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

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 overcomming each of these types of boundary are then considred 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.1139.00.02 **Draft** 22 February 1994

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Superseded by: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.1139.00.02

22 February 1994

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(0223) 323010
INTERNATIONAL	+44 223 323010
FAX	+44 223 359779
E-MAIL	apm@ansa.co.uk

Contents

1 Introduction

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 that 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 "client/server"; 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".

Note: Currently the location and organization of the text are better considered than their semantics.

Note: This is an incomplete document. There are various editorial notes in this draft document.

This document develops an overview of the types of infrastructure that are required to overcome the two principle boundary related problems in large scale distributed systems:

1.1 Scope

The boundary problems addresses in this document are those that occur during the design, implementation and operational use of distributed computer-based services. The primary goal is to control (enable and prevent) interoperation between and within such services. The consequent subgoals are federation and independence.

1.1.1 Federation

Federation is commonly required when a service user wishes to make use of a service in a different environment or when two service are provided in two or more environments and a corresponding single service is desired in the combined environment. The co-operation required to achieve this state of affairs is not limited to technical alignment.

1.1.2 Independence

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.

1.2 Overview

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.

Interoperation between objects in a large scale distributed system potentially requires a number of different types of co-operation including sharing responsibility for joint operation, sharing authority over it, sharing the management of a joint operation, 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. A classification of the different types of boundary that give rise to these forms of co-operation is chosen and the relationships that need to be established (or inhibited) across these boundaries is described.

Mechanisms capable of establishing and inhibiting these relationships are outlined and finally a number of infrastructural proposals supporting these mechanisms are made.

2 Domains, Boundaries and Services

2.1 Introduction

This chapter provides an overview of the nature of domains, boundaries and services.

2.2 Domains of Homogeneity

Boundaries delineate differences. The dual of the kind of difference that defines a boundary is the sameness, or homogeneity, that exists elsewhere. Boundaries define the limits of domains of homogeneity. Just as there are many kinds of difference (e.g. as defined in the categorization in Chapter 2), there are many properties that may be the same throughout a domain of homogeneity. These properties are fundamental properties of a domain and its boundaries. The type of entity that the property applies to is also fundamental, indeed the property may be meaningless without this information. Between them the property and the entity type define the objects in a domain and the “position” of its boundaries.

Table 2.1: 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
Supplier	Application
Authentication mechanism	Operating system

2.2.1 Domains (of homogeneity) are not 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 object 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.

2.2.2 Domains (of homogeneity) are not defined by an authority

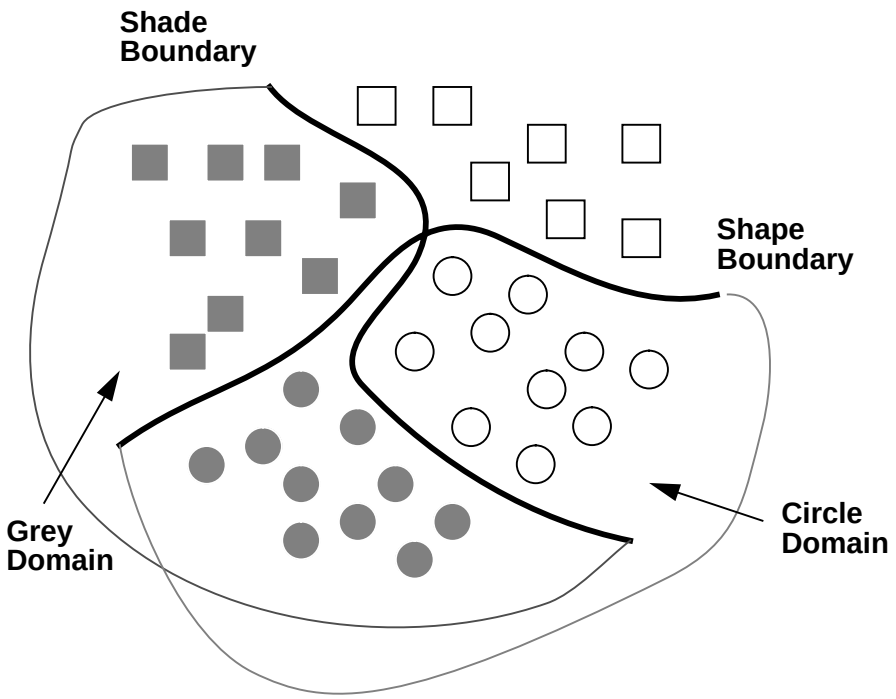
Similarly the type and property do not necessarily define a common authority (e.g. one that dictates behaviour relative to the property). An authority is not a fundamental property of a domain. Boundaries do not exist only where the scope of different authorities meet.

