Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

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

ANSA Phase III

Federation and Interoperability

Yigal Hoffner and Ben Crawford

Abstract

The need to exploit new markets as they are opened up by new technology, combined with greater competition and greater market accessibility, are causing business processes to change at an ever-increasing rate.

To survive in this environment, organisations must become capable of rapid and flexible reorganisation, and must be ready at very short notice to trade in new markets with different suppliers. In order to achieve this, their information systems must be able to accommodate diversity in technology, administrative policies and procedures, and incompatibilities between applications.

This document presents a clear and comprehensive model of interoperability issues. It is intended to help anyone wishing to understand the general issues raised by interoperability requirements, and to provide strategic guidance. It also provides context and expansion of the issues contained in the OMG Interoperability proposal and contains much information of practical benefit to platform designers and system integrators.

APM.1514.01 **Approved** 6th October 1995

Architecture Report

Distribution:
Supersedes:
Superseded by:

Federation and Interoperability

Federation and Interoperability

Yigal Hoffner and Ben Crawford

APM.1514.01

6th October 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

1.1 The business problem

For organisations to survive in todays competitive environment of rapidly changing technology, they must:

- be capable of rapid and flexible reorganisation, both internally and in terms of creation and modification of the relationships between businesses [HAMMER & CHAMPY 93]
- be ready at very short notice to trade in new markets with different organisations, across political and cultural boundaries.

Consequently, their IT systems must:

- enable cheap and reliable integration of existing and new systems to preserve investment and business stability
- support rapid interconnection with IT systems of other organisations
- support rapid interconnection of diverse applications to enable new and changing business processes
- interwork across national and organisational boundaries in worldwide distributed systems.

To satisfy these requirements, systems must be able to interoperate.

1.2 Interoperability

Interoperability is concerned with co-operation between different IT systems which may service different organizations. In the context of the business problem outlined above, this requires IT systems to be able to:

- facilitate the interaction between different systems where this becomes desirable
- restrict or prevent interaction between them where this is or becomes undesirable
- audit the interactions between different systems.

Differences among systems can be classified by whether they are administrative or technological in nature.

Administrative boundaries demarcate the authorities in charge of systems, reflecting different management policies and issues such as ownership, trust, fiscal policy and legal framework. These boundaries do not necessarily coincide with technical boundaries and must therefore be erected where necessary. Administrative boundaries are where the vetting of interactions can take place. It is also where monitoring takes place for auditing, billing and accounting purposes.

Technology boundaries are caused by differences which prevent safe interaction, such as those found between different distributed platforms such as CORBA, DCE and ANSAware. Such differences have to be overcome where interaction between them is desirable.

Interoperable information systems must be able to co-operate across both administrative and technical boundaries. The technical aspects of diversity are currently of greatest relevance to most organisations, particularly those involving legacy systems. Once these have been addressed, increased interest in administrative issues and application incompatibilities is anticipated.

Technical boundaries can be bridged by agreement on common protocols, such as the CORBA Universal Network Object (UNO) proposal, or by the use of gateways (also called transformers, bridges, wrappers or fire-walls) which perform the required transformations.

Although the main focus of current interoperability work is the development of common protocols to overcome technological differences, common protocols do not offer a solution for administrative boundaries. However, our experience indicates that techniques and technologies which are needed to manage technical boundaries can be applied equally well to administrative boundaries.

The most important technique introduced by this document is the use of interception at domain boundaries.

1.3 The solution: Domains and Interception

Domains address the encapsulation of areas which are homogeneous as a result of a common management policy, and the ability to make a distinction between interaction inside and outside a domain. Domains offer a structured way of benefiting from homogeneity and of dealing with heterogeneity.

Interception is the process which creates and inserts the appropriate gateways when a binding between a client and a server is created across domain boundaries. One of the major problems of interception concerns how to propagate the necessary gateways to subsequent bindings across the same boundaries. The inserted gateways can perform the required transformations in the case of technical differences, checking in cases where administrative boundaries are present, and monitoring where auditing is required.

1.4 This document

1.4.1 Audience

This document presents a comprehensive model of interoperability issues. It is intended to help anyone wishing to understand the general issues raised by interoperability requirements, and to provide strategic guidance as well as information of practical benefit to platform designers and system integrators.

The document should be of use to:

- designers and developers of distributed platforms and applications
- platform and application integrators
- standards bodies concerned with the distributed systems arena
- system managers and administrators.

1.4.2 Aims

The aims of this document are:

- to examine the implications of interoperability for distributed platform design
- to be a summary of the federation work done to date by the ANSA team
- to provide context, explanation and expansion of the issues addressed in the OMG interoperability (UNO) proposal [UNO 95].

1.4.3 Content

This document:

- describes a process by which co-operation between different systems can be defined and agreed
- defines a policy-independent framework for solving specific federation problems
- discusses issues which should be taken into consideration when developing a corporate strategy for managing diversity
- classifies and explain the differences encountered in heterogeneous federations, and discuss what must be done to overcome them
- explains the design and implementation options and trade-off available to system integrators producing interoperability solutions.

The structure of the document is discussed in more detail in the overview (§2 *Overview*).

1.4.4 Background material

Some familiarity with the following is a prerequisite for this document:

- ODP [ODP 93] computational and engineering models
- OMG CORBA specification [OMG 92]
- ORB Interoperability RFP submission [UNO 95].

OMG terminology is followed where possible, The main exception is the term “interface” which is used here with the ODP connotation of “an access point to a set of operations provided by an object”, rather than the OMG connotation of a “type”. Each interface has a type; there may be several interfaces of the same type in a typical system.

1.4.5 Future work

Work in the following areas is envisaged:

- multi-party federation
- integration of federation architecture with management and security architectures

1.4.6 Validation

Aspects of this work have been prototyped in CORBA-ANSAware gateways [APM.1303 95] and WWW-CORBA gateways [REES et al. 95].

1.5 Relation to other documents

This document supersedes “ORB Inter-operability” [APM.1021 93].

A shorter and more concise discussion of the ANSA interception model with particular emphasis on the affect it has on distributed platform design can be found in “Interoperability and Distributed Platform Design” [APM.1507 95].

More information about design and implementation issues for gateways is given along with illustrative case studies in “Gateway Design and Implementation” [APM.1303 95].

1.6 Acknowledgements

The authors would like to acknowledge the contributions of the members of the ANSA Federation team over a number of years.

They would also like to thank the following people for their input and help: Professor Peter Linington and Chris Scott from the University of Kent at Canterbury, Nic Holt and Tony Drahota from ICL (UK) Ltd., and David Iggulden.

Many thanks also to Dave Otway, Mike Beasley, Mark Madsen, Nigel Edwards, Rob van der Linden and Andrew Herbert from the ANSA core team at APM in Cambridge, UK, who reviewed and commented on the document.

2 Overview

This chapter summarises the issues addressed by this document. It is intended to help the reader to determine which areas may be of interest, and to clarify the structure of the document.

2.1 Document summary

In order to set up co-operation between organizations and their information systems, the parties must define and agree a relationship and then undertake to maintain it. In commercial environments, this relationship often takes the form of a contract or at least a memorandum of understanding.

The process of reaching agreement can be difficult because the parties differ in ways such as the technology used, administrative policies, management procedures, security policies, fault models, performance guarantees.

In order to manage these differences it is convenient to treat each pocket of homogeneity as a domain - an area of similarity which exists by the influence of an authority and the adherence to its policies.

When information crosses domain boundaries it may be necessary to transform it; different kinds of domains and boundaries require different kinds of transformations to be applied. These transformations are carried out by gateways or bridges.

In distributed systems, links between objects are created, used and discarded dynamically. Where such links are created across domain boundaries, the relationship defined between the domains has to be propagated to all the created links. To achieve this, gateways may be used as interceptors of the information used to set up links across the domain boundary and as creators of subsequent gateways.

There are several different approaches to the creation and placing of gateways which offer designers and system integrators different options for resource allocation and QoS.

To achieve transparent interoperation, gateways must be managed as part of the infrastructure, and this has implications for the design of distributed platforms.

2.2 Document overview

2.2.1 Chapter 3: Co-operation between systems

To achieve co-operation between systems, after some initial contact, agreements must be made, contracts negotiated, and interfaces defined, created and made available. Any incompatibility between the systems must be overcome. The process of exchanging information to achieve these goals must

occur, no matter whether it is performed inside or outside computer systems. This process and the activities within it are described in §3 *Co-operation between systems*.

2.2.2 Chapter 4: Heterogeneity and incompatibility

Much of the effort required to achieve co-operation is due to incompatibilities between the systems, which must be overcome. Classification and description of these incompatibilities helps to facilitate the management of diversity, and provides a context for the separation of issues in both standards and software (§4 *Heterogeneity and incompatibility*).

2.2.3 Chapters 5-6: Domains, boundaries and transformations

A domain is an area of homogeneity in one or more respects; these may be technical, business related or administrative. A set of autonomous domains that have established a contract for co-operation between themselves constitutes a federation (§5 *Domains and boundaries*).

To manage diversity, there is a need to bound domains and manage them; there is also a need to manage the crossing of domain boundaries in accordance with the policies of the domains and the differences between them.

Information has to be intercepted and transformed as it is passed across domain boundaries; knowledge of these transformations helps designers to characterise the mechanisms needed to make systems interwork, and helps in estimations of effort and feasibility (§6 *Transformations at domain boundaries*).

2.2.4 Chapter 7: Application level transformations and gateways

The transformations required at domain boundaries can be implemented using gateways. It is often preferable to implement many of these transformations in terms of application models. Gateways are also a convenient place to provide other facilities such as monitoring and the enforcement of administrative policies (§7 *Application level transformations and gateways*).

2.2.5 Chapter 8: Using gateways to manage domain relationships

Once contracts have been agreed, gateways must be put in place between the co-operating systems to ensure that the agreed **relationship** is maintained. The agreed relationships must be propagated from one binding to another by the appropriate creation and configuration of gateways (§8 *Using gateways to manage inter-domain relationships*).

2.2.6 Chapters 9-12: Using interception to manage transformation of information

Information requiring some action to make it compatible with the system on the other side of a boundary needs to be recognized. Some actions may then be performed on the information to make it compatible. Such actions may be performed at different times, by different agents and using information held in different places (§9 *The phases of the Interception process*).

Dividing the interception process into phases allows the development of a range of different strategies for intercepting information and acting on it. The description of interception identifies these strategies (§10 *Interception resolution strategies*).

Interface references, or object references, are a case of special interest, because exchange of interface references is a prerequisite for cooperation. Interface references passing is used to illustrate the interception process (§11 *Passing interface references through domain boundaries*).

A structured approach to management of non-functional trade-off is offered (§12 *Interception implementation issues*).

2.2.7 Chapters 13-14: Binding and interface references

In order to support different interception strategies, enhancements to the binding process are required (§13 *Binding*).

It is important to consider the requirements which interoperable interface references should meet, and agree what aspects should be standardised and what may remain diverse. The CORBA interoperable object reference is a good start, but further refinements are needed (§14 *Interface references and interception*).

3 Co-operation between systems

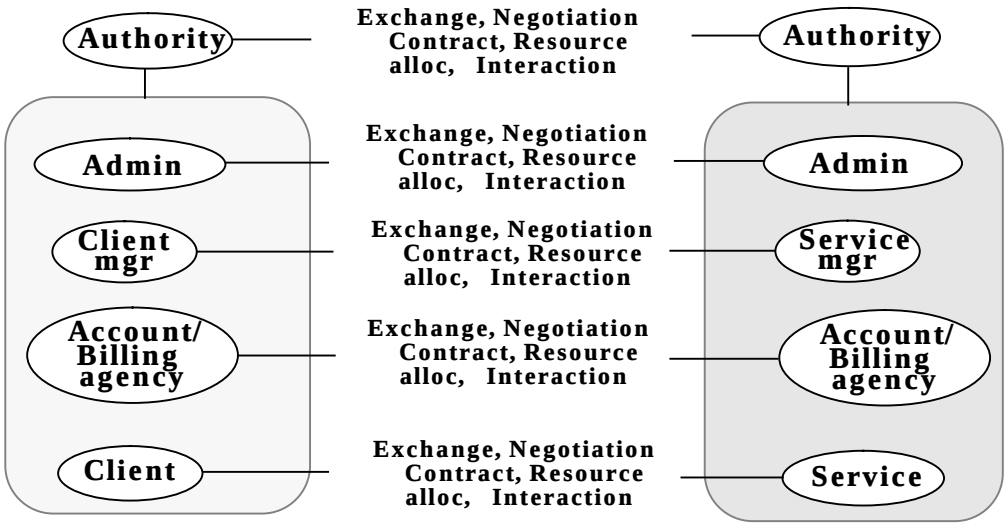
Protocols are needed for agreeing the relationships between systems. A single such protocol is unlikely to suffice for all situations, because many factors may vary, such as the set of things which must be agreed on and the way in which the relevant information is distributed among the agents. This chapter discusses the issues which should be considered when a specific protocol is to be developed.

A set of processes for reaching agreement are described. In different contexts, these may be conducted in different orders and by different agents, either inside or outside the system, and some of the processes may be null. Co-operation is described in the context of two systems but could be generalised to any number.

3.1 Processes necessary to establish co-operation

The processes necessary to establish co-operation between two systems are shown in Figure 3.3.

Figure 3.1: The processes necessary to establish co-operation between two different systems



The processes involved in co-operation set up are described below. Note that in the following processes the client and server can be any two objects such as the authorities or managers, not just the client and service shown in Figure 3.3.

- ***Trading*** *(broking)* is primarily concerned with:
  - clients finding out sufficient information about servers to provide an initial point of contact from which further exchange of information and negotiation can take place

— preliminary decisions about benefit to the parties involved. Trading can involve different levels of match-making depending on he information provided by the client and server, and should preferably be based on conformance or substitutability of types and properties

- ***Exchange of information*** is a two way dialogue concerning:

  — what the server offers to the client and what it requires from a client

  — what the client requires from the server and what the client offers as a service consumer

  — what happens if things go wrong because of failure or a decision not to continue to cooperate

  This exchange of information may supplement the information supplied during an earlier process of trading, for example in cases where it is not appropriate to advertise such information publicly. Also, it allows the servers to find out more about the clients, for example, their identity, credentials and ability to pay

- ***Negotiation***: the process of reaching an agreement acceptable to both parties concerning service provision, consumption, remuneration and any other issues which may effect their co-operation.

