MASTER IN COMPUTER SCIENCE

Introduction to Directory Services

Heuse, Bernard

Award date:
1989

Awarding institution:
University of Namur

Facultés Notre Dame de la Paix, Namur
Institut d'informatique

Année académique 1988-1989

Introduction to
Directory Services

Bernard Heuse

Promoteur : Philippe van Bastelaer

Mémoire presenté en vue de l'obtention
du titre de licencié et maître en informatique

Abstract

The Directory has grown to be an important OSI application as it acts as a focal point and general support for a number of other applications.

This work first points out directory requirements in the OSI framework and other OSI applications, as the Mail Handling System.

The first version of the X.500 standard is then described and some Directory related issues are discussed. In particular, the X.500 Directory as a database system is examined and some directory service implementations are presented.

Keywords : Directory, X.500, route, routing, address, addressing, name, naming, database, distributed system.

Condensé

Le 'Directory' s'est imposé comme étant une application OSI stratégique, puisqu'elle fournit un point de convergence et un soutien pour un certain nombre d'autres applications.

Ce travail met d'abord en évidence les besoins en répertoires dans le cadre de OSI et d'autres applications, tel le 'Mail Handling System' (le service de messagerie électronique).

La première version du standard de Service de 'Directory', X.500, est décrite. Quelques sujets relatifs aux répertoires sont ensuite abordés. En particulier, le Directory X.500 est examiné du point de vue des systèmes de base de données, et quelques implantations de services de répertoires sont présentées.

Mots-clés : Directory, répertoire, X.500, route, routage, adresse, adressage, nom, nommage, base de données, système distribué.

Acknowledgements

The realization of this work follows a six month traineeship at the European Organization for Nuclear Research (CERN). I would like to thank Professor Philippe van Bastelaer who trusted me and gave me the opportunity to live this experience. With great attention did he also supervise the redaction of this dissertation.

I would like to thank people and friends met at CERN, who welcomed me and helped me in the sometimes difficult moments of this period. I think particularly to Concepcion Merino, Ann-Kari Amundsen, Hanne Evensen, Simon Jaffer and Andy Bloor.

Maria Dimou, Tor Bothner and Denise Haegerty guided me, and together with Frederic Hemmer and people of the Communication and Software groups helped me to understand more about E-Mail and other things which are not always easy to learn at University. Maria Dimou is also the main source of information for this work.

Other staff members at CERN helped me one way or the other. In particular, François Fluckiger, Brian Carpenter, Marie-Therese Monnet and people of the User Consultancy Office.

Special thanks to Claire Bawin who kindly read and commented this study.

Table of Contents

Table of Contents I

Introduction 1

Part 1 Directory Needs 3

1. About Names, Addresses and Routes 4
   1.1. Names and Aliases 4
   1.2. Addresses 6
   1.3. Routes 8
2. Directory Needs in the OSI Reference Model 10
   2.1. The OSI reference model 10
      Naming, Addressing and Routing 10
      The three OSI functions 11
   2.2. Use of the OSI reference model 14
      2.2.1. The upper level layers 14
         Addresses 14
         Names 15
         Routes 16
      2.2.2. The problem of the network routing function 17
         The problem of internetworking 17
         The ECMA solution 18
         Difficulties of the ECMA solution 20
         The IP approach 21
         Directory needs in the IP approach 22
      2.2.3. The lower level layers 23
3. Directory Needs in the Mail Handling System 24
   3.1. The Mail Handling System functional model 24
   3.2. Naming and Addressing 25
      Directory Names 26
      O/R Names 26
      O/R Addresses 26
      Name resolution 26
   3.3. MHS use of Directory Services 27
      User-friendly naming 27
      Distribution Lists 27
      Recipient UA capabilities 28
      Authentification 28
   3.4. Further use of Directories in the MH Environment 28
      3.4.1. Information Service 28
      3.4.2. Routing 29
         Classification of E-Mail Systems 29
         X.400 30
         MTA Naming 31
         Functions required for routing in X.400 31
   3.5. Analogy with the OSI reference model 31

4. Other Directory Needs 33
4.1. Distribution Lists 33
4.2. Applications needs 34
4.2.1. FTAM needs 34
FTAM Functional Model 34
Directory needs 35
4.2.2. MTS needs 36
4.2.3. Future Needs 36
4.3. General and generic Directory Service functions 36

Part 2 The X.500 Directory Service 38

5. Models and Concepts 40
5.1. Functional Model 40
5.2. The Directory Information Base 41
5.3. The Directory Information Tree 42
5.4. Directory Entries and Attributes 43
5.6. Naming 44
5.7. Directory Schema 46
5.7.1. DIT Structure Definitions 47
5.7.2. Object Class Definitions 48
5.7.3. Attribute Type Definitions 48
5.7.4. Attribute Syntax Definitions 49
5.8. Access Control 50

6. Directory Abstract Service Definition 52
6.1. The Client-Server Model 52
6.2. Layered Model of the Directory System 54
6.3. Directory ports and services 55
6.4. Common Operation Parameters 56
Service Controls 56
Security parameters 57
Other Common Arguments 58
Other Common Results 58
Entry Information Selection 58
Entry Information 58
Filter 59
Error reports 59
6.5. The Abstract Service Operations. 60
6.5.1. Directory BIND and UNBIND operations. 60
Bind operation 60
Unbind operation. 61
6.5.2. Directory READ operations. 61
Read operation. 61
Compare operation 61
Abandon operation 62
6.5.3. Directory SEARCH operations. 62
List operation 63
Search operation. 63
6.5.4. Directory MODIFY operations 64
Add-Entry operation 64
Remove-Entry operation. 65
Modify-Entry operation. 66
Modify-RDN operation. 67

7. The Distributed Directory....68
7.1. The Distributed Directory System Model....68
7.2. Directory Distribution....69
7.3. DSA interaction model....71
7.3.1. Chaining....71
7.3.2. Multicasting....71
7.3.3. Referral....72
7.3.4. Mode Determination....72
7.4. Knowledge....73
7.4.1. Minimal Knowledge....73
7.4.2. Root Context....74
7.4.3. Knowledge References....75
Internal Reference....75
Subordinate Reference....75
Non-Specific Subordinate Reference....75
Superior Reference....75
Cross Reference....76
7.4.4. Knowledge Administration....76
7.5. The Distributed Directory Operations....76
7.5.1. DSA-Bind and DSA-Unbind operations....76
7.5.2. Other Distributed Operations....77
7.5.3. Distributed Operation Arguments....78
7.5.4. Distributed Operation Results....79

Part 3 Directory Issues....80

8. X.500 : Difficulties and Shortcomings....81
8.1. TR/32....81
Similarities with X.500....81
Replication and Distribution....82
8.2. Evolution of the Directory Standardization....83
8.3. X.500 Difficulties and Shortcomings....84
Distribution....84
Replication....85
Access Control....85
Schema....85
Knowledge....85
Charging and Accounting....85

9. The X.500 Directory as a Database System....87
9.1. Database Modelisation of The Directory....88
9.2. The Global Conceptual Schema....89
9.3. The Local Level Schemata....90
9.4. Mapping....92
9.5. MetaInformation....93

10. The RARE Directory....94
10.1. The need for Directory Services....94
10.2. Network Addresses....95
10.3. Application Entity Titles....97
10.4. The Proposed Interim Name Structure....98
10.5. Some Restrictions to the general X.500 name Structure...99

11. Directory Service Implementations....101
11.1. QUIPU....101
Design Principles....101
DSA database....102
Access Control....102
Schema....103
Distributed operations....103

Other characteristics 103
11.2. THORN 103
Design issues 104
THORN features 104
11.3. The Janet NRS 105
Names 105
Information update and distribution 105
Mappings 105
Characteristics 106
11.4. The ARPA Domain Name Service 106
Names 106
Information distribution and update 106
Functions 107
Characteristics 107
11.5. The EARN NetServ 107
File Server 108
Node Management 108
User Directory Service 108
Access to NetServ 108
Characteristics 108

Part 4 Directory Services at CERN 110

12. EMNODES 111
12.1. The database and the provided service 111
12.2. Definition of the new service 111
12.3. Technical analysis and adopted solution 112
Programming Language 112
Level of Service 113
E-Mail Address 113
User Interface 114
Implementation 115
12.4. The future of EmNodes 116

13. The EMDIR NameServer 117
13.1. The database and the provided service 117
13.2. Definition of the new service 117
13.3. Technical analysis and adopted solution 118
Remote Procedure Calls 118
Message Syntax 118
Implementation 119
Evaluation 120
13.4. The future of the NameServer 120

14. AutoRouter 121
14.1. Definition of the new service 121
14.2. Technical analysis and adopted solution 121
Principle of the Solution 121
Implementation 122
14.3. EmDir issues 123
Some Decisions 123
Security and Reliability 123
Matching Emdir entries and E-Mail Addresses 124
14.4. The future of the AutoRouter 124

Conclusions 126

Annexes 129

Annex A : The Network Address Format A1
A.1. ISO 8348/DAD 2 ..... A1
A.2. Topological example A2
A.3. ECMA-117. ..... A3
A.4. The X.121 Addressing Scheme A4
A.5. The ISO-DCC Addressing Scheme A5

Annex B : Distributed Operation Procedures B1
B.1. DSA Behavior B1
B.2. Managing Distributed Operations. B2
B.3. DSA Procedures. B3
The Operation Dispatcher B3
Name Resolution B4
Evaluation B4
Results Merging B4
B.4. Specific Operations. B4
Single-Object Operations B5
Multiple-Object Operations B5
Abandon Operation B5

Annex C : EmNodes C1
Screen of the EmNodes user Interface C2
Result of the query C2

Annex D : Code of the NameServer D1
Example of message request sent to the NameServer D2
NameServer reply message D2

Annex E : Code of the AutoRouter E1
Emdir.pol E2
Example of message rejected by the AutoRouter E4
Example of message routed by the AutoRouter E4

Annex F :List of Abbreviations F1

Bibliography ..... Biblio I

Introduction

The Directory has grown to be perhaps the most important OSI application as it acts as a focal point and general support for a number of other applications. It is urgently needed in both public and private Message Handling Systems. In the public sector, all current telematic services and the telephone service will be using the Directory as an alternative and, in the longer run, as a replacement for paper directories. In the private sector, a number of future application standards, as FTAM, will make use of the Directory. Directory services are also required for routing functions, both in the Mail Handling System and, at a lower level, in networking activities. Last but not least, it is planned that OSI management standards will use the Directory to store management information.

This all point to an urgent need for Directory implementations. Applications will benefit from having a standardized multi-purpose Directory rather than specific tailor made versions which are different from supplier to supplier, and hence unreachable unless the customer has installed the proprietary access protocol.

The Directory standardization process started in 1984. In 1988, a first Directory standard, known as the X.500 Directory Service, was jointly released by the ISO and CCITT standardization organizations. The X.500 Directory Service provides for accessing, browsing and maintaining a large database distributed over an OSI network, ant to which users are given remote access using OSI.

This work is an introduction to Directory Services. The first part aims to point out directory needs. Chapter 1 introduces the concepts of name, address and route. This will help to understand the next chapters. Chapter 2 describes needs of directory functions in the OSI reference model and in real systems. In particular, the need of directory services at the OSI network level is discussed. Requirements for Directory services in the Mail Handling System are examined in chapter 3, and chapter 4 mentions further needs for various other applications.

The second part introduces the X.500 Directory Service, as described in the standardization documents. Chapter 5 presents the models and concepts of the X.500 Directory. The abstract Directory Service is described in chapter 6, and the way Directory distribution and the Directory distributed service are achieved is examined in chapter 7.

The third part discusses some directory related issues. Chapter 8 presents TR/32, the ECMA Directory, and points out some shortcomings of the X.500 Directory. Chapter 9 considers the X.500 Directory as a database system and examines how a classical database structure can be mapped onto the X.500 Directory structure. Chapter 10 presents the RARE Directory, and chapter 11 describes five directory service implementations.

The fourth part describes the directory services implemented at the European Laboratory for Particle Physics (CERN), where the author followed a traineeship and upgraded two directory services relative to Electronic-Mail. EmNodes, discussed in chapter 12, is a directory of E-Mail nodes, whilst EmDir, described in chapter 13, is a directory of E-Mail users. The 'AutoRouter' described in chapter 14 is a new service based on EmDir which allows to use more user-friendly E-Mail addresses.

Annex A presents the network address format as standardized by ISO and recommended by ECMA. Annex B discusses how distributed operation procedures can be implemented in the distributed environment of the Directory. Annex C show the user interface developed for EmNodes, and annexes D and E contain the listings of the set of programs described in chapters 13 and 14. A list of abbreviations used throughout this study can be found in annex F.

Part 1 Directory Needs

This first part aims to point out directory needs in telecommunication systems, and in particular in the Open System Interconnection (OSI) framework.

The need for a Directory Service arises from the contrast between the constant change of an OSI network as a whole, and the need to isolate (as far as possible) the user of the network from those changes. Thus a user of the Directory Service is able to view the network as a more stable entity than a network user who is not a user of the Directory. For example, if the location of a resource in the network changes, then the user of that resource will not be affected by that change, provided that a 'name' rather than the physical location was used to reference the resource.

Another need for Directory Services arises from the desire to provide a more 'user-friendly' view of the OSI networks. For example, the use of aliases, the provision of a yellow page service, etc... help to relieve the burden of finding and using network information.

After having introduced the concepts of name, address and route, directory needs in the OSI reference model and the Mail Handling System will be examined. Further needs are also mentioned in the fourth chapter.

1. About Names, Addresses and Routes

This chapter takes up the concepts introduced in [Shock-78], [Tan-81] and [Huit-88].

Before beginning to speak about any kind of directory service, it is important to clarify the differences between names, addresses and routes. While closely related, these three concepts provide different functions in the telecommunication world.

In an extremely large definition, one could say [Shock-78]:
- the name of an object indicates what we seek,
- an address indicates where it is, and
- a route tells us how to get there.

1.1. Names and Aliases

An object can be a person, a resource, a machine, a set of objects, a set of information about an other object or anything else the user chooses.

A **name** is a symbol identifying an object in a certain world. To have such a property, the same name must not be allocated to more than one object. It has to be unique in a single world. Therefore, naming strategies have to be defined. Two of the most common are described hereafter [Huit-88].

1) A central registration authority can be set up within a world, which will guarantee the uniqueness of names. Before allocating a name to an object, the proposition has to be submitted to the registration authority. The name will be refused if it is already used, or accepted and added to the set of allocated names if it is not the case. This method is called **flat naming**. It requires a central registration authority, and the procedure to allocate a name may become heavy when the world contains many objects.

2) As a single authority for the whole world is not practical, **hierarchical naming** is used. The name space is divided into domains. Object names are then composed of two parts:
- A domain identifier (e.g. the domain name), and
- An identifying name within the domain.

The domain identifier is allocated by a central registration authority, but the procedure is faster and easier as domains are far less numerous than objects.

Within a domain, an authority guarantees that the allocated names to objects are unique. Domains can themselves be divided into subdomains by the domain authority, or use a flat naming scheme.

When the domain identifier is missing in a name, it can be defaulted to, for example, the domain identifier to which the user which specified the name is linked. This has to be done by a local process when names are specified.

An object can be identified by several names, i.e. different names can be allocated to the same object.

A name needs not to be meaningful to all users and needs not to be drawn from a uniform name space. For example, the division of different domains into subdomains and the subdomain name allocation can be done in accordance with different independent criteria, according to the different domain structures or local name syntaxes used in the domains, and unknown from other users.

To be useful in a telecommunication context, a name has to be mapped into an address. But the name has not to be bound to the address until the mapping takes place.

A name referring to an object may be disallocated. Thereafter, this object cannot be anymore referred to with this name. The same name may be latter allocated to another object, in which case it refers to the new object and not the old one.

An object may be located at several different addresses (e.g., an object representing a person may have two telephone numbers, or two E-Mail addresses). The address or addresses associated with a particular name may change over time.

The interest of referencing an object by a name rather than by its address is that names are independent from the location of the objects, whilst addresses are not (an object address may change over time).

An **alias** is an alternate name for an object. It's worthwhile to distinguish two kinds of aliases, depending on the scope of their usage.

1) As previously stated, an object may have different names. It is the case if the same object has to be referenced inside several domains, and has been allocated names by each domain in order to facilitate object naming by users (see figure 1.1). Such names can be potentially used by any user to refer to an object in the world, even if they will prefer to use only one of them. These **alias-names** have thus a global scope.

2) The way to allocate names may lead to user-unfriendly names (complex hierarchical name structure or meaningless names). A user (or a set of users) may wish to use a synonym to refer to an object. This synonym is only significant for this user, as the same symbol can be used by other users as synonym for other names. Such an alias has thus a local scope. We will call it a **synonym**.

In fact, a synonym does not directly refer to an object, but to the object name which directly refers to the object. So a synonym to address mapping requires two steps:
- the synonym to object name mapping;
- the object name to address mapping.

The first function has to be provided locally by the user or a user's process.

An example is illustrated in figure 1.1.

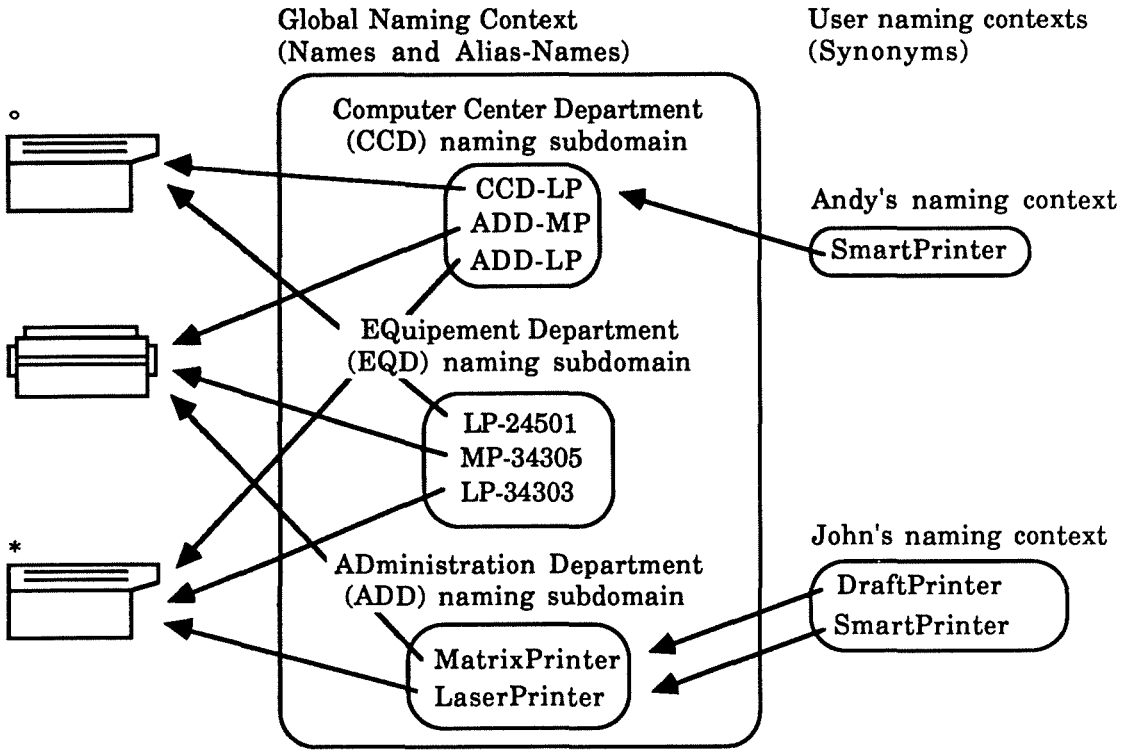


Figure 1.1 : Alias-names and synonyms.

In the above schema, the laser printer * may be referred to as "ADD-LP" in the CCD subdomain naming context, as "LaserPrinter" in the ADD subdomain naming context, and as "LP-34303" in the EQD subdomain naming context. If it is assumed that hierarchical naming is used and that global names are the concatenation of the object name in a subdomain and the subdomain name separated by a dot, all the names "ADD-LP.CCD", "LaserPrinter.ADD" and "LP-34303.EQD" globally refer to the laser printer *. From the Computer Center Department point of view, both "LaserPrinter.ADD" and "LP-34303.EQD" are alias-names for the object it refers to as "ADD-LP". For John, "SmartPrinter" refers to the laser printer *, while for Andy, "SmartPrinter" refers to the laser printer °. Both are synonyms used to refer the objects respectively globally known as "LaserPrinter.ADD" and "CCD-LP.CCD".

1.2. Addresses

An **address** is a data structure whose format has to be recognized by all the elements responsible for communications, and which defines an addressable object.

As an address is designed to locate an object within a world, it must be meaningful to these elements throughout the world, and must thus be drawn from some uniform address space.

This address space may be a flat one which spans the entire world (such as Ethernet addresses), or it may be a hierarchical address space (such as telephone numbers). Address assignment may be done in the same way as name allocation.

An object may be located at several different addresses (e.g. a person may have two E-Mail addresses).

To enter in communication with an object, the object name has to be mapped into an address. If the name to address mapping has produced several different addresses for a particular object, some form of information may prove useful in selecting a preferred address.

In the case of alias-names, the alias-name to address mapping can be proceeded in two ways:

1) If the addresses of an object are associated to every of its alias-names, there is no difference between a name and an alias-name. The alias-name to address mapping is thus quite simple: a name to address lookup in some kind of directory is enough to find the address or addresses of the object the name refers to. But a problem arises when one of the addresses of the object changes. We have to change the mapping function (the directory) for every name or alias-name referencing to the object. This can be difficult as, usually, an object does not point to all of the names referencing it.

2) In order to solve this problem, we can allocate a base-name to an object. The addresses of the object are only associated to this base-name. Alias-names refer to the base-name of the object. When an address is updated, the mapping function has to be only changed for the base-name. Alias-names can not be directly mapped into addresses. The address mapping function is a little bit more complex: it has to detect and dereference alias-names. Usually, we will only have one indirection, e.g, no alias-names for alias-names.

An example is illustrated in figure 1.2.

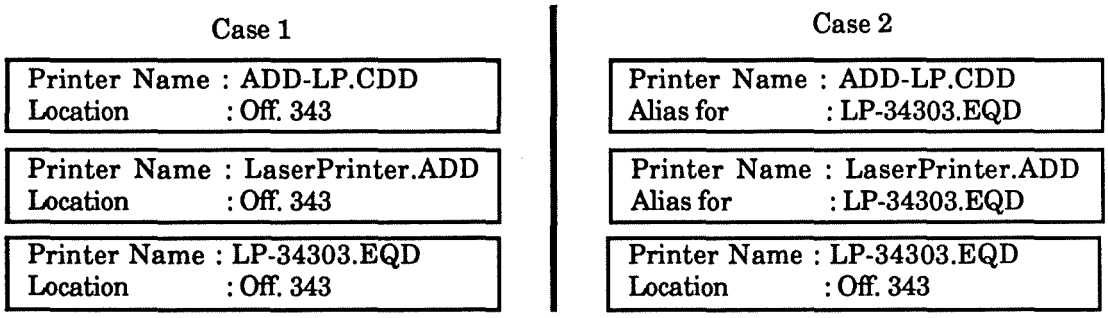


Figure 1.2 : Alias-name to address mapping.

All entries refer to the same printer object whose address is 'Off. 343'. In case 1, access to the address from the name is direct, but when the printer moves to another office, three entries have to be updated. In case 2, "LP-34303.EQD" is the base-name of the printer while "ADD-LP.CDD" and "LaserPrinter.ADD" are alias-names for this printer. When the printer moves, only one entry has to be updated (the one corresponding to the base-name), but access to the address is indirect if the object has been referred to by an alias name.

At the time one wishes to communicate with a particular address, there will be some mechanism that will map an address into an appropriate route.

The address need not to be bound to the route until this mapping takes place. The choice of an appropriate route may change over time.

The interest of locating an object with an address rather than with a route is that a path is obviously depending on the origin as well as on the destination, and that a route may change over time, depending on the availability of the intermediate switching points.

1.3. Routes

A **route** is a specific information needed to reach an object at its specified address from a given location.

The routing action may require only one step to reach a destination (a direct route), or it may require a series of steps in order to forward the information on its way.

Where there is merely one hop in a simple topology, the routing decision is usually straightforward. The routing information can only consist of the destination address, and the address to route mapping function is useless.

When the path to an address requires several steps, the route defines a path through intermediate switching points. In some cases, several routes are possible to reach a particular address.

In categorizing routing mechanisms, one dimension is the place at which each intermediate routing decision is taken [Shock-78]:

1) The source may specify all of the intermediate routing decisions, and include this information along with the data being sent (source routing). The source must have fairly comprehensive information about the environment, but the switching points do not need to maintain routing tables.

