ANSA

Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

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

ANSA Phase III

The ANSA Naming Model

Rob van der Linden

Abstract

Many naming systems and standards for naming exist or are being independently developed: every computer system and every application contains or has to deal with several naming schemes at once. The ANSA naming model accommodates the resulting heterogeneity, allows the interconnection of diverse naming systems, and leads to distributed systems in which naming aspects can be effectively managed.

This report (1) defines a naming model, which permits the discussion of federation of separately administered naming systems, (2) explores the relationship between the concepts of decentralised name management in distributed computer systems, and (3) provides examples of the application of the resulting naming model in the analysis of existing systems, designs and standards.

APM.1003.01 **Approved** 15 July 1993

Architecture Report

Distribution:
Supersedes:
Superseded by:

The ANSA Naming Model

Architecture Report

The ANSA Naming Model

Rob van der Linden

APM.1003.01

15 July 1993

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

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

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

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

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

Architecture Projects Management Limited takes no responsibility for the consequences of errors or omissions in this Report, nor for any damages resulting from the application of the ideas expressed herein.

Contents

1 1 Overview

1 1.1 Scope

2 1.2 Approach to naming

5 1.3 Summary of the ANSA Naming Model

6 1.4 Roadmap

7 1.5 Acknowledgements

9 2 Basic Naming Concepts

9 2.1 Name

10 2.2 Naming conventions

11 2.3 Naming domain

11 2.4 Binding

11 2.5 Naming context

11 2.6 Structuring the naming context

13 2.7 Naming network

14 2.8 Name space

14 2.9 Naming system

15 3 The Naming Model

15 3.1 Naming domain

15 3.2 Name set

16 3.3 Naming context

17 3.4 Naming network

19 4 Name Resolution

19 4.1 Common model

20 4.2 A general name resolution model

21 4.3 An alternative model

23 5 Federation and Naming

23 5.1 Origins of federation

24 5.2 Federation and naming

25 6 Naming System Components in Federation

25 6.1 Naming domains

25 6.2 Naming contexts

26 6.3 Naming networks

29 6.4 Name sets

30 6.5 Summary

31 7 Application of the Naming Model

31 7.1 Name service

31 7.2 Interworking and context relative naming

33 7.3 The ANSA Interface Reference

35 7.4 Binding
37 7.5 Relocation
39 7.6 Trading

43 8 Configuring Name Services
43 8.1 Quality of service concerns
43 8.2 Distributing the name service
44 8.3 Configuring name servers

51 9 Summary of Concepts
51 9.1 Introduction
51 9.2 Summary of ANSA naming concepts
52 9.3 Summary of additional concepts

1 Overview

1.1 Scope

Many naming systems and standards for naming exist or are being independently developed: every computer system and every application contains or has to deal with several naming schemes at once. The ANSA naming model accommodates the resulting heterogeneity, allows the interconnection of diverse naming systems, and leads to distributed systems in which naming aspects can be effectively managed.

This report

- defines a naming model based on context relative naming, which permits the discussion of federation of separately administered naming systems
- explores the relationship between the concepts of decentralised name management (federation and naming) in distributed computer systems
- provides examples of the application of the resulting naming model in the analysis of existing systems, designs and standards.

1.1.1 Aims and objectives

The aims of this report are:

- to show that there exists a consistent framework in which relevant concepts can be defined (Chapter 2)
- to demonstrate how this general framework can be specialised to express conformance rules for ANSA (Chapters 3 and 4)
- to illustrate how the framework can be applied to discuss federation (Chapters 5 and 6)
- to exhibit the design decisions with respect to specialisation of the architecture (Chapters 7 and 8)

After studying the material, the reader should be able to:

- design a naming system
- assess the possibilities for the federation of existing or proposed naming systems
- assess the extent to which an existing or proposed naming system conforms to the ANSA naming model

These are the objectives for the reader.

The reader is expected to be familiar with computer systems, and have some knowledge about directory structures in file systems, electronic mail systems and their structure, communications protocols and routing, and some basic operating system principles. No in depth knowledge of naming systems is assumed, as the concepts are introduced from first principles.

1.2 Approach to naming

Names are the principal means of referring to entities. In information systems there are many different entities to be named, and many different ways of naming them. Any large-scale distributed computing system will inevitably involve a range of naming policies.

Naming systems come in many different guises. Parts may be hidden in the structure of application programs, while other parts are present in the infrastructure which supports the application programs. In networked and distributed systems, application level naming is commonly visible through the provision of name services. Most applications, even those which are not distributed, use and combine a variety of naming schemes.

Most existing systems are built upon the assumption that global naming is acceptable in all circumstances[1]. Global naming schemes can provide elegant technical solutions in systems which are under the control of a single authority. However, it proves an unsatisfactory basis for a general solution to the naming problem in large evolving distributed systems which may span many organisations. The problems relate to:

- the need for autonomy within interconnected systems belonging to different administrations
- the need to avoid commitment to centralised management of interconnected systems
- the increasing cost of resolving absolute names as the object universe expands.

ANSA specifies an alternative solution, which protects autonomy: *context relative naming*. This approach also accommodates the existence of many different global naming schemes without resorting to the overly restrictive principles which underpin global naming.

1.2.1 The case against compulsory global naming

The co-existence of many different global naming schemes, leads us to ask why this should be the case? Surely, if global naming worked, there should be one instance of one (global) naming scheme.

Many systems employ a so called “global naming scheme”. In such a scheme anyone using a name for a particular entity is restricted to using a *unique name* or *absolute name*[2] for that entity. Such names are always the same, no matter when they are used, from where or by whom.

Some designers actively promote the use of unique or absolute names. Amongst the reasons are ease of use, and ease of implementation. Users will be able to use a name from anywhere in the system and get to the entity they require. This implies that the context in which names are resolved is invariant. In loosely coupled distributed computer systems which allow evolution and which scale to large size, such invariance cannot be achieved for the reasons set out below.

[1] In a global naming system, a name for an entity is the same, no matter who uses the name, when, or from where.

[2] Chapter 9 contains a glossary which defines the terms.

1.2.1.1 *Agreements*

For unique or absolute names to work, there has to be an agreement that the chosen names are the names that should be used for the chosen entities, and this agreement has to be achieved in the context of the whole system. This works in small systems and also in larger systems which are under the strict administrative control of a single entity, such as a country's PTT (under the umbrella of CCITT). In systems which consist of parts that are under separate administrative control, it is becoming increasingly difficult to achieve universal agreement on the use of particular names. As distributed systems evolve, the chances of interconnecting two systems with clashing naming policies increases, and the opportunities for resolving such clashes decreases.

1.2.1.2 *Commitments*

Once agreement on unique or absolute names has been reached, there needs to be a firm commitment by all parties involved to adhere to the agreement under all circumstances. There can be no in-built safeguards against a breakdown of the agreement. Should a breakdown nevertheless occur, this would result in failure of the whole system or significant parts of it.

1.2.1.3 *Recognised authority*

With the agreement and commitment in place, there is a need for the creation, funding, and trusting of a single centralized name allocation authority. Although such schemes are possible in a limited context (cf. Internet addresses), there are severe doubts whether a single agency for all naming would be politically possible. Its central role would place it in a position of extreme power. This will be unacceptable by many parties, leading to the rapid breakdown of any commitments or agreements on a global naming scheme.

1.2.1.4 *Enforcing the law*

What could be done in the event of a rival scheme appearing despite the agreements, commitments, and recognition of authority alluded to above? Would there be a black market in naming systems? There would have to be some enforcement of effective sanctions against rival schemes. It remains difficult to see what those sanctions might be, or how they might be implemented.

1.2.1.5 *Conclusion*

The conclusion must be that global naming schemes can only work in some specialised circumstances, in closed systems, or in systems which are strictly controlled by a single administration.

1.2.2 The case for context relative naming

Given the problems identified above, it is likely that the imposition of a global naming scheme will fail. Indeed, if history is anything to go by, that lesson should have been learned already. As a result, a world-wide distributed system, composed of multiple subsystems, each based on global naming, will contain several roots to the naming tree; a situation which is already in evidence. Suppliers with a flexible naming architecture have considerable commercial advantage over those offering global schemes, since only they can provide applications which can both use and supply services to all other systems. Global name users remain locked in their respective subsystems. Those offering more flexible naming systems can thus obtain a disproportionate share of new business, squeezing out those supporting global naming schemes.

The alternative to a global naming scheme is a strictly context relative naming scheme where each name is *bound to an entity in a particular context*. Global naming can be regarded as a special (restricted) case of context relative naming, since global naming assumes that the context in which a name assumes its validity is the whole system. As described above, when the system is large or evolves, the maintenance of a single system wide context becomes unmanageable.

1.2.2.1 Retaining autonomy

The administration of a distributed system which spans the boundaries of companies and countries will not be coherent and consistent. Each entity, whether country, company, or division, will insist on retaining control over its resources, including naming resources. The administration of a large distributed system will thus comprise a number of independent administrative domains in a decentralized management structure. The administrative domains have very well defined boundaries which are carefully policed by any organisation that wishes to *survive* its connection to other systems.

1.2.2.2 Agreements and commitments

The controls on the boundaries with other administrative domains imply a restriction on the visibility of, and access to, the resources in the domain. This can be translated into a relaxation of the constraints on the requirements for unambiguous naming: names now only have to be unambiguous within the administrative domain that controls the named resources. As a consequence each administrative domain can coincide with a naming context. The chances of reaching agreement on and ensuring commitment to naming policies in small communities are much better than in large.

1.2.2.3 Multiple roots

A system based on context relative naming tolerates the existence of multiple naming “roots”. Existing naming systems, based on global naming can thus be accommodated. As more of these roots appear, the notion of a naming root disappears and a root becomes a node in the naming network, just like all other nodes.

As names are transferred from one part of a system to another, name translations may be required when passing from one naming sub-system to another. This is the price that has to be paid in return for greatly relaxed requirements on consistency of names. Name translation at boundaries does however scale much better than insisting on consistency of names in the whole system.

1.2.2.4 Conclusion

The structure of interconnected naming contexts can reflect the structure of the organisations it serves, leading to a flexible naming structure, which is able to respond to changes in its environment.

Context relative naming systems have much better scaling characteristics than global naming systems.

The ANSA naming model is based on context relative naming.

1.3 Summary of the ANSA Naming Model

1.3.1 General naming concepts

The components that make up a naming system each serve to provide an answer to one of the four questions that any naming system should answer:

- the *naming domain* provides an answer to the question: "Which things can be named?" and so defines the universe of discourse for a naming system [HAUZEUR86]
- the *name sets* define one or more collections of names that can be used in the naming system, and answer the question: "Which names are allowed or prescribed?" [COMER87]
- the *naming contexts* bind particular names to particular entities. In one naming context, all names must be drawn from one name set; all entities must be members of a single naming domain. Naming contexts help provide answers to questions like: "What are the names for any particular thing or set of things?" [SALTZER79]
- the *naming network* expresses the constraints on the extent to which entities that have names in one naming context can be referred to from other naming contexts. It helps answer questions like: "Can any one thing name any other thing or are there restrictions?" [COMER87]

1.3.2 ANSA naming rules

The naming model has been defined by placing constraints on each of the components of a naming system. Such constraints must not impede the extensibility of a system in which the naming system is implemented. In particular, where several naming systems have been developed in the same architectural framework, the resulting systems should interwork. Naming systems must also allow a variety of name generation and resolution processes. The ANSA naming model can be summarized by the following rules:

- the architecture places no constraints on what can be named
- the architecture places no constraints on the nature of the symbols that can be used to construct names
- all naming shall be context relative
- the architecture places no constraint on the structure of the naming network
- all names shall be *structured* and consist of a *handle* and an optional *remainder*. When resolving a name, and if a remainder is present, it should be resolved in the context identified by the handle. If there is no remainder, the handle is a name for an entity
- there shall be no assumption about a single outermost (i.e. global) context. In particular no optimization may result from this assumption, since doing so will severely limit the extensibility of the system.

The justification for these rules can be derived from the case that was made against global naming schemes.

1.4 Roadmap

This report is logically divided into four parts:

- this overview
- naming model and theory
- federation and naming
- examples.

For each part, a short overview follows.

1.4.1 Overview

Chapter 1 (this chapter) contains a general introduction and a roadmap to help the reader navigate through the rest of the document. It explains the background to the work, the aims and objectives, as well as the assumptions that have been made about the readership. A summary of the ANSA Naming Model is included.

1.4.2 Naming model and theory

The emphasis in Chapters 2, 3 and 4 is on the development of the naming model. This model is a general framework of naming concepts which allows designers to reason about the applicability of existing and proposed naming systems in the context of ANSA. It should facilitate discussions about the interconnection of systems in which different naming models apply, and assist in leading to an efficient implementation of the combined system. For this to be possible, the terminology must be well understood. Rather than setting out to redefine terms, representative research papers have been consulted. A summary of concepts can be found in Chapter 9.

Chapter 2 defines the basic concepts as the components out of which a naming system is built: the *naming domain*, the *naming conventions*, the *naming context*, and the *naming network*. Each component is defined and a justification for each definition is given.

The ANSA Naming Model is defined in Chapter 3. It is constructed by placing (architectural) constraints on each of the components of the naming system. These constraints are partitioned into *rules* (the mandatory constraints), *recipes* (the optional constraints that illustrate how to achieve certain effects), and *guidelines* (statements of “good practice”).

Chapter 4 discusses name resolution in the context of the ANSA naming model. Name resolution allows the recovery of an entity given its name. All computer systems require this basic functionality. Many implementations exist. The aim is to expose the underlying principles, rather than describe a particular mechanism.

1.4.3 Federating naming systems

Chapters 5 and 6 describe the concept of federation in the context of naming systems and their components. The origins of federation are set out in Chapter 5, and the essential characteristics of federation are exposed. In Chapter 6, the relevance of federation to each of the naming model components is described.