  Automating negotiation is a complicated task and human intervention may be required. Negotiation is often simplified to selection from a range of options presented in the information exchange stage. Negotiation aims to finish with a contract or with an agreement to cease interaction

- **Agreement/Contract**: a bilateral statement of promises made by the parties to the contract. At its simplest one party agrees to do something and in return the other party promises to pay, for example. The idea of a contract can be generalised to consider more than two parties and will generally include statements about duration, liability and arbitration in the case of disputes.

  A contract defines the relationship which the entities (usually authorities) signing it undertake to uphold. All subsequent interactions are affected by the contract

- ***Resource allocation (Binding)***: setting up the binding between the client and server requires the allocation of resources in order to provide the required quality of service

- ***Interaction*** or ***Service provision and consumption***: delivery and acceptance of services. In client server terms this is when the client invokes the server

- ***Remuneration***: it is necessary to set up the links between the remuneration agents. These measure the amount of work carried out, calculate the payment due, issue the bill, pay and send of the receipt

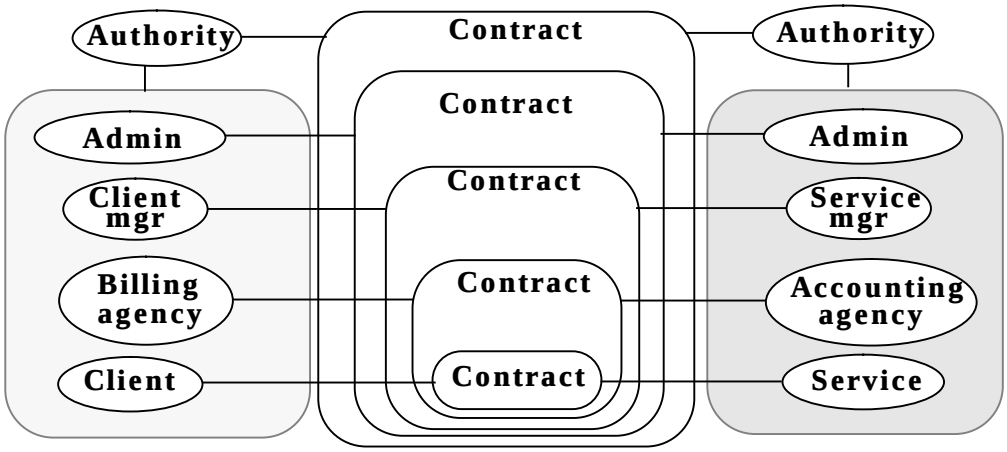
- ***Termination of co-operation***: dismantling of the binding and the resources associated with it (garbage collection).

-

3.2 Agreement/Contractual relationship

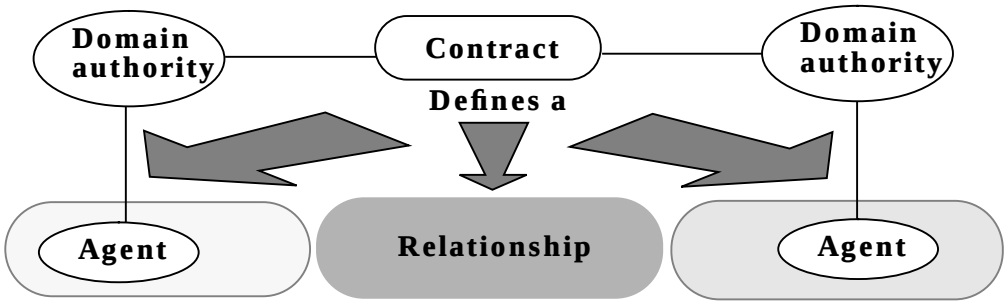
A model of the contractual relationship resulting from the processes described in section 3.1 is shown in Figure 3.2.

Figure 3.2: The contract defines the relationship between the systems



Note that although conceptually the negotiations and contracts are carried out by different objects in Figure 3.2, in practice they may be carried out entirely by the authorities or the managers as shown in Figure 3.3.

Figure 3.3: The relationship between agents is defined by the contract



3.3 Scenarios of co-operation set up

The steps necessary to ensure that effective (i.e. purposeful, meaningful and successful) co-operation between objects can take place can be described by different scenarios. The following are two such possible scenarios.

3.3.1 Scenario 1

This scenario assumes that an introduction between the authorities of the two systems forms the necessary preliminary step to any subsequent interaction between objects in the two systems.

This scenario sequence is:

1. the **authorities** of the system are introduced to each other
2. the system authorities can now exchange information and negotiate a contract for the co-operation between the two systems

3. the contract describes a relationship both systems agree to adhere to and uphold in all subsequent interactions between objects in the two systems
4. the contract is then given to the **system administrators** who are expected to fulfil it within the remit of the system policy laid out by the system authority
5. the system administrators establish a link between **client** and **service managers**
6. the client and service managers provide the **client** and the **server** with the information necessary to establish a binding between them
7. the client and the server establish the binding and set up the **accounting** and **billing agents** for dealing with remuneration
8. the client can then proceed with consuming the service provided by the server.

If the **client** and **server** in this scenario are **traders**, then the scenario establishes co-operation between them, enabling clients and servers of the two systems to find out about each other.

3.3.2 Scenario 2

This scenario assumes that the client obtained information about a potential server either through explicit trading or through a third party.

This scenario sequence is:

1. the **client** gets hold of an interface reference for the **server**
2. the client asks for an interface reference for the **service manager** and passes the result to its **client manager**
3. the client manager asks the server management for a reference to its **authority**. The client management then passes this interface reference to its authority
4. the client and server **authorities** exchange information about themselves and the two systems, and negotiate a contract
5. once a contract is established, the authorities instruct their respective **client** and **service managers** to proceed in accordance with the contract and the system policy
6. the managements can set up the appropriate **accounting** and **billing agents** for dealing with remuneration, and then instruct the **client** and **server** to proceed.

3.3.3 Issues concerning scenarios

The following issues arise from the two scenarios:

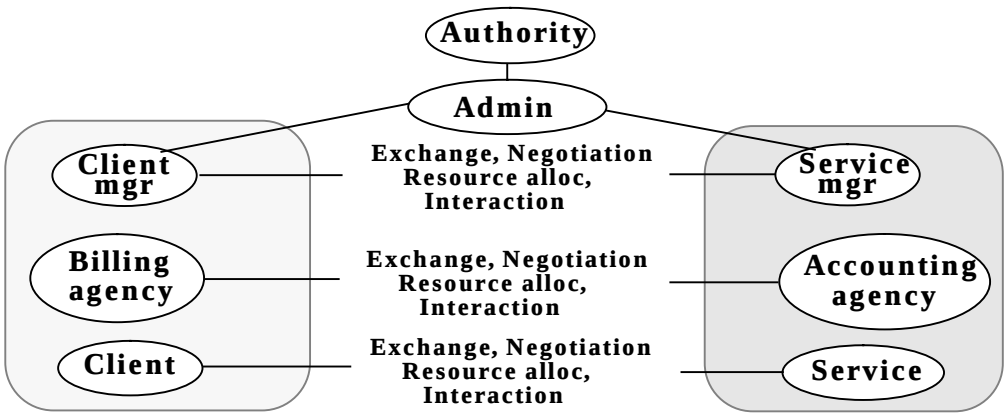
- additional variants of the above scenarios are possible
- the processes involved may take place at different times. For example, when setting up the binding between the client and server, the binding managers may communicate with each other. It is thus possible to have a more specialized and detailed information exchange and negotiation dialogues at different stages of overall process
- the functionality of some agents (e.g. authority, management, client/server managers, accounting and billing) may be embedded in other objects, or

completely bypassed where inappropriate. For example, in simple applications, the client or service managers may not actually exist and the client and server will manage themselves. In some applications, there is no payment required and therefore it is not necessary to set up the remuneration configuration

3.4 Hierarchy and Federation

Where two systems have a common authority, the situation described in Figure 3.3 and section 3.1 can be simplified into a hierarchical configuration where a single administration exists (Figure 3.3). Variants of this configuration are possible.

Figure 3.4: The relationship between two different systems



The negotiations between client and server managers, the client and server and the remuneration facilities may be subsumed in the policy laid out by the authority. They may therefore not take place explicitly.

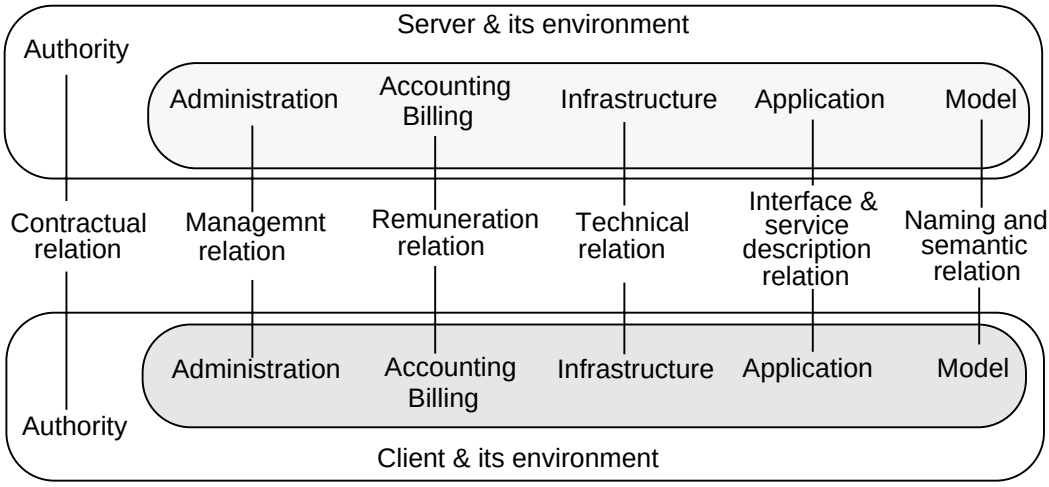
4 Heterogeneity and incompatibility

A variety of incompatibilities may be found between organizations and information systems. The classification of differences given in this chapter can be used as a checklist to help ensure that all the different aspects of the organisations and systems being interconnected are analysed for incompatibilities. It also suggests a structure for specifications and contracts.

4.1 Differences between systems

The relationship between organizations and their information systems is shown in Figure 4.1. This highlights the areas where incompatibility between them is likely to occur. **Almost always there will be differences in the information, procedures and mechanisms used in each one of those areas in different systems**.

Figure 4.1: The relationship between organizations and their systems - possible areas of incompatibility



4.1.1 Authority and contracts

In a federated system there are multiple authorities, each responsible for different parts of the federated system. The contractual relationship between the authorities will:

- reflect the institutional and legal context in which the contract between them is made, for example, whether international or local law has to be referred to, whether the agreement is between some union members or not (e.g. EEC, commonwealth, etc.)
- reflect the relationship the authorities are undertaking with respect to their organizations/systems. This can be expressed in terms of the areas discussed in the following sections: Administrations and management,

Accounting, billing and remuneration, Infrastructures and technical, Applications and service description and Models, semantics and naming.

4.1.2 Administrations and management

An administration is a body of people and facilities which manages operations, subject to the constraints and policies laid out by its authority.

In a federated system there are multiple administrations, responsible to the multiple authorities. In a federation, each administration is responsible for achieving what was undertaken in the contract between the relevant authorities and ensuring that guarantees given by the clients and servers are kept.

There are several areas where guarantees have to be met end-to-end and are therefore dependent on the combined behaviour of the client and server sides, as well as the communications between them and any other intermediate objects.

In those cases, the federated administrations will have to co-operate in order to achieve the combined effect. Therefore it is not only functional data which has to be exchanged between the client and server but also non-functional such as QoS/transparency, security, fault-tolerance, remuneration, configuration, and monitoring and debugging.

4.1.3 Accounting, billing and remuneration

Remuneration consists of four primary steps:

- *accounting*: measuring the amount of work carried out by a server and calculating the charge for the service
- *billing*: charging the user for services provided
- *paying*: transferring some currency from the user to the provider of the service
- *receipt sending* and *receiving*: exchanging proof of payment.

Different systems may have different views and implementations of each of these processes. In particular, conflicts may arise between systems which have different accounting and billing strategies.

An important issue raised by remuneration is that of accountability. This in turn raises the need for monitoring to be performed where interactions occur between different organizations, so that disputes can be resolved by an external arbitrator (e.g. a court of law).

4.1.4 Infrastructures and technical relation

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

Different infrastructures provide different facilities through different interfaces. We can classify the differences between such infrastructures according to the categories of information necessary for binding objects:

- *interface references*: DCE binding handle [OSF 92], ANSA interface reference [ARM 93], or CORBA object reference [OMG 92]

- *communications protocols*: different protocols are used, with different guarantees and features
- *RPC protocols*: e.g. DCE [OSF 92], CORBA based [OMG 92], ANSAware [ARM 93] RPC
- *transparency mechanisms*: declarative Quality of Service (QoS) requirements translate into the information, procedures and mechanisms necessary to provide the service with the required guarantees. Different systems will use different and often incompatible mechanisms.

4.1.5 Applications and service description

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

- *language of description:* what characteristics of services are described, and how different characteristics are named
- *interface type (signature)*: description of interfaces in terms of operations and data types, for example, with interface definition languages (IDLs) and abstract data types. Such descriptions can be used to check that the operations provided by a server meet the requirements of the client has the benefit of preventing interaction errors
- *quality of service (QoS)*: how to specify QoS requirements in a declarative manner which is translatable to a variety of platforms and mechanisms. Also, how to translate the mechanism specific information from one platform to another
- *semantics*: this is the hardest to achieve as describing semantics formally is still an open area of research.

4.1.6 Models, semantics and naming

Modelling concerns the way people describe the world around them. The major problem with semantics and naming in federated systems is that use of different models, different languages for describing models and different naming schemes, makes it difficult to compare names to discover whether the objects, or properties, denoted by those names are compatible. Differences in semantics result in identical or similar concepts being represented in fundamentally different ways, and conversely, in different things being represented in the same way.

It may be difficult to discover whether the semantics of an operation offered by a server is compatible with the semantics required by a client. This may be because a sufficiently complete semantic description is not available to the client, or it may be because the description is not understandable by the client.

Given these problems with semantic descriptions, it is likely that it will always be necessary to integrate automated compatibility checking with manual negotiation; for example:

- lack of semantic knowledge will cause automated checking to accept interfaces with correct syntax and incorrect semantics
- lack of information about incompatibilites will cause automated checking to reject cases where incompatibilities could be resolved through negotiation and information exchange.

5 Domains and boundaries

This chapter introduces terminology which is required in order to enable negotiation about differences, and to understand how they should be managed. A discussion of domain management is given, to encourage designers to avoid the design errors commonly introduced by the assumption that there is only a single authority with a single set of policies.