2) Alternatively, the source may only specify the destination address, and the intermediate switching points choose the next portion of the route (hop-by-hop routing). In this case, the source only needs enough information to reach the first switching point, but each of the switching points must then have a routing table of some sort.

3) There may be hybrid routing combining these two, in which the source specifies certain major intermediate points, but allows the underlying system to choose routes between those points.

A second dimension of the routing mechanisms is the time constant of the information upon which the routing decisions are based, i.e. when that information is specified, and how frequently it is modified [Shock-78]:

1) Routing tables may be set once, left unchanged for a relatively long period of time, and only changed to reflect major modifications to the system (fixed routing).

2) Alternatively, routing tables may be updated relatively frequently, reflecting shorter term changes in the environment (dynamic routing).

If there is a dynamic routing system, providing periodic updates of appropriate information, a third dimension of the routing process is the control mechanism [Shock-78]:

1) Individual switching points may try to update their information in an isolated manner, e.g. by periodically trying various routes and observing performances.

2) Changes in the environment, connectivity or performance may all be reported back to a central point, which is responsible for promulgating this information to the sources (if source routing is used) or to the switches (if hop-by-hop routing is used).

3) Alternatively, control of the dynamic update process may be distributed among all the sources or switches, which individually obtain information about the state of the system.

Thus, a name may be used to derive an address (by a local process which has to enter in communication with an object), which may then be used to derive a route (by the system which is responsible for the communication to take place).

Note that a route can be the name or the address of an object (e.g. of a switching point or of the destination object), and that an address can be used as a name if one and only one object is located at this address.

2 Directory Needs in the OSI Reference Model

After having briefly described it, the first section points out directory requirements raised in the OSI reference model. The second section discusses directory needs in real systems. In particular, requirements for directory services for routing at the network level is examined.

2.1. The OSI reference model

This section takes up the concepts relative to naming, addressing and routing described in the OSI reference model [ISO-7498].

Naming, Addressing and Routing

An Entity is an active element in an open system.

An entity within the OSI environment is identified by its Global-Title (GT), composed of two parts:
- the Primary-Title-Domain-Name (PTDN), which identifies the layer, and
- the Local-Title (LT), which identifies the entity inside the layer.

A Global-Title is a name.

Local-titles may have a hierarchical structure.

> An example of GT could be 'Network:CH.CERN.CERNVAX-X25', where 'Network' is the PTDN and 'CH.CERN.CERNVAX-X25' the LT.

In the following, (N) refers to a layer in the OSI reference model, i.e an application, presentation, session, transport, network, data link or physical layer.

(N)Title refers to the Global Title of a (N)Entity.

A (N)Service-Access-Point ((N)SAP) is a point where the services of the (N)Entity are provided to the (N+1)Entity.

A (N)Service-Access-Point-Address ((N)SPA or (N)Add) is the address of a (N)SAP. A (N)Add identifies a (N)SAP.

A (N)Entity can be attached to one or more (N)SAP. A (N)Suffix (also called (N)SAP-Selector) is a part of the (N)Add allowing to select the particular addressed (N)SAP among the (N)SAPs attached to the (N)Entity.

A (N)SAP is attached to at most one (N)Entity.

A (N)Entity can be attached to one or more (N-1)SAP.

A (N-1)SAP is attached to at most one (N)Entity.

Figure 2.1 shows a possible configuration of attachment of (N)SAPs and (N-1)SAPs to (N)Entities.

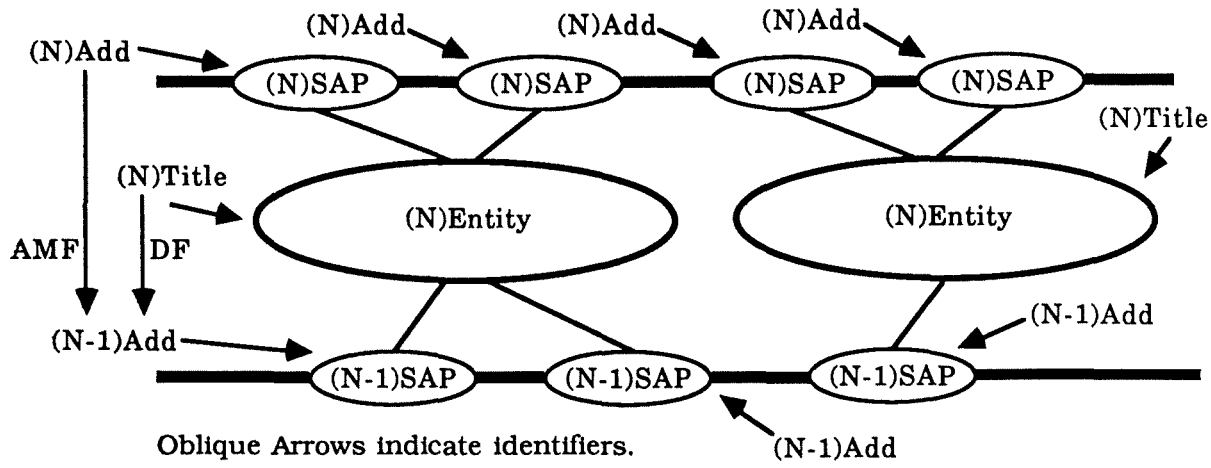


Figure 2.1 : Entities and SAPs.

For example, in the above figure, each entity could represent a transport-entity. A transport-entity is attached to a network-SAP. A network-SAP address could be an X.121 address (e.g.: 02281234567890). The address of the transport-SAP through which this transport entity provides services to a particular session-entity could be 02281234567890-44, where '44' is the transport-suffix allowing the transport-entity to determine to which session-entity a particular service request has to be forwarded (i.e. through which Transport-SAP the service has to be provided). AMF represent the Address Mapping Function, and DF the Directory Function (see OSI functions).

The (N-1)Addresses of (N-1)SAPs to which a (N)Entity is attached are the addresses of this (N)Entity.

A (N)Relay is a (N)function by which a (N)Entity forwards data received from one (N)Entity to an other (N)Entity. (N)Relays correspond to intermediate switching points.

The three OSI functions

The OSI reference model specifies three functions which should be achieved at the layer level:

- (N)Directory function: by which the GT of a (N)Entity is translated into the (N-1)Addresses of a (N-1)SAP to which the (N)Entity is attached (A name to address mapping function).
- (N)Address-Mapping function: which provides the mapping between the (N)Addresses and the (N-1)Addresses associated with a (N)Entity (An address to address mapping function).
- (N)Routing function: which translates the GT of a (N)Entity or (N-1)Add of the (N-1)SAP to which the (N)Entity is attached into a path by which the (N)Entity can be reached (e.g. a (N)Relay to use). Note that a route depends on both the current location of the information being sent and its destination.

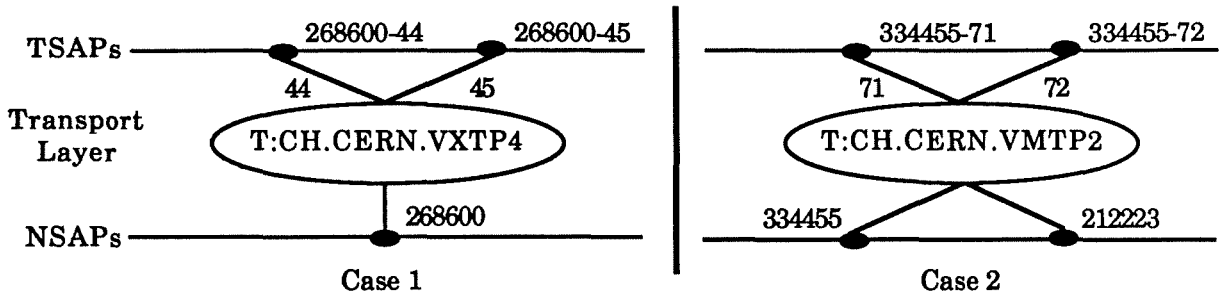


Figure 2.2 : Example of entities and SAPs at the transport level

To illustrate the three functions, let's consider the example of figure 2.2, taken at the transport level. In this example, Network-SAP-Addresses are 6 digits addresses (e.g. 268600), and two digits numbers (as 44 and 45) are Transport-suffixes. T:CH.CERN.VXTP4 is the Global-Title of a transport entity, and it is assumed that T:CH.CERN.TPRelay is the Global-Title of a transport relay, whose NSAP-Add is 102030.

Let's first consider case 1.

- the Transport-Directory function maps transport entity titles into Network-SAP addresses:

e.g. : T:CH.CERN.VXTP4 -> 268600.

- the Transport-Address-Mapping function maps a transport-SAP-address into a network-SAP address:

e.g. : 268600-44 -> 268600.
268600-45 -> 268600.

- the Transport-Routing function maps an identifier of transport entity into a route, e.g. here the Global-Title of a transport entity into the name of a transport-relay:

e.g. : T:CH.CERN.VMTP2 -> T:CH.CERN.TPRelay

This function could map the NSAP-Add giving access to the destination transport entity into the NSAP-Add of a relay to use in order to reach the destination entity:

e.g. : 334455 -> 102030.

In the more complex example of case 2,

- the Transport-Directory function maps:

T:CH.CERN.VMTP2 -> (334455, 212223).

- the Transport-Address-Mapping function maps, if the hierarchical structure (see below) of addresses is used:

334455-71 -> 334455.
334455-72 -> 334455.
212223-71 -> 212223.
212223-72 -> 212223.

With this mappings, from an external point of view (e.g. from the point of view of a session entity), a TSAP has several addresses. In fact the different addresses denote different routes in the OSI stack (usage of one or another NSAP). If a remote session entity fails to enter in communication with a TSAP with a given address because of an unreachable network address, it can try another TSAP-Add.

If an address routing function using tables is used, which maps:

334455-71 -> (334455, 212223).
334455-72 -> (334455, 212233).

then a TSAP may be identified by a single TSAP-Address from an external point of view. When an attempt to enter in communication with a TSAP fails because of an unreachable network address, the originator's transport entity can try another network address produced by the Transport-Address-Mapping function. The transport entities are able to manage incidents at the network level and choose other routes in the OSI stack. This Address-Mapping function allows more flexibility, and the unavailability of some NSAPs is transparent to the session layer, which is better.

As a (N)SAP is attached to at most one (N+1)Entity, a (N)Add identifies a particular (N)SAP to which a (N+1)Entity is attached (see figure 2.1). When the (N+1)Entity is detached from the (N)SAP, the (N)Add no longer provides access to the (N+1)Entity. If the (N)SAP is reattached to a different (N+1)Entity, then the (N)Add identifies the new (N+1)Entity, and not the old one.

The use of a (N-1)Add to identify a (N)Entity is the most efficient mechanism if the permanence of attachment between (N)Entities and (N-1)SAPs can be assured. If there is a requirement to identify (N)Entities regardless of their current locations, then the entity Global-Titles ensures correct identification, and the (N)directory function provides the mean to translate entity Global-Titles into (N-1)Adds to which they are attached. This function corresponds to the name to address mapping mechanism pointed out in chapter one.

As a (N)SAP can be attached to one and only one (N)Entity, a (N)SAP-Add also identifies the (N)Entity to which the corresponding (N)SAP is attached. Thus when Global-Titles are not used to identify a (N)Entity, to use a (N)SAP-Add of one of the (N)SAPs to which the (N)Entity is attached is a proper way to identify this (N)Entity. In this case, the use of the (N)Directory function is not possible. Thus, interpretation of the correspondence between the (N)Adds served by a (N)Entity and the (N-1)Adds used for accessing the (N)Entity is performed by the (N)address-mapping function.

Two particular kinds of (N)address-mapping functions may exist within a layer:
- hierarchical (N)address-mapping, and
- (N)address-mapping by tables.

If a (N)Add is always mapped into only one (N-1)address, then hierarchical construction of addresses can be used. The (N)address-mapping function needs only to recognize the hierarchical structure of a (N)Add and extract the (N-1)Add it contains. In this case, a (N)Add consists of two parts:
- a (N-1)Add of the (N)Entity which is supporting the current (N)SAP of the (N)Entity.
- a (N)Suffix which makes the (N)SAP uniquely identifiable within the scope of the (N-1)Add.

For example, at the transport level, a transport-SAP address can be constructed with an X.121 address plus the transport suffix identifying a particular session service, e.g. : 02281234567890-44. Transport-address-mapping consists of extracting the X.121 network-SAP address out of the transport-SAP address, i.e. 02281234567890.

Within a given layer, a hierarchical structure of addresses simplifies (N)address-mapping functions because of the permanent nature of mapping it presupposes. It is not imposed by the model in all layers in order to allow more flexibility in (N)address-mappings, in particular in the case where a (N)Entity attached to more than one (N-1)SAP supports only one (N)SAP. In this case, a (N)SAP may be accessed through several (N-1)SAPs and thus a (N)Add may be mapped into several (N-1)Adds. Hierarchical construction of addresses is not suitable if any of these (N-1)Addresses may be used, and if one does not want to impose to always use the same (N-1)SAP and but wants to keep using always the same (N)Add (see example of figure 2.2).

If a (N)Add can be mapped into several (N-1)Adds, or if a (N)Add is not permanently mapped into the same (N-1)Add, then hierarchical construction of addresses is not possible and the (N)address-mapping function may use tables to translate (N)Adds into (N-1)Adds.

The structure of a (N)Add is known by the (N)Entity which is attached to the identified (N)SAP. However, the (N+1)Entity does not know this structure.

The (N)routing-function translates a (N)Entity identifier (i.e. a (N-1)Add or a (N)Title) into a path or route by which the (N)Entity may be reached. This path or route is an information ((N-1)Add or (N)Title) identifying either the destination (N)Entity if a direct route can be used, or a (N)Relay entity if the communication cannot take place via a direct route.

2.2. Use of the OSI reference model

This section describes the way the OSI reference model is used in the reality, and points out directory needs in real systems.

Let's here use the term **directory** to refer to a global system which holds information, and provides access and update operations on this information.

2.2.1. The upper level layers

Addresses

In the upper level layers (Application, Presentation, Session and Transport layers), the addresses follow a hierarchical structure.

An application address is the address of the Presentation-SAP (PSAP) to which it is attached. As a hierarchical address structure is used in the upper level layers, the PSAP-Add is composed of the Session-SAP and the Presentation-Suffix. Applying the same decomposition to the Session-SAP-Add and the Transport-SAP-Add, one could say an application address is composed of 4 parts:

Application address = P-Add = N-Add + T-suffix + S-suffix + P-suffix.

Each suffix has to be optional. It is not necessary in the case where a (N)Entity always provides its services to the same (N)SAP, i.e. to the same (N+1)Entity. For example, many applications use Presentation service entities which are dedicated to an application.

Figure 2.3 illustrates a possible layer configuration.

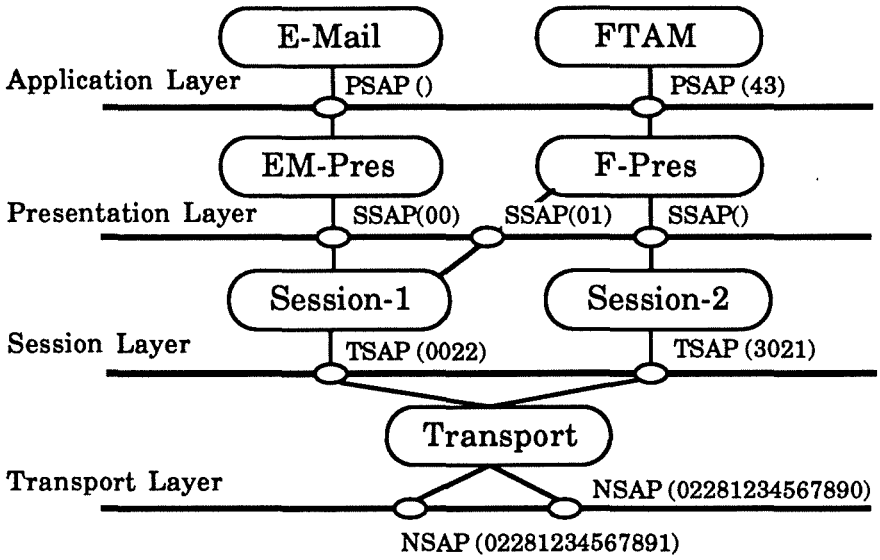


Figure 2.3 : Example of layer configuration.

In the above example, an address of the E-Mail application is <N=02281234567890; T=0022; S=00>, while an address for the FTAM application is <N=02281234567890; T=3021;P=43>. The Session-1 entity may also be used for the FTAM activity, so <N=02281234567890; T=0022; S=01;P=43> is an alternate address of the FTAM entity. An alternate NSAP may also be used, so an alternate address for the FTAM application is also <N=02281234567891; T=0022; S=01; P=43>.

The address mapping function can thus be performed by a relatively simple algorithm extracting the (N-1)Add out of the (N)Add. This is done by removing the (N)Suffix out of the (N)Add. There is no need of any address-mapping function to be held in tables, which could be stored in a directory.

Note that several differents applications can be accessed through the same Network-SAP as their addresses can just differ by the suffixes.

Names

Applications are usually referred to by an Application-Entity-Title (AET), whilst presentation, session and transport entities are not referred to by any kind of title. They can only be identified by the (N-1)Add of the (N-1)SAPs to which they are attached. This can be justified by the fact that their location is stable. Giving AETs to applications is done in order to provide more user-friendly identifiers to applications. A Directory function has to map AETs into Presentation-Addresses.

In real systems, an AET is associated to only one Presentation-Address. This facilitates the directory requirements, but indeed involves problems when an entity is moved from a SAP to another one. All applications which have to use its services must be warned of the address change. This would have been avoided if a (N)Directory-function associating the name of a (N)Entity to its (N-1)Add would exist (a (N)Add being formed of a (N)Entity name plus a (N)Suffix). In this case, we just would have to update the directory function. But every layer would have to use a directory function, which would be quite inefficient.

This way of proceeding also prevents to associate alternate addresses to an application. If, for example (see fig. 2.2), a TSAP is unavailable (e.g. because its corresponding session-entity is overloaded), and that a similar session-entity is available on the same system through another TSAP, we are unable to use it because the application address identifies a particular TSAP as being the only one to access the application. If fact, the problem arises because this address structure also imposes a route through the OSI stack. An address-mapping function done by tables, associating more than one (N-1)Add to an (N)Add, could have solved this problem. Another way to deal with it is that a connection-refused diagnostic also specifies an alternate address to access the target (N+1)Entity. But this has to be specified in the (N)protocol. We will see that the application-directory-function will at least allow to specify more than one network address.

The first interesting service which could be provided by a directory is thus the application-directory-function, which translates an AET into the PSAP-Address of the application entity.

Routes

The use of (N)Relay-Entities is relatively limited in the upper level layers of the OSI stack.

The first kind of such (N)Relay could be called a switching point. This has to be used if, for some reason (e.g. because no direct connection is possible, for cost considerations or because we don't know the route to a destination entity address, but we know the address of a relay knowing it), a direct route cannot be used to reach or communicate with the destination. Therefore, we have to use a (N)Relay-Entity. In the Mail Handling System (MHS), a MTA is a relay.

A particular case occurs when two entities cannot communicate directly between each other because they are using different protocols. Some protocol translation has to be performed in order to make them understand each other. Such a relay is usually called a gateway. But this is relatively rare in the upper level layers. One example would be a TCP/IP-TP4 gateway. A major exception is the use of such a protocol translator at the application level in the Mail Handling System, in order to interconnect E-Mail systems using different protocols.

The (N)routing-function provides a path to reach another (N)Entity. A path is a specific information needed to reach an object at its specified address. The (N)routing-function can be achieved by providing the (N)Global-Title of the destination entity (and then using the (N)directory-function to get the (N-1)Add), or directly the (N-1)Add to which the (N)Entity is attached if both entities can directly communicate with each other. If not, the (N)routing-function has to provide the same information about a (N)Relay-entity.

The routing-function is usually provided by tables associating a destination identifier to the identifier of the next entity to reach (either the destination, or a relay). As a routing function depends both on the source and destination location of the communicating entities, the tables have to be particular for every (N)Entity.

A directory could play a role as routing information provider. At the moment, in most pragmatic solutions, no use of a real global directory in such a role is made. Entities have their own routing tables which have to be changed by some static or dynamic mechanisms when information of changes in the environment are received. So, the two following proposals are strictly prospectives:

- The tables could be held in the directory (one per entity). These tables could be updated by some central or distributed mechanisms external to the entity. In some way, these mechanisms will have to be informed of changes in the configuration topology. When this occurs, it computes every new table and updates changes.
- Information about the configuration topology could be held in the directory. Entities have access to this information and use it to update their tables (either on a periodic basis, or when a connection fails). Changes in the configuration topology have to be echoed in the directory.

2.2.2. The problem of the network routing function.

This section is related to the problem of addressing and routing at the network layer level, in the case where several networks are interconnected. This subject has deeply been debated in the RARE working groups 3 and 4. The main ideas discussed here are picked up from [WG3-88].

Describing the NSAP format defined by the standards and the network address formats are out of the scope of this document. Readers who have no knowledge in this field should first read Annex A.

The problem of internetworking

Addressing conventions are local to a network (e.g.: Ethernet addresses for Local Area Networks (LANs), X.121 addresses for Public Data Networks (PDNs), ...). When an entity wants to reach another entity attached to the same network, it can use the local addressing scheme, and routing is performed by the network service provider on base of this address.

When several such networks are interconnected, the term 'subnetwork' is used to refer to an individual network, whilst the term 'network' is reserved to refer to the global set of the interconnected subnetworks. The physical network addresses (e.g. Ethernet or X.121 addresses) of entities or relays on a subnetwork are referred to as the 'SubNetwork Point of Attachment Addresses' (SNPA-Adds).

When an entity attached to a subnetwork wants to reach another entity attached on another subnetwork, the two subnetworks being interconnected (as shown in figure 2.3), a network relay has to be used, which picks up data from a subnetwork and forwards them onto the other subnetwork. Routing to reach the relay is performed by the (sub)network service provider and based on the SNPA-Add of the relay. But the decision to use the network relay to route the connection toward the other subnetwork has to be issued by some higher routing mechanism.

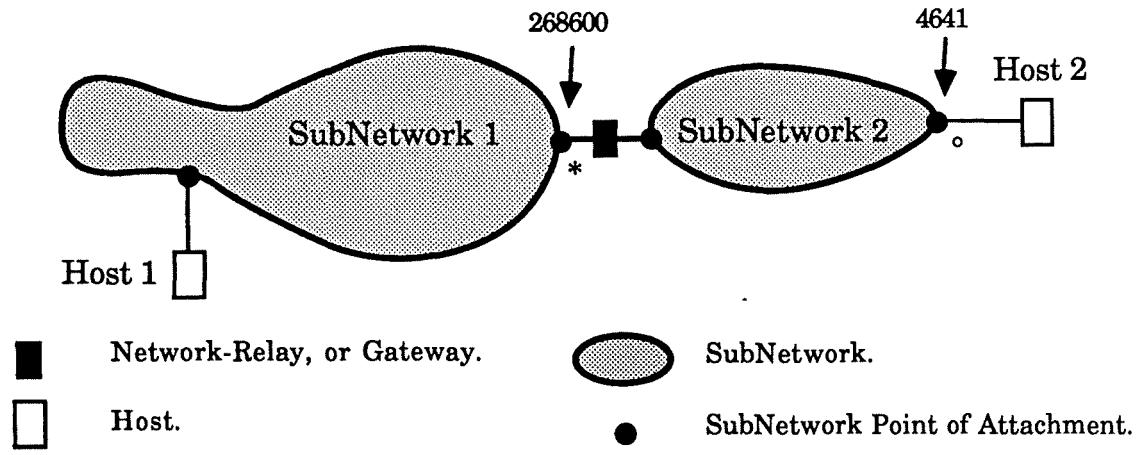


Figure 2.4 : Simple SubNetwork Configuration.

In the case of figure 2.4, when 'Host1' wants to communicate with 'Host2', the first step is to set up a connection with the relay (this has to be determined in some way), using the SNPA-Address of the relay in the first subnetwork. Then, in some way, the relay must know the address of the entity (SNPA-Add of Host2) the originator wants to enter in communication with.

The ECMA solution

A possible way of solution is that the local address of the relay is followed, in the address field of the connection request packet (NSAP-Add of the call packet), by the address of the entity to reach on the other subnetwork. So, the NSAP-Add has the following structure:

SNPA-Add of the relay	SNPA-Add of the destination

E.g., in the case of figure 2.4, when Host 1 wants to communicate with Host 2, the communication has to pass through the relay interconnecting the two subnetworks. The SNPA of the relay to subnetwork 1 is pointed by the character '*'. Its SNPA-Add could be '268600'. The SNPA of Host 2 to subnetwork 2 is pointed by the character 'o'. Ley's assume its SNPA-Add is '4641'.