1.4.4 Examples

Chapter 7 illustrates the use of the naming model in other parts of the architecture and with respect to existing systems, such as OSF DCE [OSF92a] and standards, such as CCITT X.500 [ISO 9594].

Chapter 8 describes several name server configurations and considers the quality of service characteristics of each configuration. References to existing systems and standards are included.

1.5 Acknowledgements

The author acknowledges contributions from throughout the industry. Valuable contributions were also received from all members of the ANSA Team, in particular Alastair Tocher, when seconded to the ANSA Team by BNR Europe Ltd., and Dave Otway, previously seconded to the ANSA Team by GEC Marconi Instruments Ltd., now with Architecture Projects Management Ltd.

2 Basic Naming Concepts

The basic concepts described in this chapter are *generic*, that is they are valid in many fields of application, including computer systems[1].

2.1 Name

A *name* is a linguistic entity, that singles out a particular entity from among a group of entities [HAUZEUR86], [COMER87].

A name may be a substitute for the action of pointing. When two parties are in direct visual contact, pointing at entities would be a viable alternative to naming them, subject to the availability of the entities. When there is no such visual contact, or when the entities in question are not available to be pointed at, communications about the entities may be continued by use of mutually acceptable linguistic entities or “names”. *Naming conventions* define the acceptable forms of names. If two or more parties agree that a linguistic entity **x** *denotes* an entity **y** then **x** may stand as a *name* for **y** [COED82]. Simultaneously, all parties agree that the entity y is distinguishable from everything else [BASRI66]. They have therefore agreed some part of their respective models of the world[2]. A naming domain defines the set of distinguishable entities that can be named [HAUZEUR86].

The above concentrates on a particular use of names: two parties use a name to communicate about another entity. Such a name is known as an *attributive name*; the name has been attributed to the entity about which the interaction takes place. Names may also be used to address a particular entity. Such names are known as *invocation names*; they are used to invoke some reaction from the entity that is addressed. The distinction between this different usage of names is often not made. Doing so allows the distinction of two related naming systems. The attributive names are used at the application level, the invocation names in the infrastructure, which mechanises and mediates the interactions between the application entities. In a heterogeneous distributed computer system which adheres to context relative naming and which allows invocation names for entities to be passed around, this distinction allows the specific problems of addressing, relocation and name translation to be dealt with separately from application naming issues.

[1] A summary of concepts appears in Chapter 9.

[2] The conditions for which this statement is regarded to be true in any particular situation are difficult to express. They are the subject of much philosophical debate. This is clearly outside the scope of this work.

2.2 Naming conventions

Choosing to name entities in a particular way is known as adhering to a *naming convention*. A naming convention is a predicate that defines a *name set*, the set of names that are considered valid in a particular naming system. The choice of a naming convention is governed by an extremely complex process which takes account of both the environment in which those using the names operate and of their background[1]. In practice, and in computer systems in particular, the cultural differences between those that generate names and those that resolve them are small. A workable agreement on naming conventions can thus be reached.

Different sets of constraints can be imposed on the process of constructing valid names. A particular group of people may decide that their workstations are to be named after plant species, such as Primrose, Hawthorn, Poppy, and so on. They may decide that their common file services are to be named after department stores, such as Libertys and Harrods. One may also decide that names are to be structured in some way. For instance,

usr/tmp/wk.c,

C:\pctcp\temp.log

apm@ansa.co.uk, and

192.005.254.006

are names that have been structured to reflect certain concerns. All these names were formed by applying different constraints to ordering of some set of characters.

Names and naming conventions may be chosen so that the names evoke some feeling for what they denote. In many filing systems for instance, a dot-extension is used to differentiate between various sorts of files. For instance, in a C-programming environment, it is common to use the following convention:

xyz.c for source files,

xyz.o for object files,

xyz.h for header files, etc.

If all programmers adhere to this convention there are few communication problems between them. Programmers adhere to this convention because they have all been conditioned to understand the development environment in the same way. They know that if they deviate, chaos will ensue.

Names and naming conventions can be chosen so that it is hard to attach a meaning to the name. This may be done for security reasons, or to avoid a situation in which people start to rely on certain names, thus reducing the scope for name management.

[1] The nature of this process is outside the scope of this paper. It is the subject of philosophical argument [PLATO348BC]. With respect to the extremist views on the significance of a naming model we quote from Klein and Hirschheim [KLEIN87]:

> “The difference is whether one *believes* that the model *reflects* reality or consists of subjective meanings and thereby *constructs* reality.”

2.3 Naming domain

A *naming domain* defines the set of distinguishable entities that can be named [HAUZEUR86]. It forms the universe of discourse in which names may assume a meaning.

A naming domain may be defined by set enumeration or by a suitable predicate. The definition for any particular domain may however be misinterpreted, thus leading to potential confusion over the entities in the domain. It is impossible to guarantee that there will be unambiguous agreement on the naming domain. Although people seem to cope admirably, designers would do well to be aware of potential problems in this area.

2.4 Binding

If **x** may stand as the name for a given entity, then there exists a *binding* from **x** to that entity. A binding is the association of a name with a particular entity. Following Saltzer [SALTZER79], the activity of associating a name with an entity is also called binding. Examples of names which may be bound in computer systems are an interface identifier, a port or socket identifier, and a block or sector number as a name for a space on a disk.

2.5 Naming context

A *naming context* is defined as a particular set of bindings of names to entities [SALTZER79]. Whilst a naming domain defines the entities that can be named, a naming context adds the names for those entities.

A naming context can thus be seen as a relation between a set of entities (the naming domain) and a set of names (the name set).

2.6 Structuring the naming context

With the concepts defined so far it is possible to construct one closed naming system with a single naming context. Being limited to these concepts alone would require agreement on a universal naming domain, a universal name set, and a universal mapping between the two. Many independently developed naming systems have already been implemented and many more are being proposed. The ANSA naming model must accommodate these different naming systems and allow their interconnection. A naming system with a single naming context would not support the necessary heterogeneity, and cannot be managed.

Even in a very large naming context naming must remain unambiguous, and the set of names must therefore be equally large. Each new binding which is added must be checked for consistency with all existing bindings. The time required to add a binding will thus become related to the number of bindings already held, making the availability of the binding process unpredictable. The performance of the binding process may be increased at the cost of reduced consistency; some names may be used more than once and name resolution becomes ambiguous.

Structuring the global context into a number of smaller contexts solves many of these problems. Name generation and management of consistency of names become tenable problems. Existing naming systems may be regarded as separate naming contexts.

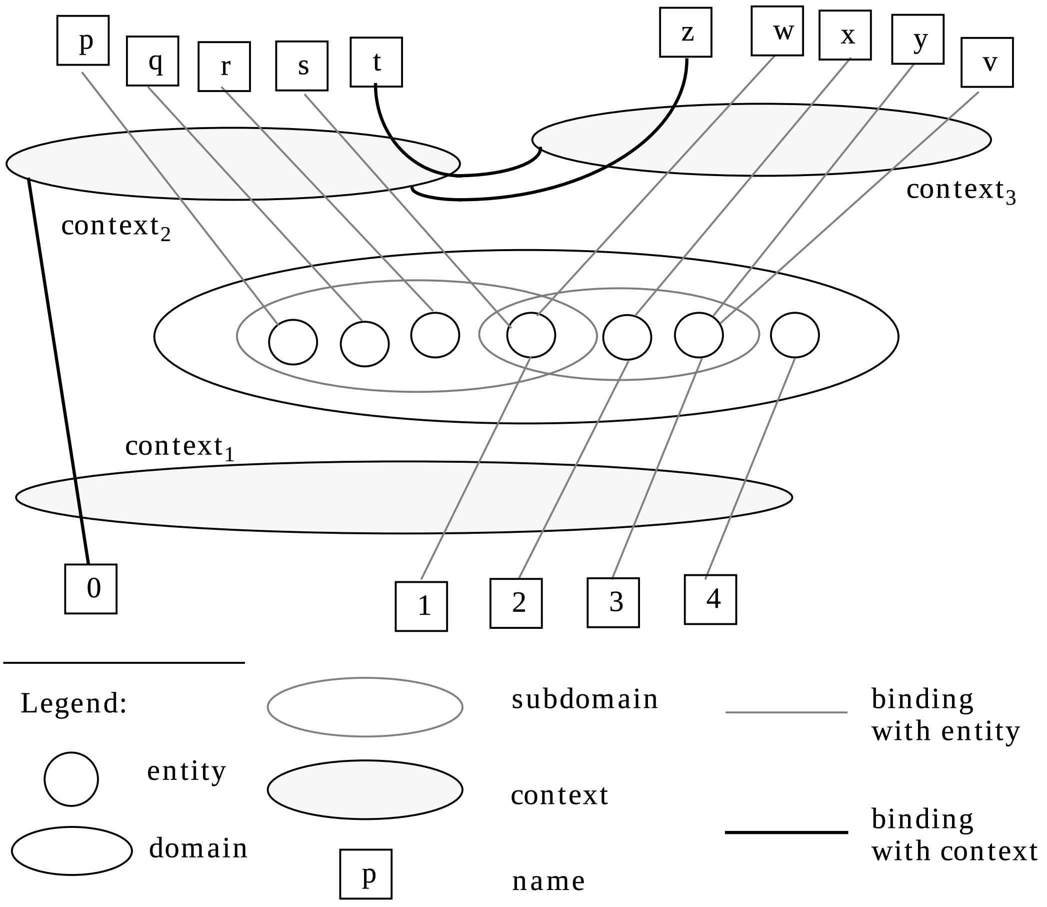
The resulting contexts can be related to one another, so that it is possible to refer to an entity in another context. This implies that in addition to binding names to entities, a naming context must allow the binding of names to other naming contexts. Since a naming context is something of interest, it is an entity and can be named.

2.6.1 Application of structured contexts

For organisational reasons, large domains are often broken up into manageable parts. The sub-sets are called *subdomains*. In each subdomain, naming can be carried out independently of any other subdomain. For each subdomain there may exist a naming context which defines the bindings between the entities in the subdomain and their names.

Despite the division in subdomains it may at times be necessary for an entity in one subdomain to name an entity in some other subdomain. To denote an entity in another subdomain would require a name for the entity and the identification of the context, associated with the target subdomain, in which the name resolves to the entity.

Figure 2.1: Example of a naming domain with subdomains and contexts



A naming context associated with a particular subdomain can therefore be described as a view onto a larger naming domain. The entities within the subdomain are bound to names. Other naming contexts may also be bound to names. Some naming contexts may be left unnamed and entities in associated subdomains cannot then be identified. Figure 2.1 shows an example of a naming domain and how its entities and subdomains relate to naming contexts.

In the example of Figure 2.1, a naming domain has been split into naming subdomains, each containing a number of entities. Each of the naming subdomains has been associated with a naming context. Entities are bound to names in contexts. Table 2.1 shows how the names for the entities vary with the context in which the names are resolved.

Table 2.1: : Context relative names for entities in Figure 2.1.

name from context$_1$	name from context$_2$	name from context$_3$
0.p	p	z.p
0.q	q	z.q
0.r	r	z.r
1 or 0.s	s or t.w	w or z.s
2	t.x	x
3	t.y or t.v	y or v
4		

From context$_1$ four entities can be named directly, and four indirectly via context$_2$. From context$_2$ four entities can be named directly, and a number of other entities can be named via context$_3$. From context$_3$ three entities can be named directly, and three others via context$_2$. Note that there are entities with more than one name. In context$_3$ one entity has two names, “y” and “v”. In context$_2$ one entity can be denoted by the names “s” and “t.w”. These names denote the same entity in the same context and are known as *aliases*.

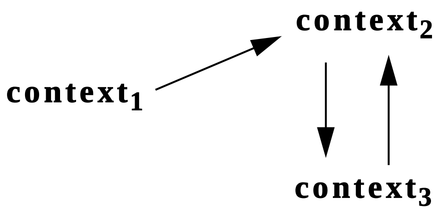
2.7 Naming network

A naming network is a description of the restrictions on the extent to which entities that have names in one context can be named from other contexts. The naming network can be seen as a directed graph, where each vertex denotes a context and each edge denotes a binding with a context. The naming network identifies the way in which naming contexts have been organised and are related to one another [COMER87].

The naming network for the example in Figure 2.1 is illustrated in Figure 2.2. It illustrates that context$_2$ can be reached from context$_1$, and that context$_2$ and context$_3$ can refer to one another.

Figure 2.2: The naming network of the example of Figure 2.1.

.



2.8 Name space

Potentially, every naming context in a naming network may define its own naming conventions. In practice however, different naming conventions are agreed in different regions of the naming network. A *name space* is that part of a naming network within which a single naming convention is used [COMER87], [MOCKAPETRIS88], [SCHWARTZ87]. Within a particular name space, all names are drawn from one name set.

2.9 Naming system

A *naming system* consists of

- a *naming domain*, which defines what entities that can be named
- one or more *name sets*, each associated with particular *naming conventions* used in a part of the naming system
- a set of *naming contexts*, that determine what actual names have been chosen for the entities in the naming domain
- a *naming network*, that places restrictions on the extent to which entities can be named from each of the naming contexts.

3 The Naming Model

The naming model states the architectural constraints imposed on each of the components of a naming system (introduced in Chapter 2). The naming model gives guidance to implementors in the construction of ANSA-conformant naming systems.

In this chapter a set of rules and recipes are stated for each of the components of a naming system. Guidelines for their use are provided where appropriate.

3.1 Naming domain

3.1.1 Rules

1. The architecture places no constraints on what may be named. Any thing, of any type, including things external to a system, can be named.