The multitude of potential differences between organizations and their systems produces different pockets of homogeneity which must be connected. Immediate questions arise from this:

- how are the relationships between these pockets to be modelled?
- how are links and the subsequent interactions between them to be constructed, maintained, managed and terminated?

5.1 Terminology

5.1.1 Domains

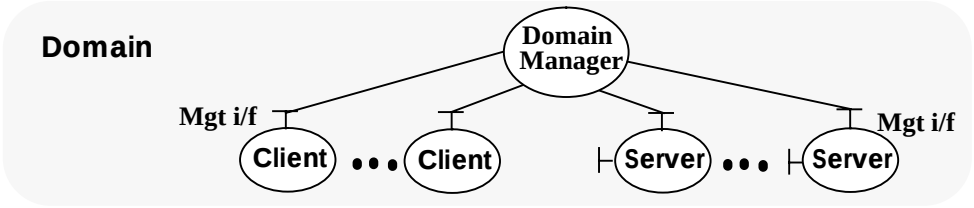
A ***domain*** comprises of a set of entities that are under a common authority and administration; this enforces some common policy on the members of the set. This results in a pocket of homogeneity with respect to the issues determined by the enforced policy. There may be areas of homogeneity which are not co-terminus with domains of authority and this may have implications concerning changes

5.1.1.1 Internal view of a domain

This view of a domain is in terms of the management of the entities which belong to or are in the domain (Figure 5.2). This is a provincial view of management and is concerned with:

- membership (joining and leaving): creation, destruction and migration
- resource allocation and binding to local resources
- object creation and destruction
- garbage collection
- other administrative procedures.

Figure 5.1: Internal view: a domain in terms of the management of its members

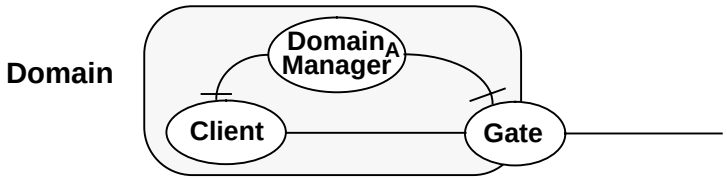


5.1.1.2 External view of a domain

This view of a domain is in terms of the management of the interactions between objects in the domain and in others (Figure 5.2). It takes into account the existence of other domains and the possibility of control of interactions between their members in terms of:

- trading across the domain boundary
- binding across the domain boundary
- interacting across the domain boundary.

Figure 5.2: Domain in terms of the management of external interactions



5.1.2 Boundaries

A ***logical boundary*** delineates a domain, i.e. that which defines the extent of the authorities' control (on the characteristics of the domain); beyond this, other domains exist which may or may not adhere to the same policies. A logical boundary is a delineation of a homogeneous environment, and thus of one or more common characteristics.

A ***physical boundary*** (border, fence, wall, firewall, encapsulation) is a physical implementation of a logical boundary and is concerned with the encapsulation of a domain. The setting up of physical boundaries is usually closely linked to the notion of protection and issues of trust.

Systems need to incorporate facilities for crossing these boundaries where this is desirable, and prevent it where it is undesirable.

A reason for distinguishing logical and physical boundaries is that a boundary may have no physical manifestation, or it may have one or more walls associated with it.

5.1.3 Paths and gateways

A ***gateway*** is a recognized and controlled opening in a physical boundary which can control and monitor information passing into or out of a domain.

Paths (*or bindings*) are access routes which lead to a physical boundary. Where encapsulation is used paths should lead to a gateway.

In computers, physical boundaries and gateways are almost invariably associated with communications mechanisms due to the fact that the boundaries between entities are often accompanied by physical separation, i.e. they manifest themselves by a subroutine, macro call, library procedure or remote object.

5.2 Model of interconnected domains

In the general case where autonomous domains are being interconnected, each domain has a gateway at its entry and exit points (Figure 5.3).

Figure 5.3: Interconnected domains and domain management

The benefits of this model are:

- *each domain retains control of interaction*: This enables each domain to check interactions and/or prevent interactions from crossing the domain boundary, carry out information transformations and perform monitoring
- *domains allocate their own resources*: Each domain retains authority over its own gateways and control of their creation and destruction, thereby retaining full control of how and when its resources are allocated and de-allocated
- *protecting domains from changes in other domains*: In cases where gateways are used as translators to overcome differences between the domains, the intermediate representation between the two domain gateways buffers changes in one domain from the other domain
- *O(n) versus O(n^2) translators*: By agreeing on intermediate forms for communication between domain gateways it is possible to reduce the number of translators which have to be created in order to interconnect multiple domains.

5.3 A classification of domain boundaries

This classification divides domains and boundaries into classes concerned with:

- technology differences
- administrative differences
- application differences.

5.3.1 Technology boundaries

Technology or **"Ability/Compatibility" boundaries** raise the question: ***can we interact safely?***

Technology boundaries deal primarily with issues such as:

- are we talking in the same language, dialogue structures and encoding
- do we talk about and do things the same way: issues of information modelling, policy and management procedures.

A **technology domain** is generally established by managers exercising a procurement policy or authorities controlling the use of a particular technology. A technology domain may include several diverse technologies or distributed platforms, such as DCE, ANSAware or CORBA. Within the domain, support for interaction and conversion across the platforms will be necessary. In a technology federation, neither party is prepared to (or has the ability to) introduce the other domain's technology into its own.

Examples of types of technology domains are:

- different distributed system platforms such as DCE [OSF 92], ANSAware [ARM 93], CORBA [OMG 92]
- communication domains: the variety of communication protocols create technology boundaries. For example, the RPC communication protocols of the different distributed system platforms
- data representation domains. Examples of such domains are:
  - language specific data representation: the way in which data is encoded in the system, for example, language bindings and problems such as little and big-endian ordering of bytes
  - interface reference domains: different structure and representation of information necessary for a client to be able to bind and invoke a server (e.g. ANSAware Interface references [ARM 93], DCE binding handles [OSF 92] and CORBA object pointers [OMG 92])
- quality of service (QoS) domains: where guarantees of performance differ it may be necessary to intercept any attempts to create bindings across such boundaries depending on the requirements of the client and server
- descriptive languages: type description languages such as IDLs. There are two possible differences among them:
  - syntactic: e.g. ANSAware versus CORBA IDL
  - semantic: e.g. CORBA versus DCE.

5.3.2 Administrative boundaries

Administrative or **"Permission and Management" boundaries** raise the question**: *do we want to interact, are we allowed to interact, and also how are we allowed to interact?***

An **administrative domain** consists of a set of objects whose security, accounting, monitoring and other management functions are under a single administration and are subject to policies enforced by the administration.

Administrations may wish to impose their own access controls for such purposes as security, accounting and monitoring, in addition to controls imposed by the objects themselves. Similarly, objects will want to add their own restrictions and conditions to those imposed by an administration.

Examples of administrative domains are:

- authority, permission and control:

  - where it is necessary to receive the permission of the management of the domain in order to interact with objects outside a domain
  - the right to create and destroy (resource allocation), to provide, charge or withhold a service
- resource management: areas of responsibility for such things as resource allocation and dependability guarantees
- security: where a common security policy is applied for issues such as authentication, secure communication channels and access controls
- remuneration: where differences exist in notions of measurement of work, the relation between work and pay, costing of effort, billing and payment strategies, receipt exchange procedures, and so on.

5.3.3 Application boundaries

Application boundaries raise questions:

- are we talking about the same things (semantic issues)
- do we know what we are interacting about.

Different applications are based on different models and may also describe such models in different ways. In order for applications to interact meaningfully, translation between different models and different languages of descriptions must occur.

6 Transformations at domain boundaries

The nature of the transformations required at different kinds of domain boundaries are examined in this chapter. This is intended to give support to designers and system integrators for determining the characteristics of transformation mechanisms and the information which they require. This knowledge is important in determining the feasibility and difficulty of establishing cooperation between systems.

6.1 Types of transformations carried at boundaries

In general, information can be considered in terms of three different aspects: representation, language and model.

1. The **representation** describes how information is physically encoded. For example, an integer may be represented using different byte ordering, or using a character to denote each digit.
2. The **language** describes how information is structured (its syntax), and how information types are composed from other information types. One example of such a language is an IDL; different IDLs may be used to describes the same interface in different ways.
3. The **model** describes the meaning of the information itself (its semantics), and the types used to represent it. As a very simple example, one domain might use a SignStatus with values {STOP, GO} to represent the state of a traffic signal, whilst another might use RedLightStatus, AmberLightStatus and GreenLightStatus with values {ON, OFF}. Movement of information from one model to another may involve alteration of both value and structure.

A transformation at a domain boundary may include alterations in any or all of these characteristics.

6.2 Scenarios for information crossing domain boundaries

6.2.1 Marshalling and unmarshalling

When information passes across a remote link, it is marshalled and unmarshalled to and from an agreed common representation, to overcome differences of concrete representations of data used by different environments.

6.2.2 Activation and passivation

Often, different media necessitate use of incompatible data formats, so the physical representation of the information must be translated as it is moved from one physical media to another.

6.2.3 Addition of information

It may be appropriate to add information items to a collection of information passing a domain boundary because:

- assumptions are made in the source domain that do not hold true in other domains, so information to describe the assumed values may be needed. Within a domain, characteristics such as the physical location may be assumed to remain constant and known, but must be added explicitly to information which is passed outside the domain boundary; for example, by adding headers to RPC messages
- the information cannot be represented, or cannot be used, in the target domain without further information from the source domain being added For example, where two instances on the source side of a boundary are modelled as one on the target side, it may be impossible to instantiate the instance on the target side unless information taken from both source instances is available. Consequently, whenever one of the source instances is migrated across the boundary, the other one must also be supplied
- the information is transformed as it passes the boundary and extra information is needed to describe how the transformation can be reversed in the target domain, for example by adding a specification of decompression algorithm to information which has been compressed.

6.2.4 Removal of information

It may be appropriate to remove information items from a collection of information passing a domain boundary because:

- there is no way of representing the information in the target domain
- there is no use for the information in the target domain
- the information is confidential to the source domain.

6.2.5 Encryption and decryption

Information may be translated into an encrypted representation for security, to be decrypted at the recipient side. It may also be done to compel the recipient to go back to a known and trusted agent in order to decode the information; for example, encryption of interface references.

This involves transformation of the representation, but also addition and removal of information needed to perform the correct encryption and decryption, such as a key.

6.2.6 Name translation

Names which pass across naming domain boundaries into domains where they are not recognised must be transformed into one of the following:

1. A name for a proxy for the named entity.
2. A name in the new naming domain for the named entity.
3. A name for a different entity which is located in the new domain, and which is conformant to the named entity in the old domain.

None of these alternatives have any semantic impact, as the name retains its meaning after transformation, at least in terms of its conformance to the expectations of users of the name [APM.1003 93].

The third approach described here should be used with **extreme caution**: in order to be able to deduce that one entity conforms to another, a large amount of (often application specific) information about the semantics and intended use of the named entity is required: information of this kind is not commonly available in systems, and so this approach must usually be considered on a case-by-case basis, where there is some difficulty with the other approaches.

6.2.7 Transferring information between type systems

A type system provides a language for describing the structure and physical representation of information in software: a set of primitive types, and a set of operators for building complex types from primitive types. Two software components using different primitive types and operators cannot represent each other's information types directly.

If an information item is to pass a type system boundary, the description of the type of the information must have already been passed from the source domain to the target domain and translated from the source type system to the target type system; only then can target domain know how to represent the translated information item.

Where software components use different complex types, an application domain boundary is being crossed, irrespective of whether there is any difference in type systems or not.

In cases where two components exist in different application domains, information about different types should preferably be exchanged by passing references to type specifications, rather than by agreeing the meanings of type names, as this places fewer constraints and allows different application domains to use independent type repositories. This approach may be difficult to apply across a type system domain boundary, as it requires that the differences between the type systems be resolved by gateways at run-time.

In general, any language which describes a model, for example as a type system describes an application model, also embodies a model in itself which can be described using another language; for example, a language could be developed to describe type systems. This regress may be infinite, and in practice, type systems have been chosen as the most practical point from which to begin.

6.2.8 Transferring information between different applications

When information passes an application domain boundary, it must be transformed from one model to another. In each domain, some language or type system is used to describe the model: usually in practice this describes how the information in the model is structured but not what it means.

Since semantics are not usually defined by type systems, different applications may use types which are structurally identical for different purposes, or structurally different types for the same purpose. This means that transformations of information exchanged across application domain boundaries are often application-specific rather than type-specific.

6.2.8.1 Approaches to structural differences in models

Structural differences may be surmounted by adding new types to the model in the target domain so that the information may be represented as is. However, this may not be possible, for example where static models are used.

The alternative is to map the information into one or more types in the target domain which are (partially) equivalent.

In the latter case, removal and/or addition of information may be required, in addition to translations. Such translations between models can be divided into two categories; value-generic and value-specific.

6.2.8.2 Value-generic and value-specific mappings

Some translations are value-generic in the sense that a mapping function can be defined between two groups of types, which holds no matter what values the instances of the types take. For example, “the value of the engine load is equal to the engine r.p.m. divided by the gear ratio for the current gear”. Translations of this kind can be modelled in gateways using virtual types to represent the two sets of types being mapped: these virtual types are shared in the sense that they are present as independent concepts in both domains.

Some care must be taken to manage ranges with such mappings, since permitted values on one side may be transformed into values which are out of range on the other.

Some translations cannot be described with value-generic rules. In these cases, transformations must be described for all the different values which instances can take. For example “DEFCOM 1 = Red Alert; DEFCOM 2 = Yellow Alert; DEFCOM 3 = Yellow Alert”.

7 Application level transformations and gateways

Gateways can be used to implement and manage the relationships between systems, by performing the type of translations described in Chapter 6 and by enforcing policies and monitoring interactions.

7.1 Application level transformations

The transformations described in Chapter 6 which occur at technology, administrative or application domain boundaries, require knowledge of one or more of the following:

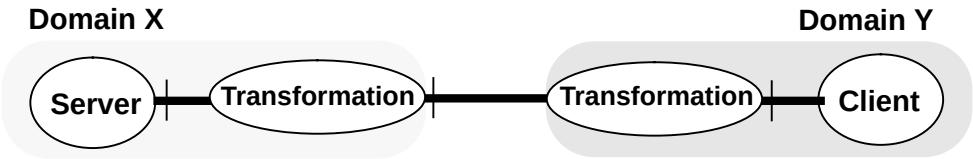
- the type of information passed
- the value of the information passed
- the source and destination of the information passed.