When communicating with Host 2, Host 1 gives the following address in the packets submitted to subnetwork 1 : '268600-4641'. Subnetwork 1 only interprets the first part of the address (i.e. '268600') to route the packets toward the relay. When the packets arrive at the relay, the second part of the address is extracted ('4641'), and used by the relay to forward the submit the packets to subnetwork 2 in order to forward them toward the destination, i.e. Host 2. A similar situation is found when a connection oriented communication is considered.

Such a kind of solution is proposed and refined by ECMA-117. This solution is particularly suitable when simple topologies are involved, and when the originator of the call is connected to a X.25 network.

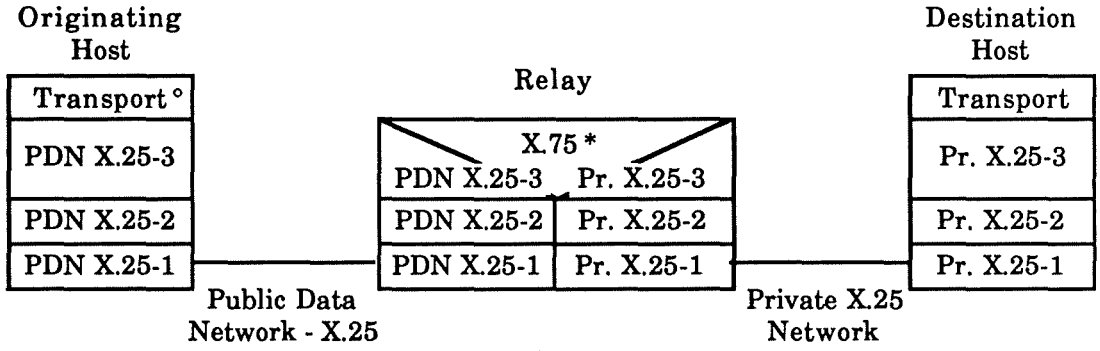
That's why such a solution has been adopted by public networks to solve the problem of internetworking when the originator is directly connected to a PDN. Addresses in PDNs are based on NSAP-Adds containing X.121 IDIs and, optionally, another subnetwork address in the DSP (both are SNPA-Adds). The X.121 address identifies a point of attachment of an other non-PDN subnetwork to the PDN (i.e. a relay), and the information in the DSP identifies the entity to reach on this subnetwork.

Note that several interconnected PDNs can be considered as forming a single subnetwork. The X.121 addressing scheme unify the different address spaces, as a X.121 address identify both the PDN to which an entity is attached, and the point of attachment of this entity to its PDN (see Annex A). Interconnection of PDNs is transparent to end systems.

When such addresses are used, the X.25 network sets up a connection with the relay, and then the relay can extract the addressing information out of the call packet in order to set up a connection with the destination entity.

This kind of solution requires that the network address specified in the packet at the network level can accept more than just an address in accordance to the local addressing scheme. This is possible in X.25 networks, as the address field may be of variable length, and only the first specified address is interpreted. Further information in the address field is ignored.

With this kind of solution, in some cases (e.g. connection of a PDN and a private X.25 network with a relay simulating a X.75 connection), the relay can be transparent to the end entities (i.e. no software has to be added in the layered architecture of end entities in order to deal with internetworking). Figure 2.5 shows such an architecture.



° Provides NSAP

* Extracts SNPA

Figure 2.5 : Possible layered architecture for the ECMA solution.

The X.121 address space is equivalent to a single subnetwork. Therefore, there is no problem for entities directly connected to a X.25-X.121 Public Data Network to have a connection with entities which are connected to a PDN, or to a subnetwork directly connected to a PDN.

Difficulties of the ECMA solution

The ECMA solution has the apparent advantage of pushing the routing to the real subnetwork service providers. It is known to work in a majority of cases, but it has the following disadvantages [WG3-88]:

- This solution only considers connections issued from a X.25-X.121 network: it does not provide an inter-subnetwork routing in non-X.25 environments (e.g. because the address field does not tolerate additional information to the local subnetwork address) or for X.25 networks not part of the X.121 addressing scheme which want to communicate with entities connected to another subnetwork via a PDN. Such systems need to know locally how to connect to the X.121 based X.25 subnetwork, and then use the specified address to set up a connection with the destination entity.
- If the PTTs change an X.121 address, then the NSAP for all attached systems will need to be changed (IDI part). The same happens when the physical subnetwork address format change, e.g following a subnetwork technology change (DSP part).
- The sequence of addresses in the NSAP determines a path for the connection through the relays specified by their SNPA-Adds. It does not provide alternate routes (e.g.: if a relay is down) for local systems or real subnetworks that are connected to more than one X.25 provider, or connected at more than one X.121 address. The generality can however be achieved by recording more than one NSAP with the PSAP address in the directory.
- In complex topologies as in figure 2.6, it is necessary to have a more sophisticated mechanism (e.g., a NSAP-Add comprising a X.121 address, followed by a private X.25 network address, followed by an Ethernet address). Such a sequence of addresses will not fit in the NSAP-Add field. In fact, it is not possible to specify more than two addresses in the NSAP-Add field.

ECMA-117 tries to deal with this last problem by only specifying the last SNPA-Add in the DSP, prefixed by a subnetwork identifier identifying the subnetwork to which this SNPA-Add is relevant. The relay has to know how to set up a connection with this subnetwork thanks to his local knowledge of the configuration of the subnetworks and their characteristics. So a NSAP-Add following the ECMA-117 recommendation has the following structure:

X.121 Add (IDI)	SubNet. Id. + SNPA-Add (DSP)

In the case of figure 2.6, let's assume that the SNPA-Add on the PDN of the relay interconnecting the X.25 PDN and the Private X.25 network (Pr. X.25), pointed by the character '*', is '268600'. The SNPA-Add on the Pr. X.25 of the relay interconnecting the Pr. X.25 and LAN8, pointed by '^', is '2738'. The SNPA-Add of Host 2 on the LAN8, pointed by 'o', is '4641'. Eventually, it is assumed that '08' identifies the LAN8 in tthe set of subnetworks interconnected to the Pr. X.25.

According to the ECMA address structure, when Host 1 wants communicate with Host 2, the following address will have to be used : '268600-08+4641'.

As in the former example, the PDN interprets only the first part of the address and routes the packets toward the relay between th PDN and the Pr. X.25. When packets arrive at the relay, the second part of the address is extracted by the relay. The subnetwork identifier '08' is used by the relay to determine (by some local mechanism, e.g. table lookup) the SNPA-Add of the next relay to use (i.e. '2738'). The packets are forwarded to the relay interconnecting the Pr. X.25 and LAN8, where the last part of the address ('4641') is used to send the pachets to the destination, i.e. Host 2. A similar situation is found when a connection oriented communication is considered.

In complex topologies, ECMA-117 suggests that it could be necessary to use logical rather than physical addresses in the DSP, and thus to do at least one look-up of NSAP-Add to SNPA-Add.

Figure 2.6 shows a possible configuration of subnetworks, which can help to understand the problems involved by such a solution.

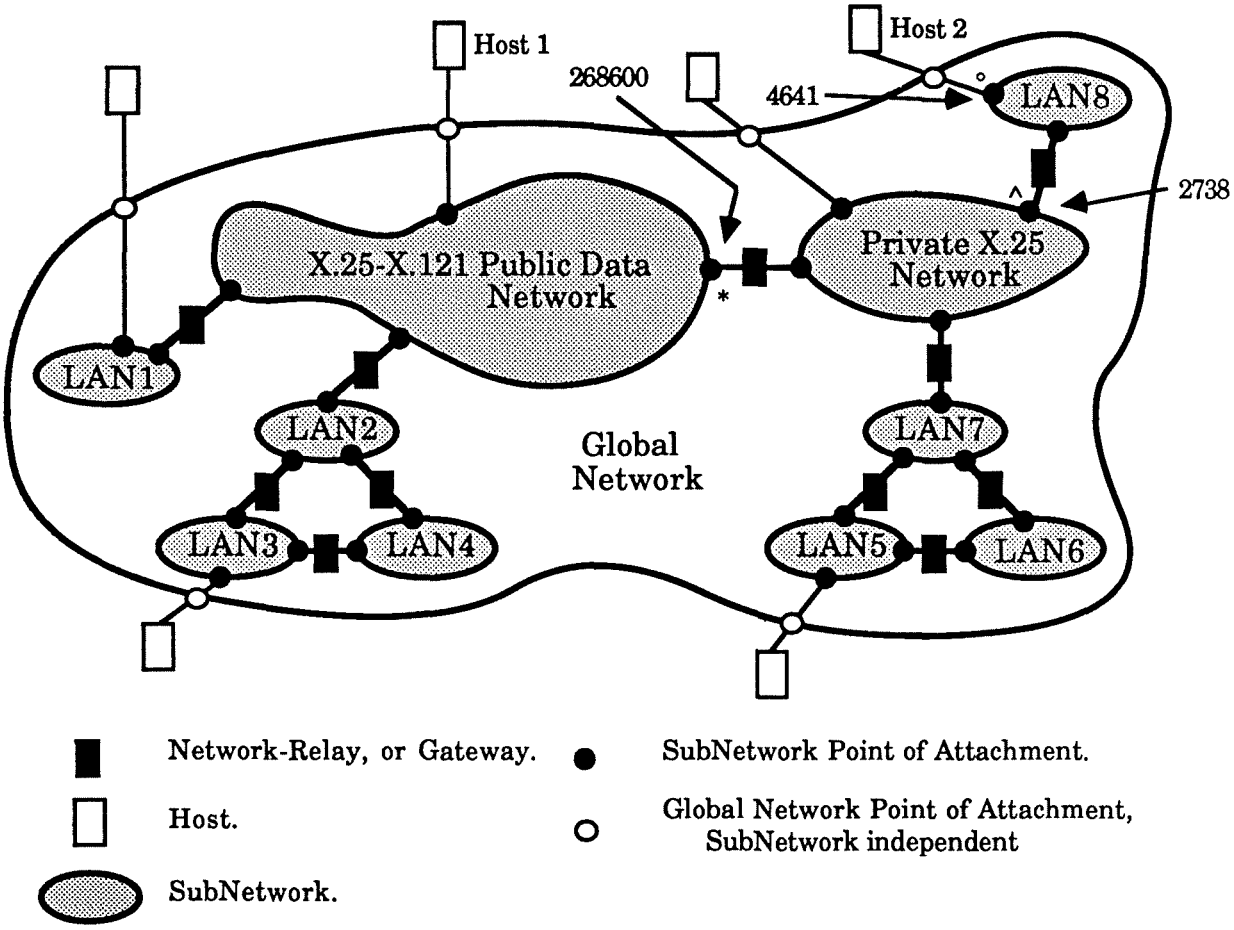


Figure 2.6 : Possible SubNetwork Configuration.

The IP approach

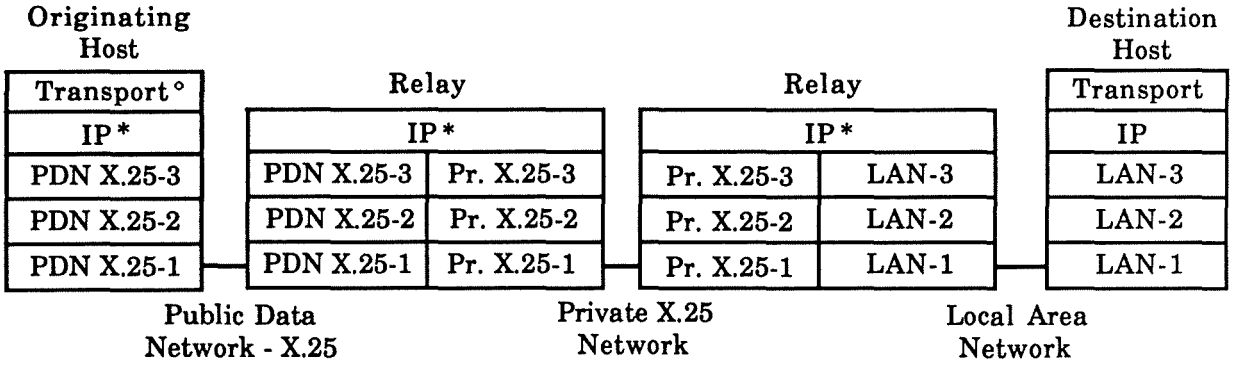
In fact, using a sequence of physical SNPA-Adds in NSAP-Adds is equivalent of putting routing information in the addresses. But an address should be independent from the routes, as routes vary according to the source and destination locations and the availability of paths. Moreover, this imposes a routing structure which may not be valid in the future (e.g. if a physical point of attachment of an relay entity to a subnetwork changes).

Rather than using some kind of sequence of physical or logical addresses in the NSAP-Add, another solution is to have an addressing scheme identifying an entity at the global network level, independently from the real subnetwork to which this entity is attached (see figure 2.6).

The NSAP should be a unique global address, carried fully in the calling and called address fields. At the originating entity, and at the boundaries of the subnetworks (i.e. in relays between, for example, a X.25 and an Ethernet networks), the subnetwork address (e.g. X.121 or Ethernet address) that defines the SubNetwork Point of Attachment (SNPA) of the target or relay entity on the next subnetwork must be derived from the NSAP (normally by table look-up). The relays between subnetworks will have to derive from the NSAP address, the addressing information to use in the real next subnetwork when relaying information.

This approach which is called 'Internet Protocol' (IP) has been developed by the DOD as part of the ARPA Internet project (the well-known TCP/IP). A protocol similar to IP has been developed by ISO. Both provide a connectionless or datagram service between end systems. Note that the TCP layer of TCP/IP provides a connection oriented service between end systems.

This approach imposes a different layered architecture in the systems involved in order to support internetworking. An IP layer is inserted between the transport and the (sub)network layer, whose function is to perform the NSAP to SNPA mapping. This is shown in figure 2.7.



° Provides NSAP
* Derives SNPA

Figure 2.7 : IP layered architecture.

ISO-DCC is a mechanism for allocating such subnetwork independent addresses. These addresses do not have any physical relationship with the topology or technology of the subnetworks to which entities are attached. The network address is a unique physical subnetwork independent identifier.

Directory needs in the IP approach

The use of a network-independent NSAP format (as the ISO-DCC scheme) implies that a mapping of NSAPs to SNPAs is required in every station and relay entity, e.g. by some directory look-up mechanism. It must be noted that the NSAP to SNPA relationship is not globally applicable. It is relevant only to the local subnetwork on which the information has to be forwarded. The solution to this NSAP-Add to SNPA-Add resolution problem requires thus a series of directory services at the IP level that allow the originating entity and the intermediate relay systems to progress the network service through interconnected subnetworks.

This implies that the total body of information needed to perform internetworking is not just one global directory for the mapping of NSAPs to appropriate SNPAs, but several such directories, potentially one per real subnetwork.

Note that this approach requires two level of routing at the network layer level:
- The first is done at the global network level (IP level), in order to route through the different subnetworks (deriving SNPAs), and
- The second is done internally in the subnetwork in order to route until the next SNPA.

So, difficulties arise when one considers the choice of the NSAP format. Subnetwork independent schemes are preferred because they are not affected by subnetwork reorganisation, changes in the subnetwork provider nor in the subnetwork technologies. They may also accommodate more easily multiple connections to the public subnetworks and alternate routes involving different subnetwork technologies. But adopting such a scheme will require some directory look-up mechanisms, because it requires some binding to SNPA-Adds in order to provide a route.

2.2.3. The lower level layers.

On a pure esthetical point of view, it would be interesting to see to what the three functions identified in the OSI reference model are reduced at the Network, Data Link and Physical levels. In reality, this is highly technology dependent.

Functions can have no existence (there is no need of physical routing function, as the signal can only be sent through the media, whatever the destination is), or can be provided by external means to the entity, and without it even being aware of them (a bridge performs a routing function at the network level in interconnected LANs).

There is a large variety of solutions. Describing such solutions falls out of the scope of this discussion. But as it is generally admitted that they are depending on the technology, the network topology, and the systems configurations, the needed functions are provided locally by the system or the technology itself. So they do not use any function which requires a high level global directory service.

3. Directory Needs in the Mail Handling System

The Electronic-Mail system upon which this chapter is based is the standard towards which all current E-mail implementations are heading: the CCITT X.400 series of recommendations, also known as the X.400 Message Handling System (MHS). The 1988 version of the X.400 MHS makes use of some kind of directory services.

3.1. The Mail Handling System functional model

A user of the MHS is either a person or a computer process wishing to send or receive a message by means of the MHS services. A MHS user is referred to as either the 'originator' (when sending a message) or the 'recipient' (when receiving a message).

The model described by the X.400 recommendations is composed of two levels [Der-88]. The lower level is the Message Transfer System (MTS). The MTS comprises a number of Message Transfer Agents (MTAs) operating together in a store and forward manner in order to transfer and deliver messages to the intended recipients.

The upper level is composed of User Agents (UAs), which provide a kind of interface between the MTS and the users they support. There is one UA per user. The originator prepares messages with the assistance of his or her UA. A User Agent is an application process that interacts with the Message Transfer System to submit messages on behalf of the user. The MTS delivers the messages submitted to it to one or more recipients. The recipient UAs receive messages delivered by the MTS and help users to interpret them.

A UA can offer other services, as storage and retrieval of previous messages, editing and naming facilities, information or directory services, ...

A UA is associated to one and only one MTA. The main interaction between UAs and their respective MTAs are the submission and the delivery of messages. A MTA first accept a message submitted by an originator UA, forwards it to the next MTA towards the destination and so on until the message arrives at the MTA of the destination UA, which delivers it.

Messages transferred by the MTS between UAs are composed of an envelope and a content. The envelope carries information that is used by the MTS when transferring the message, i.e. mainly the originator and recipient identifiers (O/R Names). The content is the piece of information that the originator UA wishes delivered to one or more recipient UAs, and which is normally left untouched [X.400-88].

A Distribution List (DL) is a component of the message handling environment that represents a pre-specified group of users and Distribution Lists, and that is a potential destination for the messages the MHS conveys [X.400-88].

The Distribution List service enables an originating UA to specify a Distribution List in place of all the individual recipients (users or other DLs) mentioned therein. The MHS will add the members of the list to the recipient of the message and send it to those members. DLs can be members of DLs, in which case the list of recipients can be successively expanded at several places in the MHS.

MTAs are grouped in management domains which own and control them. These domains can be managed by a public administration, in which case we speak of ADministration Management Domains (ADMD), or by private organizations, in which case they are called PRivate Management Domains (PRMD) [Der-88].

The MTS has been designed to be able to support any kind of connectionless communication. Cooperating UAs can use a protocol of their own in the message content to communicate. The InterPersonal Messaging Service (IPM Service) provides a user with features to assist in communicating with other IPM Service users. The IPM Service (e.g. Telex) uses the capabilities of the Message Transfer Service for sending and receiving interpersonal messages.

Figure 3.1 illustrates the functional model of the Mail Handling Environment.

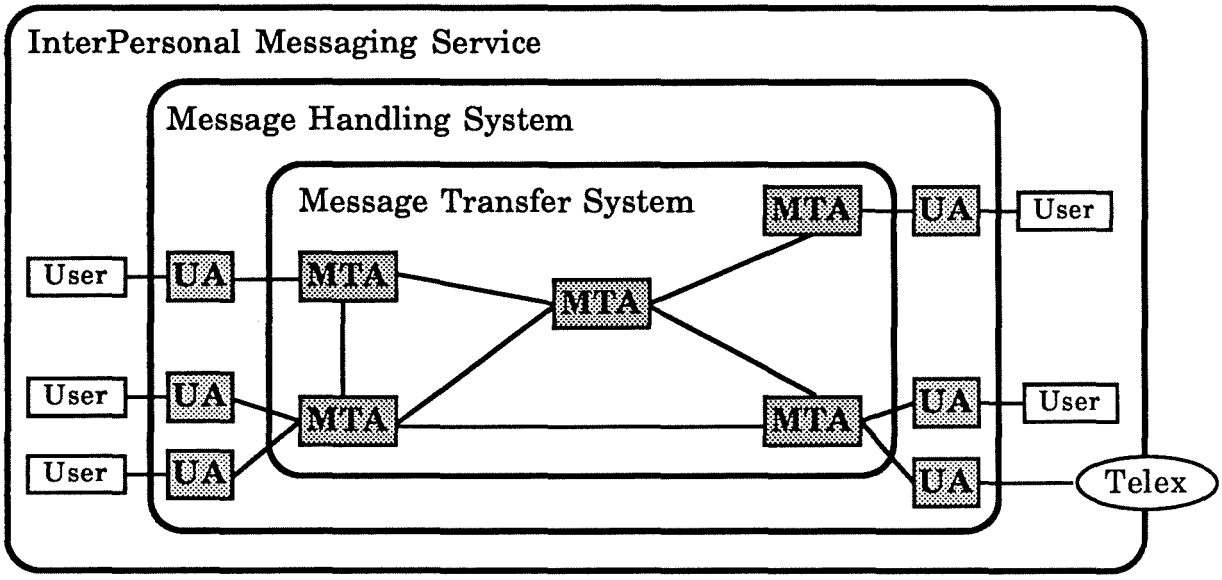


Figure 3.1 : The MHS Functional Model.

3.2. Naming and Addressing

This section takes up descriptions of [X.400-88].

In the MHS, the principal entities which require naming are the users, i.e. originators and recipients of messages. In addition, Distribution Lists have names for use in the MHS.

Users of the MHS and DLs are identified by O/R Names. An O/R Names is composed of a Directory Name or an O/R Address or both.

The prefix 'O/R' recognizes the fact that user can be acting as either the Originator or the Recipient of the message.

Directory Names

Users and DLs can be identified by a name, called a Directory Name. A Directory Name must be looked up in the Directory to find out its corresponding O/R Address. The structure and the components of Directory Names are described in Part 2 and defined in [X.521-88].

The typical Directory Name is more user-friendly and more stable than the typical O/R Address because the latter is necessarily expressed in terms of the organizational or physical structure of the MHS while the former need not to be.

MHS users or DLs will not necessarily have a Directory Name, unless they are registered in a Directory. As Directories become prevalent, it is expected that Directory Names will be the preferred method of identifying MHS users to each other. In fact, many users will lack Directory Names until the Directory is widely available as an adjunct to the MHS [X.402-88].

O/R Names

An O/R Name is an identifier by means of which a user can be designated as the originator or as a potential recipient of a message. Every user or DL has one or more O/R Names. An O/R Name distinguishes a user or DL from another.

An O/R Name comprises a Directory Name or an O/R Address or both. The O/R Addresses have to be used in order to identify users or DLs if they have no Directory Names.

O/R Addresses

An O/R Address contains information that enables the MHS to uniquely identify a user to deliver a message, and permits the MTS to locate the user relative to its own organizational or physical structure.

An O/R Address is a collection of information called attributes. O/R Addresses are allocated in a hierarchical way. [X.402-88] specifies how O/R Addresses are constructed.

Name resolution

In order to reference his addressee, a user would specify the O/R Name of the latter. But since an O/R Address is also an O/R Name, it is possible to specify the addressee directly by his or her O/R Address, if the latter is known. This has the advantage of being more cost effective since the Directory lookup necessary to map an O/R Name into an O/R Address is bypassed. But an O/R Address has the drawback of being uneasily remembered and of being more likely to change than a Directory Name [Der-88].

When only the Directory Name is specified as O/R Name, a resolution mechanism has to take place. This may be performed thanks to the Directory. Both UAs and MTAs can use the Directory to resolve names:

- The UA can directly access the Directory to find out the O/R Address of a user or of a DL, and provide a full O/R Name (Directory Name and O/R Address) to the MTS at message submission.
- As an alternative, a UA can supply just the Directory Name to refer to the recipient at message submission, and the MTS would then itself ask the Directory to resolve the recipient's O/R Address and add it to the envelope. The originating MTA normally carries out the Directory Name to O/R Address lookup.

So, either one or both components of an O/R Name can be used on submission of a message. If only the Directory Name is present, the MHS will access the Directory to attempt to determine the O/R Address, which it will then use to route and deliver the message. If the Directory Name is absent, it will use the O/R Address as given. When both are given on submission, the MHS will use the O/R Address, but will carry the Directory Name and present both to the recipient. If the O/R Address is invalid, it will then attempt to use the Directory Name as described above.

The analogy between the HHS and the OSI reference model is discussed in section 3.5.

3.3. MHS use of Directory Services

Mainly due to the lack of a standardized Directory Service, the first CCITT X.400 recommendations were designed for use without any kind of standardized Directory Service. But now that Directory Service standardization is progressing, work is proceeding within MHS standardization on the use of Directory Services.