2.2.3 Relation between domains and the properties defining them

There may or may not be a corresponding relationship between domains to the relation between the properties that defined them. For example, if there is no relation between two properties there may be no relation between the domains they define, but if a property and a “stricter” version of it are considered the same type of object may form a smaller domain under the stricter property that is a subset of the domain formed under the less strict one.

Two sets of objects may share some properties and not others. Thus, depending on the property, the two sets may define either one or two domains, and there may or may not exist a boundary between them. One set of objects may define a single domain under one property and many under another. The many domains it forms under one property may or may not represent the same division as the many domains it forms under another (see Figure 2.1).

Figure 2.1: Different domains formed under shape and shade properties



2.2.4 Independent decomposition principle

It simplifies the analysis of boundary problems if one can consider the problems posed by one kind of boundary in isolation from those posed by other kinds of boundary. 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 the independent decomposition principle it might 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) The relation of this to “feature interaction” might be worth investigating too.

2.2.5 Federation and Independence of Domains

The most important boundary problems considered in this document are:

- (federation) there is a wish to co-operate between sets of objects but a boundary prevents this; and,
- (independence) there is a wish not to co-operate with other sets of objects but no boundary prevents this.

The first is the problem of creating one domain of heterogeneity 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 federation, 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 relative not only by 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). 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).[1]

Table 2.2: 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	Client/server	Payment
Function	Client/server	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

2.3 Services

Since the ultimate aim of this document is to address the co-operation that enables (or prevents) interaction with services it is important to establish what may constitute a service in the document, and, in particular, to relate it to a basic client-server model of interaction. Because an important use of this

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.

term will be to characterize what is provided in an unfamiliar domain, it needs to be defined in as broad a fashion as possible.

A service represents some set of functions that may be achieved, 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 functions to be exercised.

When using ANSA designs in which services of relevance are provided entirely by single interfaces to servers can be anticipated (although not universally).

More broadly, in terms of a client-server model, a service could be provided (for example):

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

Note: (Gray) Show this in a diagram?

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 2.3 provides some examples of services.

Table 2.3: Example services

Service	Points of access	Access method
Trading service	trader interface	RPC
OSI managed object	CMIS application service object	CMIP
Telephony	telephones	physical manipulation of telephone “dial”, voice and hearing
OSI Transport	local Transport SAP	endsystem requests, indications, responses and confirms,
POSIX operating system	libraries	external procedure call
WIMP program	keyboard, mouse, screen	WIMP

2.3.1 Service Access

In a 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 the function names and server interfaces that provide the service;

2. (client/server level) for each server, identification of the server-specific platform-supplied access options selected; and,
3. (infrastructure level) identification of the infrastructure implementation-dependant 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 client/server infrastructure.

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

2.3.2 Client/Servers

Note: (Gray) We need a better, single, term for this concept ("[computational] object"?)

Two styles of interface that these objects can support are referred to in this document:

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 litteraly describe the relationship between a client and a server too.

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

3 Classification of Boundaries

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.

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 between them are likely. Such properties thus 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 or disable the crossing of these boundaries (relevant to federation and isolation).

Systems will have to incorporate facilities for crossing these boundaries where this is feasible and permissible or may have to prevent it where it would lead to failure or breach of security.

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 to develop solutions to those problems.

3.1 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 and payment);
- service semantics (modelling and naming);
- client/server interfaces and properties; and,
- infrastructure (engineering and technology).

Almost always there will be differences in the information, procedures and mechanisms used to manifest each of these properties.

Note: (Gray) I have placed remuneration and client/server 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.