The inspection and transformation of the information therefore have to be done in the same terms of the description of application interfaces In other words not encoded for communication, transparency or security purposes, but directly interpretable in terms of the client/server interface definition.

Application level transformations can be described in terms of application model(s). This links the transformations to client-server interface descriptions (e.g. IDL specifications).

Coupled with the generic model of domain federation shown in Chapters 5 and 8, the model of application level transformations is shown in Figure 7.2.

Figure 7.1: Application level transformations



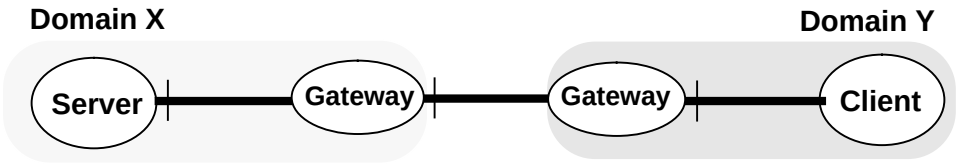
7.2 Application level gateways

Gateways which perform application level transformations are called ***application level gateways*** (Figure 7.2).

There are a number of advantages to application level gateways:

- transformations are carried out in terms of application models. Application semantics can therefore be used where available and where necessary

Figure 7.2: Application level gateways



- transformations carried out in the gateway can be specified in terms comprehensible to the application and platform builders
- since interface type information is available, gateways can detect the passing of interface references and therefore act as interceptor to the potential creation of subsequent bindings (thus the link to trading and gateway propagation explained in Chapter 8)
- IDL descriptions are sufficient to generate the gateways skeleton, although they are not sufficient for generating the transformations automatically
- application level transformations can be made independent of the lower levels translations carried out in protocol stacks (provided the invocation parameters are not encrypted before transformation takes place). This may result in better maintainability.

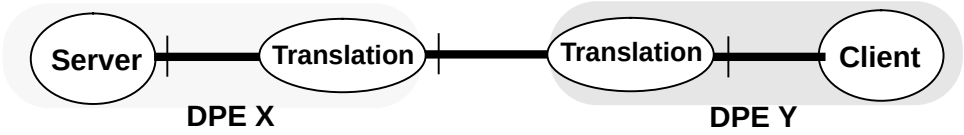
7.3 Using Application level gateways

The following sections describe how application level gateways can be used.

7.3.1 Gateway as a way of overcoming differences

Gateways can act as a domain crossing enabler in cases where there exists a difference between the domains which has to be overcome to enable objects within the respective domains to interact (Figure 7.3).

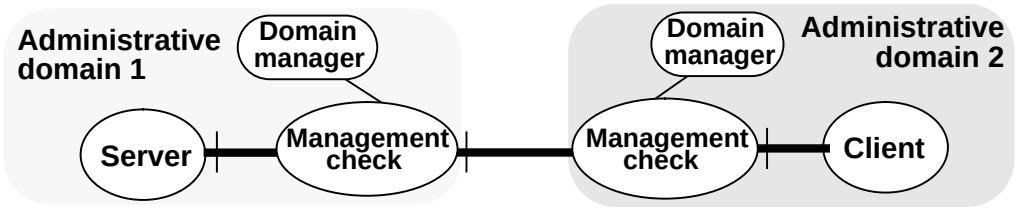
Figure 7.3: Translation to overcome differences



7.3.2 Gateway as a way of creating administrative domains

In some cases where there are no technological differences between systems, it is possible to create administrative domains by inserting gateways in the communication paths between them (Figure 7.4).

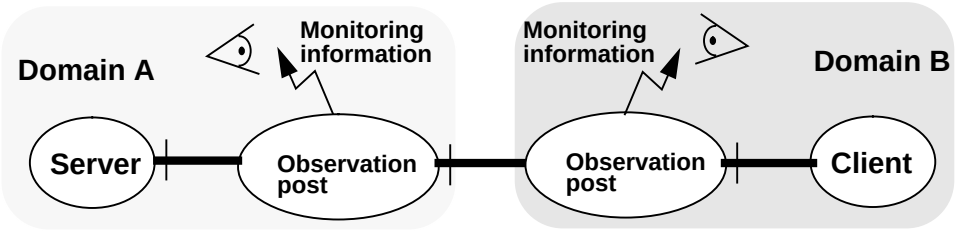
Figure 7.4: Gateways as a way of creating administrative boundaries



7.3.3 Gateway as a way of monitoring domain crossing

Gateways can help in observing the interaction between domains. Application level gateways are particularly suited for this task as the information passing through the gateway is available in a form amenable to inspection by programs or humans (Figure 7.5).

Figure 7.5: Gateways as observers of interactions between domains



8 Using gateways to manage inter-domain relationships

Initial gateways must be constructed so that cooperation between systems can commence, but it is critical to ensure that the agreed relationship is maintained as new bindings are created to support subsequent interactions. Failure to do this may result in contractual breaches such as lapses in security or failure to deliver the required QoS. This chapter shows how gateways can be used to propagate established relationships to new bindings and interactions.

8.1 Contracts and relationship between domains

Domains and their administrations can help to construct, maintain and dismantle relationships prescribed by the contract between the authorities, and in agreement with the policy of the respective authority (Chapter 3).

There are fundamentally two ways to connect different domains:

- agree standards for protocols, processes, models and so on, which all members of a federation incorporate directly into their systems
- let each member choose its own approach independently and build gateways to map between differences.

Ideally, standards are preferable because they reduce the interconnection and maintenance workload in the long term, if they are successful and widely adopted. However, in practice:

- a single universal standard which completely describes all aspects of systems is impractical, and it seems inevitable that there will always be a range of alternatives to accommodate
- it may not be economically and technically feasible to support the necessary range of alternatives in each different system
- for administrative boundaries, common protocols do not offer a solution.

This implies that transformations or at least administrative checks will usually be required for different domains to interact. The bindings between different objects in the two domains are all subject to the contractual relationship between the domains, and therefore to the necessary transformations at the domain boundaries.

It is important to note that:

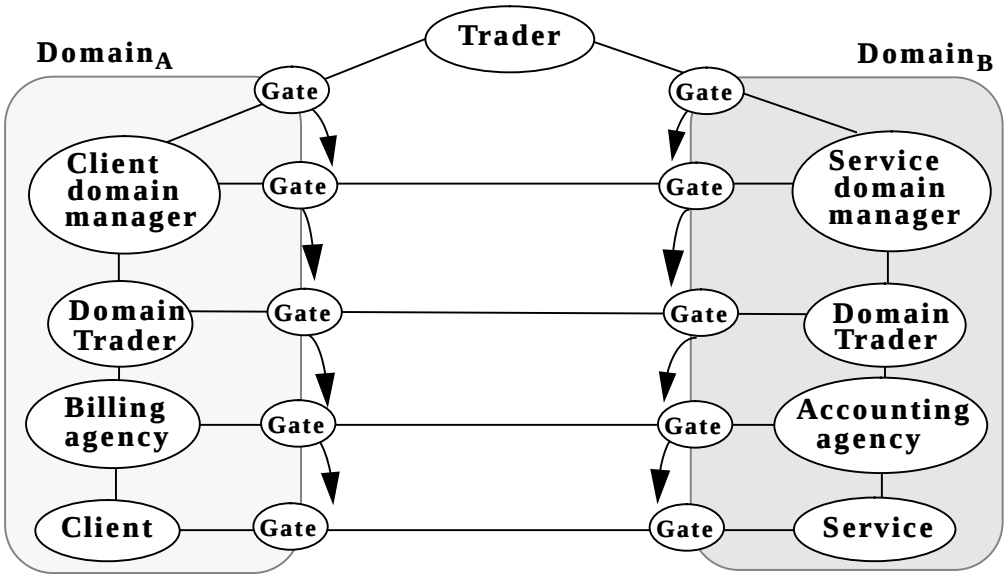
- transformations and checks at domain boundaries must be derivable from the specification of the domains and the relationship defined in the contract between the domains
- necessary transformations and checks must be implemented in all the subsequent bindings established across the domain boundaries.

8.2 Propagation of the inter-domain relationship

The problem of federating two domains using gateways can be summarised as:

- ***how to set up*** the initial link between the domains with the necessary gateways. Some of the gateways may have to be put in manually, at least in the initial bindings between the domain managers or domain traders
- ***how to propagate*** the necessary gateways to all the subsequent bindings between objects in the two domains as shown in Figure 8.1.

Figure 8.1: Propagation of the inter-domain relationship



Thus, the role of gateways is two fold:

- to manage and transform information crossing domain boundaries
- to create subsequent gateways where necessary.

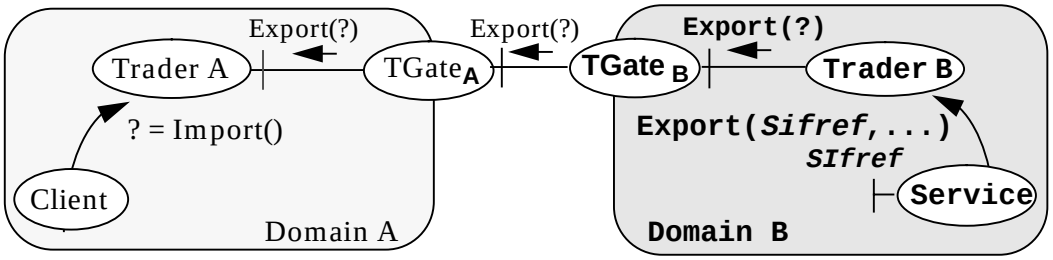
8.3 Trading and interception

A starting point for describing how to approach the problem of relation propagation is the ***trading process***. The trading process facilitates the transfer of information about services to allow bindings between clients and servers to be set up dynamically [APM.1005 93], [APM.1387 94]. One essential piece of information which must be passed to the client is the server's ***interface reference*** which contains the information necessary for the client to bind to the server. The passing of the interface reference is part of the trading process. Trading may be done explicitly through a trading service or by a third party exchange of interface references.

Figure 8.2 shows a situation where a client and a server reside in two different domains A and B. Each domain has its own trader and these are connected through gateways which deal with the differences between the two domains[1].

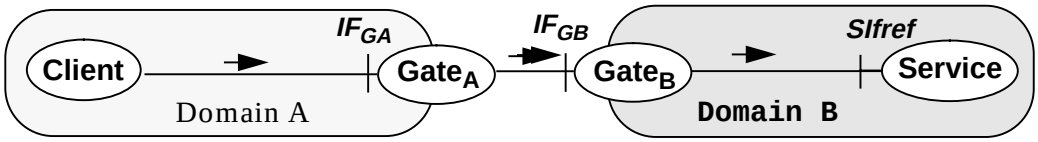
1. The federation agreement between Domains A and B has led to the establishment of trader to trader gateways embodying the policies for sharing the services across the boundaries. The scenarios described in Chapter 3 show how this can be done.

Figure 8.2: Passing interface references across domain boundaries



The server in domain B exports its interface reference to its local trader (Trader B) which in turn exports it to the trader in domain A. As a consequence of trading for a service, a gateway for that service must be established as in Figure 8.3. This is a computational view - issues of how and when to implement the gateways are discussed in the engineering model.

Figure 8.3: Binding across domain boundaries



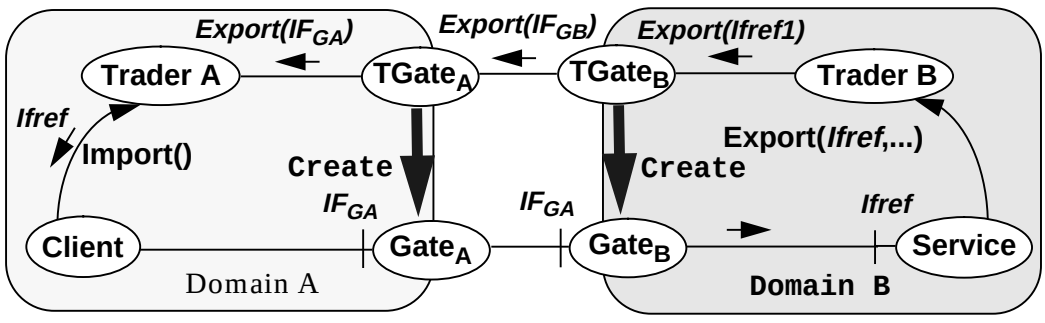
8.4 A computational model of interception

To achieve the configuration shown in Figure 8.3, both gateways in Figure 8.2 ($TGate_A$ and $TGate_B$) between the traders must be able to detect the transfer of interface references and insert the appropriate gateways capable of carrying out the required transformations, in the invocation path of the potential link before or when it is actually used.

This can be made transparent to the application programmers: interception may not appear in the computational model presented to them at all. Rather, this is a description of how the programmer would have to deal with it if it were not provided transparently.

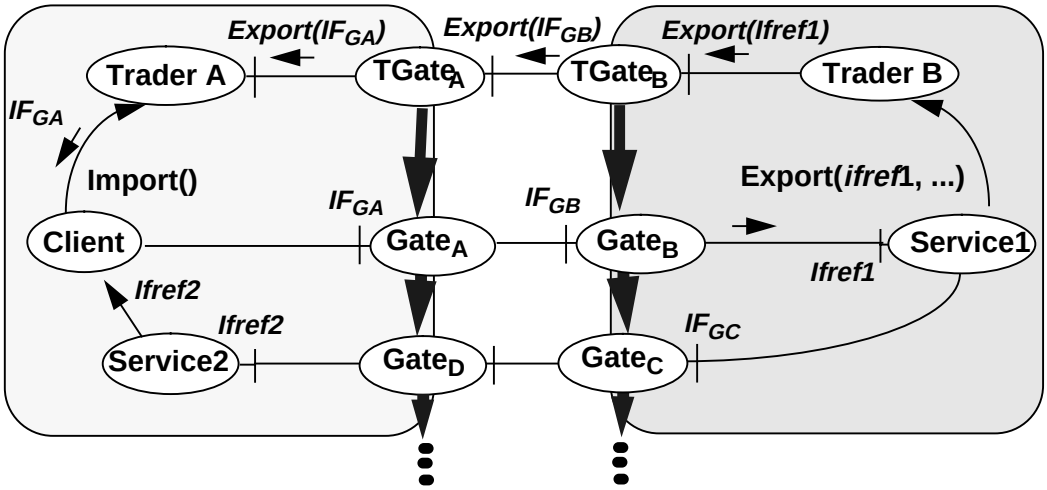
Figure 8.4 shows the relationship between the trader gateways and the gateways between the client and server which result from the interception process being applied at the gateways between the traders.