X.400-88 describes what could be the use of directory capabilities by the MHS. The use of the Directory in message handling falls into the following four categories [X.402-88].

User-friendly naming

As describe here above, the originator or recipient of a message can be identified by means of his Directory Name rather than his or her O/R Address. At any time, the MHS can obtain the latter from the former by consulting the Directory.

Distribution Lists

A group whose membership is stored in a directory can be used as a Distribution List. The originator simply supplies the name of the Distribution List. At the Distribution List expansion point, the Directory Names and O/R Addresses of the individual recipients can be obtained by consulting the Directory. Distribution Lists are further discussed in the next chapter.

Recipient UA capabilities

A MHS message can potentially include several types of encoded information (text, facsimile, picture, graphics and even sound). All UAs are not able to interpret and process all types of contents, and thus, it can appear that some messages are not deliverable to such UAs, or that such a message has to be converted from one encoded information type to another (e.g. : from graphics into pictures). Other UAs characteristics are pertinent to the MHS (e.g., maximum deliverable content length).

MHS capabilities of a recipient (or originator) UA can be stored in a directory. At any time, the MHS can obtain (and then act upon) those capabilities by consulting the Directory.

Authentification

Before two MHS functional entities (two MTAs, or a UA and a MTA) communicate with one another, each can want to establish the identity of the other. A security policy can be set up by using authentification capabilities of the MHS based on information stored in the Directory.

Aspects of an asymmetric key management scheme to support such a policy are provided by the Directory System Authentification Framework [X.509-88]. The Directory stores certified copies of public keys for MH users which can be used to provide authentification and to facilitate key exchange for use in data confidentiality and data integrity mechanisms. The certificates can be obtained from the Directory.

3.4. Further use of Directories in the MH Environment

3.4.1. Information Service

Even if Directory Names are said to be user-friendly, it is not necessarily obvious to find out the Directory Name of a particular user. It can be necessary to have some kind of service which would allow, from scattered information known about a MHS user, to find its Directory Name and/or O/R Address.

For example, a MHS user could designate the recipient of a message by filling up a form on its screen [Huit-88], e.g. :

Country	:	_____
City	:	_____
Organization	:	CERN _____
Unit	:	_____
Common name	:	Maria Dimou _____

The O/R Address and Directory Name will be found in the Directory by the UA and inserted as O/R Name in the envelope of the message.

To avoid to oblige a user to specify the Directory Name of a user every time he or she has to send a message, the UA can offer naming facilities, i.e capabilities to define and manage local synonyms for O/R Names.

3.4.2. Routing

Routing has soon been noted as a potential application of Directory services, but is not being pursued in any details. The lack of standardized mechanism to use a directory service to perform routing is surprising [Kille-6.5/M].

For those concerned with management domains, the need to specify a routing mechanism is important. It is desirable that the approach taken will lead itself to using the standardized directory service when it becomes available.

In the most general case, the routing mechanism has to be performed in two steps (source routing is not considered):
- The O/R Address to MTA mapping, which determines the next MTA to reach in order to forward the message toward its destination.
- The MTA to MTA connection information mapping, which allows the local MTA to get the information to set up a connection with a remote MTA.

In the following sections, after having classified E-Mail systems according to their topology and some addressing issues, the case of X.400 and the MTA naming issue are discussed. Then the routing functions required by X.400 are described.

Classification of E-Mail Systems

Among existing E-Mail systems, there is clearly two types of topologies [Kille-6.5/M].

1) Systems which have full connectivity (e.g. : based on packed switched networks). Each MTA can set up a connection with any other MTA.

In such a case, the address to MTA mapping function is quite facilitated if the address is of the form 'user@host' (where the 'host' part denotes the name of the MTA the user is attached to), because the name of the MTA to reach is directly derivable from the address. Messages are always transferred in a single hop. UAs are co-resident with MTAs, and the 'user' part simply identifies the UA in the context of the MTA.

Routing is performed simply by extracting the MTA name out of the O/R Address and looking up the MTA in a table to determine connection information. The table may be maintained centrally and distributed to all MTAs. This is the ARPANET old style.

But this scheme does not allow decentralized allocation of the 'host' part of O/R addresses, based on a hierarchical structure of domains. If using 'user@$subdomain_n$. $subdomain_{n-1}$. ... .$subdomain_1$. domain' like addresses, the MTA name cannot necessarily directly be derived from the address, but a lookup table mechanism has to be used to find out the corresponding MTA name. This table may also be maintained centrally and distributed to all MTAs.

Having found the MTA name, deriving the MTA connection information is done as described above. This is the extended ARPANET old style.

This approach involves that each MTA needs to know every other MTA, and it implies the distribution and maintenance of central tables. Strictly, these are the directory issues.

2) Systems based on bilateral links between MTAs, which have a rather sparse connectivity.

Any link is established by bilateral agreement between a pair of MTAs. Each MTA issues a list of its connections. These lists are collected centrally, and then re-issued to every MTA. Each MTA can then process a full topological sort of the network, and use this to determine the next hop to any given remote MTA. Processing and distributing this map clearly become expensive as the number of nodes increases. This is the UUCP style.

If domain hierarchy is used in the addressing scheme, messages can be routed on the basis of the domain hierarchy, without necessarily identifying the complete path to the final MTA (i.e. : a message is first routed towards its destination domain or subdomain rather than directly towards its destination MTA). Such a scheme reduces the map processing and update effort, at the expense of increasing message traffic through key nodes of the topology. This is the Australian ACSNet style.

X.400

X.400 falls somewhere between full and sparse connectivity. An OSI purist would say that all routing should be done at the network layer, and the only reason for a connection using one or more MTA hops is due to the lack of adequate directory service. But there are good reasons for MTA relaying [Kille-6.5/M]:

- The need to interact with systems which support the InterPersonal Messaging Service with protocol other than the one specified in X.400 (e.g.: other E-Mail systems, Telex, ...);
- To facilitate Encoded Information content conversion;
- In some cases, a sequence of MTA level hop may be more efficient than a single hop. For example, funnelling traffic through a pair of MTAs may optimize utilization of an expensive communication link;
- To enable transparent operations to take place within Management Domains boundaries. Having a fixed set of entry points will allow complex or dynamic reorganization, and yet appear static externally. The extra cost of tracking changes with directory lookups might well exceed the cost of an extra MTA level hop;
- If connection between any pair of MTAs cannot be fully automated but requires bilateral agreements and/or human intervention, full connectivity will be difficult to reach;
- When authorization and charging criteria are taken into account on the basis of E-Mail message level parameters, it is desirable to apply them at the MTA level. For this reason, PTTs will wish to continue to provide an MTA service, rather than a directory service which simply announces where other MTAs are. Because this indeed would permit private MTAs to avoid to use the public MTA service when charging becomes heavy.

It is clear that some MTA relaying is desirable. But it should be minimized within the above constraints.

MTA Naming

A key issue will be whether or not to give MTAs a namespace separated to that for O/R Addresses [Kille-6.5/M]. Most of non X.400 systems have the MTA identified within the O/R Address. However, X.400 has not adopted this solution. This probably comes from the historic evolution of X.400. The first decision to identify UAs by address rather than by name, and the wish to have user friendly identifiers leads to using a hierarchical domain addressing scheme, independent from MTA names. Separating the MTA namespace allows O/R Addresses to more naturally reflect the identity of the associated user, and for users' O/R Addresses to remain unchanged when his or her MTA changes. As a consequence, there is no natural way to extract the MTA component off from an O/R Address.

Also, O/R Addresses identify UAs. MTAs are separate from UAs, and therefore should have a separate namespace. This is particularly important to identify intermediate MTAs, which might have no associated UAs.

However, none of the X.400 protocols requires a separate MTA namespace, and managing a second name space could be unnecessary. When directories will be fully used, UAs will be identified by Directory Names, not by O/R Addresses. Therefore, O/R Addresses could be structured in a manner which is convenient to the MHS.

It is clear that either approaches can be used to construct a workable scheme. But the difficulty of labelling intermediate MTAs and small MTAs (e.g.: on workstations) suggests that it is useful for MTAs to have a separate namespace.

Functions required for routing in X.400

So clearly, the X.400 will, in a first phase at least, have a sparse connectivity and use a MTA namespace separated from the O/R Address space. This will require two kinds of functions which could be provided by a directory:

- Routing function, in order to determine, from the O/R Address, the next MTA to reach on the path to the destination of the message, and
- MTA to MTA connection information mapping function, to determine how to set up a connection with a remote MTA given by the first function. The connection information could include the PSAP address of the MTA, the supported protocol versions, and other MTA capabilities.

Routing mechanisms are not described in X.400. A fortiori, the way routing function could be offered by a directory is not detailed. A solution based on the X.500 directory service structure is proposed in [Kille-6.5/M].

3.5. Analogy with the OSI reference model.

If we express the MHE system in terms of a layered model analogous to the OSI reference model, one can say that the MHS is composed of two levels:

- the MTS layer composed of MTA entities, and whose function is to transfer and deliver messages to the intended UAs;
- the UA layer composed of UA entities, and whose function is to provide an interface between the MTS and MHS users. A UA represents, and is only associated to, one user.

This is illustrated in figure 3.2.

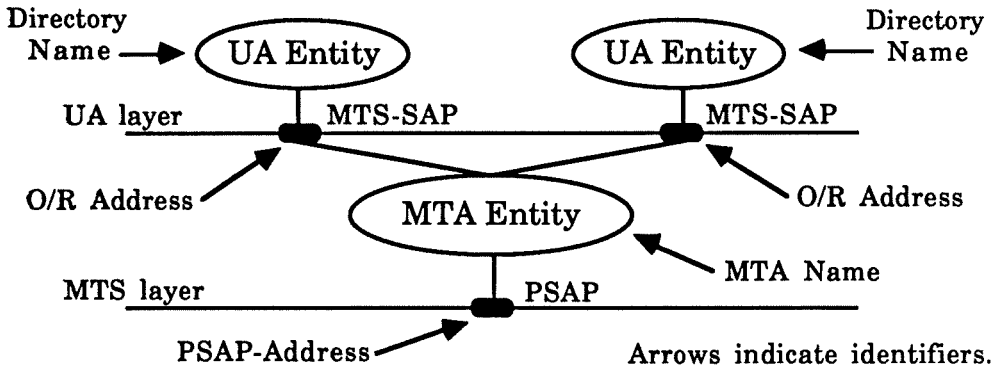


Figure 3.2 : Layered model of the MHS.

A Directory Name identifies a user, and thus its associated UA entity, in a user-friendly way. A Directory Name is thus similar to a Title.

An O/R Address is an identifier which allows the MTS to locate a UA entity. An O/R Address is thus similar to what could be called a 'MTS-SAP Address'.

The Directory function identified in the OSI reference model (which maps (N)Titles into (N-1)Adds) corresponds here to the Directory service function which maps a Directory Name into an O/R Address.

The routing function maps an O/R Address into the identifier (either the MTA name or the MTA connection information) of the next MTA to which the message has to be forwarded.

The MTA to MTA connection information function maps a MTA name into the connection information needed in order to set up a connection with the named MTA (i.e. mainly the PSAP-Add of this MTA).

Note that as previously discussed, the MTA namespace will probably be separated from the O/R Address space. And thus a function which maps an O/R Address into the MTA name of the MTA to which the UA corresponding to the O/R Address is attached will not be able to used some hierarchical structure of the O/R Address (e.g. by extracting the MTA name out of the O/R Address), but will have to use some kind of table lookup mechanism.

Such a function will probably not exist as hop by hop routing is preferred than the use of direct routes. The crucial function is thus the routing function which directly maps an O/R Address into the next MTA to reach (rather than making a routing decision in two steps, i.e. first mapping the O/R Address into its corresponding MTA (destination MTA), and then usage of a routing function which determines the next MTA to reach in order to forward the message toward the destination MTA). The routing function will anyway use tables of some sort as a route is tightly linked to the current location of the message and the topology of the network.

4. Other Directory Needs

This chapter presents other directory needs which could or should be provided by a directory service, and is aiming to show that the above described needs are not an exhaustive list.

The first section is related to Distribution Lists, an extension of the Mail Handling Service, and describes the X.400 DL service structure with its directory needs.

The second section is about application specific needs, and points out particular needs of the FTAM and MTS services.

The third section introduces more general and generic directory services.

4.1. Distribution Lists

Communications in groups often make use of Distribution Lists (DLs) to allow an originator to send a message to a list of recipients without the need to directly refer to all members. A DL can be seen as a mechanism which maps a single O/R Name (the one of the group) into a set of O/R Names (the ones of the members of the group) and hence E-Mail addresses.

A submitting entity may submit a message to the list by sending the message to the list O/R Name. The message is then distributed to the members of the list. This process is known as the expansion of the list.

List expansion is performed in the MTS by a special function of the MTA responsible for the expansion of the list, and to which the original message is first routed.

As DLs may be members of other DLs, control mechanisms have to be set up to avoid message looping and duplication problems.

Main properties of DLs are the following [X.400-88]: a DL has
- An owner : a user who is responsible for the management of the list;
- An expansion point, identified by an O/R Name. The O/R Address identifies the expansion point, which is the MTA where the O/R names of the members of the DL are added to the recipients of the message. The message is transported to the expansion point before expansion;
- Submitting members, i.e. the names of the entities (users or other DLs) who may submit a message to the list, i.e. send a message to the list;
- Receiving members, i.e. the names of the entities (users or other DLs) who will receive copies of messages submitted to the list.

Further properties may be defined [Ben-6.5/M]. A DL may have:

- A textual description of the DL functions, describing its purpose and general information relevant to the list, which should be received by any submitting or receiving member when joining the list;
- A textual description of the joining procedure to the list, both for submitting and receiving members;
- An auditor, which refers to an entity who is to receive information concerning confirmations and notifications related to the list activity;
- A moderator, which refers to an entity to whom messages are to be forwarded which have been submitted to the list without permission to do so. Such messages may be further forwarded to the list if they pass moderation;
- An administrator, which refers to an entity who performs administrative operations on the list, as addition of members or changes in the list description;
- Charging algorithms, i.e. a text describing the charging issues for the list;
- Access control, which defines who may perform operations onto a DL.

Operations on DLs may consist of adding, deleting and retrieving submitting or receiving members of a DL, checking membership of an O/R name in a DL as submitting or receiving member. One may also want to create, delete, rename, update a DL or read the description or properties of a DL.

All the properties of a DL may be stored in a directory, with appropriate access control. Processes or entities which have to manage and expand the DL perform the described operations by using a directory service.

4.2. Applications needs

4.2.1. FTAM needs

In addition to the PSAP-address of an application entity, another information that is required by some protocols during the set up of an association between application entities is the Application-Entity-Title (AET). It is the case in FTAM.

FTAM Functional Model

File Transfer Access and Management (FTAM) is concerned about interchange of files between end systems. It provides not only file transfer between heterogeneous systems, but also the ability to perform record access and file management [Handbook].

In order to allow files to be exchanged between different manufacturer equipments, they must share a common description or definition. In FTAM, this takes the form of an externally visible image of an average filestore. This is the Virtual FileStore (VFS). The VFS is defined as a collection of files, and in such a way that any real computer filestore can easily be described in VFS terms.

FTAM is an asymmetric function. There is an initiator and a responder. The initiator is the controlling side, and the image of the VFS resides at the responding end. The image provided to the user at the initiator site is out of the scope of FTAM, but could be managed by a user element plus a user interface. This is illustrated in figure 4.1.

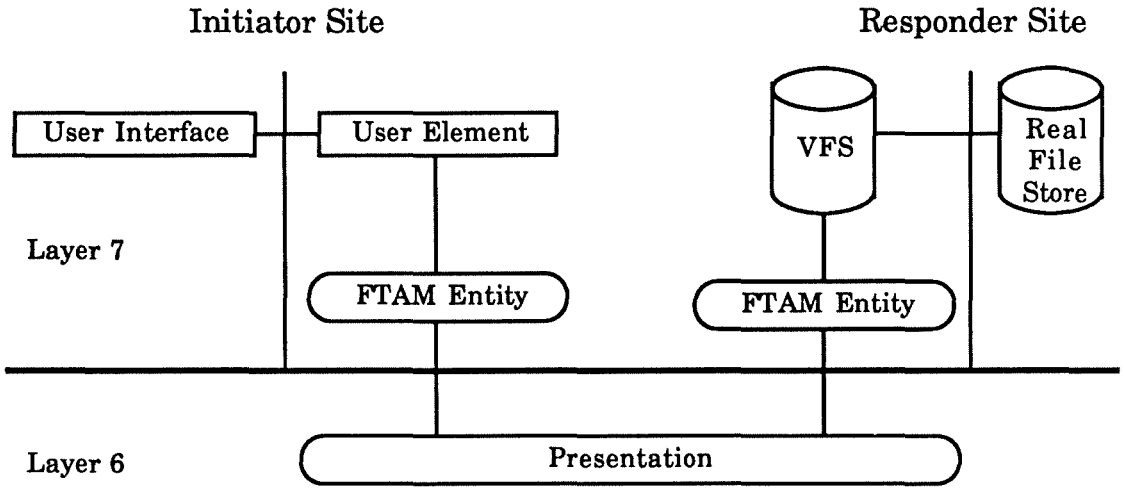


Figure 4.1 : FTAM functional model.

The initiator's activity may consist of update operations on the remote filestore, or of transfer of files from the local site to the remote filestore, or from the remote filestore to the local site.

Directory needs

FTAM requires that both Application-Entity-Titles (i.e.: filestore names) and PSAP-addresses of the called and calling FTAM entities must be provided. This information is actually passed to FTAM by the initiator when establishing the file service activity. This means that the range of this directory function must be the complete (or desired) range of all possible FTAM responders throughout the FTAM service [FTAM-DS].

This has the rather important implication that the directory service must be global. Filestore Name to PSAP-address binding must be known not only locally, but throughout the FTAM service. Which is different from the MTS, where we don't need to know the PSAP-address of the destination MTA, but only the PSAP-address of the next MTA on the route to the destination.

In a pure FTAM OSI environment, a function mapping the PSAP-address of an FTAM entity into its corresponding title is not required, as both are carried in the protocol. However, in other pre-OSI File Transfer Service, as the Janet FTS (Blue Book), it could be that there is no provision for carrying the titles of the participating entities in the protocol.

So when pure ISO FTAM service and pre-OSI file transfer protocols will have to interact, a mapping of PSAP-addresses into entity titles is required by the protocol convertor to resolve the called and calling addresses contained within the pre-OSI protocol into FTAM filestore names.

Another function related to directory needs of FTAM would be to provide a mechanism to determine the filestore name where a specific file or information is stored.

4.2.2. MTS needs

Very similar to AETs are MTA names. When an association is established between two MTAs, the parameters include a MTA name and an optional password. Each party is supposed to fill in its own name, or more precisely, the name by which the local MTA is known from the remote MTA, and to check the received information, e.g. by verifying the password or the calling network address against the MTA name [Huit-88].

This last function looks like an 'address to name' mapping function. But MTA names could very well be placed in a directory associated with the corresponding MTA address, and the address verification could then be done by using the 'name to address' mapping function as both are present in the connection information. So using the directory in the same way to perform verification and usual name resolution.

Moreover, additional information could be needed by MTAs in order to set up a connection with a remote MTA, or to decide routing. For example, it could be useful for the local MTA to know the protocol versions and the services supported by the remote MTA. If the used protocols appear to be incompatible, or if the connection would lead to a loss of functionality, another route could be selected.

A message could need some kind of function to be performed onto it before being delivered (e.g.: Encoded Information conversion). And it could be that such functions are only offered by a reduced set of MTAs. The originator MTA would then first route the message toward an appropriate special MTA before the message be routed to the destination MTA. Appropriate information about MTA capabilities to make routing decisions easier could be stored in the directory, associated with the MTA name to MTA connection information tables.

4.2.3. Future Needs

Every application has its specific directory needs. In the incoming standards, as Job Transfer Access and Management (JTAM) or Virtual Terminal, new needs will probably appear. It would be desirable that such directory needs can easily be provided by the emerging directory service standard. The constraint for such a requirement is that Application Entity Titles are Directory Names, in order to optimize directory access performance. If it is felt that Directory Names are not convenient, synonyms specific for the application may be defined, and even managed by the directory.

Network management is an important task for which use of directory services would be an helpful tool. Storage and retrieval of long term configuration information maybe provided by a directory service, because it provides simple query facilities and mechanisms for distribution of data as well as an authentification framework [COS-88].

4.3. General and generic Directory Service functions

A global directory service could be designed to satisfy larger and more general needs than the ones required for telecommunications. It could be useful that information about people, organizations, projects and location of facilities and documents be stored and accessed in a general directory.

Generic directory services are services that a global directory should be able to provide, and which could be useful for a large set of applications.

Four main generic functions are identified:

- The White Page Service: the basic function of a directory service is to map the Directory Name of an object into another object identifier. It is the case when an O/R Name is mapped into an O/R Address, or when an AET is mapped into its PSAP-address.

- The Information Service: for a more general use of the directory services, the white page service should extend the retrieved information to all or a part of the information stored about an object in the directory entry corresponding to this object. E.g.: the ability to get all the information about a project.

- The Inverse White Page Service: this directory function maps an object identifier different from the global Directory Name into its global Directory Name. It the inverse of the White Page service. E.g.: mapping a PSAP-address into the AET of the corresponding entity.

- The Yellow Page Service: this service maps a possible incomplete description of a directory entry into the list of entries matching the description. E.g.: all 'workstations' in the 'computer center building'.

Part 2 The X.500 Directory Service

The Directory is a distributed service being jointly defined by ISO and CCITT. ISO standardization documents are IS-9594 1 to 8. Corresponding CCITT Directory papers are the X.500 series of recommendations.

The three next chapters aim to introduce models and concepts of the directory, the abstract service provided by the Directory, and how Directory distribution and the Directory distributed service are achieved, as described in the standardization documents.

The Directory Service (DS) provides for accessing, browsing and maintaining a database distributed over an OSI network. The directory standardization has been produced to facilitate interconnection of information processing systems to provide directory services. It aims to allow, with a minimum of technical agreements, the interconnection of directory systems from different manufacturers, under different managements, of different level of complexity and of different ages [X.500-88].

The set of all such systems, together with the directory information which they hold, can be viewed as an integrated whole, called the Directory. The information held by the Directory is collectively known as the Directory Information Base. This information is typically used to facilitate communication between, with or about objects such as application entities, people, terminals or distribution lists.

The Directory provides the directory capabilities required by OSI applications, OSI management processes, other OSI layer entities and telecommunication services. Among the capabilities which it provides are those of 'user-friendly naming', whereby objects can be referred to by names which are suitable for citing by human users, and of 'name-to-address mapping' which allows the binding between objects and their location to be dynamic. The latter capability allows, for example, network operations not to be affected by addition or removal of objects, or changes in the object location [X.500-88].

In essence, the Directory Service is a database to which a user is given remote access using OSI. The Directory is not intended to be a general purpose database. It has certain special characteristics which have led to it not following well known systems of database structure, although it may partly be built on such systems (e.g.: the relational model) [Kille-88].

The Directory is intended to be large and distributed. This is a fundamental criterion. The other characteristics can be viewed as restrictions which facilitate this goal, but are still acceptable to users of the Directory.

It is assumed, for example, that as is typical with communication directories, there is a considerably higher frequency of queries than of updates. There is also no need for instantaneous commitment of updates. Temporary inconsistencies where both old and new versions of the same information are available, are quite acceptable.

It is a characteristic of the Directory that, except as a consequence of differing access rights to the information or unpropagated updates, the results of Directory queries will not be dependent on the identity or location of the inquirer. This characteristic renders the Directory unsuitable for some telecommunication application, for example some types of routing [X.500-88].

5 Models and Concepts

This chapter takes up the main Directory concepts presented in [X.500-88] and [X.501-88].

5.1. Functional Model

The Directory Service may be seen, from a user's point of view, as an object holding a database, called the Directory Information Base (DIB), and a service providing access to this database. The users of the Directory (e.g. a person or an application process) can read or modify the information held in the Directory, subject to having permission to do so.

In interacting with the Directory, each user is represented by a Directory User Agent (DUA), which is the application process actually accessing the Directory on behalf of the user. The Directory provides one or more Directory access points at which such accesses can take place. This is illustrated in figure 5.1.