3.1.2 Recipes

1. Multiple domains:
   The set of entities in a distributed system is scattered across a number of domains. There are many reasons for the introduction of multiple domains. For instance, domains may be used to delineate authorities; to distinguish areas in which different security, management, or naming policies apply. Where multiple domains arise, different relationships between domains may exist. There is no requirement for different kinds of domain to coincide with regards to their membership.
2. Administrations in hierarchy:
   To reflect the administration sub-administration relationship, a domain may be a sub-domain of another domain. The sub-domain may have further sub-domains. In a proper hierarchy, the domain/ subdomain relationship is transitive.
3. Administrations in federation:
   When two administrations come together in federation, their domains remain disjoint.

3.2 Name set

3.2.1 Rules

1. The architecture places no constraints on the nature of the symbols that may be used to construct names.
2. All names are *structured*. All names consist of a *handle* and an optional remainder. If the remainder is present, then the handle is a name for the naming context in which the remainder is to be resolved. If the remainder

is not present, then the handle is a name for (resolves to) the thing named (this may be a naming context).

3. One may not assume that there exists a single outermost (i.e. global) context. In particular no optimization may result from this assumption, since doing so would severely limit the extensibility of the system.

3.2.2 Recipes

1. A name as an aid to interaction:
   A name may be required in the interaction with some entity. The name is used to distinguish the entity that is the target of an interaction from all other entities. Such a name is known as an *invocation name*. An invocation name conveys an opportunity (not a right) to interact with the entity denoted by that name. In the computational model, for instance, interfaces, operations and terminations are given invocation names.
2. A name as an attribute:
   A name that cannot be used to interact with an entity, may be used to denote that entity. An *attributive name* is used in the interaction between two entities to refer to a third entity, which is itself kept outside the interaction pattern.
3. A name as an entity:
   A name can itself be seen as an entity of interest in a system. In the information projection for instance, a customer name is treated as a particular piece of information.
4. A name as (part of) a predicate:
   A name may be used as part of a predicate. For instance in “the papers written by Saltzer”, Saltzer is a name. In query languages, predicates are formed to specify a set of responses. In constructing the query, names may be used.

3.2.3 Guidelines

1. Naming conventions may specify which parts of a name are considered handle and remainder. This influences the way in which name resolution proceeds with respect to a given name: left to right as in /usr/mrx/tmp/bin; right to left as in rvdl@ansa.co.uk; or in some other fashion.
2. Individual systems may each impose a different constraint on the symbol set that may be used to construct names. Where entities in one system can be referred to from another system a name translation might be required.
3. A name set is defined by a predicate known as a naming convention. In any naming system more than one naming convention may be in use simultaneously. Thus several name sets may coexist.

3.3 Naming context

3.3.1 Rules

1. All naming must be relative to a context.
2. For any one naming context, all names must be chosen from a single name set, and all entities must be chosen from a single naming domain.

3. *Homonyms* are not allowed as invocation names. In a particular context, a particular invocation name may be bound to at most one entity.

3.3.2 Recipes

1. Version control:
   Naming contexts are often dynamic, that is the set of bindings is subject to change. This change may be caused by the definition of a new binding and/or the deletion of an existing binding. There are two views on dynamic naming contexts. In the first, changes in the bindings cause the existing naming context to be replaced; the old context is no longer available. In the second view, changes in the bindings cause the creation of a new naming context, and the old context remains available. Both naming contexts have to be named to distinguish between them. Version management uses the latter view. When a change takes place, a new version is created, the old version remains.
2. Unresolveable names:
   Not all names in the name set have to be bound to an entity. Names that are unbound, but nevertheless used are *unresolveable*.
3. Alias:
   In any particular naming context one entity may have more than one name. Such names are known as *aliases*.
4. Synonym:
   A particular entity may be named from several naming contexts, using different names. Such names are known as *synonyms*. Note that name resolution does not have to start in the same naming context as for alias.

3.4 Naming network

3.4.1 Rules

1. The architecture places no constraints on the structure of naming networks.
2. Only context-relative naming is possible.

3.4.2 Recipes

1. Path names[1]:
   The structure of a naming network can be described by a graph **G**=(**V,E**). Each vertex **V** denotes a context and each edge **E** denotes a link between two contexts. The edges are directed and labelled. The label represents the name that an entity in the source context (at the base of the directed edge) uses to identify a particular context. The link is with the target context (at the pointed end of the directed edge). Name generation is the process of generating a list of primitive names that taken together describe a path through the directed graph **G**, from the context in which the source entity resides to the target context. In that way the target context is singled out from amongst all possible target contexts in the graph. The graph **G** need

[1] This recipe is adapted from a model presented by Comer and Peterson [COMER 87, COMER 89]. They limit their use of the model to physical naming networks (Recipe §3.4.2 - 2).

not be fully connected. The connection matrix is often sparse, and in that way it specifies the constraints on the extent to which source and target entity pairs can be formed.

2. Physical path names:
A physical naming network is a naming network that is isomorphic to the perceived physical structure of a distributed system. A distributed system consists of a number of interconnected components or nodes. The system is modelled by a directed graph **G**=(**V**,**E**). Each vertex in **V** denotes a node in the system. Each edge in **E** identifies a connection between two nodes in **V**. For example, some electronic mail systems reflect the structure of the network that connects machines in the way in which names are composed and resolved.

3.4.3 Guidelines

1. Soft link:
A soft link (or symbolic link) is a name which refers to the name of an entity. To get to the entity, the first name is resolved to the second in context C_{N1}. The second name is resolved to the entity in context C_{N2}. C_{N1} and C_{N2} are often, but not necessarily in the same the naming network and, strictly, two naming systems are therefore involved.

2. Location transparent naming:
Location transparent naming requires that a name for a particular entity is independent of the location of the target entity. When the path name reflects the physical structure of a system, location transparency is not provided if such names are visible to an application. A soft or symbolic link to the full source routing name can be inserted between the application and the system that resolves the name to achieve the required location transparency.

3. Migration transparent naming:
Migration transparent naming requires that a name for a particular entity is independent of the change of location of the target entity. When the path name reflects the physical structure of a system, migration transparency is not provided if the changes in location of the target entity are reflected in changes in the name. To make this invisible, a transparency layer can be inserted that will map a name to the new source routing name, each time the target entity migrates. Note that the soft link can be the proper source routing name to start with, and only become a proper soft link when the target entity migrates for the first time.

4. Consistent or uniform naming:
End users frequently require that a name for a particular entity is the same from the set of contexts from which the entity is most frequently named by a user. In that way the entity will be known by a single name. In each context a soft link may be established. The binding between the soft link and the full source routing name can be created above each context separately or implemented through a new shared context (sometimes referred to as a pseudo-root).

4 Name Resolution

When knowledge about an entity is to be communicated, without the presence of the entity, a name must be generated for that entity. In a particular naming system, the name must be taken from the name set and the entity must be a member of the naming domain. Name generation can be characterised by a relation between names and the entities [HAUZEUR86]. This relation is the naming context of the generator. The receiver of a name resolves that name with reference to its own name set and naming context. Information transfer is only successful when the naming domains of generator and receiver overlap or coincide.

As a general rule, name resolution must take place in the same name space as the generation of the name. If this is not the case, then there must exist some process that knows about both the generator's name space and the resolver's name space, such that the name may be suitably transformed (for instance from ukc!acorn!ansa!xyz to xyz@ansa.co.uk). A mechanism that can span two or more name spaces is called a *naming bridge*.

This chapter is concerned with the generic process which characterises all name resolution. Consistency and availability of name resolution are important issues which affect the distribution of name resolution processes. Distribution strategies are described in Chapter 8.

The common name resolution model adopted in most systems is presented in §4.1. In §4.2 this model is improved, eliminating asymmetries associated with starting and terminating conditions and providing for the explicit inclusion of naming conventions. Finally, §4.3 briefly presents the name resolution model in terms of a state machine.

4.1 Common model

A name resolution model that is commonly applied in computer systems uses structured names and context relative naming but has specialised starting and terminating conditions. At any stage of name resolution, a structured name has a handle and an optional remainder. The name resolution process separates the handle from the remainder and uses the handle to determine the context in which the remainder is to be resolved. Resolution of the remainder is done in the same way. At this level the remainder, as a name, is decomposed into a handle and a remainder etc. Each step in the name resolution process may be performed at a different site; each context is potentially remote from every other context.

The resolution process thus takes a name and a context and produces a new name (the remainder) and a new context (derived from the handle). Since a naming network contains a finite set **C** of naming contexts, for any particular name space, a transition relation **T** may be specified:

T: C X N <=> C X N

where **N** denotes the name set and the relation **T** is:

1. partial because not all context-name pairs need to be resolvable;
2. many-to-one because a context can be reached from more than one starting context;
3. not onto because the naming network is not completely connected.

Although name resolution can be represented by a simple transition relation, it is not clear from this alone when and how name resolution starts, and when and how it terminates.

4.1.1 Starting and terminating name resolution

To start name resolution there is a requirement for a name and a context in which the name may be resolved. In practice however, there is only a name, generated by some entity in the system. To obtain a starting context, every entity that can refer to other entities by name, must be associated with a context in which those names are bound [SALTZER79]. A total closure is defined, that produces a context from an entity and a name:

closure: D X N <=> C

where **D** denotes a particular naming domain, **N** a name set, and **C** a set of contexts in a naming network. The closure relation is:

1. total because every entity must be associated with a context in which resolution can start;
2. many-to-one because several entities may be associated with the same starting context;
3. not onto because there may be contexts which are empty.

A name resolution process terminates when **T**(c,n) is undefined, that is, when the name cannot be decomposed, i.e. when there is no remainder.

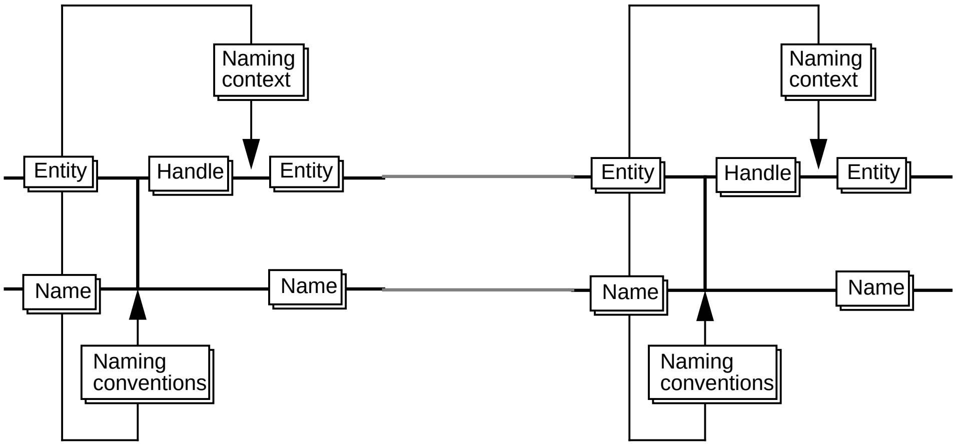
4.2 A general name resolution model

There are two shortcomings of the common model. First, name resolution is irregular when starting (when a context has to be derived from an entity) and when terminating (when the transition relation is undefined). Second, the common model does not make the naming conventions, which are needed to generate a handle and a remainder from a structured name, explicit.

A more general model can be constructed that incorporates the starting and terminating conditions and that makes the requirement for naming conventions at every stage of name resolution explicit. Figure 4.1 illustrates the generic component; it provides a single step in name resolution.

Each name resolution step starts with an entity and a name. From the entity and the name, the naming conventions and naming context are derived. The naming conventions are used to derive a handle and a name (remainder) from the name. The handle and the naming context deliver a new entity for the next step. The remainder is the new name for the next step.The generic name resolution component can thus be cascaded as illustrated.

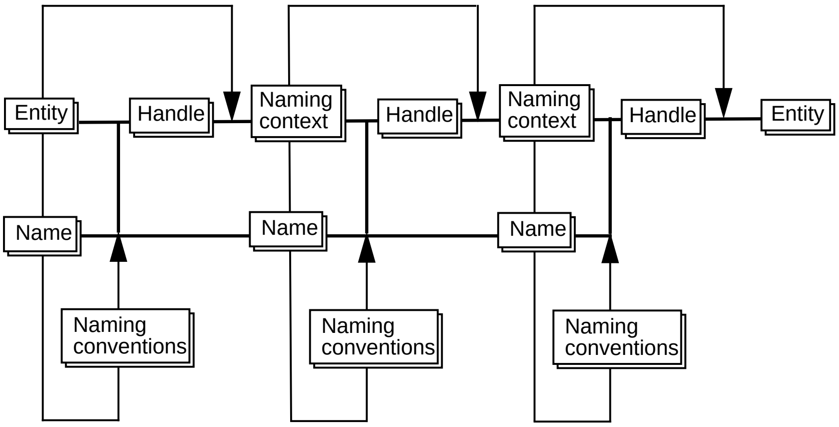
Figure 4.1: The general name resolution component



4.2.0.1 *The common model revisited*

At each stage the handle denotes an entity. This can be an entity in the naming domain or another naming context, as in the common model. The common model of §4.1 can be derived from the general model in Figure 4.1, as illustrated in Figure 4.1.

Figure 4.2: The common name resolution model in cascade



4.3 An alternative model

An alternative model allows the context graph to be processed by a state machine (Mealy Automaton). The graph is then translated into a state transition table, where the vertices are the states, and the directed edges the allowable state transitions. This model is logically equivalent to the model presented above.

Despite viewing the context graph as a data structure (the state transition table), the model does not preclude distribution. The state machine may be replicated, and the state transition table may be partitioned and distributed across the nodes of a distributed system for instance. Chapter 8 compares several options for the distribution and configuration of the context graph.

