is structured but unpopulated – even the structure, however needs to be revisited once questions arising from the chapters above have been settled.

Interception mechanisms are those that support federation, separation or monitoring across boundaries. Connecting systems which differ in any of the properties discussed above will require the delineation of the differences by boundaries and the provision of interception where activities cross these boundaries.

To enable interoperation of clients and servers on opposite sides of a boundary, it is necessary to bridge the differences in some way so that bindings can be set up, and messages and information flows can be exchanged meaningfully. 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”.

6.1 Distributed systems as domains

A physical ensemble of technology can represent a domain.

6.2 Authority boundaries

Note: Text needed

6.3 Management boundaries

Note: Text needed

6.4 Remuneration boundaries

Note: Text needed

6.5 Semantics boundaries

Differences at the semantic level need to be considered in terms of similarity of purpose.

Two services intended to do completely different things (e.g. a weapons control service and a holiday booking service) are not candidates for federation. This is not a criticism of any methodology used in solving federation problems – in some sense the requirement for federation is simply “wrong”.

Two services with overlapping goals are suitable for federation. If the goals are not identical, but overlap, the common subset of the goal needs to be isolated. Either service may address goals outside this common subset. The common service that is to be presented in the joint environment can either address this common subset or, potentially, an extension of it (e.g. the extension represented by either of the component services). Services that do not meet an extended service would need enhancement.

Note: (Mike) extensions complicates unnecessarily

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. Federation strategies include the following.

- Common semantic subset

  A common subset of these two services is a time service that provides only the time, not the date, to a 1 minute resolution. This could be chosen to be the federated service provided in the joint environment.

- Extended common semantic subset

  As an alternative to the above the federated service could be defined to be the common service extended with the date. In this case the federation strategy would require the extension of the date service that did not provide the date.

Thus the semantics of the services to be federated and the resulting federated service may all be different but are related.

6.5.1 Common semantic subset

A federated service is created with semantics that are common to both services being federated. This service is implicitly realized by both services being federated, however clients must treat the new service as a separate from those federated (since it is likely to have reduced functionality).

6.5.2 Extended common semantic subset

A federated service is created incorporating semantics that are common to both services being federated, but that also incorporate additional features. This services is realized by mechanism implementing the additional features and utilising the service the common semantic subset strategy yields. The mechanism may be provided either at a central point, co-incident with the old services, or distributed elsewhere. Again clients must treat the new service as separate from those federated.

6.5.3 United semantics superset

A federated service is created as described in the extended common semantic subset in which the extensions incorporate the semantics that each individual services has outside their common subset. This is feasible only when the extensions necessary to each service are not contradictory. When this is possible the federated service can be provided to clients transparently (i.e. as a substitute for the services federated).

6.6 Service access boundaries

Two arbitrary services can differ on two different levels: a semantic (what) level and a mechanism (how) level. The semantic level deals (above) with difference in what the goal or mission of the service is or what it is supposed to achieve, either in general or in detail. The mechanism level deals with how it has been decided to fulfil that goal/mission/objective.

6.6.1 Mechanism Level Differences

Differences at the mechanism level fall into two different categories: access differences and configuration differences.

- *Access differences*: are differences in the means through which access is established, in the communication paradigm used, and in the representation of information involved. (Much, but not all, of this is standardized in an ODP/ANSA, platform based, environment. However differences in the transparencies used and in other details of the engineering of the platform/infrastructure will still need to be considered.)
- *Configuration differences*: are differences in the organization of interacting components that provide the service and in their internal accesses.

When a federated service is to be provided from existing services there are two strategies that may be used:

1. Service interface composition:

   construct the federated service exclusively out of accesses to the services being federated (as though each were encapsulated);

2. Service mechanism composition:

   construct the federated service by inter-mixing the components of the services to be federated into a new configuration

Strategy 1 requires only access differences to be resolved (e.g. by insertion of mechanism in the access path). Strategy 2 requires configuration differences to be addressed too and, if there was no similarity in their design may not be possible.

Note: Consideration of the available strategies, their development, and their automation using tools needs to be considered in the following.

6.6.1.1 Service interface composition

This is illustrated in Figure 6.1;

Note: more text required

6.6.1.2 Service mechanism composition

This is illustrated in Figure 6.2.

6.6.1.3 Abstract syntax supporting objects

An important class of access differences is the use of different representations of things that carry the same information in different contexts, or representation domains. Abstract syntax supporting objects enable the information carried by a representation in one context to be correctly represented in another.