Figure 8.4: Setting up the gateways when interface references pass between different domains



The process of interception may also occur in the child gateways, if any of the parameters of the client-server invocations themselves contain an interface reference (third party trading). Thus gateways may create gateways which in turn create more gateways. The parent-child lineal[1] nature of the process can be seen as gateway cloning (Figure 8.5).

Figure 8.5: Computational model of interception: the cloning nature of setting up the gateways



Note that the computational model of interception is **only** a **model** which can be implemented in a variety of different ways. Different implementation strategies for interception will be discussed in detail in the following chapters.

1. ***Lineal*** describes a relation among a series of causes or arguments such that the sequence does not come back to the starting point [BATESON 79]. The opposite of lineal is *recursive.*

9 The phases of the Interception process

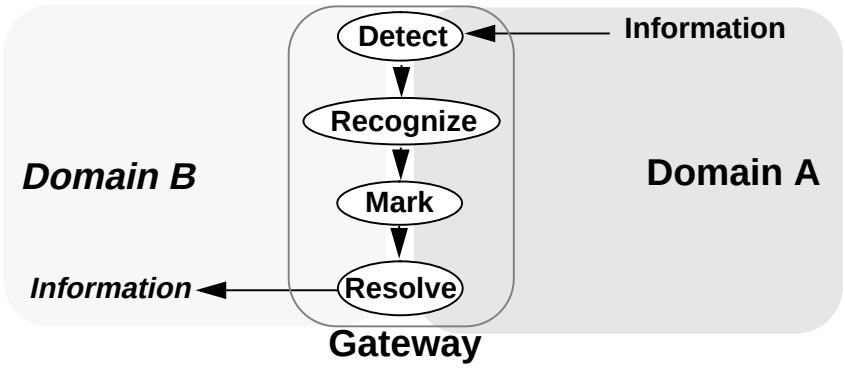
This chapter examines the action which a gateway has to take as information passes through it on the way to another domain. Dividing the action into distinct phases helps designers to decouple the issues associated with the action taken.

The phases are used in subsequent chapters to devise different interception strategies, to show how designers and system integrators can distribute the different activities to give different options for resource allocation, performance and other non-functional issues.

9.1 Interception process phases

The interception process can be divided into distinct phases, some or all of which can be carried out by a gateway (Figure 9.1). The phases are described below.

Figure 9.1: Phases of the interception process



9.1.1 Detection phase

This phase is involved with the detection of information crossing a boundary. It indicates that the potential for the crossing must have been detected at an earlier epoch and the appropriate mechanisms put in place to detect the actual crossing of an invocation.

9.1.2 Recognition phase

This phase is involved with recognizing the parts of the information e.g. invocation parameters which:

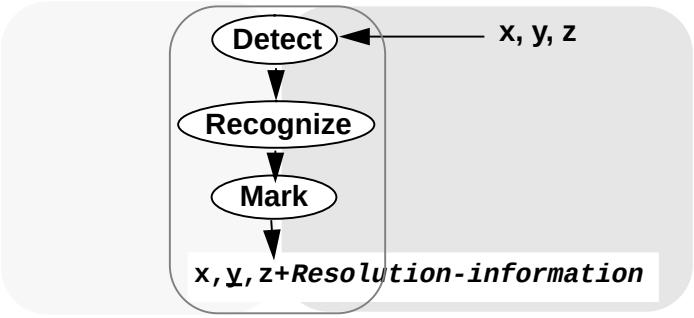
- require modification as a result of the crossing
- necessitate some action by the interception process.

Determining what parameters require action to be taken depends on the type of the boundary which is being crossed and the nature of the information.

9.1.3 Marking phase

This phase is involved with the marking of information which needs acting upon by identifying what needs to be transformed and adding information about where or how this is to be done (Figure 9.2).

Figure 9.2: The marking phase



The different ways of marking can determine or at least constrain:

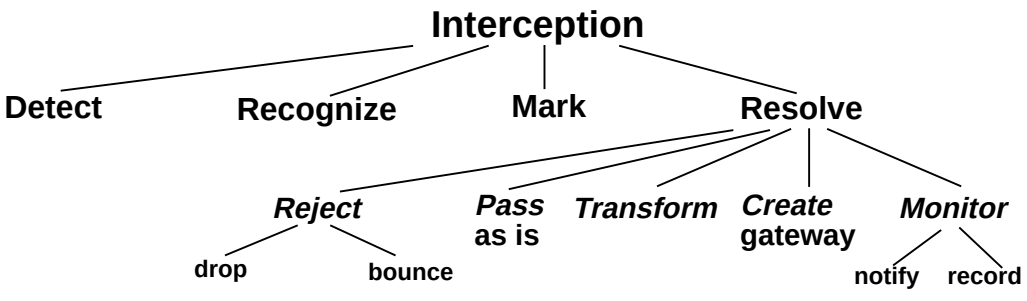
- which objects can request the resolution
- where the resolution will be carried out, and how and when

9.1.4 Resolution phase

This phase is involved with carrying out the necessary action required by the crossing of a boundary. The resolution phase may be one or a combination of the steps shown in Figure 9.3 (some of these, but not all, are mutually exclusive):

- *rejection* of the crossing with or without notification to the source
- *passing* the invocation without changes
- *transforming* the content of the invocation
- *creating* any appropriate gateway (or gateways) in case of an interface reference being passed
- *monitoring*: notifying some interested agent or recording the event.

Figure 9.3: Phases of the interception process



For example, a rejection of a crossing is unlikely to be accompanied by the creation of any gateways, whilst translation and creation of gateways do not necessarily exclude each other.

10 Interception resolution strategies

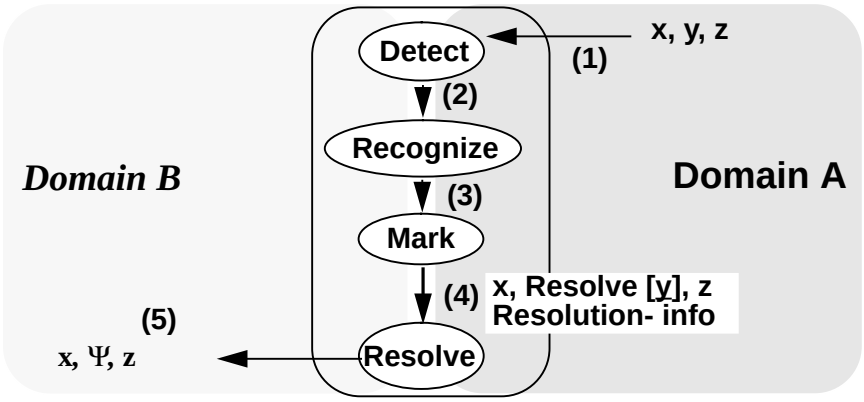
The question of when translation occurs at domain boundaries affects various non-functional issues such as latency, throughput and the effectiveness of resource allocation. For example, it is undesirable to translate information that is unlikely to be used. This chapter presents different strategies for managing whether different activities are done, when, and by whom, and discusses their impact on the design of gateways and infrastructures.

Three resolution strategies are presented, which are based on different ways in which the marking and the resolution phase of the interception process can be joined.

10.1 Immediate resolution strategy

Where the resolution process immediately follows the Detect/Recognize/Mark phases of the interception process, the result is the **Immediate Resolution** strategy (Figure 10.1).

Figure 10.1: The Immediate resolution strategy



The resources required for the resolution process are allocated to it regardless of whether the transformed information passing through the domain boundary will be used in Domain B or not. This is wasteful in cases where it will not be used. The example of interface reference passing domain boundaries will be described in detail in Chapter 11.

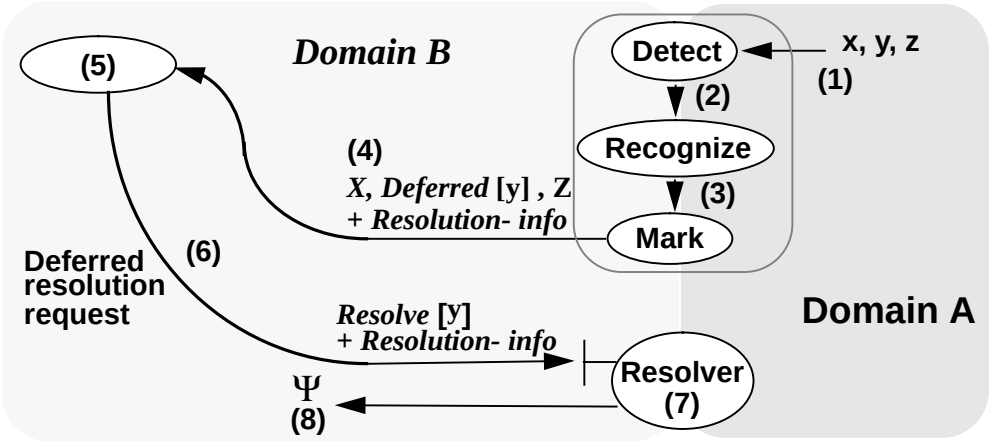
One advantage of the immediate resolution strategy is that the information passed into Domain B will require no further transformations: these may be time consuming and cause initialisation delays at a later stage.

Another advantage of this strategy is that the infrastructure of Domain B and the recipient object will require no changes as it will receive information in the form it is expecting. The entire transformation is carried out in the gateway prior to the information crossing the boundary. This is important in legacy systems - where it is not possible to modify an application or an environment.

10.2 Deferred resolution strategy

If the Detect/Recognize/Mark phases are separated from the Resolution phase, allowing an object in Domain B to decide if and when (and sometimes where) the resolution process will take place, the result is the **Deferred Resolution** strategy[1] (Figure 10.2).

Figure 10.2: The Deferred resolution strategy



In step (4) the gateway's marked information is forwarded to a recipient in Domain B. The recipient object (or any object it passes this information to) can decide to use the information (5) and request the resolution (6). This strategy will require a Resolver which can accept invocations from objects in domain B.

10.3 Leave-and-Forward resolution strategy

If information marked as deferred at a domain boundary, is sent by an object in the recipient domain back to the sending domain (as an invocation parameter, for example), it may not be necessary to resolve the deferred marking at all if at some later stage it is to be sent back to the originating domain without any changes[2].

The Resolver in this case acts as a bounce-forward agent and has to be able to recognize the marked information and be able to strip the marking off, forwarding it to the domain from which it originated (Figure 10.3).

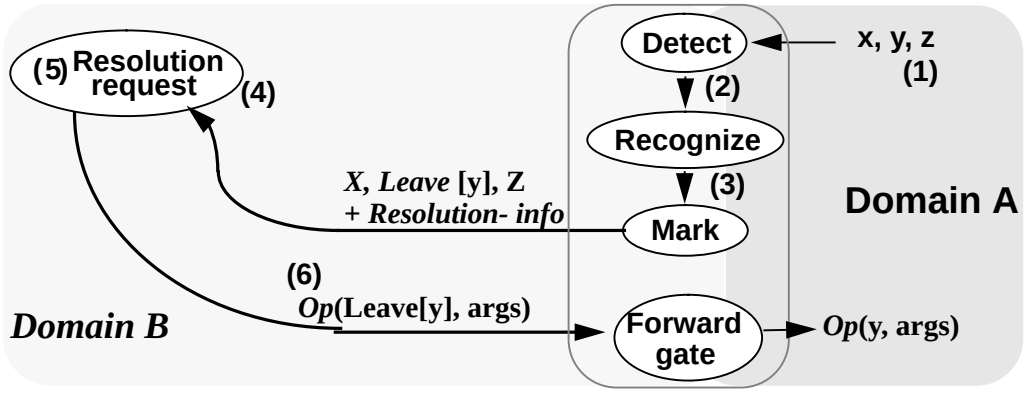
10.4 Choosing a resolution strategy

The decision as to whether to use the immediate, deferred or Leave-and-Forward resolution strategy will be dictated by the policy of the domains which the gateway spans, and the gateway management and maintenance policy. Choosing the deferred or leave-and-forward strategy will require the recipient domain to be capable of receiving marked information and dealing

1. This is similar to a situation where a tourist goes on holiday with home currency and decides not to convert the money back until she needs it.

2. This is similar to a situation where a tourist decides not to convert her home money to the local currency because the person she has to pay plans to visit her country shortly and will need money in that currency. Both gain from not having to convert the currency and pay commission.

Figure 10.3: The"Leave-and-Forward" resolution strategy



with it. This may not always be acceptable since it will either require a change in the interface definition of the client or an agreed way to mark information.

The resolver or forwarding agent may not necessarily reside in the gateway as long as it is accessible from the recipient domain. It is also possible to have a default resolver for one or more types of boundaries crossed, so that the marking does not necessarily have to include the reference to the resolver in the marked information.

10.5 Level of resolution

There following are examples of the different places where resolution can be requested and carried out:

- a server or server manager: these can create a gateway for a service where a boundary between incompatible applications is to be crossed
- a server binder: when creating the interface reference for a service the binder can consult a domain manager and create a technical or administrative gateway
- a trader: can detect a difference between the properties of a client and server and act to create a gateway to perform the required conversion. In ANSAware, for example, the trader acts as a gateway interceptor which inserts a relocator interface reference into the server's interface reference [ARM 93]
- a domain manager: can create a technical or administrative gateway on either the client or the server side
- client or client manager: similarly to the server and server manager the client side can create a gateway when it receives the server interface reference
- client binder: when binding the client to a server in a different application domain

11 Passing interface references through domain boundaries

This chapter illustrates the interception strategies described in §9 *The phases of the Interception process* by showing how each strategy handles interface references which are exchanged across domain boundaries. It aims to assist platform designers and system integrators in determining the most appropriate approach by highlighting the non-functional benefits of each strategy and the design implications for infrastructures.

11.1 Crossing domain boundaries

In order to describe what happens when interface references cross domain boundaries, the example of connecting two different distributed application platforms is used. Most existing platforms (DCE [OSF 92], CORBA based [OMG 92], ANSAware [ARM 93] for example), use different RPC mechanisms and also different notions of the information and structure of interface references (e.g. interface references in ANSAware, Binding handles in DCE and Object pointers in CORBA). Both differences have to be accommodated by the gateways bridging the two domains.

Three strategies are illustrated:

- immediate resolution strategy
- deferred resolution strategy
- leave-and-Forward strategy

In the following figures, interface references are shown in a graphical manner. The exact nature of their structure and information will be discussed in detail in chapter 14.