5 Federation and Naming

5.1 Origins of federation

The concept of federation in computer systems was introduced by Heimbigner and McLeod and was based on the federation of information bases [HEIMBIGNER81]. Their approach was directed to achieve coordinated sharing and interchange of computerised information, rather than relying on centralised control by a centralised (logical) organisation. Federation appears central to the construction of systems which allow evolution and can operate in heterogeneous environments with independent administrations. Heimbigner and McLeod identify four principles that characterise a federal approach to interconnecting computing components [HEIMBIGNER81]:

- *“A component must not be forced to perform an activity for another component. The role of centralised authority must be replaced by cooperative activity among the components supporting protocols.”*
- *“A component must have 'freedom of association' with respect to the federation. Since the federation is a dynamic entity, components must be able to dynamically enter or leave the federation. Further, a component must be able to modify its shared data interface, adding new data and withdrawing access to previously shared data.”*
- *“Each component determines the data it wishes to share with other components. Since partial, controlled sharing is a fundamental goal of the federated approach, each component must be able to specify the information to be made available as well as specify which other components may access it and in what ways.”*
- *“Each component determines how it will view and combine existing data. In a composite system, all access to the underlying data is mediated by a global schema. In a federation, each component must be able to, in effect, build its own “global” schema that is best suited to its needs.”*

As computer systems grow and evolve, it is likely that several systems which were once regarded as quite separate become linked. For example, computer systems have been separately developed and installed in different departments. If these departments are to closely cooperate with one another, then this cooperation might well result in a coupling between their computer systems. Economic factors clearly prohibit the development of a completely new system which can serve all departments, so an evolutionary approach is appropriate. Furthermore, the cooperation between two or more organisational entities will be subject to numerous constraints and from time to time it will be necessary to review and modify these.

It is not appropriate to rely on a central authority which may impose policies and implement any restrictive mechanisms, since no such body exists and mutual mistrust will prevent such a body ever being formed. The solution must be sought in a cooperative form which will allow all parties to control the extent to which they share their information. Such a cooperative form is called a federation.

5.2 Federation and naming

There are two ways in which the concept of federation can be discussed in relation to naming. The first emerges with the recognition that information is itself named; the second is related with the information structures which are required by the name resolution process.

5.2.1 Named information

Information is structured and parts of the structure are given names. The naming of information proceeds within the framework of a naming system.

When federating information systems, the naming systems that have been employed to name the information must also be reconciled since the way in which naming can be carried out in different systems can vary considerably. Different meanings may be attributed to some names. Different naming conventions may apply, resulting in different name sets. The name resolution processes may differ, and different approaches may be taken to the provision of location and migration transparencies through the naming system.

Chapter 6 relates federation to each of the components of a naming system.

5.2.2 Name services

The second concern is related to the fact that the name resolution process requires data and can thus be seen as an information system in its own right. The service which performs name resolution is known as a name service, and is performed by the name server object. Several of these objects can thus be federated as suggested in §5.1. The universe of discourse for a name service in a distributed system is the set of services that are available in that distributed system. Name services hold information that represents the relation between attributive and invocation names. The information can be held in a database and each name service exercises some authority over the information it holds. When federating systems the databases that hold the information that reflects the naming system in the universe of discourse must also be federated. The name services will be expected to share information, without necessarily giving up their control over the information.

The different ways in which name services can be configured will be illustrated in Chapter 8. One configuration is typical of a *federation*.

6 Naming System Components in Federation

In this Chapter the relevance of federation to each of the components of a naming system is described.

6.1 Naming domains

The naming domains of the naming systems that are federated are disjoint before federation and remain disjoint during and after federation.

The naming domains are disjoint since the entities in each domain are citizens of separate and autonomous administrations. Having entities in more than one domain contradicts the notion of separation and autonomy.

6.2 Naming contexts

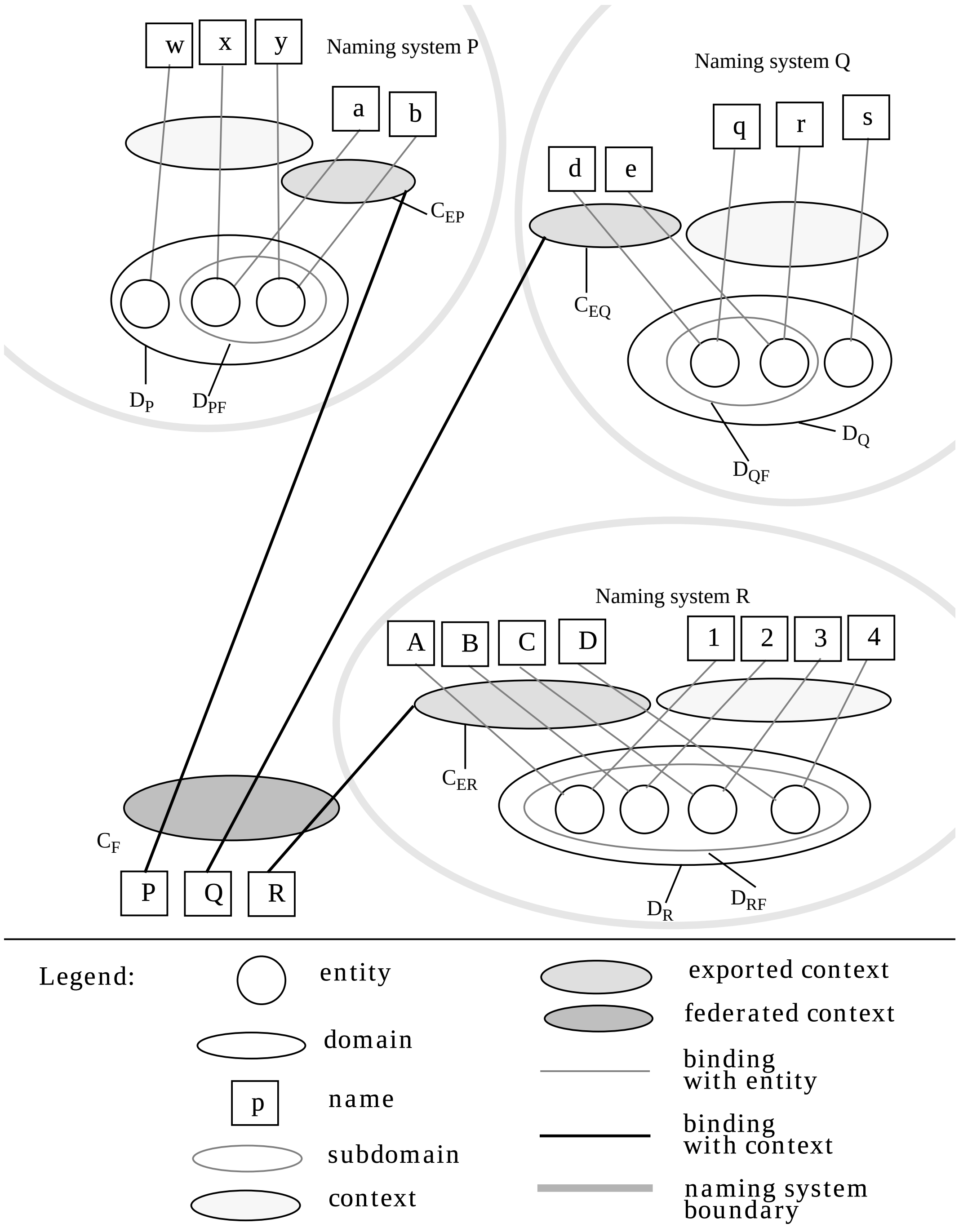
Naming systems in federation are connected through their naming contexts. There are several ways in which the linkages can be established. Consider, as an example, three parties P, Q, and R that are of interest in relation to a particular federation F. Each naming system has a naming domain and an associated context. The domain of P is denoted D_P, the domain of Q is denoted D_Q, and that of R is denoted D_R. As federations are to do with information sharing, each naming system makes a part of its domain visible to the other members of the federation. With respect to the federation F above, the parts of the domains of P, Q, and R that are made visible are denoted D_{PF},D_{QF}, and D_{RF}.

A naming context is associated with each of the domains and subdomains. The contexts associated with those domains that are made visible are called the "export schema" in [HEIMBIGNER81]. Here we refer to these contexts as *exported contexts*. In the example illustrated by Figure 6.1, the exported contexts are labelled C_{EP}, C_{EQ}, and C_{ER}.

When P, Q, and R are all part of the federation, then the naming context that is associated with the federation (C_F in Figure 6.1) is made up from the component contexts. This "federal" naming context allows access to the contexts exported by each of the naming systems[1]. Each naming system retains full control over the visibility of the entities in their naming domains, since they alone can add and delete bindings in their exported contexts.

[1] This context is itself distributed amongst the naming systems that are involved in the federation. This is further explained in §6.3.

Figure 6.1: Example of a naming domain with subdomains and contexts



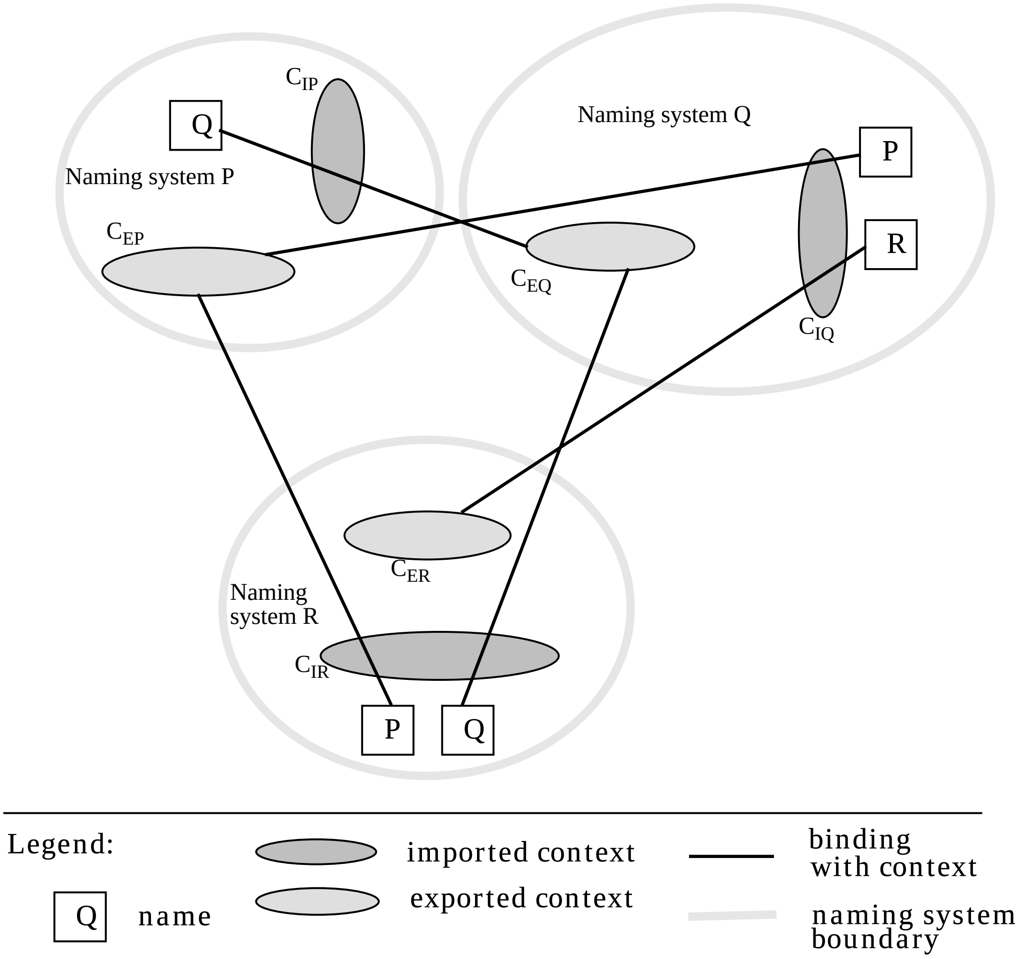
6.3 Naming networks

The effect of federation on the naming network of a naming system involved in a federation is simply the extension of that network with that part of the federal context that is selected. Since each party to a federation can impose a constraint on its own view of the federation, there is no generic rule that can be used to derive the extent of a naming network after joining a federation.

The federal context (C_F in §6.1) does not exist as an entity in a federation. Instead it is distributed over the naming systems of the federation and is implemented as a set of links between contexts. Each naming system in a federation may create a context in which it can name some or all other components to the federation. This context is called the “import schema” in [HEIMBIGNER81] and is here referred to as the *imported context*, denoted by C_{IP}, C_{IQ}, and C_{IR}. Figure 6.2 illustrates an example of how the logical federal context C_F can be distributed over the naming systems in the federation. Each naming system is free to view the federal naming context as it sees fit. Thus P may elect to use the context exported by Q, but not the one exported by R for example, leading to the single binding in C_{IP} in Figure 6.2

Figure 6.2: Distributing the federal context

.



6.3.1 The federal naming system

From Figure 6.2 it is clear that we have created a new naming system, in which the set of exported naming contexts are members of a naming domain. In the example there are three naming contexts. The naming network is flat.

6.3.2 Context links and searching strategies

A naming system that wishes to join a federation in order to import a context from that federation does not have to set up a separate import context[1]. It may include a binding to some exported context in a naming context that already exists and in which bindings with entities in the local domain are defined.

[1] It is necessary to use a separate naming context only if different naming conventions are used. See §6.4.

When this is done, it is recommended that this binding is marked as leading to another context in another naming system, such that the boundary with another administration remains visible. Searching along a link to another administration may be of lower priority; a policy may be in force that states that it is better to get connected to a service provided in one's own organisation, if one exists. For example, in a company that has its own printing facilities, it would not do to use outside agencies before checking the availability of internal printing services. A naming system in which no distinction is made between those links that represent the *is-a-subdomain-of* relationship and those that reflect the *peer-to-peer* relationship does not permit such searching policies to be implemented.