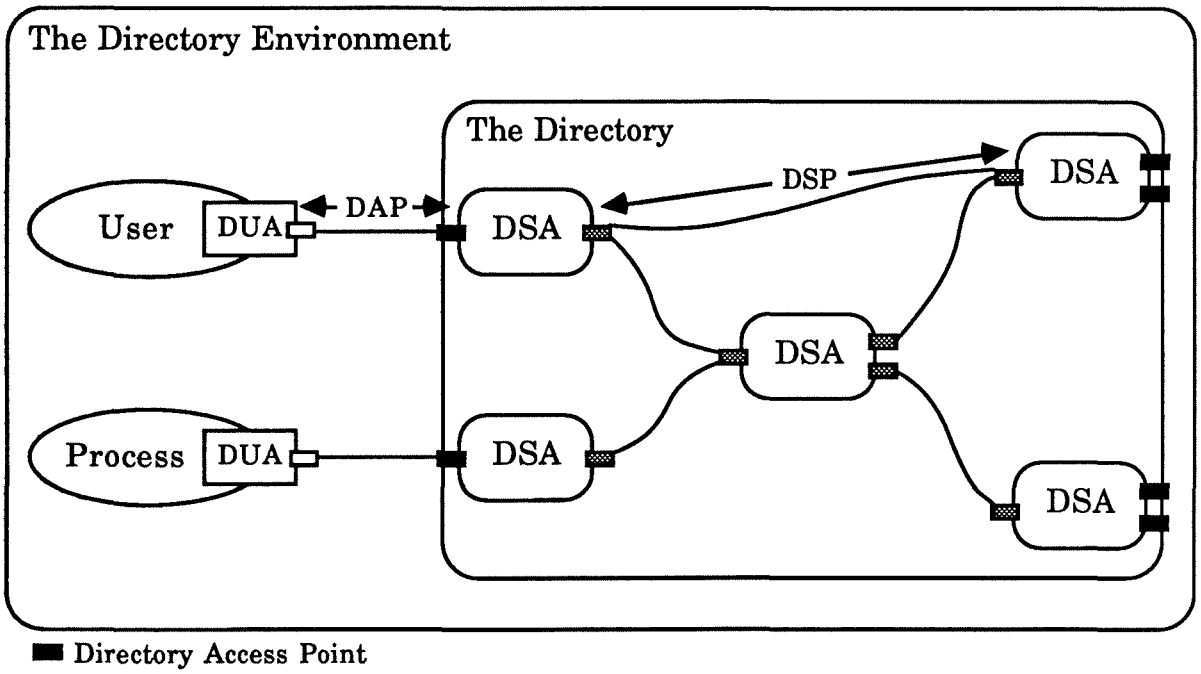


Figure 5.1 : The Directory functional model.

The Directory provides a well defined set of access capabilities to its users, known as the abstract service of the Directory. This service provides simple modification and retrieval capabilities (see chapter 6).

The Directory is manifested as a set of one or more application processes known as Directory Service Agents (DSAs), each of which provides zero, one or more access points to the Directory. This is illustrated in figure 5.1. If the Directory is composed of more than one DSA, it is said to be distributed.

Each DSA holds a part of the DIB. DSAs allow thus the physical distribution of the DIB. DSAs cooperate to achieve the Directory Service provided to users, as the information must transparently be accessed regardless of its location in the DIB.

A particular pair of application processes which need to interact in the provision of the Directory Service (either a DUA and a DSA, or two DSAs) may be located in different systems. Such interactions are carried out by means of the Directory protocols.

If a DUA and a DSA are located in different systems, the Directory Access Protocol (DAP) is used to transmit the request from the DUA to the DSA and to return the results from the DSA to the DUA.

Each DSA provides direct access to the information held by it, and, by means of communications with other DSAs via the Directory System Protocol (DSP), to the complete information base.

A set of one or more DSAs and zero or more DUAs managed by a single organization may form a Directory Management Domain (DMD). The organization concerned specifies the communications among the functional components within the DMD and certain aspects of the behavior of their DSAs. It may in fact not make use of the X.500 DS to govern any interactions among DUAs and DSAs which are wholly within the DMD, but define its own one.

Each DSA is administrated by an administrative authority. This entity has control over all the part of the DIB held by it, and in particular on its structure and for allocation of Directory Names (see sections 5.7 & 5.6).

5.2. The Directory Information Base

The Directory Information Base is made up of information about 'objects of interest' (objects) which exists in some world, primarily of interest in the world of telecommunications. An object can be anything in that world which is identifiable, i.e. which can be named.

Every object belongs to an object class. An object class specifies the characteristics which are shared by a family of objects.

> E.g., every application entity belongs to the Application-Entity object class. Every object belonging to the Application-Entity object class has a PSAP-Address and an Application-Entity-Name.

An object class may be a subclass of one or more other classes, in which case the latter are called the superclasses of the former. A subclass always automatically inherits all of the characteristics of its superclasses (these characteristics must not be redefined for the subclass). A subclass may be defined to add characteristics to an object class. The members of the subclass are also considered to be members of their superclasses. There may be subclasses of subclasses, etc, to an arbitrary depth.

> E.g., the MTA object class is a subclass of the Application-Entity object class and automatically inherits of all of its characteristics, i.e. has a PSAP-Address and an Application-Entity-Title. Moreover, the MTA object class has an additional characteristic which is the Supported-Protocol-Version. The Application-Entity object class is a superclass of the MTA object class. An MTA is an Application-Entity, and is thus also a member of the Application-Entity object class in addition to be member of the MTA object class..

The DIB is composed of Directory entries (entries), each of which consists of a collection of information about a single object.

For any particular object, there is precisely one object entry containing the information about that object in the DIB. The object entry is said to represent the object.

For any particular object, there may, in addition to the object entry, be one or more alias entries for that object entry. Only object entries may have alias entries, so alias entries for alias entries is not permitted. Alias entries point in some way to their corresponding object entry. Alias entries are used to provide alternate names to objects (see section 5.6).

Each entry contains an indication of the object class of the object to which the entry is associated. The object class indication in an alias entry indicates in fact that it is an alias entry. This is shown in figure 5.2.

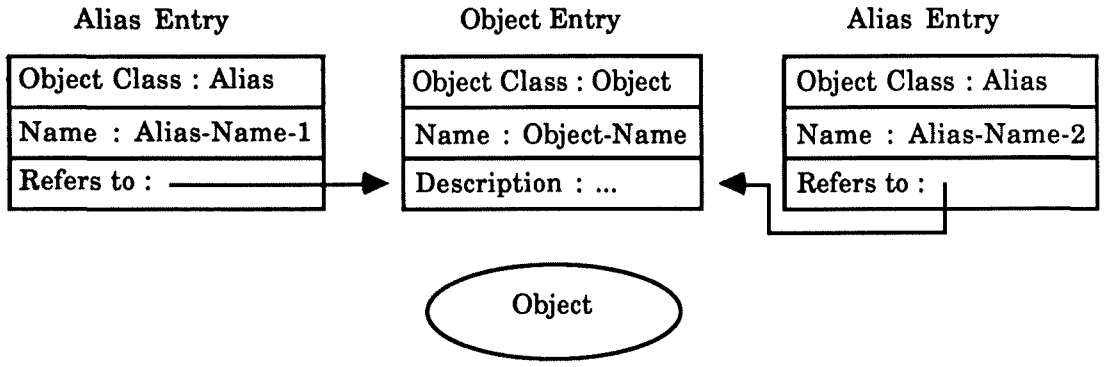


Figure 5.2 : Objects, object entries and alias entries.

5.3. The Directory Information Tree

The data in the Directory are structured in a hierarchical manner. The entries of the DIB are arranged in the form of a tree, the Directory Information Tree (DIT), where vertices represent the entries. Entries higher in the tree (nearer to the root) will often represent objects such as countries or organizations, while entries lower in the tree will represent people or application processes. This is illustrated in figure 5.3.

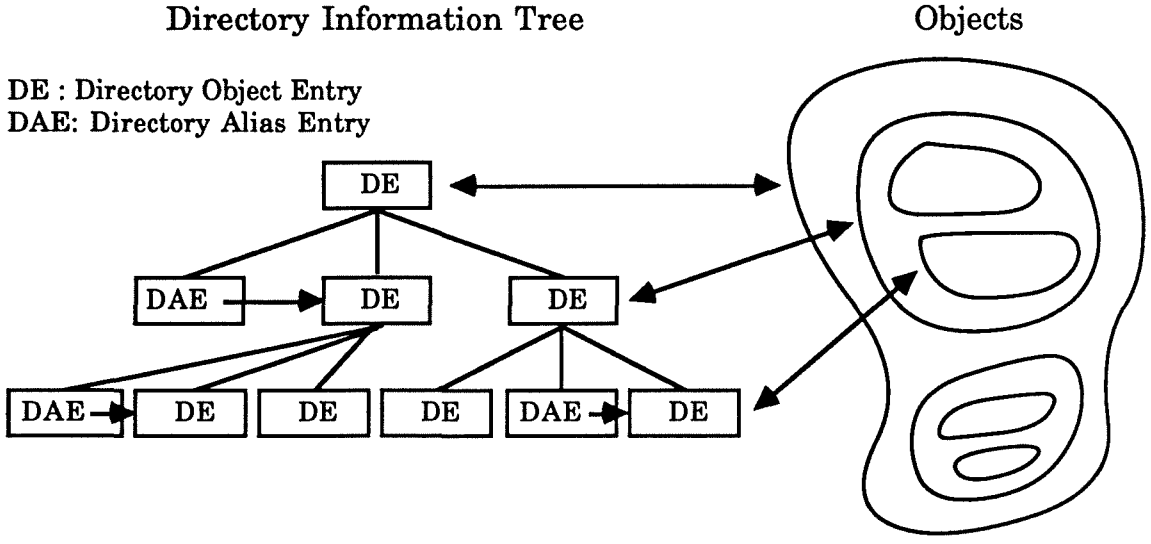


Figure 5.3 : The Directory Information Tree.

Such an organization is designed to facilitate distribution and management of a potentially very large DIB (see chapter 7), and to ensure that objects can be unambiguously named (see section 5.6) and their entries found.

The hierarchical relationship found among objects in the tree can be exploited by the arrangement of the entries in the tree.

> E.g., a person works for a department, which belongs to an organization, which is headquartered in a country.

Vertices in the tree are entries. Object entries may be either leaf or non-leaf vertices, whereas alias entries are always leaf entries. Note that an alias entry can refer to a non-leaf object entry.

The object represented by an entry is closely related with the naming authority for its subordinates (see section 5.6). The root can be viewed as a null object entry, representing the highest level of naming authority for the DIB.

Permitted superior/subordinate relationships among objects, and hence entries, are governed by the DIT structure definitions (see section 5.7.1).

5.4. Directory Entries and Attributes

An entry consists of a set of attributes. Each attribute provides a piece of information about, or describes a particular characteristic of, the object to which the entry corresponds. "Attribute" is the Directory lingo for field in a Directory entry.

Each attribute consists of a type, which indicates the class of information given by the attribute (i.e. its semantic), and the corresponding attribute values. An attribute type may have several values.

> E.g.: Attribute type : Internal-Phone-Number
> Attribute values : 3391, 2178.

This is shown in figure 5.4.

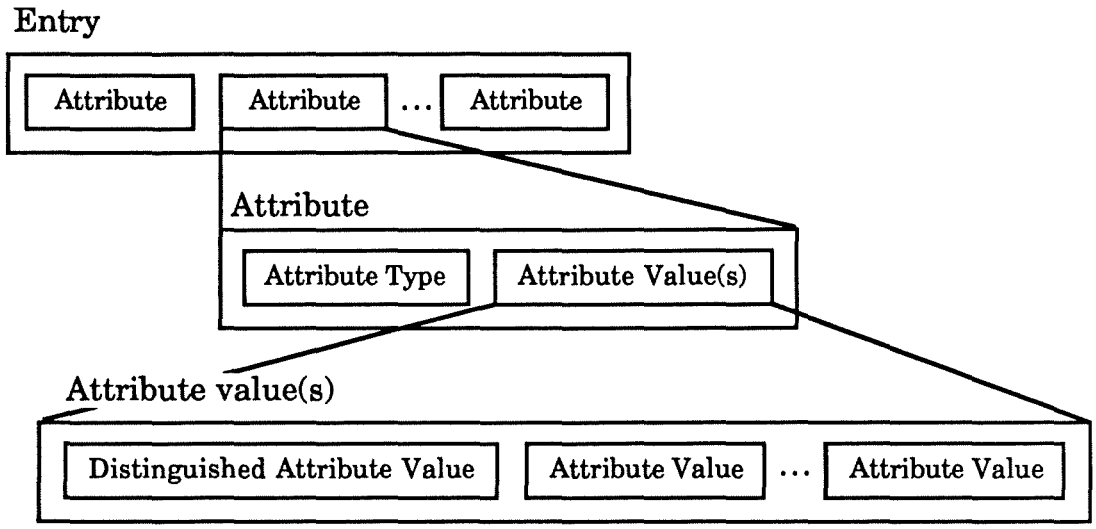


Figure 5.4 : Structure of an entry.

All attributes in an entry must be of distinct attribute types.

There are a number of attribute types which the Directory system knows about and uses for its own purposes. These include:

- the 'Object-Class' attribute, which appears in every entry and indicates the object class to which the object corresponding to the entry belongs;
- the 'Aliased-Object-Name' attribute. An attribute of this type appears in every alias entry, and holds the distinguished name (see section 5.6) of the object entry this alias entry is an alias of.

The types of attributes which must or may appear in an entry are governed by rules applying to the indicated object class: the object class definition (see section 5.7.1). The object class definition defines, among other things, the mandatory and optional attribute types which appear in an entry.

> E.g., entries corresponding to objects of the MTA object class must contain the 'NSAP-Address', 'Application-Entity-Title' and 'Supported-Protocol-Version' attribute types, and may contain the 'Contact-Person-Phone-Number' attribute type.

Defining an attribute type also involves specifying the syntax for the values of this attribute (attribute syntax), and hence the data type to which every value in such an attribute must conform (see sections 5.7.3 & 5.7.4).

> E.g., the attribute type 'Contact-Person-Phone-Number' has to be of attribute syntax 'Numeric-String'.

At most one of the values of an attribute may be designated as a distinguished value, in which case the value appears in the Relative Distinguished Name of the entry (see section 5.6).

An Attribute Value Assertion (AVA) is a proposition, which may be true, false or undefined, concerning the attribute values of an entry. It involves an attribute type and an attribute value. A matching rule specifies when the presented value of the AVA satisfies the AVA (see section 5.7.4). When no matching rule is specified, the matching rule for equality is defaulted.

> E.g.: the AVA <Application-Entity-Title = MyMTA> is true for a particular entry if the value of the attribute type 'Application-Entity-Title' in that entry matches the value 'MyMTA' for equality.

An AVA may have an undefined value when, for example, the attribute syntax of this attribute type has no equality matching rule.

5.6. Naming

Managing entries of a worldwide Directory must solve the problem of managing a worldwide namespace in a flexible and user-friendly way [COS-88]. The name structure follows the hierarchical structure of the Directory Information Tree. This approach solves the problem of assigning unique names to entries without the need of a central authority.

A Directory Name (name) is a construct that identifies a particular object among the set of all objects. However, the name needs not to be the only one identifying the object, i.e. several different names may unambiguously denote the same object.

Each entry of the Directory has a Relative Distinguished Name (RDN). A RDN consists of a set of Attribute Value Assertions (possibly made up of only one AVA), each of which is true, concerning the distinguished attribute values of the entry. The set contains exactly one assertion about each distinguished value in the entry.

The RDN for an entry is chosen when the entry is created. A RDN must identify a subordinate entry among all subordinate entries of its superior entry. This thus involves that the RDNs of the entries subordinated to a particular entry are all distinct. It is the responsibility of the relevant naming authority for the superior entry to ensure that this is so by appropriately assigning relative distinguished values.

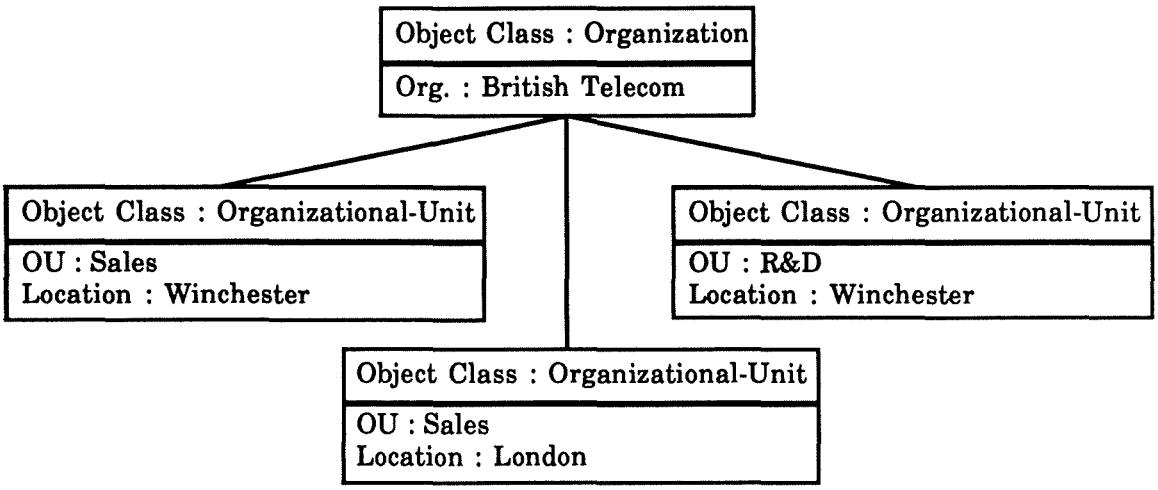


Figure 5.5 : Entries and Relative Distinguised Names.

E.g., in the example of figure 5.5, the RDN <OU = Sales ; Location = London> identifies a particular entry representing an organizational unit among all subordinate entries of the entry corresponding to the 'British Telecom' object. The value 'Sales' of the attribute type 'OU' is a distinguished value for the entry.

The RDN of the root is defined to be null.

The Directory Names of an object are the distinguished name and the alias names of the object entry corresponding to this object.

The distinguished name of a given entry (object or alias entry) is defined as being the sequence of the RDNs of the entry which represents the object and those of all its superior entries, in descending order. Figure 5.6 illustrates this principle.

DIT	RDN	Distinguished Name
ROOT	{ }	{ }
Countries	C=UK	{ C=UK }
Organizations	O=British Telecom	{ C=UK; O=British Telecom }
Org. Units	OU=Sales; L=London	{ C=UK; O=British Telecom; { OU=Sales; L=London } }
People	CN=Smith	{ C=UK; O=British Telecom; { OU=Sales; L=London }; CN=Smith }

Figure 5.6 : Determination of Distinguished Names.

The aliasing mechanism permits, in some way, entries to have multiple immediate superiors among which only one is an object entry, other ones being alias entries. Moreover, the distinguished name of an alias entry may be considered as an alias name for the distinguished name of the object entry this alias entry refers to. Therefore, alias entries provide a basis for alternate names for an object. An alias, or an alias name for an object is a name where at least one of the RDNs is that of an alias entry.

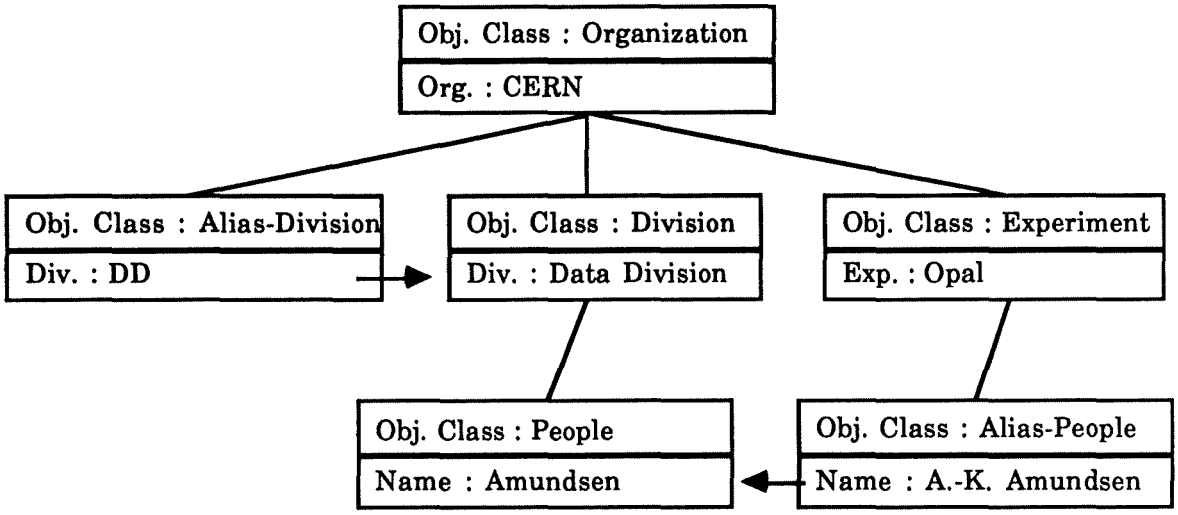


Figure 5.7 : Alias entries and alias names.

E.g., in the example shown in figure 5.7, <Org. = CERN ; Div. = Data Division ; Name = Amundsen> is the Directory distinguished name of the object (in fact a person) represented to by this entry. Alias Directory names for this person are <Org. = CERN ; Div. = DD ; Name = Amundsen> and <Org. = CERN ; Exp. = Opal ; Name = A.-K. Amundsen>. All are Directory names.

A purported name is a construct which is syntactically a name, but which has not (yet) been shown to be a valid name.

5.7. Directory Schema

The Directory schema is a set of definitions and constraints concerning the structure of the DIT, the possible ways entries are named, the information that must or may be held in an entry, and the attribute types and attribute syntaxes used to represent that information.

Defining the Directory schema is equivalent of defining the structure of a database and its related constraints to a DataBase Management System. The schema enables the Directory system to check the validity of the operations performed in the DIT and on the entries it holds. For example, the Directory system will prevent the creation of an entity if a value for a mandatory attribute is missing in the 'Add-Entry' operation, or will prevent the addition of a character-string value to an attribute if its attribute syntax is defined to be a numeric string value.

There are four kinds of definitions in the Directory schema:
- the DIT structure definitions;
- the object class definitions;
- the attribute type definitions;
- the attribute syntax definitions.

5.7.1. DIT Structure Definitions

The DIT structure definitions define the ways in which entries may be related to one another in the DIT, i.e. the permitted hierarchical relationships between entries . By the way, they also define the structure distinguished names of entries may have. The X.500 standard recommends a basic DIT structure described in figure 5.8. The table gives for each object its Relative Distinguished Name and its possible superiors.

Object Class	RDN	Root	C.	Loc.	Org.	O.U.	App. Proc.
Country	CountryName	X					
Locality	LocalityName	X	X	X	X	X	
Organization	OrganizationName	X	X	X			
Organizational Unit	OrganizationalUnitName			X	X	X	
Organizational Person	CommonName				X	X	
Organizational Role	CommonName				X	X	
Group of Names	CommonName			X	X		
Residential Person	CommonName			X			
Application Entity	CommonName						X
Device	CommonName				X	X	
Application Process	CommonName				X	X	

Figure 5.8 : Recommended DIT Structure Definition.

An 'Organizational Person' is a person employed by an organization, or strongly associated with that organization. It is distinct from the 'Residential Person'. An 'Application Entity' consists of those aspects of an application process pertinent to OSI, while an 'Application Process' is an element within a real system which performs a particular application. A 'Group of Names' may represents, for example, a Distribution List.

Note that the definition is recursive, as for example, a 'Locality' entry can be subordinated to another 'Locality' entry.

When one attempts to modify the DIT (e.g. when adding an entry somewhere in the DIT), the Directory system checks whether the operation violates the applicable Directory structure. If it is the case, the operation fails.

> E.g., in the example of the DIT structure defined in figure 5.8, an attempt to add an entry of the object class 'Residential Person' as subordinate to an entry of the object class 'Country' will fail. In fact, only entries of the object class 'Locality' may be superior entries for entries of the object class 'Residential Person'.

The DIT structure definitions also define the permitted RDNs for object class entries, and hence the possible Directory Name structures for the corresponding objects. The possible Directory name structures for an entry of a particular object class may be derived from the DIT structure (see section 5.6). They in fact depend of the possible object classes of their superior entries and their RDNs. Only one structure is valid for a given entry. The valid structure depends of the position of the entry in the DIT.

E.g., in the example of the DIT structure definitions of figure 5.8, the structure of a distinguished name for an entry of the object class 'Organization' may be either <CountryName = ... ; LocalityName = ... ; OrganizationName = ... >, <CountryName = ... ; OrganizationName = ... >, <OrganizationName = ... >, <LocalityName = ... ; OrganizationName = ... >, etc ...