11.2 Immediate resolution strategy and interface reference passing

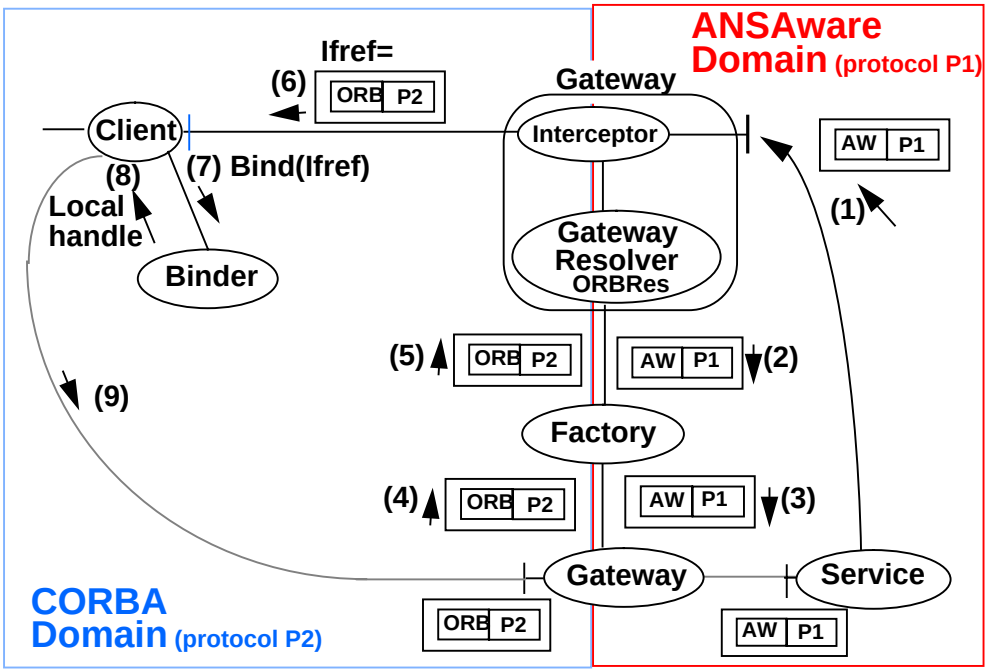
The ***immediate resolution strategy*** shown in Figure 11.1 supports the creation of the necessary gateway as the interface reference is passing the boundary.

11.2.1 The immediate resolution strategy sequence

The sequence of operations in the immediate resolution strategy (Figure 11.1) is:

1. the server interface reference is sent from the sender domain to the recipient domain but gets intercepted by the gateway
2. the gateway resolver, according to the immediate resolution strategy, requests a gateway factory to create an appropriate gateway, and passes it the interface reference of the server in the sender domain

Figure 11.1: immediate resolution strategy



3. the factory creates the requested gateway with its interface in the recipient domain. The new gateway can bind to the server at this point
4. the created gateway passes its interface reference to the factory
5. the factory returns the gateway interface reference to the originating gateway resolver and interceptor
6. the new interface reference is passed to the client in the recipient domain
7. the client passes the gateway interface reference to its Binder
8. the binder creates and returns a local handle for the client
9. the client performs invocations on the created gateway.

11.2.2 Advantages and disadvantages of the immediate resolution strategy

The main disadvantage of the immediate resolution strategy lies in the fact that resources are allocated for constructing the gateway regardless of whether the service will be used from the recipient domain.

The main advantage of the strategy is that once the resources are allocated to the gateway, it is possible to guarantee that the gateway will be available when necessary. Also, no additional time has to be spent in setting up the gateway when the binding is created. This may be important in real-time applications.

This strategy also has the advantage that it provides a way of dealing with legacy systems as it does not require any changes to be made to the supporting platforms or applications in order to deal with boundary crossing. To the recipient domain, the interface reference will appear as if it is supported by a service in the same domain.

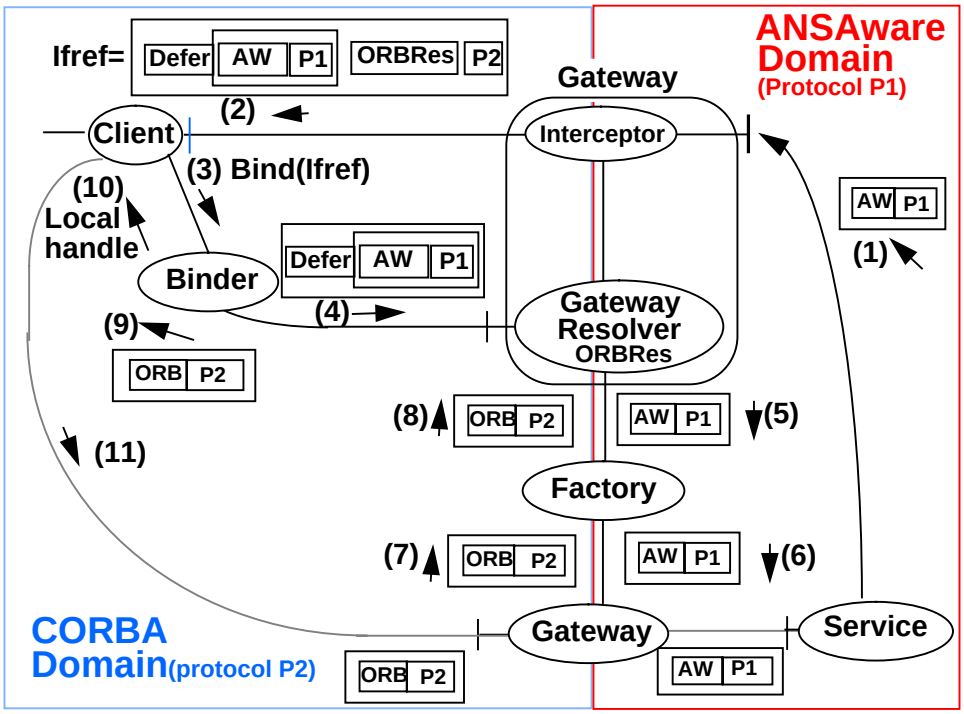
11.2.3 Requirements placed on distributed platform design

None.

11.3 Deferred resolution strategy and interface reference passing

The ***deferred resolution strategy*** (Figure 11.2) intercepts the passing of the interface reference through the domain boundary. It then marks the interface reference to indicate that it has crossed a domain boundary and that it will ultimately require the creation of the appropriate gateway before being used to invoke the server in the other domain. At some time before the invocation takes place, the marked information will have to be passed to a gateway-resolver capable of creating the gateway.

Figure 11.2: Deferred resolution strategy



The request to resolve the deferred interface reference can take place either when client requests the binder to set up the binding, or it can take place on first invocation attempt.

11.3.1 The deferred resolution strategy sequence

The sequence of operations in the deferred resolution strategy (Figure 11.2) is:

1. the server interface reference is sent from the sender domain to the recipient domain but gets intercepted by the gateway
2. the gateway interceptor, according to the deferred resolution strategy, marks the information as deferred (“defer”), adding the reference of the gateway resolver where the deferred interface reference can be resolved when required. The marked interface reference is then passed to the recipient domain
3. the client requests Binder to bind to the interface using the passed interface reference
4. the binder recognizes the interface record marked as deferred, extracts the gateway resolver reference and sends the deferred interface record (or the entire interface reference) to the gateway resolver

5. the gateway resolver requests a gateway factory to create an appropriate gateway, passing it the interface reference of the server
6. the factory creates the requested gateway with its interface in the recipient domain; the new gateway can bind to the server at this point
7. the created gateway passes its interface reference to the factory
8. the factory returns the gateway interface reference to the gateway resolver
9. the new interface reference is passed to the Binder in the recipient domain
10. the binder creates a binding to the gateway and returns a local handle for the client
11. the client performs an invocation on the created gateway.

11.3.2 Advantages and disadvantages of the deferred resolution strategy

The deferred resolution strategy is more efficient with regard to resource utilization than the immediate resolution strategy as the creation of gateways only takes place when a client requests their creation, i.e. when it wishes to use the service.

The main disadvantage of this strategy is that it places overheads on the creation of the client-server binding in terms of the time taken to set up the gateway. Also this strategy requires changes to the supporting distributed platform.

11.3.3 Requirements placed on distributed platform design

The interface reference has to be able to hold foreign interface references within it together with gateway-resolver references. The Binder has to be able to extract and use this information to invoke the appropriate gateway-resolver.

11.4 Leave-and-Forward resolution strategy and interface reference passing

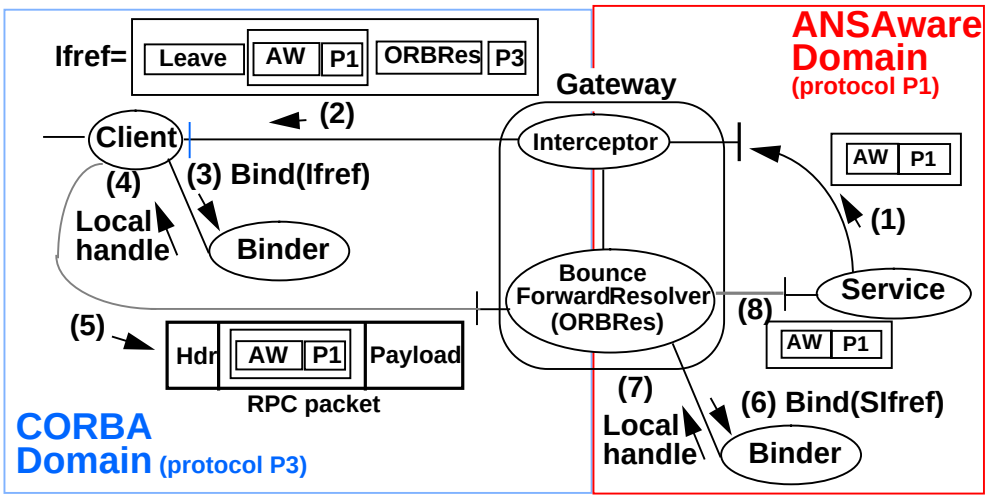
When an interface reference is passed through a domain boundary and is marked as “Leave”, it is possible to have the marked interface reference used for invoking the service it is referring to, by passing it as one of the invocation parameters without having to resolve it. The gateway resolver can then forward the invocation to the relevant service.

This however, will require a change in the interface definition of the client and the gateway to include the interface reference as a parameter and is unlikely to be acceptable. One possible implementation is to carry the destination interface reference in the RPC header so that the gateway resolver can obtain it and use it. This will not require a change in the interface definition and stubs but will require the RPC protocol and mechanisms to carry and deal with the information as necessary. Implementation of the **Leave-and-Forward** strategy at the RPC level with use of the gateway-resolver as a bounce-forward agent is shown in Figure 11.3.

The allocation of resources in this strategy leaves the following choice to the designer:

- a binding between the gateway and server can be established and held over more than one invocation from the client. This implies the gateway-resolver holds client-server specific state

Figure 11.3: The Leave-and-Forward strategy



- a binding between the gateway and server can be established for each invocation and discarded after response is delivered to client.

11.4.1 The Leave-and-Forward strategy sequence

The sequence of operations in the Leave-and-Forward strategy (Figure 11.3) is:

1. the server interface reference is sent from the sender domain to the recipient domain but gets intercepted by the gateway
2. the gateway interceptor, according to the Leave-and-Forward strategy, marks the information as “leave”, adding the reference of the bounce-forward resolver where the invocations are to be sent to, and forwards it to the client
3. the client passes its Binder the interface reference
4. the binder returns a local handle for the bounce-forward resolver, having set up the appropriate communications
5. the client invokes the bounce-forward resolver passing it the interface reference of the server as part of the RPC header information
6. the bounce-forward resolver requests the binder in sender Domain to create a binding to the service
7. a local handle for the server is returned to the bounce-forward resolver
8. the bounce-forward resolver forwards the invocation to the server.

11.4.2 Advantages and disadvantages of the Leave-and-Forward strategy

The main advantage of the Leave-and-Forward strategy is that it does not require holding any state in the gateway concerning bindings between clients and servers. This will make re-starting such a gateway after failure easier as no mappings between clients and servers will be lost.

If the binding between the server and gateway is established and discarded on per invocation basis, the overheads of establishing a binding for each invocation will be prohibitive in some real-time applications.

11.4.3 Requirements placed on distributed platform design

This method places requirements on distributed platform:

- a gateway interceptor must be able to mark an interface reference to indicate that the Leave-and-Forward strategy is to be used
- the binder has to be able to recognize the marking and act on it accordingly
- an RPC protocol supported by the platform must carry the full interface reference of the destination in an agreed part of its header
- the gateway bounce-forward resolver must support the same RPC protocol.

12 Interception implementation issues

The design of gateways determines the extent to which the non-functional characteristics of links can be managed, and constrains the non-functional characteristics of interactions occurring across those links.

This chapter discusses design options for gateways, discussing their impact on non-functional issues and showing how they relate to the interception issues introduced in previous chapters. It then discusses issues for ordering and combination of gateways and highlights potential problems.

12.1 Overview of design issues for interception and gateways

The computational model of interception shown in Figure 8.5 may be implemented by creating a private gateway for each interface reference which is passed through bindings across each domain boundary; each gateway might hold all the information and resources it requires privately.

This may be unacceptable in engineering terms because of inefficient resource usage and duplication of information. The computational gateway, generated by a parent gateway, may be implemented differently, depending on the engineering decisions taken. The interception strategy adopted may impact the design of both gateways and infrastructures.

A ***gateway*** can be viewed as:

1. information about the transformations that have to be applied at the boundary.
2. an interface for managing a set of bindings.
3. information enabling bindings to be set up and maintained (how to bind to servers; e.g. hold server interface references or the actual bindings).
4. the resources required to implement the appropriate interactions and transformations.

A gateway can thus be implemented in a variety of ways and the implementation options can be derived from the different possible answers to the following inter-related questions:

1. how should the information describing transformations be distributed?
2. what sets of bindings can be managed by one gateway?
3. how should the information about bindings be distributed?
4. what resources are allocated, when, and for how long?
5. when many transformations are applied, how should they be ordered?

In practice, there is a spectrum of design options for the design of a gateway, and for the ways in which different gateways may be positioned and combined.