The client/server interfaces property encompasses both operational [BIRRELL?] and stream-based [Gray: ODP-3?] interfaces. Increasingly, distributed systems are the result of development processes that are

themselves carried out in a distributed fashion. This is referred to as the distributed development process and recognize that these processes may differ by:

- 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 between systems is a list of the following distributed system properties:

Table 3.1: Relationship between entities

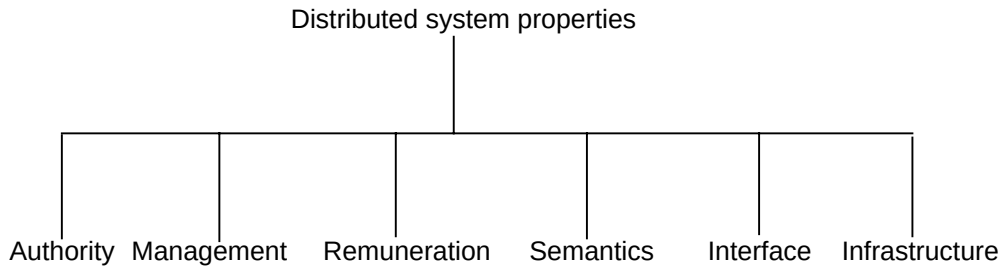
Entity	Relationship
Authority	institutional framework – agreement
Management	administration
remuneration	accounting, billing, payment
Semantics	models and naming
Interfaces	client/server service provision and consumption
Infrastructure	engineering and technical

Table 3.2: Process Relationship between processes

Process	Relationship
Development	object description agreement

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

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 the differences but also the identification of homogeneity, where it exists inasfar as the properties account for them.

As an example of the use of this classification: if it is known that two systems have a single authority and management structure/policy, is reasonable to concentrate on the other properties where differences may lie. Many assumptions made in the past can be explained by using this list. 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 the authority and management for example, are likely to continue to present obstacles to federation, as they reflect organizational and not technical variety.

3.2 Authority

An authority is a decision making body primarily concerned with issues of creation, ownership and control. The authority of a distributed system will determine the approach taken concerning all the issues where options are available. Authorities will vary on the nature of the agreement which it will regard as acceptable with another authority. Also, different authorities may have different philosophies/policies on issues such as 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 classification of this chapter represents one way of classifying these issues.

3.3 Management and administration

An administration is a body of people and facilities that manages operations, subject to the constraints and policies laid out by an appropriate authority. 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 client/server 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.

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 infrastructure resources which 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:

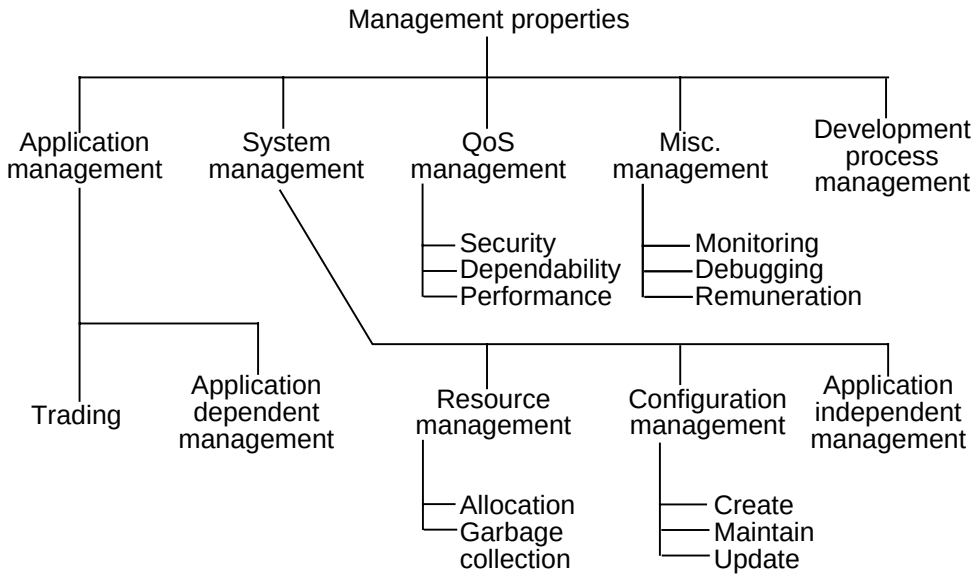
  this concerns the provision of end-to-end QoS guarantees when service provision crosses between different systems. This requires the appropriate policies, mechanisms and procedures addressing:

  — security;

  — dependability; and,

  — performance.

Figure 3.2: Management properties



- Application management:
  - — trading; and,
  - — application specific management.
- Development process management:

  this concerns the monitoring and control of the distributed development process.
- Miscellaneous management functions:
  - — monitoring; and
  - — debugging.