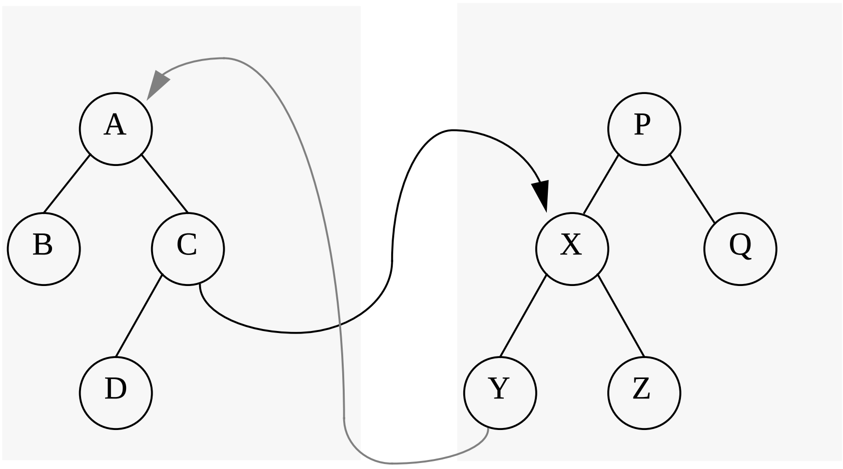
A naming system that wishes to join a federation in order to export a context into that federation, does not have to create a new exported naming context. It is possible to pass a reference to a context that already exists, to other naming systems in the federation. It must be remembered that all entities that can be named from an exported context become visible and thus potentially accessible from other naming systems. It is expected that access controls will be applied by the entities themselves, but security policies may well state the need for controlled visibility, which can be implemented by use of export contexts.

6.3.3 Circularity in the naming network

Name resolution in a federation will proceed in the same way as it would in any other naming system in which context relative naming and multiple name spaces are allowed. In a single naming system, the naming network is often restricted to be directed and acyclic. This is done to make sure that name resolution terminates.

A naming system that has individually been restricted to have an acyclic naming network may become part of a cyclic naming network if it is allowed to become party to multiple federations. For instance, a cycle may be created as shown in Figure 6.3..

Figure 6.3: Introducing cycles through multiple context links



The only way to avoid the introduction of cycles is by avoiding multiple federations. Restricting all federation through a single domain does not help, because the cycle may go through any number of naming networks and any number of federations. For instance, assume that systems S_1, S_2, and S_3 are party to federation F_1. Systems S_2 and S_3 are party to federation F_2. A cycle may be formed from S_1 to S_2 in F_1 then from S_2 to S_3 in F_2, and finally from S_3 back to S_1 in F_1. From the point of view of any of S_1, S_2, or S_3, there are no cycles, since they are either not aware of all federations (S_1) or of all links (S_1 and S_3). Non terminating name resolution as a result of cyclic naming networks can only be avoided by the introduction of a detection algorithm

The development of such algorithms is an area of active research. Most cycle detection algorithms require that contexts have unique names: precisely the situation that should be avoided. Others rely on controlling the search time. However, it is not generally possible to abort a search which has extended to a remote part of the system. Algorithms which allow the specification of search depth seem the most promising: the depth parameter is reduced by one on each hop and the search aborted when the parameter has reached zero. Whatever cycle detecting algorithm is used, it is important that all parts of the system over which the search may extend support the same algorithm. This may be hard to achieve in a federation.

6.4 Name sets

Each naming system in a federation potentially has its own name sets, based on carefully chosen naming conventions. When naming systems federate, their naming networks are extended, and the entities in their domains will have to be named if access to them is required.

As illustrated in Figure 6.1, each naming system controls the bindings in their exported contexts. Each naming system names the entities according to its own naming conventions, and any naming system that imports a context must adhere to these conventions if it is to name an entity in another domain.

In Figure 6.2 each naming system has an import context in which it may name the exported contexts of other naming systems it is interested in. These contexts are named relative to the import context and local naming conventions may be used. In the example of Figure 6.2 unique names were used for simplicity, but this is not a requirement.

The naming conventions in all naming systems that are party to a federation thus remain independent, that is they do not change as a result of the federation. In crossing the boundary between the federated systems there will generally be a need for name translations to cater for the potentially different naming conventions. There are two options:

- a naming system adheres to its own local naming conventions when naming entities in other naming systems through the federation.
- a naming system adopts different naming conventions for entities in other naming systems

The name translations must be performed at the imported context. It can be implemented by a simple name translation mechanism, often called a *naming bridge* [COMER89]. If the naming conventions for local entities are different from the naming conventions for entities in other naming systems, then it is necessary to set up an explicit import context, in which the differing name conventions can be used. If the naming conventions in both systems are the same then there is no need for an explicit import context, and any naming bridge can be characterised by the identity relation.

6.5 Summary

The federation of several naming systems creates a new naming system in which:

- the *naming domain* is formed by a set of (exported) contexts, contributed from each naming system that makes itself accessible to other naming systems in the federation
- there exists a *naming context* (imported context) for each naming system that wishes to access other naming systems in the federation
- each naming system may adhere to its own *naming conventions*; i.e. there are potentially as many name spaces as there are imported contexts
- the *naming network* is flat.

This new naming system is superimposed on the existing naming systems, thus creating one larger naming system in which:

- there exist a number of disjoint *naming domains*
- naming between entities in different disjoint naming domains is always indirect through an import and export context pair

7 Application of the Naming Model

A naming model cannot be implemented in its own right. The ANSA naming model must be adhered to indirectly by ensuring no system components break the naming rules. This chapter describes how naming concepts can be applied to:

- the notion of name service
- interworking between application entities
- the ANSA interface reference
- binding models (in ANSA and OSF DCE)
- the relocation service
- the trading service.

When naming concepts are applied to non-ANSA systems, it becomes obvious where ANSA naming rules are broken. This non-compliance does not mean that such systems cannot interwork with ANSA systems. Full generality of interworking is however lost in such systems and this places constraints on the designs. The naming model as part of the ANSA architecture thus serves the important purpose of assisting the analysis of the complexity of interworking: one of the objectives of an architecture.

7.1 Name service

A *name service* maintains a relation between attributive names and invocation names. Attributive names are usually descriptive in nature and are often meaningful to people. Unlike invocation names they have no meaning in the addressing architecture that supports the interaction mechanisms.

Attributive names are assigned to entities in one naming system, invocation names are assigned in another. The naming domains of both naming systems coincide. All entities in the naming domain associated with the name service have invocation names, some may also have an attributive name. Since attributive names and invocation names are used side by side there is a requirement to be able to map from one to another.

A *name server* is an object which provides an interface at which the name service is offered. The way in which name servers can be configured to achieve different quality of service characteristics is described in chapter 8.

7.2 Interworking and context relative naming

Two entities are said to be *connected* when it is possible for them to exchange some set of symbols. If both entities attach some significance to the symbols exchanged and if there is some common understanding of what this significance is, then the exchange of symbols is possibly a meaningful one. The entities are said to be *interworking*.

There is a need for some infrastructure which can mechanise the exchange of symbols from one entity to another to achieve connectivity. The technologies for this are well known and range from shared memory to networks and communication protocols.

There are no effective mechanisms which *ensure* a common understanding of the significance attached to the symbols exchanged. Two alternative techniques may be employed to circumvent this basic dilemma:

1. pass an *application name* (which is a name) and insist that the naming context in which the name was generated (the sender's context) is the same as the naming context in which the name will be resolved (the recipient's naming context);
2. pass a *reference* (which is a name) and claim that when the reference is used from the recipient's context, the effect will be as if the reference were used from the sender's context.

7.2.1 Passing application names

When passing application names between entities, the same strings exist on both sides of the connection. When the usual short user friendly application names are used, the most important problem is that a sender cannot have complete knowledge of the context of the recipient: a name sent may already be in use at the receiving end. In this case it is unclear whether the transmission should fail, the name be re-assigned, or a potential homonym be allowed to exist: none of these options is desirable.

Systems such as OSF DCE [OSF92] solve this problem by requiring that any names which are exchanged between clients and servers (and which refer to entities with whom interaction can take place) are allocated *centrally*[1]: hence the Universally Unique IDentifier (UUID).

When either party wishes to make use of the entity which is bound to the name it needs to resolve that name to an invocation name. Since there is a requirement for a shared naming context, it is important to know where this shared context is, since name resolution needs to take place within it. Usually, name resolution takes place in a name server, which is common to the sender and the recipient.

In OSF DCE the UUID is presented to the name service, and “Server binding information” is returned. This is presented to the application as a binding handle which can then be used to connect to and interact with the entity bound to the passed name (the UUID in this case). The extent to which applications have to deal with binding information explicitly varies as DCE provides *automatic*, *implicit* and *explicit binding management*. In each the UUID plays a central role.

When passing application names, the name allocation policy is clearly based on a single (global) name space, and this does not conform to the ANSA naming model.

[1] In OSF DCE (and most other systems) this must be interpreted as *effectively centrally*. Since the UUID contains a large random number, clashes are said to be most unlikely, thus mimicing a global (universal!) allocation policy.

7.2.2 Passing references

References introduce a level of indirection between names and the entities they denote. A name is bound to a reference and the reference is bound to the entity. From an application point of view it appears as if the name is directly bound to the entity.

When a name is sent, it is replaced by the reference during marshalling, without the application being aware of this. The reference is passed to the recipient and assigned a name after having been unmarshalled. To the recipient it is as if a name is bound directly to some entity. The names attached to the references at sender and recipient need not be the same. They can be allocated according to the name allocation policies at either end. These do not have to be the same as in the case of passing application names. It can however be agreed between the authorities that govern both applications that the naming policies are aligned, thus providing greater usability of the system. Should either authority wish to change its naming policy, it can do so independently from the other and without breaking any constraints imposed by the technology.

Name resolution at either end takes place in the local contexts, which do not have to be shared, thus preserving the context relative naming features required by the ANSA Naming Model.

Upon receipt a reference can be used immediately. There is no delay for name resolution, no need to access name servers or other devices.

7.3 The ANSA Interface Reference

In ANSA references to interfaces are the only references which need to be passed. The *interface reference* embodies an *opportunity* (but not the *right*) to use a service provided at the interface which is referred to. The information contained in an interface reference allows a client infrastructure to set up a connection through one or a number of networks and protocol stacks to the service with which the interface reference is associated.

Transparency mechanisms use the information carried in the interface reference to ensure trouble free interactions between applications, for whom the interface reference remains hidden by encapsulation by the infrastructure. An application programmer is given a pointer to the interface reference. Applications that require to take control of distribution will require access to the interface reference and a special interface to access it’s fields is provided. Note that the use of this interface can reduce the portability of applications.

Interface references are dynamically generated by the infrastructure. This is done as and when it becomes apparent that a service is to be made available to other entities in the distributed system. It can be done early (e.g. when a service is initialised) or late (e.g. transparently, as a parameter is being passed outside the local address space).

7.3.1 Features

Many of the important features of the ANSA interface reference are closely related to requirements imposed by the naming model. They are:

- *the naming scheme employed in the interface reference is completely context relative*:
  ANSA is committed to an evolutionary approach to systems integration,

making it possible to combine ANSA applications with existing applications (legacy systems) and ANSA distributed infrastructure components with components developed in other distributed systems environments (e.g. OSF DCE, OMG ORB, etc.). Thus it is most important that the interface reference can operate over a range of distributed systems platforms. The interface reference may not reflect any one naming policy of any one of the systems it is to operate over. This can be achieved by adhering to context relative naming throughout.

- *the interface reference is able to absorb or include binding information as defined for several different distributed system environments*:
  This is necessary to ensure that the validity of the interface reference is preserved when it is passed from an application on one platform to an application on another platform.
- *the interface reference consists of data types that can be marshalled just like any other data types*:
  The interface reference must be capable of being passed around the distributed system across heterogeneity boundaries in the parameter and result parts of invocations. Marshalling is the accepted method of dealing with heterogeneity of data representation.
- *an interface reference contains the information that reflects what protocols are supported by the server that created it*:
  For a client to be able to use a service it’s infrastructure must use the same protocols as the server. The client infrastructure must therefore have knowledge of what protocols to use before actually interacting with a particular server. Where a server supports more than one protocol stack, information about each stack may be included, so that the client infrastructure is effectively offered a choice.
- *an interface reference contains the information that reflects the context relative network address of the server port*:
  For the communications protocols to be able to route messages to the server, it must know the network address of the server port, relative to the clients position in the network. The interface reference holds the context relative address of the server. If an interface reference is passed from one network to another, then the addressing information is updated by the router in the gateway. As a result, global addresses are not required.
- *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 in the context of the connection that has been established with the help of the address and protocol information.
- *each interface reference can contain information which can be used to find services which have migrated*:
  When a server migrates to another location, then all interface references previously issued become invalid. Since the server has no control over interface references once they have been given out to others, this can lead to considerable disruption, unless the interface reference contains

information that can be used for remedial action. See also section 7.5 on relocation.

- *the structure of the interface reference accommodates the possibility of a service being offered by a group*:
  Where a service is actually (and transparently) provided by a group of servers (replication to increase dependability for instance), the interface reference should appear as if a single server is involved from the client point of view, but contain all information to allow the orderly progression of group execution protocols (possibly using multicast messaging) by the client infrastructure.
- *an interface reference is not a capability*:
  An entity that obtains an interface reference may use it in order to try and invoke operations on the service that generated it. The service may however decline to respond (for security reasons for instance).

7.3.2 Structure

An interface reference consists of four components:

- group data
- a nonce
- for each member in the group a member_record
- a sequence of relocator interface references.

Each member_record consists of:

- a sequence of address records.

There is an address record for each set of protocols that can be used to access a service. The address record is interpreted in the context of the distributed systems infrastructure which supports the server. In this way, ANSA specific information can be included as well as DCE binding information for instance.

Each address record contains a stack of tuples, each consisting of

- a protocol identifier
- a protocol address.

Protocol identifiers and protocol addresses are context relative *names* which have meaning within the context of a specific subnetwork and distributed systems infrastructure.

The nonce is *not* used as a global identifier. The nonce is *not* used in any name resolution process either. If the client and server hold different nonces on a set of bindings, then the bindings are broken. Matching nonces provide no guarantee that things are right however! Thus the nonce is only used in the *context* of an interaction on a connection determined by the address records. The chances of erroneous connection are so reduced to acceptable levels, without having to resort to globally unique identifiers.