12.2 Choosing gateway design options

Which design options are most appropriate for any particular gateway will be determined by the following issues:

- **Performance** and **QoS** guarantees: early versus late allocation of resources may affect trade-offs of latency versus throughput, and efficiency of resource usage versus latency. Consequently, it may constrain the QoS guarantees which can be given, particularly with regard to availability and performance
- **Management** of domain boundaries: boundaries may need to be made visible for enterprise, administrative or technical reasons; as a consequence, information exchanged between gateways may need to be monitored in specific forms. It may be important to be able to trace the parenthood relationships of gateways for management, security or billing purposes; this implies a need to make distinctions between different client-server bindings.
- **Technical** issues: for example, the availability of interface specific versus type generic stubs can have a significant effect on gateway design and implementation (generic stub facilities such as CORBA DSI/DII [UNO 95]). Sharing of resources as in the case of type generic gateways may also affect performance and influence the QoS guarantees. Availability of multi-threading together with performance requirements may dictate resource allocation issues.
- **Application semantics**: the specific requirements of the application may place constraints or indicate which option is most appropriate. Applications with extreme requirements may merit specific mechanisms.

12.3 Gateway design options

This section discusses the options for design of a single gateway. Further detail concerning gateway design options can be found in [APM.1303 95].

12.3.1 Distribution of information about transformations

Information specifying which transformations should be applied at which boundaries, to which types and values of information and under which circumstances, must be available to each gateway. Such information may be physically held:

1. privately by each gateway.
2. shared between many gateways.
3. in shared repositories, but with some specific information held privately.

No matter whether information is shared or not, it must be organised in some way. Options are:

1. per-type: all information describing transformations that should be applied to transform to or from particular type is held together.
2. per-boundary: all information describing transformations that should be applied at a particular boundary is held together.
3. per-domain: all information describing transformations that should be applied at all boundaries of a particular domain is held together.

4. various combinations of the above.

In general, information should be shared wherever this does not compromise maintainability or performance. Performance requirements may dictate that transformations be represented as executable code, in which case sharing may only occur prior to compilation. If the shared information may alter for some but not all of the parties sharing it, the sharing may be a maintainability problem.

The per-domain option will usually be the first one to be applied, since it is usually desirable for domains to remain autonomous. Usually it is a question of whether per-type information should be held on a per-boundary basis, or vice-versa. This depends upon where the greatest commonality is to be found, which in turn is dependant upon the nature of the differences, but in most cases will be per-boundary.

More general gateways (e.g. type generic stubs) allow the use of a single interface to act as a gateway for different types of bindings, thereby circumventing the need to create more gateways as interface references of different service types cross the boundary and enabling more efficient resource usage. However, during conditions of high load in a gateway which shares resources between bindings, the creation and use of private resources may be necessary to maintain performance.

12.3.2 Distribution of bindings across gateways

There are various options for the allocation of bindings to gateway instances:

1. Multiple simultaneous clients and servers for each gateway instance.
2. Multiple simultaneous clients and one server for each gateway instance.
3. One client and multiple simultaneous servers for each gateway instance.
4. One client and one server at a time for each gateway instance.

Support for multiple simultaneous clients implies that some state about server invocations is held, either in the invocation messages to servers or in gateways, to relate these invocations to clients, so that a response from a server can be returned to the correct client.

If a single gateway instance manages multiple bindings, either with clients or servers, there is the question of which set of bindings are shared. This may be constrained by the strategy selected for the distribution of information about transformations. For example, an instance of a type specific gateway may not maintain bindings of different types.

This choice affects the granularity with which certain management policies can be applied to different bindings. It may have security implications due to possible interactions between different bindings. Reliability, performance and efficiency of resource usage may be affected depending upon the assignment of resources to gateways.

Further discussion of these issues can be found in [APM.1303 95].

12.3.3 Distribution of information about bindings

Information about bindings may be held entirely in a gateway, so there is no information about the server in the client invocation message.

Alternatively, some or all of the information about the servers nature and location may be held in the client invocation message (e.g. in an RPC header)

in which case the gateway acts as a router, processing and forwarding client invocation messages solely on the basis of the information contained in each single invocation message. Note that the client infrastructure may obtain this information via an interface reference for the server.

This is analogous to the distinction between connection-oriented and connectionless protocols.

With the connection-oriented approach, the gateway must maintain a binding constantly until it is terminated, which means that the resources it requires remain allocated. The connectionless approach allows the binding to be discarded and re-created upon a later invocation, so resource allocation may be managed in a more flexible manner.

12.3.4 Resource allocation

12.3.4.1 What resources are allocated to each binding?

Essentially, there are four different kinds of resource which may be shared between bindings:

- capsules, or in other words address spaces
- threads
- interfaces
- implementations; sharing of implementations between bindings occurs when interactions across different bindings are processed using the same executable code.

Sharing in each of these aspects may be altered independently of the others.

12.3.4.2 When are resources allocated to bindings?

The allocation of resources to bindings can happen at different points:

- immediate resolution strategy: there is immediate allocation of resources to create gateways when an interface reference crosses a domain boundary
- deferred resolution strategy: this involves deferring gateway creation to
  - — any object which recognizes the deferment of the resolution and knows how to extract the information necessary to request the resolution
  - — first invocation only, with the gateway remaining in place, or initialisation on every invocation
- leave-and-forward strategy: this may result in no permanent allocation of resources for a client-server binding if used on a per invocation basis

12.3.4.3 For how long are resources allocated to bindings?

The resources required for a binding may be maintained for various different durations:

- a single invocation
- one session with one client; the end of the session must be indicated by some means
- all sessions with a client, in which case the client must signal its demise or intent not to interact with the server again

Where information about bindings is held in the gateway rather than in server invocation message (see 12.3.3), it is usually preferable to maintain a single binding for the duration of a session.

Where the server holds state, it may be necessary to maintain a client-server binding for the lifetime of a client.

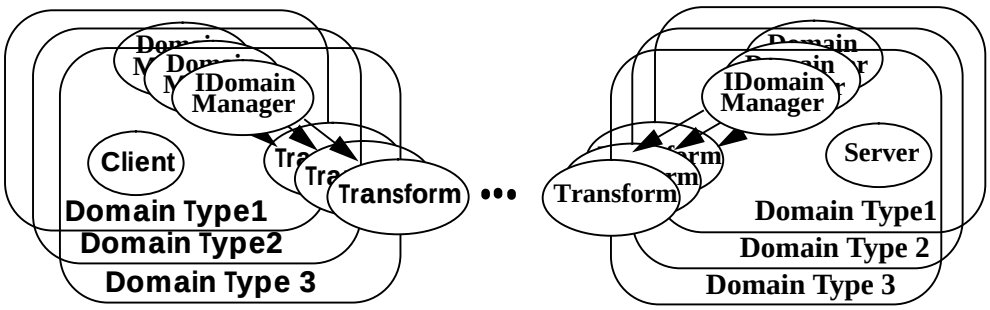
12.3.5 Ordering of composite gateways

Objects belong to several domains of different types, so multiple transformations are likely to be needed when they interact with other objects (Figure 12.1). It is usually inefficient to implement successive transformations using different gateways, so transformations are usually collocated inside a single gateway. Care should be taken:

- to retain low coupling between transformations
- to order the transformations in such a way that they will not adversely effect the transformations or prevent them from being carried out.

The order in which transformations are applied places constraints on the accessibility of the information, and there are often performance issues. These are discussed below.

Figure 12.1: Clients and servers in a multiple domain environment

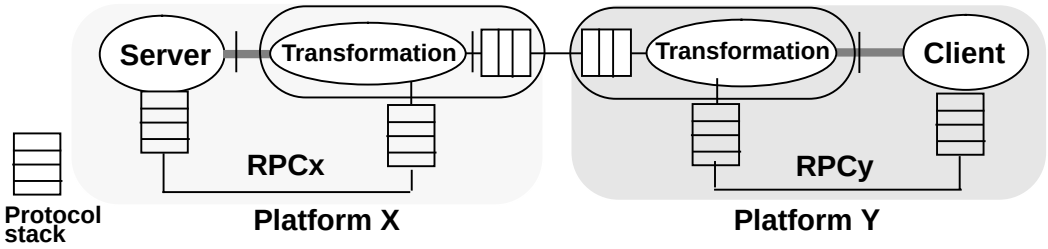


Two examples of the ordering of unmarshalling and marshalling with respect to other transformations are given below as an illustration. RPC stubs are a well-understood form of gateway.

12.3.5.1 Marshalling and unmarshalling first

Gateways may be implemented as stand-alone and therefore potentially remote from clients and servers; this may be useful where for legacy reasons the client and/or server cannot be modified (Figure 12.2). This implies that marshalling and unmarshalling must occur before other transformations.

Figure 12.2: Distributed stand-alone gateway implementation

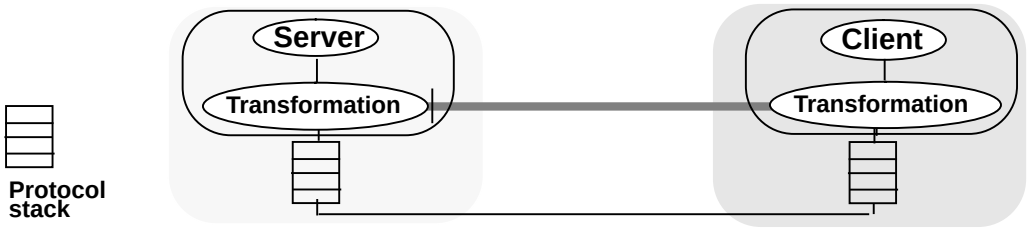


Note that this gateway configuration overcomes the difference between RPC protocols where the transformation shown is null. A more detailed discussion of this use of application level gateways to overcome RPC differences is given in [APM.1303 95].

12.3.5.2 *Marshalling and unmarshalling last*

If gateways are collocated with the client and server, it is possible to apply other transformations before communications layer transformations take place (Figure 12.3).

Figure 12.3: Pre-marshalling gateway implementation



12.3.5.3 *Problems with positioning of gateways*

Using the example of marshalling illustrated above, if gateways are implemented after unmarshalling, as shown in Figure 12.2, then the following problems may occur:

- platform specific information may be lost: the way in which a platform provides transparencies is lost by the time the information has been unmarshalled in a gateway, unless appropriate context information is added
- communication is inefficient: as shown in Figure 12.2, in order to carry out the transformations, the entire protocol stack has to be traversed several times instead of just twice
- security may be compromised: in order to be able to look at the information passing through the gateways, it will be necessary to decrypt it if it was encrypted in the server and client. this means that the de/ encryption keys will have to be given to the gateways; which may be an unacceptable security risk.

These be overcome by doing the transformations either before or during the stub marshalling stage as shown in Figure 12.3.

In general, where transformation A is implemented on information I before transformation B rather than after:

1. transformation B must be applied to A(I) instead of I
2. transformation A must be applied to I rather than B(I)
3. information types available for inspection are I, A(I) and B(A(I)) instead of I, B(I) and A(B(I)).

Issues of types 1. and 2. often have performance implications, because some transformations increase the volume of information or decrease its interpretability. Issues of type 3. have implications for monitoring and thus for security, auditing, traceability and management.

12.4 Impact of design choices on infrastructures

Many of the issues described above only affect the design of the gateways themselves, and the non-functional characteristics of gateways.

However, some of the issues may affect the design of the client and server infrastructures, and these should be given special attention.

- Deferred resolution: implies that the server infrastructure must be able to transport foreign binding information, and recognise markings indicating foreign information, and be able to pass foreign information to designated resolvers.
- Binding information in invocation message: implies that the client infrastructure must be able to hold binding information in invocation messages. This binding information must enable a binding to a foreign object, so either the foreign information itself must be included directly, or a complete translation of the foreign information must be used.

13 Binding

In order to support different interception options, some enhancements to infrastructures may be required. In particular, the resolution process chosen will influence the design of resolvers. Binders resolve interface references into local handles, and are thus affected by different interception strategies in the same way as any other kind of resolver.

The impact of the interception strategies on the design of binders is considered in this chapter, as an illustration of the influence on resolvers in general. The information which is resolved may also be affected, and the relationship between binders and interface references demonstrates this.

13.1 Binding and interface references

Since interface references may be nested, and are likely to be exchanged widely and frequently, it may be appropriate to modify the internal structure of interface references to support the interception options efficiently (§14 *Interface references and interception*). The binding process must support such modifications:

- recognise interface records nested within interface records
- recognise interface records marked as deferred or leave-as-is
- recognise relocation information
- extract information about where to resolve the marked information, i.e. the gateway resolver (or use a default gateway-resolver) in the case of deferred resolution
- extract information about where to send the forwarded invocation in the case of the leave-as-is strategy
- allow for other information concerning issues such as relocation, and passivation and activation.

13.2 Binder policy

As the order of interface records is not necessarily indicative of preferences or priorities of either the producer or the consumer of the interface reference, the binder's policy has to determine:

- whether to deal with each record at a time or adopt some other approach such as searching for immediately usable references before trying to resolve marked ones
- under what circumstances to look for short cuts in invocation paths described by marked interface records (this will be discussed in Chapter 14).

13.3 Binder algorithm

In order to construct a binding using an interface reference, a binder has to:

1. determine what order in which to try the available interface records according to dictated policy
2. process each interface record in order:
   - — immediately usable records: set up communications resources and pass the client a local handle to invoke
   - — records marked as deferred: extract the resolver interface reference and use it pass the interface reference which is currently being processed to the resolver. The returned interface reference is processed from the beginning; it too may be marked
   - — records marked as leave-as-is: extract resolver interface reference and set up the appropriate communications resources and RPC with the destination interface reference in the RPC header, and pass the client a local handle to invoke.

14 Interface references and interception

Interface references are crucial for co-operation between systems, because the exchange of interface references is a pre-requisite for interaction between them.

In accord with §13 *Binding*, this chapter considers the requirements which interoperability places on interface references. For interface references which are to be exchanged between different systems, there are a range of other desirable requirements which are also identified. These characteristics are of great importance for platform designers.

14.1 Introduction

An interface reference contains the engineering information needed to establish a remote connection between a client object and a server object or set of server objects [ARM 93]. Note that it is not a capability: the service may decline to respond to invocations, for example for security reasons.

This chapter describes the kinds of information that must be present in an interface reference and how it is used in establishing and maintaining a connection.

An interface reference may contain information that:

1. names, in the context of the client's binding manager, the end-point to which requests should be sent.
2. identifies alternative routes for reaching the same end-point.
3. enables integrity checking of end-to-end bindings against errors in transparency and network management mechanisms.
4. supports transparent migration of the interface which it refers to.
5. supports transparent replication of the interface which it refers to.
6. supports different interception strategies.
7. enables heterogeneous interface references to be exchanged between different DPEs.

The requirements which should be met for each of these aspects are discussed in three different categories:

- minimal requirements, which any interface reference must meet
- requirements for interoperability, which any interface reference should meet if it is to be exchanged between different systems. This suggests a structure for interoperable interface references (see §14.4 *Recommended interface reference structure*).
- requirements to support other transparencies; whilst these are not directly associated with interoperability, they may affect and be affected by the interoperability strategy adopted.