3.4 Remuneration

Remuneration includes three different processes (Figure 3.3):

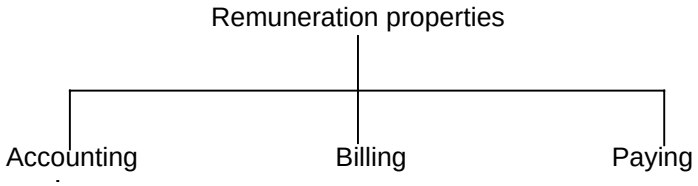
- accounting – measuring the amount of work carried out by a server and calculating the price of the service;
- billing – charging the user for services provided; and,
- paying – transferring some currency from the user to the provider of the service. This process may include sending a receipt from the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

Remuneration is closely related to issues of trust and security.

3.5 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

Figure 3.3: Remuneration properties



used to refer to it. 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., of service specifications.

3.6 Client/server interfaces and properties

Note: (Gray) need to make it plain that both operational and stream based interface use are relevant

The functional and non-functional specifications of objects have to be available in a form which allows the trading process to compare them and to find compatible objects. Object descriptions can be broken down into a number categories as follows (Figure 3.4).

1. Semantic description: a behavioural description (described in section 3.5).

   This may be expressed at many different levels of abstraction and may be either descriptive (e.g. a behavioural requirement specification) or prescriptive (e.g. an object's behavioural representation in a programming language).

2. Type specification (signature): description of interfaces in terms of operations 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].

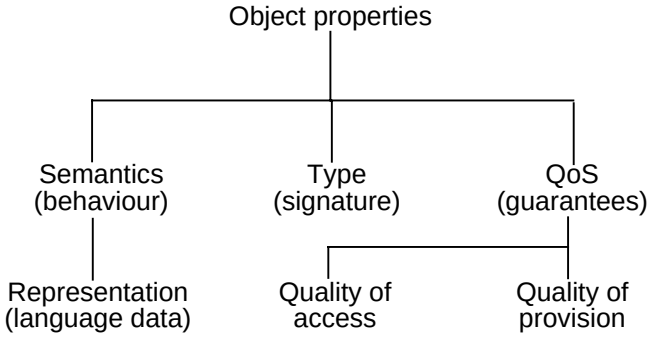
3. Quality of service (QoS) description: description of non-functional requirements such as security, performance, and dependability.

   The perceived QoS associated with an object 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).

Note: (Gray) The term "transparency mechanism" can be understood either to be a mechanism to achieve one of the ODP transparencies (of which there is a small fixed set) or it could be understood to be any Quality of Access mechanisms. If the latter is considered to be a more general interpretation of the term perhaps we should use "Quality of Access mechanism"?

Note: (Gray) QoS appears to include the specification of non-functional semantics of an object. That is it incorporates a subset of what is proposed in 1. Perhaps we need to limit 1. to "functional semantics" and 3. to "non-functional semantics"?

Figure 3.4: Client/server properties



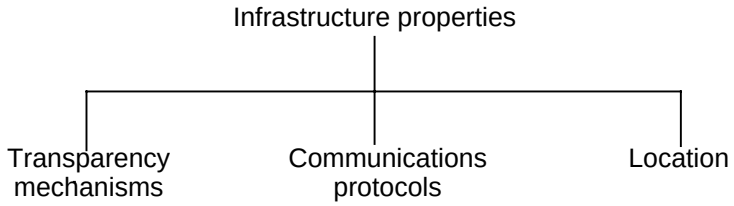
3.7 Infrastructure

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 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 3.5):

- transparency mechanisms;
- communications protocols; and,
- location.

Figure 3.5: Infrastructure properties



Transparency mechanisms and communications protocols are closely related to the quality of service issue.

3.8 Approach to using the distributed system properties

Two questions arise with regard to the differences between distributed systems and should be answered in the subsequent studies on the subject:

1. What is the operational impact of each type of difference when planning, designing, implementing, installing or invoking services which span different systems?