5.7.2. Object Class Definitions

The definition of an object class comprises:
- optionally, an identifier for the object class;
- an indication of which class or classes this object is a subclass of;
- the list of the mandatory attribute types that an entry of this object class must contain (in addition to the mandatory attribute types of all its superclasses);
- the list of the optional attribute types that an entry of this object class may contain (in addition to the optional attribute types of all its superclasses).

Example of class definition:

```
Define Object Class 'Organizational-Person'
        - Subclass-of       : 'Person' ;
        - Object-Class-Id   : 103 ;
        - Must contain      : Organizational-Role ;
                              Internal-Phone-Number ;
        - May contain       : Fax-Number ;
End of 'Organizational-Person' ;
```

Note that an object class automatically inherits of all of the attribute types of its superclasses. These attribute types are fully part of the object class definition of its subclasses.

Every object class is implicitly a subclass of the special object class called 'Top'.

Every entry contains an attribute type 'Object-Class' (inherited from the 'Top' object class) which identifies the object class to which the object corresponding to the entry belongs.

When one attempts to modify an entry of the DIT (e.g. when adding an attribute to an entry), the Directory system checks whether the operation violates the object class definition. If it is the case, the operation fails.

E.g., attempting to delete the mandatory attribute type 'Internal-Phone-Number' to an entry of the above 'Organizational-Person' defined object class will fail. The same happens if one attempts to add to an entry an attribute type which is absent from the object class definition of that entry.

A special object class 'Alias' is defined. Every alias entry shall have an object class which is a subclass of the Alias object class.

5.7.3. Attribute Type Definitions

The definition of an attribute type involves:
- assigning an identifier for the attribute type;
- indicating or defining the attribute syntax for the attribute type;
- indicating whether an attribute of this type may have only one value, or may have more than one value.

Example of attribute type definition:

```
Define Attribute Type 'Internal-Phone-Number'
        - Att.-Type-Id    : 1204 ;
        - Att.-Syntax     : Numeric-String ;
        - Multivalued     : Yes ;
End of 'Internal-Phone-Number' ;
```

The Directory system ensures that the indicated attribute syntax is used for every value of attribute of this type. The Directory also ensures that attributes of this type have one and only one value in entries if attributes of this type are defined to have only one value.

The attribute type definition may also define constraints on the attribute values, as a range of permitted values or a maximum length for an alphabetic string.

5.7.4. Attribute Syntax Definitions

The definition of an attribute syntax involves:
- optionally assigning an identifier to the attribute syntax;
- indicating the data type of the attribute syntax;
- defining rules for matching a presented value with a target attribute value held in the DIB.

Matching rules may be defined for equality, ordering or substring. A matching rule for ordering specifies when a presented value is less than, equal to or greater than a target value.

Example of attribute syntax definition:

```
Define Attribute Syntax 'Ethernet-Address-Syntax'
        - Att.-Syntax-Id : 10203 ;
        - Data-Type      : BitString ;
        - Matches-for    : Equality
                ( Presented equals target when Presented = Target) ;
End of 'Ethernet-Address-Syntax'.
```

The attribute syntax also defines how values of the data type have to be encoded.

The syntax of distinguished values should always specify a matching rule for equality.

A presented distinguished name value is equal to a target distinguished name value if and only if:
- the number of RDNs in each is the same;
- corresponding RDNs have the same numbers of AVAs;
- corresponding AVAs (i.e. those with identical attribute types) have attribute values which match for equality.

Figure 5.9 summarizes the relationships between the schema definitions on one side, and the DIT, directory entries, attribute types and attribute values on the other side.

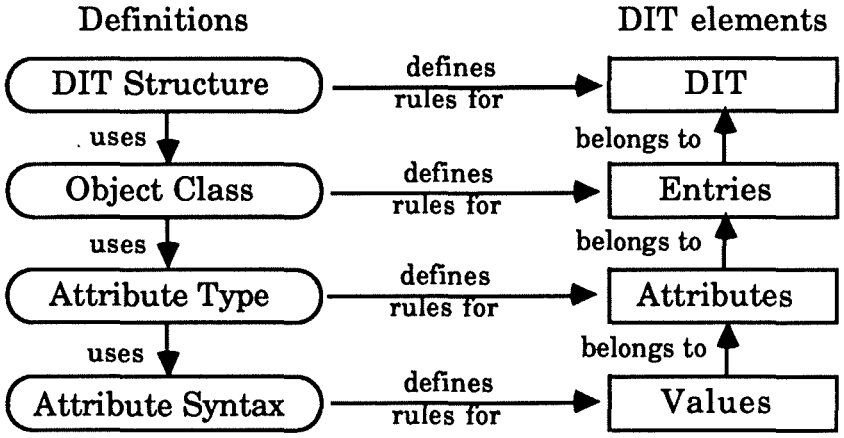


Figure 5.9 : Overview of Directory Schema.

The X.500 International Standard defines a rather extensive set of object classes [X.521-88], attribute types and attribute syntaxes [X.520-88], covering the general needs of current telecommunication operations. Specific needs of applications or of user communities within a national or private Directory system may require extensions to the standardized definitions which may be met by defining appropriate new object classes, attribute types and attribute syntaxes.

5.8. Access Control

Directory users are granted access to the information on the basis of their access control rights in accordance with the access control policy in force protecting that information. X.500 has no provision for managing access control to the information held in the DIB. Although it is recognized that implementations will need to introduce means of controlling access to the information, access control is left as a local matter by the standard. However, Annex F of [X.501-88] (which is not part of the standard), recommends some principles that are to guide the establishment of procedures for managing access control.

Four levels of protection are presently identified:
- protection of an entire subtree of the DIT;
- protection of an individual entry;
- protection of an entire attribute within an entry;
- protection of selected instances of attribute values.

A need for at least six categories of access is envisaged. If access is not granted to a protected item in any category, then the Directory, in so far as possible, responds as if the protected item do not exist at all.

The following categories of access are presently identified (the letters denote that the item which can be so protected is an attribute (A), an entry (E), or both (EA)):

- detect (A) : allows the protected item to be detected;
- compare (A) : allows a presented value to be compared to the protected item;
- read (A) : allows the protected item to be read;
- modify (A) : allows the protected item to be modified;
- add/delete (EA) : allows the creation and deletion of new components (attributes or attribute values) within the protected item;
- naming (E) : allows the modification of the RDN of, and creation and deletion of, entries which are immediately subordinate to the protected entry.

A scheme for managing access control associates with every protected item, either explicitly or implicitly, a list of access rights. Each item in such a list pairs a set of users with a set of access categories.

Determining if a user is in one (or more) of the noted sets must be possible from the information supplied with the request.

6 Directory Abstract Service Definition

This chapter describes the abstract Directory Service as defined in the current Directory standardization documents.

'Abstract' refers to the fact that functionalities are defined regardless of their implementations. In the standard papers, the abstract Directory Service is formally defined with the Abstract Syntax Notation One (ASN1).

6.1. The Client-Server Model

The Directory System is positioned in the OSI application layer, and is constructed in accordance with the Client-Server model described hereafter [TR/32].

A Server System is a functional entity that performs a set of basic and specific application services (the Server Services), by means of Service Agents within the Server System.

A Client System is a system in which one or more Clients of a Server System reside.

A Client is a functional entity that requests services provided by a Service Agent for that particular service. A Client consumes the services provided by the Server System.

A Client gains access to a Service Agent by means of an Access Protocol.

A Client System may be a User, or a Service Agent of another service (e.g. a MTA may be a Directory User).

The set of services provided by a Service Agent is made available to its Clients through Access Points. Each Access Point corresponds to a particular combination of Supplier Ports. Each Port defines a particular kind of interaction between a Client and a Server, and provides a particular set of services.

Similarly, the Server System views the Client through Consumer Ports to which the requested services have to be provided.

Communications between Consumer Ports and Supplier Ports are made thanks to the Access Protocol.

Figure 6.1 illustrates the Client-Server model.

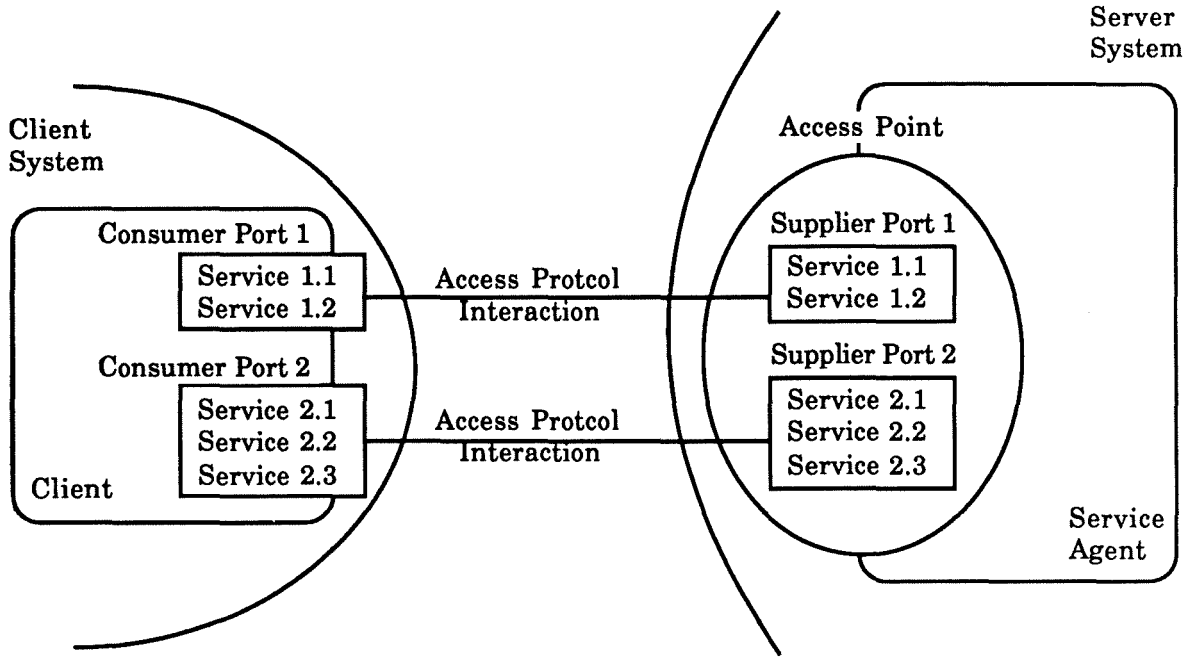


Figure 6.1 : The Client-Server Model.

One or more Service Agents connected to a network may interact to perform the requested services. In that case, they cooperate by means of an inter-service Agent Protocol. Similarly to the Client-Server interaction, the different Service Agents interact through Ports.

The Access Protocol is the standard way for Clients to gain access to a Service Agent. It is the mean that allows location of Clients to be remote from Service Agents.

An Agent Protocol is the standard way for a Service Agent to gain access to another Service Agent of the same specific service. It is the mean that allows a Server System to be distributed between a number of Service Agents.

Figure 6.2 illustrates a distributed Client-Server System.

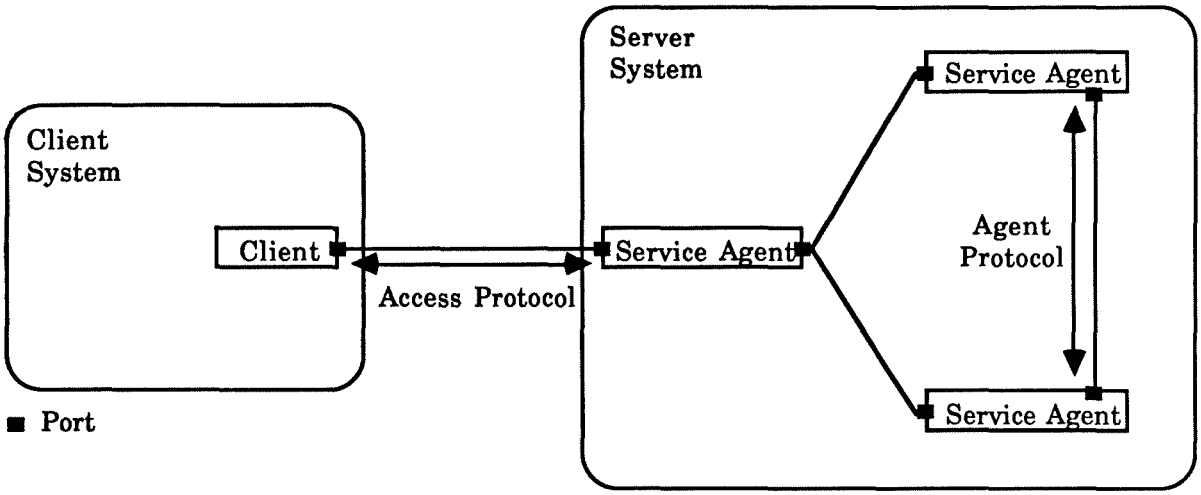


Figure 6.2 : Distributed Server System.

In the Directory Service system, Clients are made of Directory User Agents (DUAs), and Service Agents are made of Directory System Agents (DSAs), which are all supported by application processes. The Access and Agent Protocols are respectively the Directory Access Protocol (DAP) and the Directory System Protocol (DSP), which are application level protocols.

6.2. Layered Model of the Directory System

When a pair of remote Directory application processes have to interact, these interactions are supported by the DAP if these applications are a DUA and a DSA, and by the DSP if both are DSAs.

The functions of the Application-Entities are provided by a set of Application-Service-Elements (ASEs). There is a one to one correspondence between an ASE and a port.

The Remote Operation Service Element (ROSE) supports the request/reply paradigm of the abstract operations that occurs at the ports in the model. The Directory ASEs provide the mapping function of the abstract syntax notation of the Directory abstract service onto the services provided by the ROSE [X.518-88].

The Association Control Service Element (ACSE) supports the establishment and release of an application association between a pair of application-entities. An association between a DUA and a DSA may be established only by the DUA. Only the initiator of an established association can release it [X.518-88].

This is illustrated in figure 6.3.

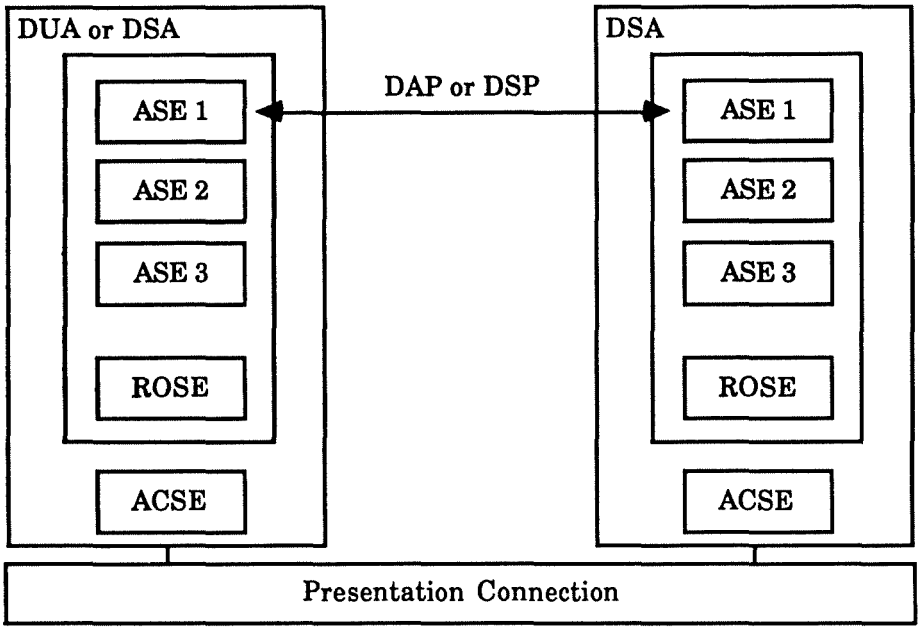


Figure 6.3 : Layered Model of the Directory System

The protocol specifications of the DAP and the DSP are further detailed in [X.519-88]

6.3. Directory ports and services

The Directory may be seen as an object accessed through Access Points. Each Access Point corresponds to a particular combination of ports, each of which providing a particular set of services. The Directory supplies services via three types of Supplier Ports:

- the Read ports, which support reading information from a particular named entry in the DIB;
- the Search ports, which allow more exploration of the DIB;
- the Modify ports, which enable the modification of entries in the DIB.

It is intended that in the future, there may be other types of Directory ports.

Similarly, the DUAs are seen from the viewpoint of the Directory, as composed of a combination of Read, Search and Modify Consumer Ports.

Each type of port gives access to a particular set of services. Each service is offered thanks to the corresponding operation.

The Read ports provide the 'Read', 'Compare' and 'Abandon' operations. The Search ports provide the 'List' and 'Search' operations, and the Modify ports provide the 'Add-Entry', 'Remove-Entry', 'Modify-Entry' and 'Modify-RDN' operations. The services corresponding to these operations are described in the following sections.

In addition to these externally visible ports (visible from the DUA point of view) a DSA supports a distributed version of these ports, namely 'Distributed-Read' port, 'Distributed-Search' port and 'Distributed-Modify' port, which allow DSAs to propagate requests for these services to other DSAs (see chapter 7).

6.4. Common Operation Parameters

This section describes the parameters (arguments and results) which are common to several Directory operations.

The 'common arguments' information may be present to qualify the invocation of each operation that the Directory can perform. When absent, the default values (defined in [X.511-88]) are assumed.

The 'common arguments' comprise five components: the 'service controls', the 'security parameters', the 'requestor distinguished name', the 'operation progress' and the 'aliased RDN'. These components are described in the following sub-sections.

```
Common Arguments
    comprise : - Service controls ;
               - Security parameters ;
               - Requestor distinguished name ;
               - Operation progress ;
               - Aliased RDN ;
```

The 'common results' information should be present to qualify the result of each retrieval operation that the directory can perform.

The 'common results' comprise the 'security parameters', the 'requestor distinguished name' and the 'alias dereferenced' components.

```
Common Results
    comprise : - Security parameters ;
               - Requestor distinguished name ;
               - Aliased dereferenced ;
```

Further arguments specific to one or more operations are also described. These are the 'entry information selection', the 'entry information', the 'filter' and the 'error reports'.

Service Controls

The 'service controls' argument contains the controls, if any, that are to direct or constrain the provision of the service.

These parameters comprise the 'options', 'priority', 'time limit', 'size limit' and 'scope of referral' components.

```
Service Controls
    comprise : - Options ;
               - Priority ;
               - Time limit ;
               - Size limit ;
               - Scope of referral ;
```

The 'options' component contains indications for the behavior of DSAs when performing the requested service. Options indicate if chaining is preferred to referrals (see section 7.3), whether chaining and multicasting are prohibited, if an operation has to be limited to a local scope (a single DSA or a single DMD), whether copy of the original information may be used to provide the service and whether aliases used to identify an entry may or must not be dereferenced while performing an operation.

The 'priority' component indicates the priority (low, medium or high) at which the service is to be provided at the Directory level. There is no relationship implied with the use of priorities in underlying layers of the OSI stack. Note that there is no guaranties that the Directory, as a whole, does implement queuing.

The 'time limit' indicates the maximum elapsed time, in seconds, within which the service should be provided. If this constraint cannot be met, an error is reported. In the case of a time limit exceeded during a List or Search operation (see section 6.5.3), the result shall be an arbitrary selection of the accumulated results.

The 'size limit' is only applicable to a List or Search operation (see section 6.5.3). It indicates the maximum number of objects to be returned. In the case of size limit exceeded, the results of List or Search operation may be an arbitrary selection of the accumulated objects, equal in number to the size limit. Any further results shall be discarded.

The 'scope of referral' indicates the scope to which a referral returned by a DSA should be relevant (see section 7.3). Only referrals to other DSAs within the selected local scope (DMD or country), will be returned. Note that should a DSA be enable to return a referral within the selected scope, it will nevertheless return any reference it holds to other DSAs able to continue the operation.

Certain combination of priority, time limit and size limit may result in a conflict. For example, a short time limit could conflict with low priority, a high size limit could conflict with a low time limit, etc...

Security parameters

The 'security parameters' govern the operation of various security features associated with a Directory operation.

A digital signature allows a recipient to check whether a received information has been produced by the indicated originator. The arguments or results of any operation may or may not be accompanied by a digital signature according to the requested level of protection.

The 'security parameters' are the 'certification path', the 'name', the 'time' with a 'random number' and the 'target protection requested'.

```
Service Parameters
    comprise : - Certification path ;
               - Name ;
               - Time and random number ;
               - Target protection requested ;
```

The 'certification path' component is used to identify the originator's distinguished name, and verify the signature on the arguments or results. A sequence of certificate pairs is required when the recipient of information does not have the same certification authority as the originator [X.509-88].

The 'name' is the distinguished name of the first intended recipient of the arguments or results. This information may be forwarded to other agents when the operation is to be distributed. For example, if a DUA generates a signed argument, the name is the distinguished name of the DSA to which the operation is submitted.

The 'time' is the intended expiry time for the validity of a signature when signed arguments or results are used. It is in used in conjunction with a 'random number' to enable the detection of replay attacks when the arguments or results have been signed.

The 'target protection request' appears only in the request for an operation, and indicates the requestor's preference regarding the degree of protection to be provided on the results. Two levels are provided: none (no protection requested), and signed (the directory is requested to sign the results). The actual degree of protection provided may be equal to or lower than that requested, based on the limitation of the Directory. The degree of protection actually provided to the result is indicated by a protection indication.

Other Common Arguments

The 'requestor distinguished name' identifies the requestor of a particular operation. It holds the name of the user who initiates the request, as identified at the time of binding to the directory. It shall be present only if the results have to be signed.

The 'operation progress' defines the role that the DSA is to play in the distributed evaluation of the request (see Annex B).

The 'aliased RDN' component indicates to the DSA that the object component of the operation was created by the dereferencing of an alias on an earlier operation attempt (e.g. during a previous read attempt in a new read operation performed after having received a referral, see section 7.3).

Other Common Results

The 'alias dereferenced' component is set to 'true' when the name of an entry which is a target or a base for an operation included an alias which was dereferenced.

Entry Information Selection

An 'entry information selection' argument indicates what information is being requested from an entry in a retrieval service. It specifies the set of attributes about which information is requested.

One may specify all attributes or a list of attributes that are to be returned (the list may be empty), and if one wants only to retrieve attribute types only, or attribute types and values. Returning all attribute types and values is defaulted.

An attribute error will be returned if none of the selected attributes are present. A security error will be returned in the case where access rights preclude the reading of all information requested.

Entry Information

An 'entry information' result conveys the selected information from an entry. The distinguished name of the entry is always included. A parameter indicates whether the information was obtained from the original entry or from a copy of the original entry. A set of attribute types are included, each of which may be alone or accompanied by one or more attribute values.

Filter

A 'filter' argument applies a test that is either satisfied or not by a particular entry. The filter is expressed in terms of assertions about the presence or value of certain attributes of the entry, and is satisfied if and only of the filter is evaluated to 'true'.

A filter is either a filter-item or an expression involving simpler filters composed together using the logical operators 'and', 'or' and 'not'.

A filter-item is an assertion about the presence of, or value(s) of, an attribute of a particular type in the entry under test. Each such assertion is either 'true' or 'false'.

A filter-item is either the indication of an attribute type (when one wants to check the presence of an attribute type), or an attribute value assertion plus a matching rule to be applied (when one wants to check the value of an attribute).

Defined matching rules for filter-items are 'equality, 'greater or equal', 'less or equal', 'substring' and 'approximate match'. The 'approximate match' matching rule includes phonetic match or spelling variations, and is determined by a locally defined approximate matching algorithm.

Error reports

The Directory provides for 'error reports'.

The Directory does not continue to perform an operation beyond the point at which it determines that an error is to be reported.

Errors are classified by families. In an error report, the family is specified, plus the particular problem encountered. The different families are shortly described hereafter.

The 'abandon' may be reported for any Directory inquiry operation (i.e. 'Read', 'Search', 'Compare', 'List' operations) if the DUA invokes an abandon operation for the corresponding operation. There is no particular problem to be further reported.

The 'abandon failed' error reports a problem encountered during an attempt to abandon an operation. The particular problem may be, for example, that the Directory has already responded to the specified operation, or that an attempt was made to abandon an operation for which this is prohibited (e.g. for the 'Modify-Entry' operation).

The 'attribute error' reports an attribute related problem. A parameter identifies the entry to which the operation was being performed when the error occurred. Each problem is accompanied by an indication of the attribute type and, if necessary to avoid ambiguity, the attribute value which caused the problem. The particular problem may be, for example, that the named entry lacks one of the attribute specified as an argument of the operation, or that there is an attribute definition violation.

The 'name error' reports a problem related to the name provided as argument to an operation. A parameter contains the name of the lowest entry (object or alias) matched in the DIB. The problem may be that, for example, the supplied name does not match the name of any particular object, or that an alias has been dereferenced which names no object.