14.2 Minimal requirements for interface references

14.2.1 End-point Naming

This is the only mandatory function of an interface reference. Interface references name end-points which represent destinations for messages. The nature of the end-point is not constrained; for example it might be:

- a network port, which might be named by a protocol name and a port name. Here, the protocol name also implicitly identifies a naming context
- an instance of an interface, which might be named by a node name, a capsule name and an interface name. Here, the node name might also implicitly identify a naming context.

Different processing environments should be free to employ different strategies for naming end-points. The only general requirement is that the naming scheme employed should be completely context relative.

This gives an evolutionary approach to systems integration and makes it possible to combine new applications with existing applications (legacy systems) and to enable interworking over a range of distributed systems platforms.

14.3 Interoperability requirements on interface references

Interoperability places the following requirements on interface references:

- Accommodating alternative routes
- Accommodating different interception strategies
- Manipulating and passing interface references.

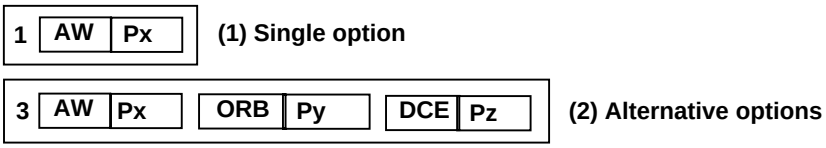
14.3.1 Accommodating alternative routes

The interface to which an interface reference enables access may be reached via many different routes. These routes may be different in that:

- They allow the interface to be invoked via different protocols.
- They allow the interface to be invoked via a different sequence of intermediate interfaces, such as proxies, interceptors and gateways.

An interface reference may have one or more interface records, each providing information about possible paths between the client and server (Figure 14.1).

Figure 14.1: Interface Reference structure - one or more alternative paths



Routes involving proxies, interceptors and gateways are discussed in subsequent subsections.

Binders will be able to choose whichever option best matches the available comms infrastructure, QoS constraints and cost in terms of resources.

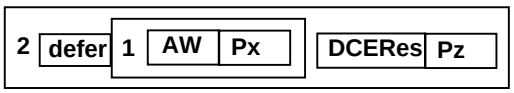
14.3.2 Accommodating different interception strategies

The deferred resolution and leave-as-is approaches are discussed in §10 *Interception resolution strategies.*

14.3.2.1 Deferred resolution strategy

Where the deferred resolution method is used, the information about the cascade of gateways can be represented in a nested fashion with the reference to the gateway-resolver appended.

Figure 14.2: Deferred resolution information inside an Interface Reference



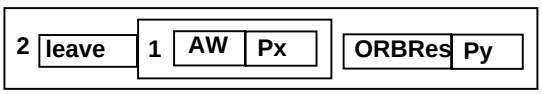
Deferred option (single nesting)

The label “defer” tells the Binder that the marked information is not to be interpreted by the Binder directly but passed to the resolver (whose reference is appended to the marked information) to be resolved.

14.3.2.2 Leave-and-Forward strategy

Where the leave-and-forward method is used, the information is marked with the “leave” label. This tells the Binder that it should be used in the RPC header of the invocation sent to the gateway resolver whose reference is appended to the marked information.

Figure 14.3: Leave-as-is information inside an Interface Reference



Leave-as-is option

The gateway in this case could be a type generic gateway implemented so that it receives the destination of the invocation from the RPC header and passes the invocation on.

14.3.2.3 Short cuts and optimisations

When using the deferred resolution method, a Binder can try and skip gateway nesting of interface references and go directly to the service, provided of course that it has access to suitable communication protocols. There are therefore two cases to consider:

- where the nested records are NOT options, binders must not try to skip a gateway and bind directly to an interface nested inside the structure
- where short cuts can be made, binders may be permitted to take a more direct route (e.g. to prevent circularity of reference), but sometimes the non-functional characteristics of a longer route may be desired.

Whether the nesting indicates a compulsory path or not depends on the type of boundary crossed and the enterprise or management issues associated with the crossing of the boundary. Either way it should be left to the gateway-resolver to decide.

The information marked as deferred can also be encrypted so as to prevent its resolution by unauthorised agents. This can help prevent taking any short-cuts where it is necessary to force the use of a gateway.

14.3.3 Manipulating and passing interface references

Interoperability requires the ability to manipulate interface references and hence some agreement on structure and content of interface references is necessary.

Different processing environments have adopted different kinds of interface references, and this seems likely to continue. Support for interoperability must therefore include the ability for platforms to manipulate, or at least to exchange, foreign interface references.

This can be achieved by allowing interface records and interface references of different types to be nested in a single interface reference, which acts as an interoperable wrapper. Each specific wrapped interface record or interface reference is prepended with a platform flag (e.g. “ANSAware RT” or “OO DCE”), a version and a length, which allows it to be ignored efficiently by infrastructures that do not understand it.

Labels for marking interception options, such as “deferred” and “leave”, are wrapped along with the interface records or references which they mark, so there is no need for these to be globally agreed.

14.4 Recommended interface reference structure

The OMG has defined an interoperable wrapper of the kind described above, called the interoperable object reference (IOR) [UNO 95]. Figure §14.4 is a pictorial representation of this, and partial IDL is given in Figure 14.5.

Figure 14.4: Interoperable Object Reference (IOR)

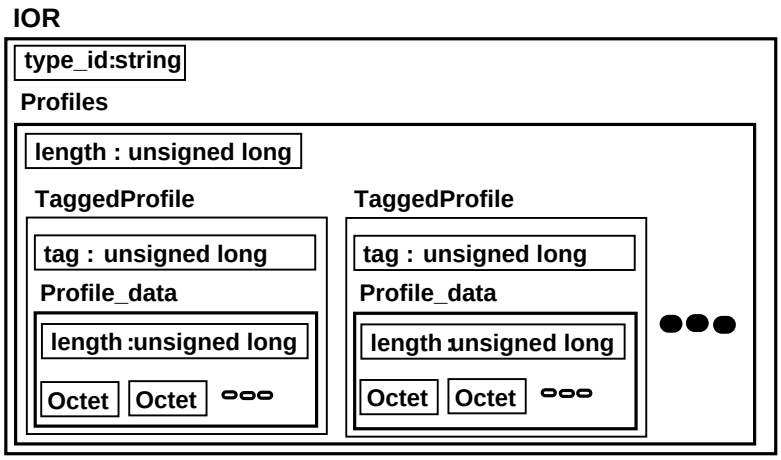


Figure 14.5: IDL for IOR structure

```
module IOR {
     typedef unsigned long ProfileId;

     struct TaggedProfile {
          ProfileId tag;
          sequence<octet> profile_data;
     };

     struct IOR {
          string type_id
          sequence <TaggedProfile> profiles;
     };
};
```

14.4.1 Tagged Profile

The *TaggedProfile* structure leaves the contents of *profile_data* unconstrained. The *profile_data* should hold sufficient information to enable interaction with some object, but this may vary from a complete interface reference to something much more lightweight, e.g. a protocol identifier and an address.

The *tag* should identify the type of the profile, e.g. 'CORBA/IIOP' or 'ANSAware'.

Tag administration should be handled with care. To minimise the overheads of tag allocation, the tag may be divided into:

- tag-foundary field
- tag-family field.

This avoids the need for a single foundary for handing out tags: each foundary, identified by a unique value for the tag-foundary field, is assigned its own subset of the tag space which is called a tag-family, and it may then allocate its reserved space autonomously, without the possibility of contention. Foundaries may be arranged hierarchically if required.

The profile_data includes the length of the profile_data in bytes, which improves efficiency by allowing infrastructures to skip profile_data for records of encoding types which are not understood. It should also include any other information about the wrapped record(s) which is required by specific platforms, such as version numbers or flags to indicate whether the records are for resolvers or gateways.

14.4.2 IOR

This is simply a sequence of *TaggedProfiles*, which will of course include the length of the sequence.

Profiles in a single IOR may hold information describing several interfaces, in arbitrary forms; this allows free interchange of interface references between DPEs.

The *type_id* in the IOR may prove problematic if used incorrectly. It is not thought practical to standardise names for types globally, so great care must be taken to identify types using context relative naming at least, to ensure that type names are not misinterpreted. This might be achieved by:

- prepending the type name with a repository name where the interface reference passed outside the assumed scope of the original repository
- identifying the type using an interface record for a type object in a repository; this could then be resolved into a gateway at a domain boundary if required.

Type names not treated in this fashion should be held in profile_data, as they are effectively type domain specific.

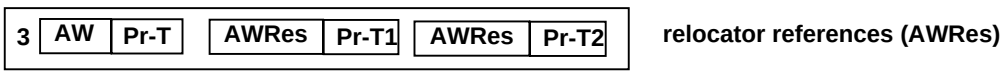
14.5 Additional requirements on interface references

These requirements, whilst not directly concerned with interoperability, are often desired. The solutions used for these requirements may constrain or be constrained by solutions to interoperability requirements, but detailed analysis of the issues has not yet been conducted.

14.5.1 Migration and Relocation

Migration or failure of the server invalidates interface references which referred to it. Interface references can include information that can be used for both detecting the change in location and relocating the interface. This can be represented as a sequence of interface references in increasing order of scope marked accordingly (Figure 14.6).

Figure 14.6: Relocation information in Interface Reference



When an invocation fails, the client infrastructure can request the Binder to intervene. The Binder then searches the interface reference either for an alternative path (in case of comms failure) or for a resolver interface reference (in case the server relocated). The Binder then invokes the Resolver/Relocator giving it the interface reference. If successful, the Resolver/Relocator will return a new interface reference of the relocated service.

14.5.2 Integrity checking

An interface reference contains a nonce that can be used to perform an end-to-end check.

When a request arrives at a server, it is necessary for the server to check that the request arose from the use of an interface reference that was generated by the server, and not by the interface reference of another server. This is necessary because where servers are mobile, the location of a server offers no guarantee about the kind of service or the service instance that is provided. The end-to-end check is performed using the nonce, in the context of the connection that has been established with the help of the address and protocol information.

The server will refuse to establish a connection with a client that cannot quote the nonce for the interface being connected. These failures should be reported to system management since they are indicative of a configuration error in the system. Since nonces are random and generated locally by each DPE they provide an overall statistical integrity check, not an absolute guarantee.

Under no circumstances should the nonce be used as part of any name resolution process.

14.5.3 Replication

The structure of the interface reference should accommodate the possibility of a service being offered by a group [APM.1002 93].

Where a service is transparently provided by a group of servers, perhaps to increase dependability for instance, the interface reference should appear as if a single server is involved from the client application point of view, but contain all the information required to allow the orderly progression of group execution protocols by the client ORB.

This can be achieved by allowing each interface record to hold a list of lists of end-points, one list for each member of the group. The size of the group (its cardinality) is held, along with a count of how many times the membership of the group has changed (its incarnation). There is only one nonce for the group.

The number of replies expected to each request is the cardinality, but the number of requests to be sent may be less than the number of recipients, because some of the recipient addresses may be multi-cast.

The relationship between replication and iterception is beyond the scope of this paper.

References

[APM.1002 93]

Oskiewicz, E. & Edwards, N. “A Model for Interface Groups”, APM.1002, APM Ltd, Poseidon House, Castle Park, Cambridge CB3 0RD U.K., February 1993.

[APM.1003 93]

van der Linden, R., “The ANSA Naming Model”, APM.1003, APM Ltd, Poseidon House, Castle Park, Cambridge CB3 0RD U.K., February 1993.

[APM.1005 93]

Deschrevel, J-P., “The ANSA Model for Trading and Federation”, APM.1005, Architecture Projects Management, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1993.

[APM.1021 93]

Herbert, A.J., “ORB Interoperability”, APM.1021, Architecture Projects Management, Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1993.

[APM.1303 95]

Crawford, B., “Gateway Design and Implementation”, APM.1303, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1995.

[APM.1387 94]

Hoffner, Y. “A Designers' Introduction to Trading”, APM.1387, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1994.

[APM.1392 95]

Otway, D. J., “The ANSA Binding Model”, APM.1392, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1995

[APM.1404 95]

Hoffner, Y. & Crawford, B., “Federation Work Plan”, APM.1404, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1995.

[APM.1427 95]

Hoffner, Y. “The ANSA Interception Model”, APM.1427, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1995.

[APM.1446 95]

Hoffner, Y., “Using the Interception Model to Describe Architectural Issues”, APM.1446, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1995.

[APM.1507 95]

Hoffner, Y., “Interoperability and Distributed Platform Design”, APM.1507, APM Ltd., Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1995.

[REES et al. 95]

Owen Rees, Nigel Edwards, Mark Madsen, Mike Beasley, Ashley McClenaghan, "A Web of Distributed Objects" to appear in the Fourth International World Wide Web Conference "The Web Revolution", Boston, Massachusetts USA, December 11-14, 1995,

[ARM 93]

"The ANSAware 4.1 manual set", Architecture Projects Management, Poseidon House, Castle Park, Cambridge CB3 0RD U.K., 1993.

[BATESON 79]

Bateson, G., "Mind and Nature: a Necessary Unity", Bantam Books, ISBN 0-553-34575-3, 1979.

[BULL 93]

Bull J. A., Gong L. & Sollins K. R., "Towards Security in an Open Systems Federation", European Symposium for Research in Computer Security - ESORICS '92. Springer Verlag Lecture Notes in Computer Science, pp 2 - 20. ISBNs 3-540-56246-X, 0-387-56246-X

[HAMMER & CHAMPY 93]

Hammer, M. & Champy, J. "Reengineering the corporation", HarperCollins Publishers, Inc., ISBN 0-88730-640-3, 1993.

[IONA 93]

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

[ODP 93]

"Open Distributed Processing Reference Model", ISO/IEC 10476, ITU-T Reccomendation X.900 (1995) Parts 1, 2, 3.

[OMG 92]

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

[OSF 92]

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

[SALTZER 84]

Saltzer J. H., Reed D. P. & Clark D., "End-To-End Arguments in System Design", ACM Transactions on Computer Systems, Vol. 2, No. 4, November 1984, pp. 277-288.

[UNO 95]

"CORBA 2.0/Interoperability - Universal Networked Objects", BNR Europe Ltd., Digital Equipment Corporation, Expersoft Corporation, Hewlett Packard Corporation, IBM Corporation, ICL, plc., IONA Technologies, Sunsoft Inc., OMG Document number 95.3.10, March 20, 1995.