2. When is it possible and what has to be done to bridge the differences between distributed systems?

4 Relationships across Different Types of Boundary

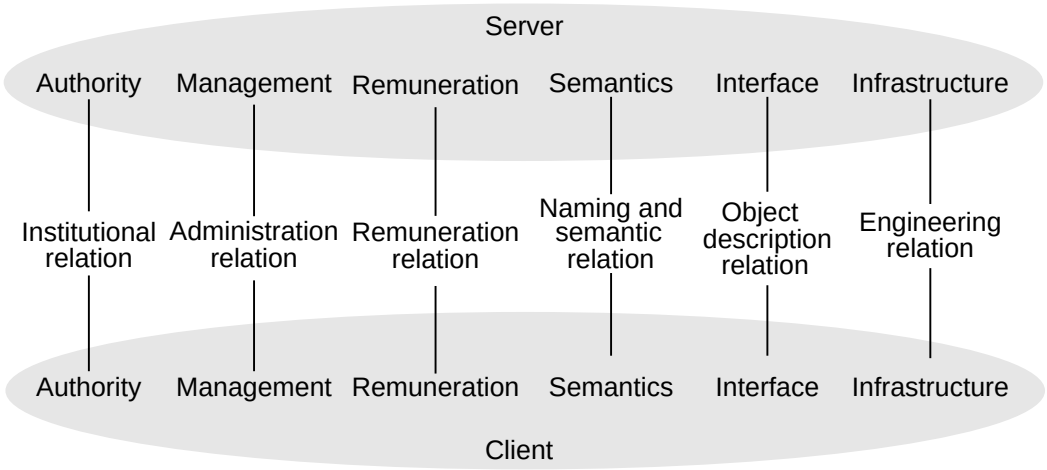
The following sections describe the boundary problems that each of the properties of distributed systems (as shown in Figures 3.1-3.5) give rise to when considered independently.

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 two distributed systems entities (Figure 4.1).

Figure 4.1: The relations between properties of two distributed systems



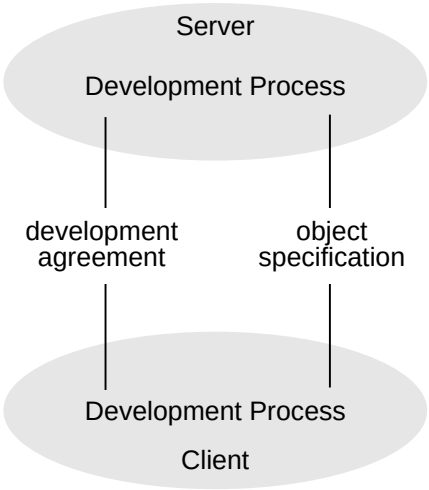
Note: (Gray) By talking about the relation between a client and a server this diagram avoids discussion about the relation between a service and its user (assuming that not all services are implemented on a single server).

Note: (Yigal) Feature interaction??? – (Gray: i.e. issue of section 2.2.4?)

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 processes will have to communicate at some epoch(s) and exchange these specification in order to ensure the co-operation between objects. Thus the relationship between two development processes consists of agreement and object description (Figure 4.2). The relationship

Figure 4.2: The relations between two development processes



between development processes is based on the description of the objects being developed.

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

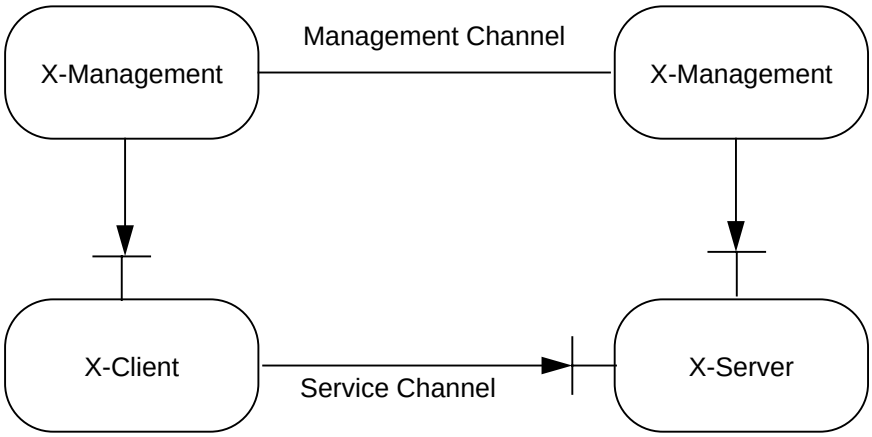
In each one of these areas the differences between administrations may manifest itself in different:

- management policy;
- 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 so that these systems can be compared and 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:

Note: (Gray) further text needed here:

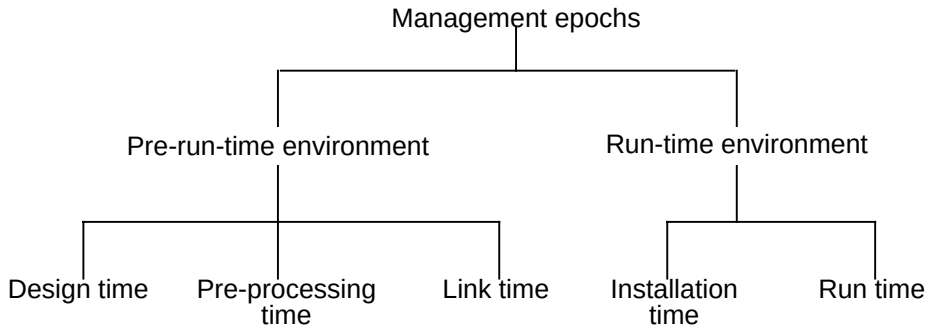
- Monitoring across a management boundary:
- Failure management across boundaries: what happens if a client and server from two different systems co-operate whilst a failure occurs in one of them. What if the idea of dealing with failure is contradictory?

Note: (Yigal) roll back versus transactions

Note: (Gray) 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 three processes identified in Section 3.4. 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 trading to succeed it is necessary to have the relevant information concerning remuneration.

4.5 The naming and semantic relation

Differences in semantics, where two domains do not share the same understanding of something, result in fundamental barriers to interaction.

Differences in the syntax associated with the same semantics result in identical or similar concepts being represented in fundamentally different ways.

4.5.1 Service specification

An example of a difference in semantics between two services: 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.

Semantic differences will range from entirely different applications to ones that deal with the same problems but may have a slightly different model of the problem. In such cases it is possible to imagine “semantic interceptors” that convert invocations suitable to the semantics and syntax in domain A to a form which is suitable to the semantics and syntax in domain B.

A 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 incomprehensible to the client.

One way to make information defining semantics available is to make it part of the service specification that traders maintain. 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.

4.5.2 Transfer of representations and naming

Production of a representation of something in one context (e.g., in representation domain A: the current time, a specification of a service, or the name of a file) and the transfer of the representation to another context (e.g., representation domain B) does not necessarily convey the meaning attached to the time, specification or file name. The two contexts may have different understandings of the representation used.

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.

Note: (Gray) the case where the semantics are the same and the "syntax" is different is not an example of a semantic boundary, it's an example of a syntax or representation boundary. As such it doesn't really belong here (although it probably needs to go somewhere).

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.

4.5.3 Naming models

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 of 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 The object description relation (interfaces)

Note: note Gray's comment about difference between objects and interfaces

A client/server, or object, description can be broken down as shown in Figure 3.4. Semantic differences are discussed in Section 4.5.

The signature of an interface is described in terms of the types of interface operations and data structures [AR.01]. A number of type systems that differ in the way interfaces are defined and described are likely. Notions of access compatibility are also part of the type system.

Quality of service is a measure of the quality attributed to or required of a service or activity. Major areas which are 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), and are thus addressed in Section 4.7. However, the translation between quality of service requirement and is part of the development process which is expected to be automated by tools as much as possible.

4.7 The engineering relation (infrastructure)

Several infrastructures exist and problems arise when:

- the functionality and/or quality of different infrastructure domains differ; or,
- the interface to the infrastructure functionality varies (link to Abstract and Automate [ref.]).

The differences between distributed system infrastructures that concern interoperability can be broken down according to Figure 3.5:

- Access QoS mechanisms: declarative QoS 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 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.

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 functionality may incorrectly appear compatible.

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

meaningfully be exchanged. 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”.

Note: (Gray) Are the mechanisms that we'll propose for integrating otherwise incompatible infrastructures (given the arbitrary range of functionality that they may encompass) actually any different from the mechanisms that we'll propose for the integration of large scale systems with arbitrary semantics? If not, there is a case for ceasing consideration of this class of problem here and referring instead to the more general case covered by the other sections in this chapter.

5 Dealing with Different Classes of Boundary

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.

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

5.1 Authority boundaries

Note: Text needed

5.2 Management boundaries

Note: Text needed

5.3 Remuneration boundaries

Note: Text needed

5.4 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.

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.