7.4 Binding

In the engineering model the notion of “binding” is loosely associated with the existence of an end to end “connection” between two entities, so that any forthcoming interactions are possible.

7.4.1 Binding in ANSA

ANSA applications need never be aware of the activities in the infrastructure, necessary to provide communications facilities, although transparency can be relaxed to allow explicit management of these facilities.

The concept of binding as defined in the ANSA Naming Model can be applied to describe the engineering model concept of binding more precisely.

A server in ANSA is “bound” when

1. the server’s infrastructure has been set up with the necessary communications facilities, such that incoming calls can be received and results returned,
2. an interface reference has been created, which reflects the nature of these communication facilities,
3. a name in the server’s application space has been *bound* to the interface reference.

In naming model terms, the naming domain for the server binding is the interface reference.

A client in ANSA is bound when

1. an interface reference has been received which reflects the characteristics of the communications facilities required to interact with a server,
2. an application name has been *bound* to the interface reference,
3. and *either* the client infrastructure has been set up with the necessary communications facilities, such that calls can be made and results received *or* such an infrastructure can be set up before interaction actually takes place.

In naming model terms, the binding between the application level name and the interface reference is all important.

7.4.2 Binding in OSF DCE

Applications in OSF DCE can choose to manage bindings themselves (explicit method), share the management with the run-time (implicit method), or leave all binding management to the run-time (automatic method). When using the explicit method, applications also get some control over the nature of the protocols that are included in the communications infrastructure[1]. Binding of clients and servers in DCE is similar to that in ANSA.

A DCE server is bound when

1. the server’s infrastructure has been set up with the necessary communications facilities, such that incoming calls can be received and results returned (note that an explicit listen call is required),
2. binding information (including the UUID) has been created,
3. a name (binding handle) in the server’s application space has been *bound* to the binding information.

A DCE client is bound when

[1] In ANSA such control is to be exercised through the quality of service parameters and the tool chain, which makes protocols available for a range of quality of service characteristics.

1. binding information has been received (either via the name server using the UUID or as a string binding),
2. an application name (binding handle) has been *bound* to the binding information,
3. and the client infrastructure has been set up with the necessary communications facilities, such that calls can be made and results received.

7.5 Relocation

To allow flexible configuration and resource management policies, servers should be allowed to migrate freely from one location to another. As servers relocate, clients may experience disruptions in service provision. Transparency mechanisms are put in place to hide these disruptions from application code. In designing these mechanisms, care should be taken not to break the naming model rules.

7.5.1 Relocation in ANSA

A client's infrastructure, upon detecting the absence of a response from a server, may attempt to rebind using a different address record. If the service has migrated, then none of the address records will yield a successful interaction. In that case the client infrastructure uses information contained in the interface reference to obtain a new interface reference, to rebind and then continue interaction. All (re-)binding and access to location services is performed by the client's infrastructure, invisible from the application, and completely within the context provided by the local infrastructure[1]. Client application code should experience no more than a delay in response.

The server nominates a location server (or relocator) and includes the interface reference of this server in its own interface reference. There is no reliance on a "well known" address, such as that of a name server, as this is a solution which does not scale to very large systems. Relocators may migrate as well, as their interface references can contain interface references for relocators which can in turn be consulted.

When migration is as liberal as suggested, then the likelihood that a particular location is visited by two or more services who have each nominated the same relocator, increases. In that case neither the service location, nor the relocator, nor the combination of the two provides sufficient context to disambiguate the entries for the services in the relocator database. Two solutions to this problem can be provided: either the location of interface creation or the relocator itself can be used as a disambiguating context.

7.5.1.1 Solution 1: Use of location of interface creation

As no two services are ever created in the same location at the same time, each location is associated with a monotonically increasing counter, increased each time a service interface is bound for the first time. The location of creation, together with the value of the counter are sufficient to disambiguate relocator database entries.

[1] The resulting migration transparency may be relaxed and applications may be allowed access to relocators.

The allocation of keys to interface references is done in a distributed fashion, at each server site. To avoid the potential bottleneck in number allocation, counters can be maintained by capsules[1].

This scheme relies on the location as a sufficient identifier for a context, in which the name <value of the counter> can be resolved. The naming convention for locations should be strong enough to support this.

7.5.1.2 Solution 2: Use of the relocation context

If the assumption on naming conventions for locations is unsafe, then the following solution could be adopted

- when the server infrastructure constructs the interface reference for the first time, it asks the relocator, whose interface reference it is about to include, for a key,
- it includes the key in its interface reference, together with the relocator interface reference;
- the key obtained from the relocator is strictly valid in the context of that relocator only;
- the relocator assigns keys once only, thus ensuring they are unambiguous;
- any clients who contact the relocator pass the “old” interface reference, which contains the key (as before).

The advantage of this scheme is that servers can only insert the interface reference of a relocator if that relocator has agreed to perform the relocation function. It effectively agrees by handing out keys. Thus there is a measure of control on the amount of business a relocator is willing to get involved in, which is missing in solution 1.

A disadvantage is that the relocator could become a bottleneck for key allocation if the number of interfaces it looks after is very large. However, it can be expected that system designers would guard against this and distribute the location service instead of concentrating everything in place[2]. Furthermore there is a possible optimization by allowing a series of keys to be allocated at once, thus requiring only one relocator access per set of interface creations.

7.5.2 Other relocation schemes

Traditional relocation schemes generally take a more centralised approach and rely on one well known (and invariant) address such as that of a name server, where location information is provided. In distributed systems, where federations must be considered, there is no guarantee that a single well known place exists (or would be practical). Centralised or semi-centralised schemes are nevertheless still being proposed for distributed systems [OSF92].

In OSF DCE, automatic rebinding can only occur when the automatic method of binding management has been selected, and then only if either the rpc has not started to execute, or the operation is idempotent. In explicit binding

[1] A capsule is a unit of resource allocation and can support several objects, each with several interfaces [ISO 10746-3].

[2] The distribution policy for relocators may result in a relocator for all objects (1) on a particular node, (2) in a particular capsule, (3) belonging to a particular principal (owner), (4) all interfaces of a particular type, or some other classification.

management mode it would be possible to provide relocation services similar to those employed transparently in ANSA. There are several possible shortcomings though. The RPC system on each node contains an *rpc deamon*. This process provides the *endpoint map service*, which manages a node specific database where servers register their endpoints and associated addressing information (host address, communications protocol information etc.). The endpoint operations provided by the DCE RPC runtime allow the insertion and deletion of entries from the endpoint map, thus allowing interfaces to migrate within a node[1]. When a client makes a remote procedure call not specifying any endpoint, a search for a *compatible* server is started. If successful, the endpoint is given back to the client, who can then reissue the rpc with the correct endpoint. The endpoints are clearly disambiguated within the context of the server's RPC runtime.

Relocating a DCE server to another node should be done whilst preserving the Object UUID, and insisting that the Object UUID is included in all remote procedure calls. If the Object UUID is provided, then the rpc runtime will take it into account when trying to find a compatible server. The client will thus not be returned an endpoint from the node from which the requested server has migrated. The only way open to the client is to access the DCE name service to attempt to find the new node to which the server has migrated. The rpc deamon at that node is then to be consulted with the Object UUID to obtain an endpoint for subsequent remote procedure calls. It is clear that UUIDs break the ANSA Naming Model rules.

7.6 Trading

Trading [APM1005.1 93] is defined as:

the activity of choosing services, such that they match some service requirement. The choice is based on the specification of a required service provided by a prospective service consumer and the service specifications supplied by service providers or their agents.

Note that the trading service is itself a service and can be advertised, possibly by the same trading service that offers it. Note also that the trading service is not the only service that can pass knowledge about other services around the system. By virtue of the ability to pass interface references in ANSA, this capability is granted to all ANSA objects.

The provision of a trading service involves a number of naming systems, each originating from a separable set of concerns about the operation of a system in which services are provided.

The primary aim of the trading service is to allow the contact between service consumers and providers to be made. As a consequence, one of the naming systems involved relates service instances with their invocation names. We shall call this the *invocation naming system*.

Before service consumers can be given an invocation name for a particular service instance, it is necessary to check that the service instance is appropriate, or stronger still, that it is best suited for the stated requirement.

[1] It is unclear what happens if a client makes a remote procedure call specifying an endpoint which is no longer in the endpoint map, nor is it clear what happens if the same endpoint is now occupied by another server.

An *attributive naming system* relates invocation names for service instances with attributive names.

There are several other sets of entities whose naming affects the trading system. They are types, properties and administrations. For each a separate naming system is devised and related to the others.

7.6.1 Invocation naming system

Invocation names for service instances take different forms, depending on the projection in which these names are considered:

- in the computational projection, a service instance is provided at an *interface*. A service instance is named by naming the interface at which it is provided, with an Interface Reference [APM1001.1 93]
- in the engineering projection, a service is provided at a *socket*. Services are named by the socket at which they are provided. Sockets are named relative to a capsule by a socket number.
- in the technology projection, capsules are named relative to the host on which they reside, and hosts are named relative to the network to which they are attached.

7.6.1.1 Invocation naming contexts

Independent of projection, an invocation name is only an invocation name relative to an appropriate context. Ultimately, the invocation name need only be an invocation name in the context in which it will be used to invoke a service, that is at the service consumer. In particular, the trading service need not be able to invoke a service instance for which it has an invocation name.

An *invocation naming context* is a context in which an invocation name is unambiguous and works. Invocation naming contexts are introduced to allow a description of the mechanisms that are required when an invocation name is passed to a service consumer (in another invocation naming context) who cannot use the invocation name immediately.

The ANSAware trading service assumes that all service providers and consumers are in one and the same invocation naming context. Context relative addressing and differences in addressing schemes are absorbed by Interface References and the mechanisms that handle them.

7.6.2 Attributive naming system

There are two kinds of attributive names for service instances: typenames and property name/value pairs. For each a naming system can be distinguished.

7.6.2.1 Type naming system

A *type* is a set of permissible interactions. Types can be named, and the bindings between types and typenames are defined in *type naming contexts*.

The ANSAware trading service employs simple *type naming conventions*: typenames are unstructured strings and must be *absolute names*[1]. This leads

[1] This precludes the federation of different type naming systems and will cause scaling difficulties. Future releases of ANSAware will be brought into line with the architecture.

to a view in which there exists a single type naming context in a *flat type naming network*.

Types themselves are not present in a computer system, only their names are. The bindings in the type naming context are abstract; the meaning of each typename must be agreed between the designers of the system in question.

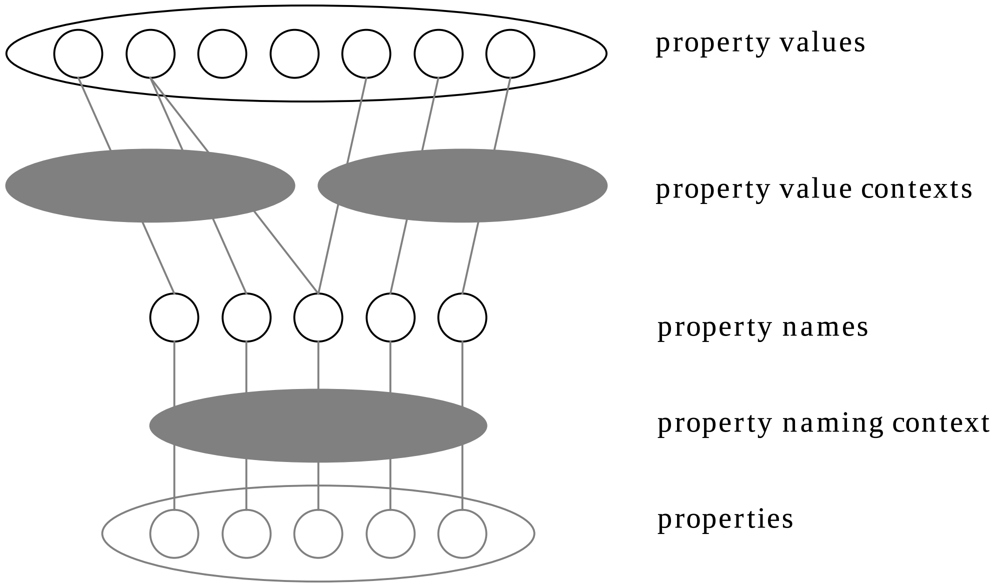
The ANSAware trading service allows the reflection of the *is-a-subtype-of* relationship between the types. This takes the form of a relationship between typenames. This relationship is not a part of the naming system; it is a way in which the names are used[1].

To reflect the *is-an-instance-of* relationship between service instances and types, the type naming system and the attributive naming system are linked through the use of particular typenames. A particular typename may be bound to an invocation name for a service instance in the attributive naming system, and to a type in the type naming system.

7.6.2.2 *Property naming system*

A *property* is some characteristic (in this case of a service). Properties can be named, and the bindings are defined in a *property naming context*. Each property name is bound to a property value in a *property value context*. This leads to a two level naming system as illustrated in Figure 7.1

Figure 7.1: Property naming system



Properties themselves are not present in a computer system, only their names are. The bindings in the property naming context are abstract; the meaning of each property name must be agreed between the designers of the system in question.

The *naming conventions* for properties and their values in the ANSAware trading service are fixed: property names are strings, property values may be strings or integers. All names are *unstructured*.

[1] The relationship is formed by a directed acyclic graph (DAG), where each vertex denotes a type and each directed edge denotes the *is-a-subtype-of* relationship. The relationship is transitive.

The ANSAware trading service has a flat *property value network*. To reflect the *has-properties* relationship between service entities and a set of property name/value pairs, each property value context is bound to an invocation name for a service instance, and as such functions as an attributive name, containing potentially many *property name/value pairs*.