The 'referral' is not really an error. It redirects the service-user to one or more Access Points better equipped to carry out the requested operation. The error has a single parameter which contains the references of the Access Points which can be used to progress the operation.

A 'security error' reports a problem in carrying out an operation for security reasons, e.g. if the requestor does not have the right to perform the requested operation, or if there is an authentification problem (e.g. and invalid password), or if an invalid signature for the request was found.

A 'service error' reports a problem related to the provision of the service. This may occur when the Directory is unwilling to perform or proceed an operation, for example when a part of the Directory is unavailable or busy, or if a time limit was exceeded.

An 'update error' reports a problem related to attempts to add, delete or modify an information in the DIB. Such problems occur when an operation would violate the integrity rules of the Directory, i.e. the Directory schema (e.g. removal of an attribute type which is part of the RDN).

6.5. The Abstract Service Operations.

6.5.1. Directory BIND and UNBIND operations.

The Directory 'Bind' and 'Unbind' operations are used by a DUA at the beginning and at the end of a particular period of accessing the Directory.

Bind operation.

A Directory-Bind operation is used at the beginning of a period of access to the Directory.

```
Directory Bind-Operation
    Arg :   - port ;
            - versions ;
            - credentials ;
    Res :   - version ;
            - credentials ;
            - error report ;
```

The Bind operation parameters specify as arguments:
- the 'port' to which the service will be requested;
- the 'versions' of the service which the DUA is prepared to participate in;
- the 'credentials' which allow the directory to establish the identity of the user. They may be either simple (consisting of a name and optionally a password), or strong (additional information has to be provided to allow secure authentification of the identity of the originator).

Should the bind request succeed, the following results will be returned:
- the 'version' which indicates which of the versions of the service requested by the DUA is actually going to be provided by the DSA;
- the 'credentials' which allow the user to establish the identity of the Directory, i.e. of the DSA that will directly provide the Directory Service. They are of the same form as those supplied by the user (i.e. simple or strong);
- a 'security error' or 'service error' report if there is an authentification problem, or if the Directory is unavailable.

Unbind operation.

The Directory-Unbind operation is used at the end of a period of access to the Directory and specifies the 'port' to which the unbind is requested.

```
Directory Unbind-Operation
    Arg :   - port ;
```

6.5.2. Directory READ operations.

There are two 'read-like' operations: Read and Compare. The Abandon operation is grouped with the read operations for convenience.

Read operation.

A Directory-Read operation is used to extract information from an explicitly identified entry. It may also be used to verify a distinguished name. The arguments of the operation may optionally be signed by the requestor. If so requested the Directory may sign the result.

```
Directory Read-Operation
    Arg :   - object name ;
            - entry information selection ;
            - common arguments ;
    Res :   - entry information ;
            - common results ;
            - error report ;
```

Arguments are:
- the 'object name' which identifies the object entry from which information is requested. Should the name involve one or more aliases, they are dereferenced (unless this is prohibited by the relevant service controls);
- the 'entry information selection' which indicates what information from the entry is requested;
- the 'common arguments', which specify the service controls applying to the request.

Results are:
- the 'entry information' result which holds the requested information;
- the 'common results' qualifying the results of the operation;
- an 'error report' if the operation fails.

Compare operation.

A Directory-Compare operation is used to compare a value (which is supplied as an argument of the request) with the value(s) of a particular attribute type in a particular object entry. The arguments of the operation may optionally be signed by the requestor. If so requested the Directory may sign the result.

```
Directory Compare-Operation
    Arg :   - object name ;
            - attribute value assertion ;
            - common arguments ;
    Res :   - distinguished name ;
            - matched ;
            - from entry ;
            - common results ;
            - error report ;
```

Arguments are:

- the 'object name' which identifies the object entry concerned. Should the name involve one or more aliases, they are dereferenced (unless this is prohibited by the relevant service controls);
- the attribute type and value to be compared with in that entry (an 'Attribute Value Assertion');
- the 'common arguments', which specify the service controls applying to the request.

Results are:

- the 'distinguished name' of the object itself (present only if an alias was dereferenced);
- the 'matched' result parameter, which holds the result of the comparison. The parameter takes the value 'true' if the values were compared and matched, and 'false' if they did not;
- an indication ('from entry') of whether the information was compared against the original entry information, or against a copy of the original entry information;
- the 'common results' qualifying the results of the operation;
- an 'error report' if the operation fails.

Abandon operation.

Operations that interrogate the Directory may be abandoned using the Directory-Abandon operation if the user is no longer interested in the result. Abandon is only applicable to interrogation operations (i.e. Read, Compare, List and Search).

```
Directory Abandon-Operation
    Arg :   - operation ;
    Res :   - error report ;
```

There is a single argument which identifies the 'operation' that is to be abandoned. The way the operation is identified is not clearly defined.

Should the request succeed, the original operation will fail with an 'abandon' error, and the returned results will not convey any information.

Should the operation fail, the 'abandon failed' error will be reported.

6.5.3. Directory SEARCH operations.

There are two 'search-like' operations: List and Search.

List operation.

A Directory-List operation is used to obtain a list of the immediate subordinates of an explicitly identified entry. Under some circumstances, the list may be incomplete (e.g. size limit exceeded). The arguments of the operation may optionally be signed by the requestor. If so requested the Directory may sign the result.

```
Directory List-Operation
    Arg :   - object name ;
            - common arguments ;
    Res :   - distinguished name ;
            - set of subordinates ;
            - limit problem ;
            - unexplored ;
            - common results ;
            - error report ;
```

Arguments are:

- the 'object name' which identifies the object entry whose immediate subordinates are to be listed. Should the name involve one or more aliases, they are dereferenced (unless this is prohibited by the relevant service controls);
- the 'common arguments' specify the service controls and security features applying to the request.

The request succeeds if the object entry corresponding to the specified named object is located, regardless of whether the entry has immediate subordinates or not.

Results are:

- the 'distinguished name' of the object entry whose subordinates are listed (present only if an alias was dereferenced);
- a 'set of subordinates' parameter, conveying the information on the immediate subordinate entries, if any, of the named entry. Should any of the subordinate entries be aliases, they will not be dereferenced. For each subordinate, the following information is specified:
  - the relative distinguished name of the subordinate;
  - an indication of whether the information was obtained from the original entry, or against a copy of the original entry;
- a 'limit problem' parameter indicates whether the time limit, size limit or an administrative limit has been reached;
- an 'unexplored' parameter, which shall be present if regions of the DIT were not explored. The parameter consists of a set of continuation references, allowing the DUA to continue the processing of the List operation by contacting other Access Points;
- the 'common results' qualifying the results of the operation;
- an 'error report' if the operation fails.

Search operation.

A Directory-Search operation is used to search a portion of the DIT for entries of interest, and to return selected information from those entries. The arguments of the operation may optionally be signed by the requestor. If so requested the Directory may sign the result.

```
Directory Search-Operation
    Arg :   - base object name ;
            - filter ;
            - entry information selection ;
            - common arguments ;
    Res :   - distinguished name ;
            - set of entry information ;
            - limit problem ;
            - unexplored ;
            - common results ;
            - error report ;
```

Arguments are:

- the name of the object ('base object name') which identifies the object entry relative to which the search is to take place, and an indication on whether the search is to be applicable to the base object only, the immediate subordinates of the base object only (one level in the DIT), or all subordinates of the base object (all subtree). Aliases may be dereferenced during the search (except if this is prohibited by the relevant service controls). If an alias is dereferenced, the search continues from the aliased object entry in its subtree;
- the 'filter' which is used to eliminate entries from the search space which are of no interest. Information will only be returned on entries which satisfy the filter;
- the 'entry information selection' which indicates what information from entries is requested;
- the 'common arguments' which specify the service controls and security features applying to the request.

The request succeeds if the base object is located, regardless of whether there are any subordinates to be returned.

Results are:

- the 'distinguished name' of the base object itself (only present if an alias was dereferenced in the base object name):
- the 'set of entry information' which conveys the requested information from each entry (zero or more) which satisfy the filter;
- a 'limit problem' parameter indicates whether the time limit, size limit or an administrative limit has been reached;
- an 'unexplored' parameter, which shall be present if regions of the DIT were not explored. The parameter consists of a set of continuation references, allowing the DUA to continue the processing of the Search operation by contacting other Access Points;
- the 'common results' qualifying the results of the operation;
- an 'error report' if the operation fails.

6.5.4. Directory MODIFY operations.

There are four operations to modify the Directory Information Base: 'Add-Entry', 'Remove-Entry', 'Modify-Entry' and 'Modify-RDN'. Each of these operations identifies the target entry by means of its distinguished name (so alias names cannot be used).

Add-Entry operation.

A Directory-Add-Entry operation is used to add a leaf entry (either an object entry or an alias entry) to the DIT. The arguments of the operation may optionally be signed by the requestor.

Note that the operation does not provide any facility for controlling the physical placement of the entry (i.e. in which DSA the entry will be stored). In general, the Directory Service only supports adding an entry to the same DSA in which its immediate superior entry resides. An entry may be added to another DSA than the one in which its superior entry resides, but this has to be done through bilateral agreements.

```
Directory Add-Entry-Operation
    Arg :   - entry name ;
            - entry ;
            - common arguments ;
    Res :   - result ;
            - error report ;
```

Arguments are:

- the 'entry name' argument, which is the name of the entry to be added. Its immediate superior entry, which must already exist for the operation to succeed, can be determined by removing the last RDN component which belongs to the entry to be created;
- the 'entry' argument, which contains the information which constitutes the entry to be created. This includes the attributes of the RDN, and a consistency check with the RDN derived from the entry name argument is made. The Directory shall ensure that the entry conforms to the Directory schema. If the entry being created is an alias entry, no check is made to ensure that the 'aliased-object-name' attribute points to a valid entry.
- the 'common arguments' include the specification of the service controls applying to the request.

Aliases are never dereferenced by this operation.

Should the request succeed, a 'result' will be returned although no information will be conveyed in it.

Should the operation fail, an 'error report' is returned.

Remove-Entry operation.

A Directory-Remove-Entry operation is used to remove a leaf entry (either an object entry or an alias entry) from the DIT. The arguments of the operation may optionally be signed by the requestor.

As with Add-Entry, in general, the Directory Service only supports removing an entry which is in the same DSA in which its immediate superior entry resides. An entry which is on another DSA than the one in which its superior entry resides may be removed, but this has to be done through bilateral agreements.

```
Directory Remove-Entry-Operation
    Arg :   - entry name ;
            - common arguments ;
    Res :   - result ;
            - error report ;
```

Arguments are:

- the 'entry name', which is the name of the entry to be deleted. Aliases in the name will not be dereferenced;
- the 'common arguments' include the specification of the service controls applying to the request.

Should the request succeed, a 'result' will be returned although no information will be conveyed in it.

Should the operation fail, an 'error report' is returned.

Modify-Entry operation.

A Directory-Modify-Entry operation is used to perform a series of one or more of the following modifications to a single entry:

- add a new attribute;
- remove an attribute;
- add attribute values;
- remove attribute values;
- replace attribute values;

The arguments of the operation may optionally be signed by the requestor.

```
Directory Modify-Entry-Operation
    Arg :   - entry name ;
            - changes ;
            - common arguments ;
    Res :   - result ;
            - error report ;
```

Arguments are:

- the 'entry name', which is the name of the entry to which the modification should be applied. Any aliases in the name will not be dereferenced;
- the 'changes' argument, which defines a sequence of modifications that are to be applied in the specified order. If any of the individual modifications fails, then an error is generated and the entry is left in the state it was prior to the Modify-Entry operation (i.e. the operation is atomic). The end result of the sequence of modifications shall not violate the Directory schema. However it is possible, and sometimes necessary, for the individual entry modification changes to do so. The following individual types of modification may occur:
    - addition of a new fully specified attribute (i.e. attribute type and attribute values) to the entry. Any attempt to add an already existing attribute results in an attribute error;
    - removal of an attribute (specified by its attribute type) from the entry. Any attempt to remove a non-existing attribute, a mandatory attribute, or an attribute type present in the RDN results in an attribute error;
    - addition of one or more specified values to a specified attribute. An attempt to add an already existing value results in an error;
    - removal of one or more specified values from the specified attribute. If the values are not present in the attribute, or if an attempt is made to modify the object-class attribute, an error occurs.

    Attribute values may be replaced by a combination of remove and add value changes in a single modify entry operation.
- the 'common arguments' include the specification of the service controls applying to the request.

Aliases are never dereferenced by this operation.

Should the request succeed, a 'result' will be returned although no information will be conveyed in it.

Should the operation fail, an 'error report' is returned.

Modify-RDN operation.

The Directory-Modify-RDN operation is used to change the relative distinguished name of a leaf entry (either an object entry or an alias entry) in the DIT. The arguments of the operation may optionally be signed by the requestor.

```
Directory Modify-RDN-Operation
    Arg :   - entry name ;
            - new RDN ;
            - delete old RDN flag ;
            - common arguments ;
    Res :   - result ;
            - error report ;
```

This operation may only be used on a leaf entry.

Arguments are:

- the 'entry name' which is the name of the entry whose RDN is to be modified. Aliases in the name will not be dereferenced;
- the 'new RDN' which specifies the new RDN of the entry. The attribute values specified in this argument become the distinguished values of the corresponding attribute types. If a specified attribute value in the RDN does not already exist in the entry (either as part of the old RDN or as a non-distinguished value), it is added. If it cannot be added, an error is returned.
- a 'delete old RDN flag', which, if set, indicates that all attribute values in the old RDN which are not in the new RDN are to be deleted. If this flag is not set, the old values should remain in the entry as non-distinguished values of the corresponding attribute types. The flag should be set when a single valued attribute type in the RDN has its value changed by the operation;
- the 'common arguments' include the specification of the service controls applying to the request.

Should the request succeed, a 'result' will be returned although no information will be conveyed in it.

Should the operation fails, an 'error report' is returned.

7. The Distributed Directory

This chapter aims to present the Directory as a distributed system, and to show how information distribution and the distributed operations which permit the distributed service to be performed are achieved. In particular, the knowledge information, which is the basis for determining the DSA holding a particular entry, is described.

This chapter follows the description given in [X.518-88]. For further details, Annex B presents a way the distributed operation procedures can be implemented.

7.1. The Distributed Directory System Model

DSAs are defined in order that distribution of the DIB can be accommodated and that a number of physically distributed DSAs can interact in a prescribed, cooperative manner to provide the Directory Services to users of the Directory (DUAs).

The ports associated with a DSA are of two types: service-ports and distributed-service-ports.

The service-ports of a DSA are the externally visible ports (i.e. from the DUA point of view): namely the Read, Search and Modify ports. The service-ports associated with a DSA constitute an Access Point through which Directory Services described in chapter 6 are made available to DUAs.

In addition to the service-ports of the DSAs which accommodate access to the Directory, a second set of ports for DSAs are defined: the distributed-service-ports. These permit communication between DSAs, allowing DSAs to propagate requests for services to other DSAs, in order that the Directory Service can be achieved in a distributed environment.

The types of distributed-service-ports and the operations provided through them are in direct correspondence to the similarly named service-ports, and are the 'Distributed-Read' ports, the 'Distributed-Search' ports and the 'Distributed-Modify' ports.

The 'Distributed-Read' ports support the 'Distributed-Read', 'Distributed-Compare' and 'Distributed-Abandon' operations, the 'Distributed-Search' ports support the 'Distributed-List' and 'Distributed-Search' operations, and the 'Distributed-Modify' port support the 'Distributed-Add-Entry', 'Distributed-Remove-Entry', 'Distributed-Modify-Entry' and 'Distributed-Modify-RDN' operations. These operations are described in the following sections.

Figure 7.1 illustrates the distributed Directory model.

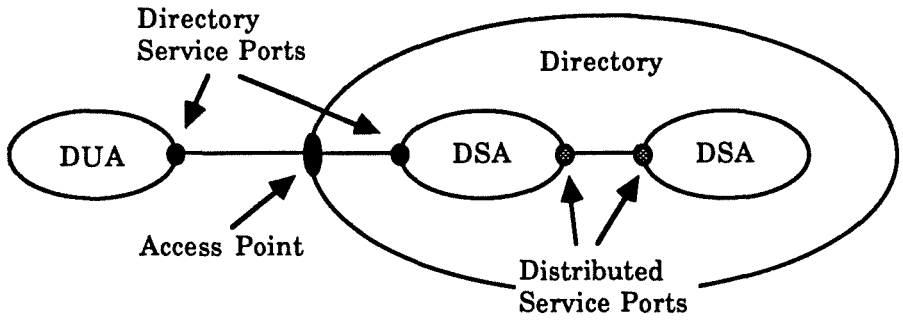


Figure 7.1 : Distributed Directory Model.

7.2. Directory Distribution

This section describes the principles according to which the DIB can be distributed.

Each entry within the DIB is administered by one, and only one, DSA's administrator who is said to have 'administrative authority' for that entry. Maintenance and management of an entry must take place in a DSA administered by the administrative authority for the entry.

Each DSA within the Directory holds one or more fragments of the DIB. The DIB fragment held by a DSA are described in terms of the DIT, and are called 'naming contexts'. A naming context is a partial subtree of the DIT defined as starting at the vertex the closest from the root denoting an entry held by a DSA and extending downwards to leaf and/or non-leaf vertices denoting entries held by the same DSA. Vertices denoting entries belonging to other DSAs denote the start of further naming contexts.

It is possible for a DSA's administrator to have administrative authority for several disjoint naming contexts, i.e. which not share a same superior entry held in the DSA. For every naming context for which the DSA has administrative authority, it must logically hold the sequence of RDNs which leads from the root of the DIT to the initial vertex of the subtree comprising the naming context. This sequence of RDNs is called the 'context prefix'.

A DSA administrator may delegate administrative authority for any immediate subordinate of any entry held locally to another DSA. Delegation of administrative authority begins with the root and proceeds downwards in the DIT; that is it can only occur from an entry to its subordinates.

Figure 7.2 illustrates a hypothetical DIT logically partitioned into five naming contexts which are physically distributed over three DSAs.

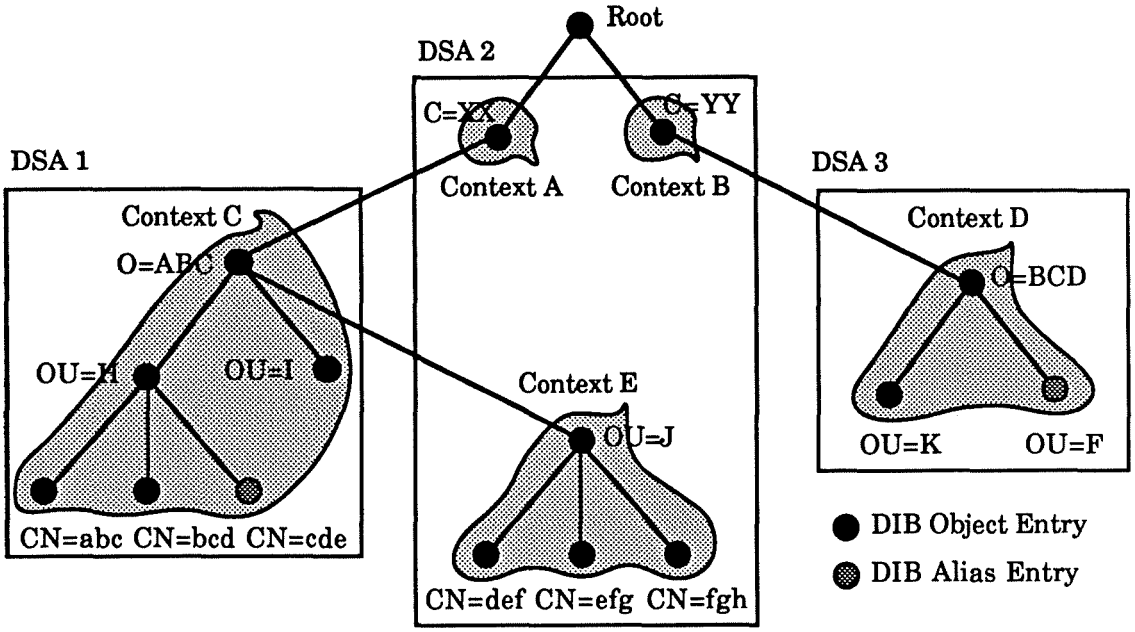


Figure 7.2 : Hypothetical DIT.

The DSA2's administrator has administrative authority for the naming contexts A,B and E. The administrator of the naming context A has delegated its authority to the administrator of DSA1 for the administration of an immediate subordinate of the naming context A (namely the entry whose name is <C=XX;O=ABC>). This subordinate starts the naming context C. The context prefix of the naming context C is <C=XX;O=ABC>.

Note that the root is not held by any DSA.

From the above definitions, the limiting case for a naming context can be either a single entry, or the whole DIT.

During the process of modifications of entries, it is possible that the Directory may become inconsistent. This will be particularly likely if modifications involve aliases and aliased entries which may be in different DSAs. The inconsistency must be corrected by specific administrator actions, for example to delete aliases if the corresponding aliased entries have been deleted. The Directory must continue to operate during this period of inconsistency.

In order for a DUA to begin processing a request, it must hold some information , specifically the presentation address, about at least one DSA that it can contact initially. How it acquires and holds this information is a local matter.

Although the Directory does not provide any support for the replication of entries, it is nevertheless possible to realize replication in two ways:

- copies of an entry may be stored in other DSAs through bilateral agreements. The means by which these copies are maintained and managed is a function of the bilateral agreement;
- copies of an entry may be acquired by storing locally and dynamically a copy of an entry which results from a request.

The originator of the request is informed as to whether information returned in response to a request is from a replicated entry or not. A Service Control is defined which allows the user to prohibit the use of replicated entries.

7.3. DSA interaction model

A basic characteristic of the Directory is that, given a distributed DIB, a user should potentially be able to have any service request satisfied (subject to security and access controls) regardless of the Access Point at which the request originates. In accommodating this requirement, it is necessary that any DSA involved in satisfying a particular service request have some knowledge of where the requested information is located and either return this knowledge to the requestor, or attempt to have the request satisfied on its behalf (the requestor may be either a DUA or another DSA).

Three modes of DSA interaction are defined to meet these requirements, namely 'chaining', 'multicasting' and 'referral'. 'Chaining' and 'multicasting' are defined to meet the latter of the above requirements whilst 'referral' addresses the former.

7.3.1. Chaining

This mode of communication may be used by one DSA to pass on a remote operation to only one other DSA when the former has specific knowledge about naming contexts held by the latter.

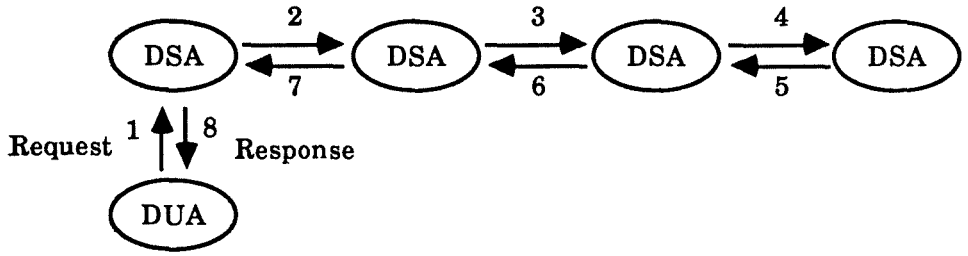


Figure 7.3 : Chaining Mode.

7.3.2. Multicasting

This mode of communication is used by one DSA to pass on an identical remote operation in parallel or sequentially to one or more DSAs, when the former does not know the complete naming contexts held by the other DSAs. Each of the DSAs is passed on the identical remote operation.

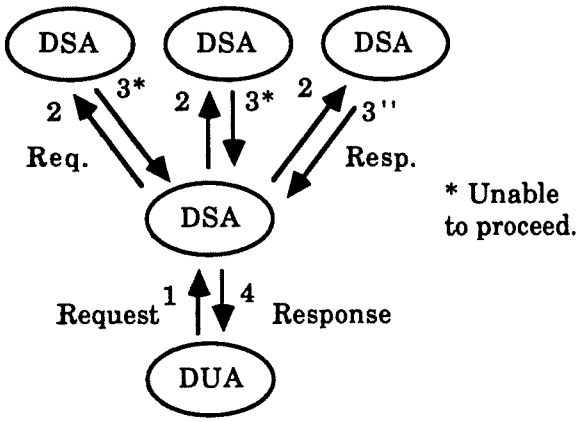


Figure 7.4a : Multicasting Parallel Mode.