5.4.1 Common semantic subset

Note: Text needed

5.4.2 Extended common semantic subset

Note: Text needed

5.5 Client-Service 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.

5.5.1 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

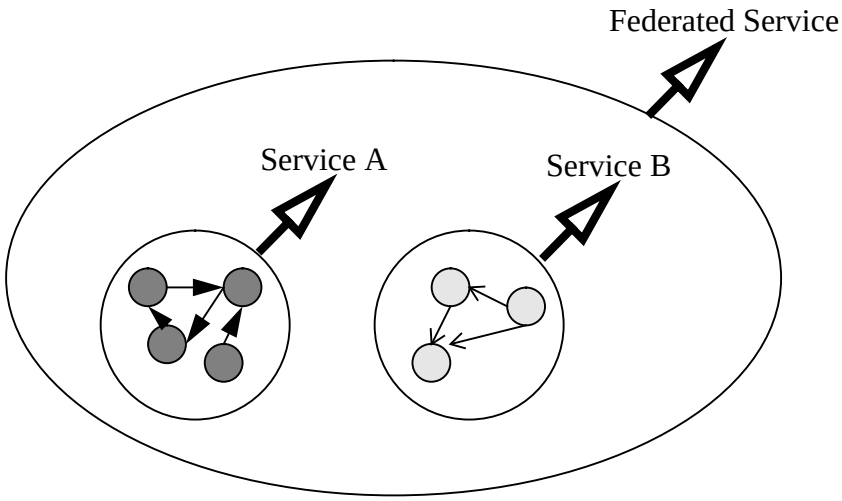
Strategy 1 requires only access differences to be resolved (e.g. by insertion of mechanism in the access path). Strategy 2 requires configuration differences to be addressed too and, if there was no similarity in their design may not be possible.

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

5.5.1.1 *Service interface composition*

This is illustrated in Figure 5.1;

Figure 5.1: Encapsulated Service Federation

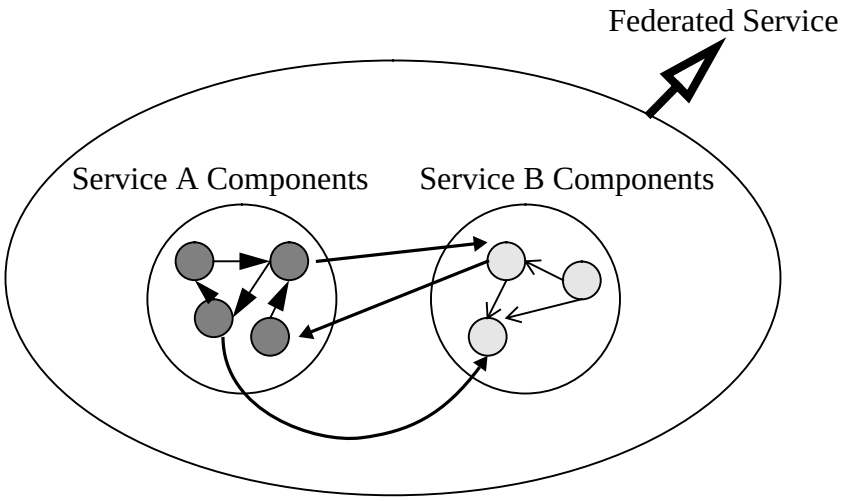


Note: more text required

5.5.1.2 *Service mechanism composition*

This is illustrated in Figure 5.2.

Figure 5.2: Amalgamated Service Federation



5.5.1.3 *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.

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.

5.5.2 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 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

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.

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.

5.6 Infrastructure boundaries

Note: Text needed

6 Infrastructure Supporting Federation

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.

6.1 Wrappers

Note: Text needed

6.2 Interceptors

Note: Text needed

6.3 Dynamic configuration

Note: Text needed

6.4 Trading support

Note: Text needed

6.5 Summary

Federation and independence 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 independence during the development or operational use of distributed computer based services.

7 APPENDIX

7.1 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 AR.05].

Table 7.1: Information for co-operation and boundaries

Information	Boundary type (AR.05)
Semantics	Properties
Type (signature)	Type system
QoS	Security, performance, dependability
Transparency mechanisms	Engineering
Communications protocols	Interconnection

7.2 ---------- 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

[ANSA 91]

ANSA: A Systems Designer's Introduction to the Architecture, APM Ltd., Cambridge U.K., April 1991.

[APM1003.1 93]

Van der Linden, R.J., **The ANSA Naming Model**, 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.