Figure 6.1: Encapsulated Service Federation

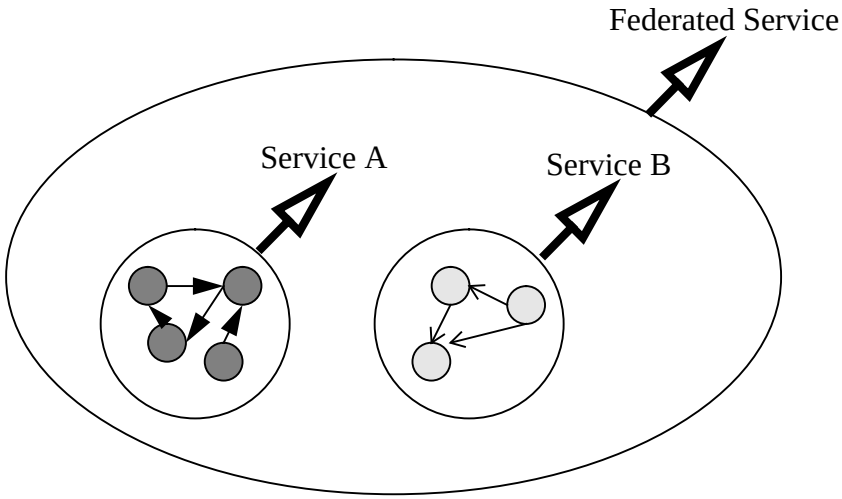
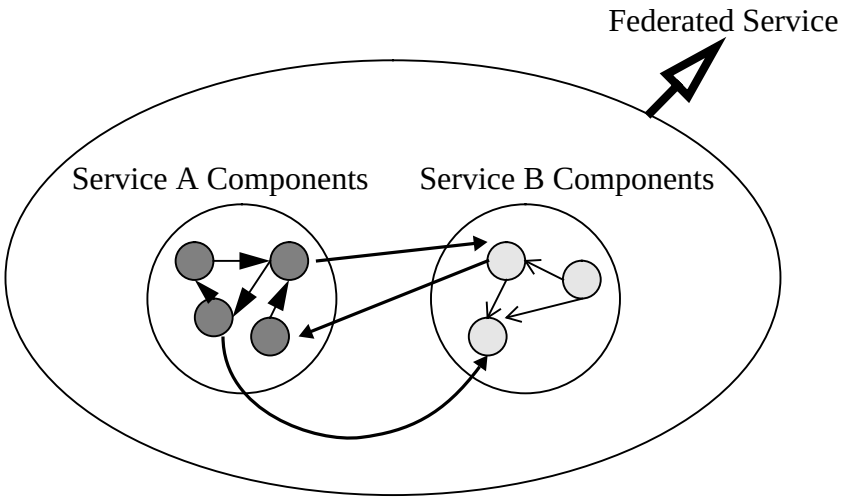


Figure 6.2: Amalgamated Service Federation



Different representation domains occur both within systems, where the use of different storage data structures or media (e.g. in memory, as a string or on a database) may befit different representations, and among systems, where a representation may be required for communications purposes that may be different to a system's local representation.

An abstract syntax supporting object can encapsulate a representation domain in such a way that objects that require representation in that domain do not then require different mechanisms to deal with different representation domains.

Objects that export or import representations (e.g. of themselves) do so with reference to a common abstract syntax notation that consists of:

- a set of base types (e.g. in the limit just the `BIT` type, but larger sets will confer greater efficiency) that have representations, and the same semantics, in all the representation domains of interest; and,
- a set of type structuring constructs (e.g. `CHOICE, SEQUENCE, SET`).

For each representation domain an abstract syntax supporting object is created with mechanisms that provides a means to export and recognize each of the chosen base types and each of the chosen type constructs.

Each object that may be represented has an associated abstract syntax that defines its representation, not in terms of a specific representation domain, but in terms of the abstract syntax notation supported. This is realized by mechanisms that *represent* its value; and *construct* its value through interaction with an abstract syntax supporting object.[1]

Different representations of an object are then possible by interacting with different abstract syntax supporting objects.

Translating one representation of an object to another is achieved by using the an object's *construct* mechanism in one representation domain and *represent* mechanism in another.

A formal specification of an object's abstract syntax enables representation translating interceptors to be constructed mechanically by compiling the object's *construct* and *represent* mechanisms from it.

Communicating an object between a sender and receiver involves finding a(ny) representation domain that the sender and receiver have in common (the transfer syntax), the sender *represent*ing the object to the common representation domain, transferring that representation to the receiver, and the receiver finally *construct*ing the object from that representation domain.