7.6.3 Administration naming system

An administration is a source of authority, in pursuit of a collection of goals. Administrations are named and the bindings are defined in the *administration naming context*.

The ANSAware trading service employs simple administration *naming conventions*: names are unstructured strings and are *absolute names*. This leads to a view in which there exists a single administration naming context in a *flat administration naming network*.

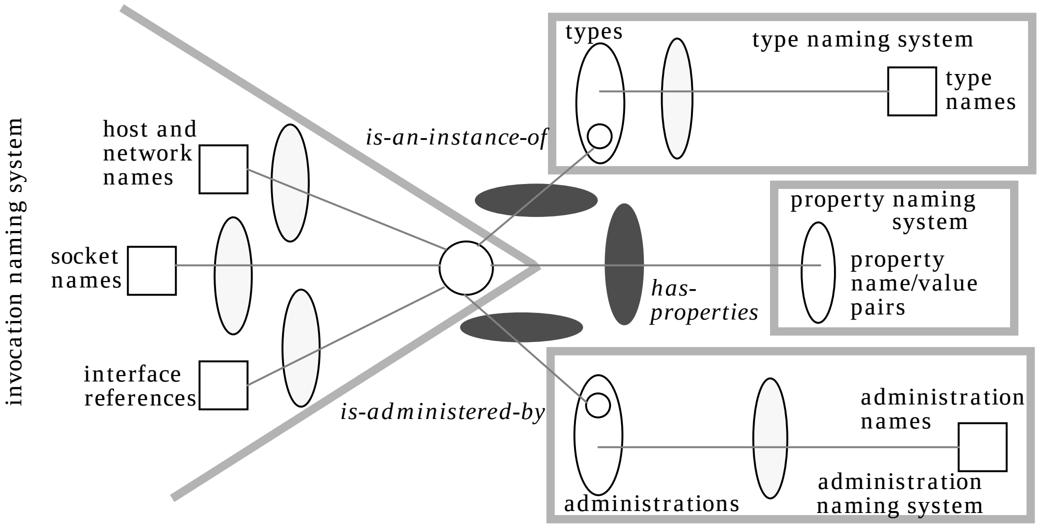
Administrations themselves are not present in a computer system, only their names are. The bindings in the administration naming context are abstract; the meaning of each name must be agreed between the system designers.

The ANSAware trading service allows the reflection of the *is-a-subadministration-of* relationship between administrations. This takes the form of a relationship between administration names. The relationship is not a part of the administration naming system; it is a way in which the names are used[1]. The relation becomes explicit in the trading network.

7.6.4 Summary

Figure 7.2 illustrates the relationships between the various naming systems employed in the trading service.

Figure 7.2: Relationship between naming systems in trading



[1] The relationship is formed by a directed acyclic graph (DAG), where each vertex denotes an administration and each directed edge denotes the *is-a-subadministration-of* relationship. The relationship is transitive.

8 Configuring Name Services

A name server is an entity that offers the name service at least one of its interfaces. This chapter identifies several configurations of name servers and examines the quality of the resulting name service.

Centralised name services will not be examined. The obvious performance penalty of a single name resolution process forming a bottleneck in anything but the smallest system is not acceptable in distributed systems.

8.1 Quality of service concerns

Some of the ways in which the quality of a name service may be measured can be expressed as:

- *how can ownership and autonomy issues be reflected?*
  The name service database contains information about other services. The rules about ownership and autonomy over these services vary. At the same time it is convenient to partition the information about services in such a way as to reflect these ownership and autonomy issues. The name service is long lived and subjected to many organisational changes over its life span. It will thus be necessary to provide a flexible approach to the partitioning of the name service data structure throughout it's life time.
- *how accessible is the name server, and how long does access take?*
  A name service database can be very large. There will be few if any name service clients that will require frequent access to all of the database. It is common for several name service clients with similar access requirements, to be clustered together. If the location of name service clients is static or does not vary too much (e.g. they remain in a single LAN), then it is possible to partition the name service database over name servers which can serve certain client communities locally. Name service clients may of course migrate and alter their usage pattern, and this again requires a flexible approach to the partitioning of the name service data structure.
- *what is the effect of parts of the service failing?*
  Fault isolation is an important factor in deciding how to partition or replicate the name service data structure over name servers. Each name server is considered as a separate unit of failure. By partitioning and replicating the data structure over several name servers, the failure of one name server will not lead to a total loss of service to all users.

8.2 Distributing the name service

The name service maps attributive names to invocation names. To do this its state consists of a data structure which reflects the relation between both kinds of names. Distribution of this data structure is achieved by

- partitioning the relation to reflect ownership and autonomy issues, and to increase accessibility by clients and
- replicating the relation to improve resilience to failure, and again to increase accessibility by clients.

There exists a conflict between the consistency and availability of the data structure. As a result, distribution criteria will vary from system to system and from application to application. The following options for distributing the name service exist:

- one name server per name space
- one name server per service type
- one name server per service provider or consumer
- one name server per administration
- one name server per application or utility, e.g. a directory server in a file system
- one name server per failure domain
- one name server per host node
- any combination of the above.

The data structure reflects the form of the naming network. Viewing the naming network as a graph helps when reasoning about the distribution of the data structure used by name services. The network can be carved up and parts of it replicated. Each possibly replicated partition is assigned to a name server. The name servers are then connected together to reconstruct the original naming network.

The architecture does not place any constraints on the structure of the naming network. Particular naming systems will do this. For instance, in OSF DCE [OSF92a], the Cell Directory Service (CDS) entries are organised in directories, which are structured hierarchically. The directory structure is then partitioned over CDS databases, each called a *clearinghouse*[1].

In X.500 [ISO 9594] a similarly hierarchical structure gets partitioned over a set of Directory System Agents (DSA), which are able to interwork using the Directory System Protocol (DSP).

8.3 Configuring name servers

8.3.1 Basic model

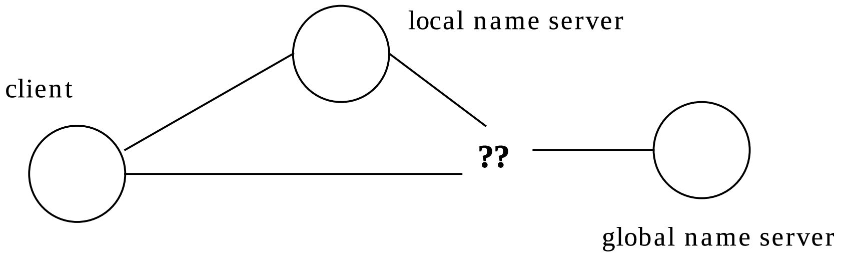
A name service in a distributed system is implemented by a collection of interconnected name servers. There are two kinds of name servers. Local name servers look after a set of entities in a local domain, which is determined by the server placement policy (see §8.2). The entities in the local domain can only see other entities that are also in that domain. Visibility of entities in other domains is strictly through “global” name servers[2].

[1] The use of Clerks and caching strategies complicate this simple model. It is referred to in §8.3.

[2] The term global should not be taken literally. It is used in the sense of non-local, not in the sense of spanning a global or galactic space time continuum.

The way in which local and global name servers are implemented and access one another determines the configuration. In all cases, a client of a name service must have access to at least a local name server, and optionally to a global name server, as in Figure 8.1.

Figure 8.1: A name service client has access to a local and a global name server



Sometimes, the global name server consists of a client and a server part. In X.500 for instance, the Directory User Agent and its cache form the client part of the global name server. The DUA and it's cache can clearly be seen as a local name server in the model of Figure 8.1.

The four basic name server configurations that can be built from these components are "direct access" or "referral", "re-registration", "chained", and "federal" configurations.

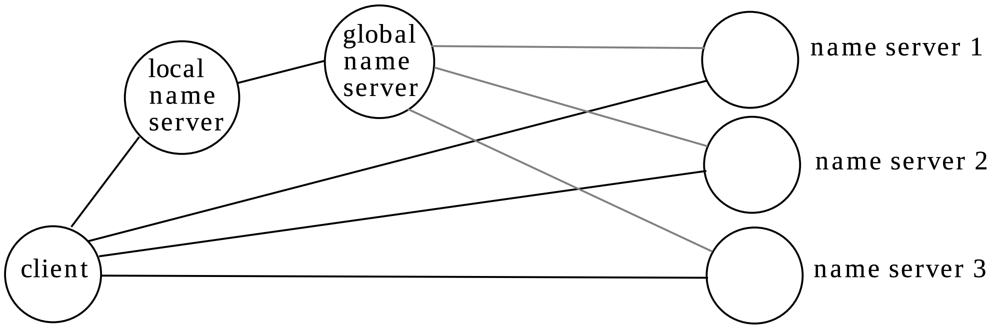
Before a client can use any name service, it needs to determine which name server should be used. Once this has been determined, the second activity is to access the selected server. Depending on the configuration, the local and global name servers take part in either or both activities.

8.3.2 Direct access (or referral) approach

In the direct access approach the name server client accesses remote name servers directly. First port of call is the client's own local name server. When a name cannot be resolved there, the global name server is consulted. The global server knows about other local name services, and holds a single level mapping from global names to local name servers. The name service client thus obtains a handle, allowing direct access to other local name servers. It is as if the global name server refers to the other name servers (hence the term referral).

The knowledge which the global name server has about other name servers has been represented by dashed lines in Figure 8.2.

Figure 8.2: Direct access approach:
Global name server hands back reference to other local servers



The name service client is aware of the different local name servers and access, location (and migration) transparency are not offered.

Referral is one of the methods by which Directory System Agents in X.500 search the directory structure (chaining being the other).

8.3.2.1 *Local name servers in different name spaces*

Different local name servers may be in different name spaces. Before the name service client can access the local name servers, it needs to know about the name spaces in which they operate. This information could be provided by the global name server. Any functionality to deal with the differences is firmly positioned at the client end. Adding a new local name server that supports a different name space would require an update to all potential name service clients as well as the global server; a solution that clearly does not scale.

It is possible to place the functionality for the provision of name space transparency at the server end. This would require a front end to each local name server in the system. This solution scales better because adding a new name server means the addition of a single new mapping.

The transparency may also be included in the infrastructure which supports the interactions between the name service client and the local name servers. The global name server hands back a handle to a so called Naming Semantics Manager[1] (NSM). This manager can handle the translations as part of the RPC mechanism, as demonstrated in the Heterogeneous Computer System project [SCHWARTZ87].

8.3.2.2 *Other solutions*

To hide the differences between the local name servers (servers 1, 2, ..., N in Figure 8.2), it is necessary to isolate the name server client from the local servers. This can be done in two ways: re-registration or chaining.

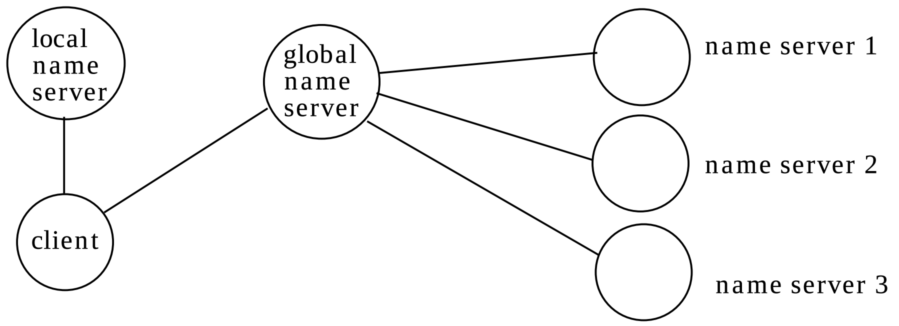
8.3.3 Re-registration approach

In the re-registration approach a name service client accesses the global name server for access to the information about services outside of the local domain. No access to other local name servers is required, as these have all passed their responsibility for maintaining the mapping between attributive names to invocation names to the global name server. They have re-registered everything that is registered locally and that needs to be made available globally at the global name server. Figure 8.3 illustrates the relationships between the local name servers and the global name server.

The re-registration approach imposes several constraints on the local and global name servers. The local name server is not at liberty to change any of the names of entities whose names have been re-registered. To do so would require the information held in the global name server to be changed as well. Conversely, the global name server needs to keep track of the source of the registrations, so as to disambiguate any potential homonyms. The global name server represents an administration that is hierarchically above the local name servers.

[1] It would have been better to use the term “Naming Syntax Manager”, since it is the syntax that is at issue here. Schwartz uses the term “semantics” however.

Figure 8.3: Re-registration approach



It is clear that this approach does not scale, as it relies on a single global name server that is expected to keep track of all mappings. The global name server would soon become a bottleneck.

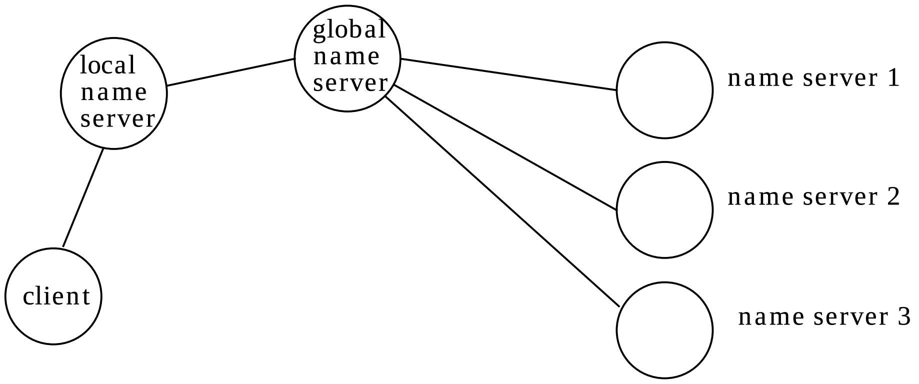
8.3.3.1 *Caching and involuntary re-registration*

Copies of name server entries in the local name servers may appear in the global name server when the global name server caches information obtained in queries. The client part of the global name server, such as the Clerk in CDS [OSF92a] and the Directory User Agent and it's cache in X.500 [ISO 9594], perform this function. This leads to much better scaling characteristics with respect to availability, but increases the burden on the name service as a whole to keep its data consistent.