6.6.2 Quality of Service

In a federated system there may be few end-to-end quality of service guarantees, because it is difficult to guarantee something which you don't control. To the extent that there are any, they may work something like this: Services in domain A have particular QoS guarantees within domain A. There is a translator or interceptor or gateway between domain A and domain B. This gateway looks like a client in domain A and like a server in domain B. It provides QoS guarantees to clients in domain B for services in domain A, based on its internal overhead and the system in domains A and B and the guarantees it is getting from the servers in domain A. Even within a domain, there may be a guarantee on response time as measured at the server, but no guarantee on response time as seen by a client, since the server may have no control over network congestion. Thus, one may want to think of end-to-end QoS guarantees as something which the client (or a third party QoS computation service) pieces together from guarantees of individual components.

6.7 Infrastructure boundaries

Note: Text needed

1. This interaction may be indirect since an object may be composed of other objects each of whose *represent* and *construct* mechanisms may be utilized.

7 Interception Infrastructure

Both the creation of enhancements to address semantic level differences and the creation of mechanism in the access path to address mechanism level differences need to be positioned architecturally. Architectural positions wrapping, or encapsulating, objects providing access to services and positions intercepting service access are considered. Tools dealing with mechanism level configuration differences are include those enabling dynamic re-configuration.

The information required in the federation process, at any phase during development and operational use, for the automatic generation of such mechanism needs to be investigated. Such information applicable to server access is currently made available in a trader.

7.1 Wrappers

Note: Text needed

7.2 Interceptors

Note: Text needed

7.3 Dynamic configuration

Note: Text needed

7.4 Trading support

Note: Text needed

7.5 Summary

Federation and separation should be addressed both at a semantic level, at which what something is must be addressed, and at a mechanism level, at which how something is configured must be addressed.

Semantic level tools to support semantic level federation include both those that subset and those that extend existing functionality.

Mechanism level tools to support mechanism level federation include both those that address access differences and those that address configuration differences.

The tools in the scope of this work are those that support federation or separation during the development or operational use of distributed computer based services.

8 APPENDIX

8.1 Infrastructures and application boundaries

It is interesting to compare the sub-classes in the infrastructure and application with the classes of boundaries mentioned in [APM 1005].

Table 8.1: Information for co-operation and boundaries

Information	Boundary type (APM 1005)
Semantics	Properties
Type (signature)	Type system
QoS	Security, performance, dependability
Transparency mechanisms	Engineering
Communications protocols	Interconnection

8.2 ---------- garbage line ----------

CGT: Thus, one may have to deal with different authorities to get authorization to use different services, and one may have multiple domains of identity and authentication, requiring one to maintain multiple IDs and passwords (or whatever) and present the appropriate ones for each service invoked (ideally with transparency to the end user – but without compromising security by writing passwords into files or over-using “trusted hosts” approaches). The different philosophies may also require one to use different approaches to getting authorization and establishing authentication. That is, not only may there be different mechanisms (in the realm of “system”), but the different mechanisms may be based on different philosophies. For example, in some domains one may authenticate oneself once a day and get a “key” which can be presented with each invocation, while in other domains one may need to authenticate oneself with each invocation. Or in some domains one may negotiate authorization which lasts indefinitely, while in other domains one may have to go through some kind of software process daily to renew authorization.

Note: dependability - where do different transaction models fit, i.e. what happens if an invocation crosses transaction model boundaries.

References

[APM1001.1 93]

Rees, R.T.O.R., *The ANSA Computational Model*, 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, Herbert A J, *The ANSA Model of Federation and Trading*, APM Ltd, Cambridge, UK, 1993.

[APM 1014.1 93]

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

[APM 1015.1 93]

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

[APM1017.3 93]

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

[APM1042.1 93]

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

[Birrell?]

?

[ISO ODP]

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

[ISO ODP-1 93]

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

[ISO ODP-2 94]

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

[ISO ODP-3 94]

ISO/IEC JTC1/SC21/WG7 and ITU-T SG VII Q.ODP, *Information Technology – Open Systems Interconnection, Data Management and Open Distributed*

Processing – Basic Reference Model of Open Distributed Processing – Part 3: Prescriptive model, **ISO/IEC DIS 10746-3** and **Draft ITU-T X.903**, Feb 1994

[ISO ODP-4 93]

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

[ISO Management 91]

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

and,

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

[ISO RDA 90]

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

and,

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

[ITU-T MHS 88]

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

[ITU-T Directory 88]

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