8.3.4 Chaining name servers

Chaining yields a similar name server configuration to re-registration. The difference is that the global name server holds a single level mapping as in the direct access approach. Unlike the direct access approach, the name server client does not interact with any other local name servers. The global name server acts as an agent, on behalf of the name server client in accessing name servers. Figure 8.4 illustrates the configuration.

Figure 8.4: Chaining



Because the global name server accesses the name servers (1-3), then from the point of view of the client, these name servers are invisible, since the decision about which name server to access is also internal to the global name server. In that way the global server offers complete access and location transparency.

Chaining is one of the ways in which the Directory System Agents in X.500 [ISO 9594] search the directory structure (direct access or referral is the other).

This scheme allows some freedom in the mapping between the global names and the local names. Since the global server maps from global names to a single name server name, the local name servers are allowed to change the bindings between local names and the entities in the local domain. The combination of global and local name servers can be made to have the effect of a level of indirection in the name translation.

8.3.4.1 *Following a chain*

There are two strategies in searching a number of chained name servers. The shallow search strategy will search all of the information held in a particular name server and only follow a chain if name resolution is not successful at that server. The deep search strategy will follow the chain to another server as soon as such a chain is encountered. It will return when the end of a chain has been reached to continue searching locally.

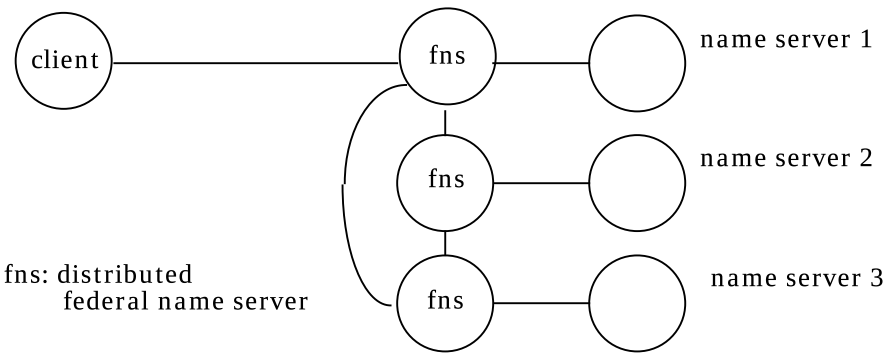
8.3.4.2 *Caching*

The global name server may want to cache some of the information it has already got from one or several of the local name servers. If this is done, then the problems described in the re-registration approach may return. Using out of date name mappings will cause misdirection of interactions later on. If mappings change infrequently (as is expected in X.500) and resulting misdirection errors can be dealt with efficiently, then a decision to have a cache and lazily update it (i.e. periodically or on problems only) may be acceptable. This increases availability of the name server at the cost of reducing the consistency of the name server data base.

8.3.5 Federal approach

Local name servers may be connected in a federal manner by distributing the global name server over all the name servers that take part in the federation. The resulting completely distributed “federal” name server is characterised and implemented by the protocol of interaction between the local name servers in the federation.

Figure 8.5: Federal structure: distributing the global name server



In any one federation, each name server is able to use the service offered by any one of the other name servers in the federation. The tasks of (1) determining which server holds the required information and (2) accessing that server are devolved over all members of the federation. Each component in a federation can be described as a set of protocols that must be adhered to as a condition of belonging to a federation.

Each server must hold a set of mappings between locally known external names it knows about and the name server that implements the mapping between attributive and invocation name.

Figure 8.5 illustrates the structure of a system built using the notion of federation in terms of the components introduced in the descriptions of the other approaches to combining name servers.

8.3.5.1 *Scaling*

Note that the federal approach scales better than any of the other approaches. The federation places a bound on the context in which a name is assumed resolvable: it is restricted to the scope of the federation. Name servers are allowed to be a member of more than one federation at the time.

9 Summary of Concepts

9.1 Introduction

This Chapter contains a summary of concepts that are related to the subject of Naming. The descriptions are in English and have been based on common usage of the terms in the literature. References have been included where appropriate.

The descriptions are independent of any one particular projection or application.

9.1.1 Organisation

The glossary has been organised in two parts. ANSA naming concepts are considered *essential* concepts. *Additional* concepts can be expressed in terms of the essential concepts. Each part has been organised alphabetically. All terms in italics are included in the summary.

9.2 Summary of ANSA naming concepts

Bind (verb) To associate a *name* with an *entity*. [SALTZER79]

Binding (noun) The association of a *name* with a particular *entity*. [SALTZER79]

Denote (verb) To stand as the *name* for. [COED82]

Entity Any thing that may be of interest.

Handle That part of a *structured name* that is resolvable in the current *naming context*. [COMER87], [COMER89]

Name (noun) A linguistic entity, that singles out an *entity* from amongst a group of *entities*. [HAUZEUR86], [COMER87], [COMER89]

Name (verb) To select a *name* from a *name set*.

Name resolution The action of isolating a particular *entity* from amongst a group of *entities*, given a *name* and an *entity* (and by implication a *naming context* and *naming conventions*).

Name set A set of *names*. [COMER87], [COMER89]

Naming context A set of *bindings* [SALTZER79]. Within a given *naming context,* all *names* must be chosen from a single *name set* and all *entities* must be chosen from a single *naming domain*. A *naming context* may be treated as an *entity* for the purpose of naming.

Naming convention A predicate that defines a *name set*.

Naming domain A set of distinguishable *entities* that can be named. [HAUZEUR86]

Naming model A set of constraints on the components of a *naming system*.

Naming network The constraints on the extent to which *entities* that have *names* in one *naming context* can be pointed at or referred to from other *naming contexts*. [COMER87], [COMER89]

Naming subdomain A subset of a *naming domain*.

Naming system The combination of
- a *naming domain*,
- one or more *name sets*,
- one or more *naming contexts*,
- a *naming network*.

Structured name A *name* that is composed of at least one part. (cf. *handle*)

9.3 Summary of additional concepts

Abbreviated name A *synonym* that is shorter than the alternative *name*.

Absolute name A *target entity* is *denoted* by an *absolute name* if all *source entities* use the same *name(s)* for it [COMER87], [COMER89]. Unlike *unique names*, *absolute names* allow *synonyms*.

Address A *name* that *denotes* the location of an *entity*.

Alias Two or more distinct *names* are *aliases* if and only if when they are *resolved* in the same *context*, they *denote* the same *entity*. [COMER87], [COMER89] (cf. *synonym*)

Anonymous entity An *entity* that is not *bound* to any *name* (in a particular *context*).

Attributive name A *name* which one *entity* attributes to another *entity*. It might not be possible to interact with an *entity* using its *attributive name*.

Capability An *invocation name*, knowledge of which confers the right to interact with the so named *entity*. Capabilities are thought of as unforgeable.

Design enterprise Those (people/roles) responsible for the design of a system or part of a system, and who use *attributive names* to *name* the *entities* in the system.

Directory A *naming context*.

Dynamic name A particular *entity* is said to have a *dynamic name* if, in a particular *naming context*, the *entity* is associated with several *names*, but only one *name* is *resolvable* at any time. [COMER87], [COMER89] (cf. *synonym*)

Dynamic naming context A *naming context* in which the *bindings* are of limited validity in a spatio-temporal sense. [COMER87], [COMER89]

Flat name space A *naming network* which consists of a single node and no edges. [COMER87], [COMER89] A *flat name space* has a single *naming context* and only *absolute names* are used within it.

Hierarchical name space A *name space* in which the *naming network* forms a tree. [COMER87], [COMER89]

Home directory A default *naming context*.

Homogeneous naming context A *naming context* in which all *names* have the same structure. [COMER87], [COMER89]

Homonym A *name* which *denotes* more than one *entity*, in a particular *naming context*.

Invocation name A *name* that must be used to interact with an *entity*.

Location transparent name A *name* that does not reflect any information about the location of the *entity* that is *denoted* by that *name* or the *path* that is followed to get to that *entity*.

Migration transparent name A *name* that does not reflect any change in the *path* between the current *context* and the *named entity*, in the face of migration of the *entity* that is *denoted* or any *context* in the *path*.

Name server An *entity* which maps *attributive names* to *invocation names*.

Name space That part of a *naming network* within which all *names* are taken from a single *name set*. [COMER87], [COMER89]

Naming bridge A mechanism that permits two *name spaces* to be connected in a *federation* [HEIMBIGNER81], [HEIMBIGNER85], and that performs the *name* translations as required.

Path That part of a *naming network* which must be traced to advance from one *naming context* to another *naming context*.

Path name A *structured name*, in which the structure of the *name* reflects the physical *path* along which *name resolution* proceeds, when *resolving* each part of the *structured name* in the order defined by the *naming conventions*. [COMER87], [COMER89] (cf. handle)

Plexus (of a *context*). The set of *entities* that can be named from a particular *naming context*.

Relative name A *target entity* is denoted by a *relative name* if *source entities* that are in different *naming contexts* may use different *names* to *denote* the same *target entity*. [COMER87], [COMER89]

Route A sequence of *contexts* in which successive parts of a *structured name* can be *resolved*.

Scope of a name The set of *naming contexts* from which a given *name*, when *resolved*, *denotes* a particular *entity*.

Simple name A *structured name* that consists of exactly one part. A *simple name* cannot refer to the *context* in which that *name* is currently *resolved*. [COMER87], [COMER89]

Simple name space A *name space* in which only *simple names* are used. In a *simple name space* there are no *naming contexts* that define *bindings* onto themselves. [COMER87], [COMER89]

Source entity An *entity* that uses an *invocation name* interact with the *target entity*.

Source routing name space A *routing name space* in which all *names* are *path names*. [COMER87], [COMER89]

Static name In a particular *naming context*, a *static name denotes* the same *entity* for the duration of the epoch.

Synonym An alternative *name*. Two or more distinct *names* are *synonyms* provided that when they are *resolved*, they single out the same *entity*. [COMER87], [COMER89] Note that the *resolution* of the *synonyms* does not have to start in the same *naming context*, as for *alias*.

Target entity An *entity* that is being *named* with an *invocation name* by a *source entity* in the course of interaction.

Unique name A given *name* is *unique* if and only if (a) the name is *absolute* and (b) no other *name* in any *naming context* of the given *naming system resolves* to the *entity* to which the *unique name resolves*.

Unresolvable name A *name* for which there does not exist a *binding* in a particular *context*.

References

[APM1001.1 93]

Rees, R.T.O., “The ANSA Computational Model”, Architecture Projects Management Ltd., Cambridge UK, (1993).

[APM1005.1 93]

Deschrevel, J.P.,"The ANSA Model for Trading and Federation", Architecture Projects Management Ltd., Cambridge UK, (1993).

[BASRI66]

Basri, S.A., “A Deductive Theory of Space and Time”, in the series “Studies in Logic and the Foundations of Mathematics”, North Holland Publishing Company, Amsterdam, (1966).

[COED82]

Concise Oxford English Dictionary, Seventh Edition, 1982.

[COMER87]

Comer, D.E. and Peterson, L.L., “Names and Name Resolution”, in Concurrency Control and Reliability in Distributed Systems, Bhargava, B.K. (ed), Van Nostrand Reinhold Company, New York, 489-524 (1987).

[COMER89]

Comer, D.E. and Peterson, L.L., “Understanding Naming in Distributed Systems”, Distributed Computing, 3(2), 51-60 (May 1989).

[HAUZEUR86]

Hauzeur, B.M., “A Model for Naming, Addressing and Routing”, ACM Transactions on Office Information Systems, 4(4), 293-311 (October 1986).

[HEIMBIGNER81]

Heimbigner, D. and McLeod, D., “Federated Information Bases - A Preliminary Report”, In Infotech State of the Art Report: Database., Infotech State of the Art Reports, Vol. 9, Pergamon Infotech Limited, Maidenhead, UK, 1981, pp.383-410.

[HEIMBIGNER85]

Heimbigner, D. and McLeod, D., “A Federated Architecture for Information Management”, ACM Transactions on Office Information Systems 3(3) 253-278 (July 1985).

[ISO 7498-3]

ISO 7498-3 “Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 3: Naming and Addressing”, First edition, 1989.

[ISO 9594]

ISO 9594 and CCITT X.500 Directory Services.

[ISO 10746-3]

ISO/IEC JTC1/SC21 Information Retrieval, Transfer and Management for OSI, “Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model”, ANSI, New York, November 1992.

[KLEIN87]

Klein, H.K. and Hirshheim, R.A., “A Comparative Framework of Data Modelling Paradigms and Approaches”, The Computer Journal, 30(1), 8-15 (Feb. 1987).

[MOCKAPETRIS88]

Mockapetris, P.V. and Dunlap, K.J., “Development of the Domain Name System”, Communications Architectures & Protocols, ACM SIGCOMM '88, Stanford, California, 1988.

[OSF92]

OSF DCE Revision 1, Update 1.0.1, Application Development Guide, Open Software Foundation, Cambridge MA, July 1992.

[OSF92a]

OSF DCE Revision 1, Update 1.0.1, “Introduction to DCE”, Open Software Foundation, Cambridge MA, July 1992.

[PLATO348BC]

Plato, “Cratylus”, translated by Benjamin Jowe, Great Books of the Western World, Vol.7, Encyclopedia Britanica, Chicago Il, USA, 85-114.

[SALTZER79]

Saltzer, J.H., “Naming and Binding of Objects”, in Operating Systems: An Advanced Course, Bayer et.al. (Eds), Springer-Verlag, Berlin, 99-208 (1977).

[SCHWARTZ87]

Schwartz, M.F., Zahorjan, J. and Notkin, D., “A Name Service for Evolving Heterogeneous Systems”, ACM Operating Systems Review, 21(5), 52-62 (November 1987).