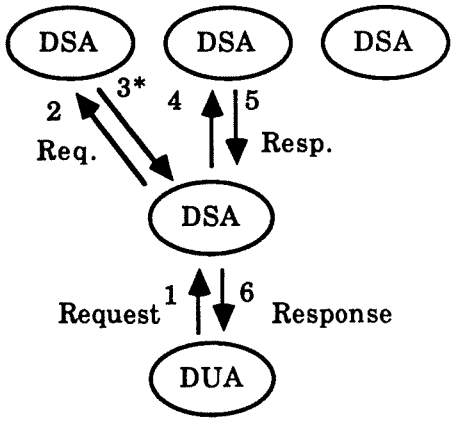


Figure 7.4b : Multicasting Sequential Mode.

7.3.3. Referral

A referral is returned by a DSA in its response to a remote operation which it has been requested to perform, either by a DUA or another DSA. The referral contains a knowledge reference to another DSA able to proceed or continue the operation.

The DSA or DUA receiving the referral may use the reference contained therein to subsequently chain or multicast the original operation to other DSAs. Alternatively, a DSA receiving a referral, may in turn pass the referral back in its response.

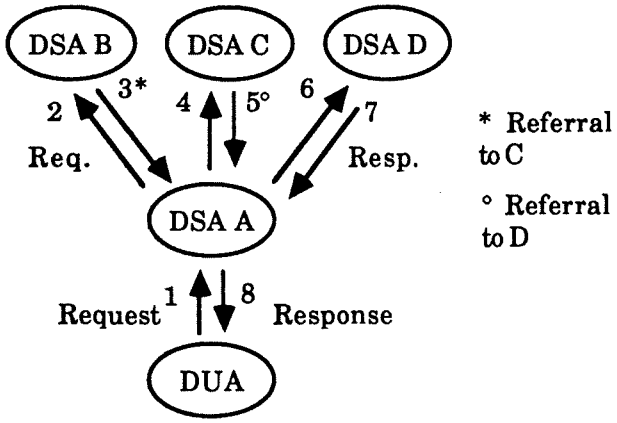


Figure 7.5a : Referral Mode, DSA with chained port.

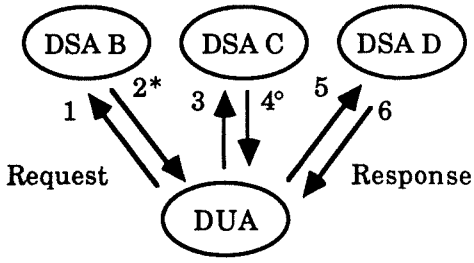


Figure 7.5b : Referral Mode, DSAs with no chained port.

7.3.4. Mode Determination

If a DSA cannot itself fully resolve a request, it must chain or multicast the request (or a request formed by decomposing the original one) to another DSA, unless chaining is prohibited by the user via the 'service controls' (in which case the DSA must return a referral or a service error), or unless the DSA has administrative, operational or technical reasons for preferring not to chain (in which case the DSA must return a referral).

7.4. Knowledge

The DIB is potentially distributed across multiple DSAs with every DSA holding a DIB fragment. It is a requirement that, for particular modes of interaction, the distribution of the Directory be rendered transparent, thereby giving the effect that the whole of the DIB appears to be within each and every DSA.

In order to support these operational requirements, it is necessary that each DSA holding a fragment of the DIB be able to identify and optionally interact with other DSAs holding other fragments of the DIB.

This section describes the knowledge as the basis for the mapping of a name to its location within a fragment of the DIT.

Conceptually, DSAs hold two types of information:
- the Directory information, which is the collection of entries comprising the naming context(s) for which the administrator of a particular DSA has administrative authority;
- the knowledge information, which denotes how the naming contexts held by a particular DSA fit into the overall DIT hierarchy.

The knowledge possessed by a DSA is defined in terms of a set of one or more knowledge references where each reference associates, either directly or indirectly, entries of the DIB with DSAs which hold those entries.

The context prefix of a naming context is a sequence of RDNs leading from the root of the DIT to the initial vertex of the naming context and corresponds to the distinguished name of that vertex.

7.4.1. Minimal Knowledge

The Directory must insure that every entry can be accessed from every DUA, whatever the Access Point the DUA uses to enter in communication with the Directory. Therefore a reference path, which is a continuous sequence of references, must exist from each DSA to all naming contexts within the Directory. The set of all these references and DSAs can be modelled as a graph. To ensure a correct behavior of the Directory, a minimal set of references must exist. This set can be regarded as the skeleton of the graph. Additional features should be added to the skeleton in order to guarantee an acceptable performance.

Accordingly, a DSA shall maintain:
- superior knowledge, i.e each DSA shall have a reference path to the root context;
- subordinate knowledge, i.e. each DSA shall have a reference to those DSAs (if there are any) which hold naming contexts that are subordinate to a naming context for which it has administrative authority. From the root context exits a reference path to each other naming context.

These two conditions guarantees that a reference path exists from every DSA to any naming contexts held by other DSAs.

Superior and subordinate references are illustrated in figure 7.6.

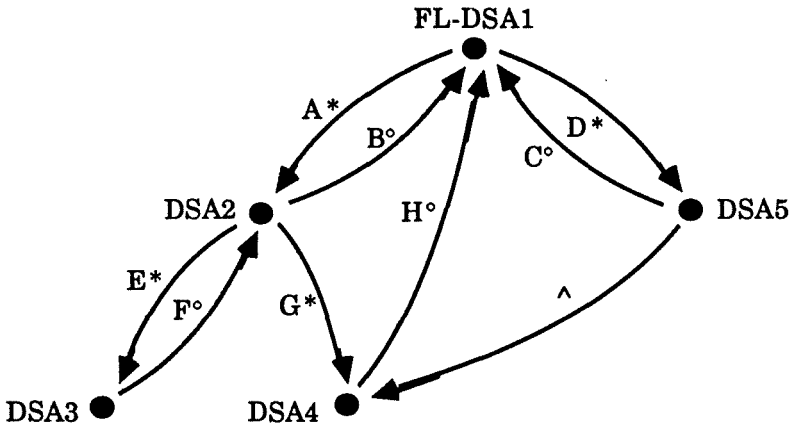


Figure 7.6 : Subordinate and superior references.

In figure 7.6, an arrow indicates a reference held by the DSA at the origin of the arc. A '*' represents a subordinate reference, while a '°' represents a superior reference. A '^' represents a cross reference (see section 7.4.3). FL-DSA1 is a First-Level DSA which holds the root context (i.e. a reference to every other First-Level DSA, see section 7.4.2).

For DSA3, the reference path to the root context is made of the two superior references F and B, while for DSA4, the reference path to the root context is made of the single superior reference H. From every DSA exists a reference path to every other DSAs, and thus to all the naming contexts of the Directory held by these DSAs.

7.4.2. Root Context

Because of the autonomy of the different countries, or global organizations, there is no 'single' DSA which holds the root context. The functionality of a 'root-DSA' concerning the name resolution process has to be provided by those DSAs which have administrative authority for naming context that are immediately subordinate to the root. These DSAs are called 'First-Level DSAs'. Each First-Level DSA must be able to simulate the functionality of the 'root-DSA'. This requires full knowledge about the root naming context. The root context is replicated onto each First-Level DSA and therefore has to be administered commonly by the autonomous first level administrative authorities. Administration procedures have to be determined by multilateral agreements.

Each First-Level DSA shall hold the root context, which implies a reference to each other First-Level DSA.

Each non-First-Level DSA shall have a superior reference, which implies a reference path to any arbitrary First-Level DSA.

If a new non-First Level DSA is introduced, it must have a minimal initial knowledge, which is represented by a superior reference. Any further knowledge will be added by subordinate references or cross references (see section 7.4.3).

If a new First-Level DSA is introduced, it must acquire the root context and advise all other First Level DSAs.

7.4.3. Knowledge References

To be able to fulfill the requirements to reach every DIB entry from any DSA, every DSA is required to have knowledge about the entries which it itself holds, and about subordinate and possible superior entries thereof (minimal knowledge). This gives rise to the following types of knowledge references: 'internal references', 'subordinate references', 'non-specific subordinate references' and 'superior references'. Additionally, for optimization purposes, optional 'cross references' are defined.

Internal Reference

An 'internal reference' consists of:
- the distinguished name corresponding to a DIB entry held by the DSA;
- an internal pointer to where the entry is stored in the local DIB.

All entries for which a particular DSA has administrative authority are represented by internal references in the knowledge information of the DSA.

Subordinate Reference

A 'subordinate reference' consists of:
- the distinguished name corresponding to an immediate subordinate DIB entry held by another DSA;
- the distinguished name (Application Entity Title) of the DSA to which administrative authority for that entry was delegated;
- that DSA's Presentation-Address.

All subordinate references held by a particular DSA (corresponding to those subordinate entries for which this DSA has delegated administrative authority to another DSA) must be represented by subordinate references or non-specific subordinate references.

Non-Specific Subordinate Reference

A 'non-specific subordinate reference' consists of:
- the Distinguished Name (Application-Entity-Title) of a DSA which holds one or more immediately subordinate Naming Contexts;
- that DSA's Presentation-Address.

This type of reference is optional, to allow for the case in which a DSA is known to contain some subordinate entries of the DSA but the specific distinguished names of those entries is not known.

For each naming context which it holds, a DSA may hold any number (including zero) of non-specific subordinate reference, which will be evaluated if all specific internal and subordinate references have been pursued.

Superior Reference

A 'superior reference' consists of:
- the distinguished name (Application-Entity-Title) of a DSA holding a superior entry of the naming context;
- that DSA's Presentation-Address.

Each non-First-Level DSA maintains precisely one superior reference, which is related to the context prefix of a superior naming context (not necessarily the one of its First-Level DSA). The superior reference held by a DSA shall refer to a DSA which holds a naming context with a context prefix with a fewer number of RDNs than the context prefix of the closest to the root naming context held by the former DSA.

Cross Reference

A 'cross reference' consists of:
- a Context Prefix;
- the Distinguished Name (Application-Entity-Title) of a DSA which has administrative authority for the corresponding naming context;
- that DSA's Presentation-Address.

This type of reference is optional and serves to optimize Distributed Name Resolution (see Annex B). A DSA may hold any number (including zero) of cross references.

In the event that the set of knowledge references associated with a particular DSA contain only internal references, the DSA has no knowledge of other DSAs and the DIB is therefore centralized.

7.4.4. Knowledge Administration

To operate a widely distributed Directory with an acceptable degree of consistency and performance, procedures are required to maintain and extend the knowledge held by each DSA. The same procedures are appropriated for the creating initial knowledge of a DSA.

Knowledge can be maintained by:
- the DSA, or its administrative authority, propagating changes of knowledge to those DSAs holding all kinds of references to it, whenever changes at that DSA cause these references to become invalid. This is the only way superior, subordinate and non-specific subordinate references can be maintained. Procedures for propagating knowledge changes must locally be established by bilateral agreements;
- the DSA requesting and obtaining cross references to improve the performance of the service. The local set of cross knowledge references can be expanded using ordinary Directory operations.

The Directory has to support consistency checking mechanisms to guarantee a certain degree of knowledge consistency. After a DSA has detected an invalid reference, it should try to re-establish knowledge consistency. Depending on the role and the importance of the reference, this can be done by simply deleting the invalid reference or by replacing it with a correct one which can be obtained using the above mechanisms.

7.5. The Distributed Directory Operations

7.5.1. DSA-Bind and DSA-Unbind operations

DSA-Bind and DSA-Unbind operations are respectively used by a DSA at the beginning and at the end of a period of accessing to another DSA.

A DSA-Bind operation is used by a DSA to bind its 'Distributed-Read', 'Distributed-Search' and 'Distributed-Modify' ports to those of another DSA.

A DSA-Unbind operation is used to unbind the 'Distributed-Read', 'Distributed-Search' and 'Distributed-Modify' ports of a pair of DSAs.

The components of these operations are identical to their counterparts in respectively the Directory-Bind and Directory-Unbind operations defined in the previous chapter.

7.5.2. Other Distributed Operations

A distributed operation is used to propagate between DSAs a request which (normally) originated from a DUA invoking an operation at a DSA, that DSA having elected to chain or multicast it.

Corresponding to each of the ports of the Directory Service is a port of the DSA which allows the service to be provided by cooperating DSAs.

The operations in the corresponding ports are also in one to one correspondence. The names of the ports and operations have been chosen to reflect this correspondence, with the port or operation name in the DSA distributed service being formed from that of the Directory Service prefixed by the word 'distributed' (e.g. Distributed-Read port, Distributed-Read operation,...).

The arguments, results and errors of the distributed operations are, with one exception, formed systematically from the arguments, results and errors of the corresponding operations in the Directory Service (as described in section 6.5).

The one exception is the Distributed-Abandon operation, which is syntactically equivalent to its Directory Service counterpart.

The arguments of the distributed operations may optionally be signed and, if so requested, the performing DSA may sign the results.

```
DSA Distributed-X-Operation
    Arg :   - Distributed Operation Arguments ;
            - Arguments of the Directory X-Operation ;
    Res :   - Distributed Operation Results ;
            - Results of the Directory X Operation
                (including error reports);
```

The 'Distributed Operation Arguments' contain the information needed in order for the performing DSA to carry out the distributed operation. This information is described in the following section.

The 'Arguments of the Directory X-Operation' contain the original DUA supplied arguments, as described in section 6.4.

The 'Distributed Operation Results' contain the information which may be needed by previous DSAs in a chain or a multicast. This information is described in section 7.5.4.

The 'Results of the Directory X-Operation' contain the results which are being returned by the performer of the operation , and which are intended to be passed back in the results to the originating DUA.

7.5.3. Distributed Operation Arguments

The 'Distributed Operation Arguments' are present in each distributed operation to convey to a DSA the information needed to successfully perform its part of the overall task.

The Distributed Operation Arguments comprises the following components:

```
Distributed Operation Arguments :
    comprise : - originator ;
               - target object ;
               - operation progress ;
               - trace information ;
               - return cross-references ;
               - reference type ;
               - info ;
               - time limit ;
```

The 'originator' component conveys the name of the originator of the request unless already specified in the common arguments.

The 'target object' component conveys the name of the object entry to which the operation is being routed to. The role of this object entry depends on the particular operation concerned: it may be the object entry to which the operation is to be operated on (e.g. in a Distributed-Read or a Distributed-Modify operation), or which is the base object for a request or sub-request involving multiple objects (i.e. in a Distributed-List or a Distributed-Search operation).

The 'operation progress' component is used to inform the DSA of the progress of the operation, and hence of the role it is expected to play in the overall performance. It specifies if the 'Name Resolution' phase is not started, in process or completed and, if relevant, the next RDN to be resolved (see Annex B).

The 'trace information' component is used to prevent looping among DSAs when chaining is in operation. A DSA adds a new element to the 'trace information' prior to chaining the operation to another DSA. On being requested to perform an operation, a DSA checks, by examination of the trace information, that the operation has not formed a loop.

The 'alias dereferenced' component is a boolean value which is used to indicates whether or not one or more alias entries have so far been encountered and dereferenced during the course of the distributed name resolution.

The 'return cross-references' component is a boolean value which indicates whether or not knowledge references used during the course of a distributed operation are requested to be passed back to the initial DSA.

The 'reference type' component indicates to the DSA being asked to perform the operation, what type of knowledge was used to route the request to it. The DSA may therefore be able to detect errors in the knowledge held by the invoker.

The 'info' component is used to convey DMD specific information among DSAs which are invoked in the processing of a common request. This information may be of any type.

The 'time limit' component indicates the time by which the operation is to be completed.

7.5.4. Distributed Operation Results

The 'Distributed Operation Results' are present in the results of each distributed operation and provide feedback to the DSA which invoked the operation.

The 'Distributed Operation Results' comprise the following components:

```
Distributed Operation Results :
    comprise : - info ;
               - cross references ;
```

The 'info' component is used to convey DMD specific information among DSAs which are invoked in the processing of a common request. This information may be of any type.

The 'cross reference' component is not present in the distributed operation results unless the 'return cross-references' argument component of the corresponding request held the value 'true'. This component consists of a sequence of cross references.

Part 3 Directory Issues

This third part discusses some Directory related issues.

Chapter 8 points out some of the difficulties which arose during the elaboration of the X.500 standard. It first presents TR/32, the ECMA Directory, and ends with some shortcomings of the X.500 Directory standard.

Chapter 9 considers the X.500 Directory as a database system and describes how a classical database can be mapped onto the X.500 Directory structure. Chapter 10 presents the RARE Directory, and chapter 11 describes five Directory Service implementations.

8 X.500 : Difficulties and Shortcomings

The Directory Service standardization elaboration has not been an easy process. Difficulties have led to a rather limited service in comparison with what was first expected. This chapter first presents TR/32, the pre-X.500 ECMA Directory Service standard, and then relates the evolution of the directory standardization. The last part of the chapter is devoted to the X.500 Directory Service difficulties and shortcomings.

8.1. TR/32.

The first strong requirements for a Directory standard was expressed in the 1984 CCITT X.400 recommendation for Message Handling Systems. A group was formed in CCITT with the mandate to start work immediately in 1984. In parallel to CCITT, work was also started in ISO and ECMA [OSN-87].

ECMA which has the reputation to be quick on new subjects, took the lead temporarily. In December 1985, ECMA issued its TR/32 Directory Service report.

X.500 has been deeply influenced by the ECMA Directory Service recommendation, and most of concepts described in TR/32 are found in X.500. That's why this section only points out the major difference between X.500 and TR/32: replication.

Similarities with X.500.

The ECMA Directory has a hierarchical naming structure similar to X.500, with the option to find the way to an entry also using an alias name. Wildcard characters may be used in the last part of a name (i.e. last RDN) to allow only a partial specification of a name. This partial name is taken as a pattern by the Directory Service which attempts to find a name or a set of names in the information base that match it. Entries are structured in a tree, as in X.500.

The ECMA Directory is, as X.500, modelled on the client-server paradigm. However, the Directory System Protocol (i.e. the protocol which supports the performance of the distributed operations) is achieved thanks to the Directory Access Protocol. Each Directory Service Agent is independent in the sense that there is no specified protocol to communicate directly between DSAs. A DSA which accesses another DSA has to act as a DUA, using the Directory Access Protocol and services. A DSA may use the DAP to query other DSAs in order to satisfy the original DUA request, but will generally send back some kind of referral (called 'hint').

One of the main ideas of ECMA is that there will not have only one worldwide Directory tree, but many small trees, each of which held by one or more DSAs. When a Directory System wants to use Directory Services provided by another Directory System, the former will have to act as a DUA to interact with the latter. And, except if implementation specific protocols are defined, the same is true when a DSA of a Directory System distributed over several DSAs wants to access another DSA of the same Directory System. Note that X.500 does not preclude the existence of several DITs.

The ECMA Directory Service does not explicitly support Directory schema, but offers more flexibility than X.500 to perform exploration of the DIT thanks to its larger set of Directory operations.

ECMA provides for limited authentification mechanisms, but no access control nor security features.

ECMA provides specific operations to manage alias entries (create alias / delete alias), and in particular, the interesting function which returns all alias names of a particular entry. Other operations defined in ECMA roughly provide the same functionalities than the ones defined in X.500. The ECMA Directory Service does not support a search-like operation in a whole subtree of the DIT as in X.500, nor filtered search.

ECMA defines two kinds of attributes: 'item properties', similar to the attributes defined in X.500, and 'group properties'. The value of an item property is not inspected by the Directory System and may consist of any data the user wishes. On the other hand, the value of a group property is understood by the Directory System to be a sequence of names called members. The Directory System knows how to add, delete, enumerate and search for members in a group property. Specific Directory operations are provided in order to manage group properties. The same functionality is offered by X.500 thanks to the multivalued attributes, but there are no specific operations to manage them.

Replication and Distribution.

In order to improve efficiency, replication of part or of all of the Directory Information Base may be desirable.

The ECMA Directory Service is designed to allow distribution and replication of its DIB.

Distribution is opposed to centralization. If the local information base held by a DSA is the entire DIB, there is no need for communications between DSAs. This situation is said to be 'centralized'. Distribution of the information base is defined as the property that there are multiple DSAs, each of which has a local information base which is a proper subset of the DIB.

Replication is the property of some portions of the DIB of being held by multiple DSAs. The DIB may be partially or totally replicated.

Replication allows reliability of the DIB (if a copy is damaged, others are still available), availability of the DIB (despite server or communication failures) and efficiency of access to the DIB (despite geographically remote DSAs).

Replication within the DIB implies the problem of propagating updates in the multiple copies of the information. Therefore, a Directory System Protocol for DSA to DSA communications has to be developed. But this is out of the scope of the ECMA report, and is thus subject to bilateral agreements. Replication also involves that the user has to deal with temporary inconsistencies, as there may exist a appreciable delay before all copies are updated.

Distribution mainly provides the ability to handle a large DIB with modest means. But database distribution implies the necessity of 'navigating' the database in order to find the data. When a DSA cannot satisfy an operation locally, it tells the DUA that it has contacted the wrong DSA and gives the DUA a 'hint' about where to look next.

8.2. Evolution of the Directory Standardization.

As previously said, work on Directory standardization was started in parallel by the three CCITT, ISO and ECMA working groups during 1984.

Around mid-1985, all three groups had converged onto the 'ECMA solution' described here above.

During 1985, additional requirements surfaced, which led CCITT to a new more advanced model for the Directory Information Tree, as well as features for distributed reading and updating of the Directory information [OSN-87].

ISO was primarily happy with the earlier, simpler solutions, but had in the meantime seen the great advantage of close cooperation with CCITT on the subject, and it was formally decided to have joint documents and joint meetings.

The number of features increased and included the concept of shadowing which allows a part of the total Directory information to be copied to another part of the Directory Information Tree, in order to save search time on frequent queries (replication). The going-in position at the Egham meeting in September 1986 was to simplify and stabilize the texts from the previous meetings in order to produce a complete and coherent set of documents for publication as ISO Draft Proposals. Although a few simplifications were achieved, a number of contributions, primarily from CCITT members, requested the inclusion of a number of new features (e.g. that the different read and search operations be combined into something internally referred to as 'fat read', which allows a huge part of the directory information to be read in single access protocol interaction).

This was the scenario set for the Munich meeting beginning 1987. In the meantime, the national standards organizations (ANSI, BSI, DIN, AFNOR etc. which are the ISO members), had scrutinized the Draft Proposals texts and their official ballot comments were the most input of the meeting. The US, Canada, UK, Australia, France, Holland, Sweden, and Fed. Rep. of Germany had all voted NO on all parts of the Draft Proposals. The only two major countries who had voted YES were the USSR and Japan, but the Japanese also had a whole bunch of comments, which they requested to be taken into account.

Threatened by the possibility of not having a set of Directory recommendations at all for 1988, CCITT proposed to cut out all features which were not necessary from a CCITT point of view, and concentrate on stabilizing a subset which was agreeable to everybody. This would not preclude further developments of the standards in a second step.

The most important features of the proposed slimmed down version were:
- to limit the directory capabilities to 'read only'. The simple read and some more advanced search capabilities would be kept, but all updating functions for the Directory Information would be outside the scope of the first set of standards. Any functions associated with 'shadowing' would also be dropped, as they were closely related to the updating problems;
- the number of possible name types would be reduced (e.g. no provision for short names). Only two types of names would be allowed, the 'distinguished' names, which uniquely identify a Directory entry through its placement in the Directory Information Tree and 'alias' names, which follow the same rules as the 'distinguished' names, but end with a pointer rather than an entry;
- the earlier requirement that all entries must be, at least potentially, members of one universal directory tree was dropped and it will be possible to claim conformance to the Directory standard even with a separate tree. This however would not change the objective within CCITT to have one universal tree for all the Directory entry information relevant to public PTT services;
- the security features in the form of explicit standards for authentification and access control were reduced to be a local implementation matter. However some 'hooks' will be provided in the protocol to allow for private intermediate solutions.

The resulting structure of the new proposal of Directory consisted of two Application Service Elements (ASEs), a Directory Read ASE (with the 'Read' and 'Compare' operations) and a Directory Search ASE (with the 'Search' and 'List' operations).

Later on the third Directory Modify ASE has been added, allowing simple updating facilities for the DIB (with the 'Add-Entry', 'Remove-Entry', 'Modify-Entry' and 'Modify-RDN' operations). Moreover, provision for simple authentification has been reintroduced. This has led to the X.500 Directory standard described in Part 2.

8.3. X.500 Difficulties and Shortcomings.

Distribution

The current specifications of the 'Add-Entry' and 'Remove-Entry' operations do not provide enough functionalities for a distributed environment. These operations are restricted to the case that exactly one DSA is affected. Thus, the entry to be added or deleted and its superior object entry must be held in the same DSA [COS-88].

In the context of a distributed Directory, the 'Add-Entry' and 'Remove-Entry' operations may affect two DSAs: the DSA holding the superior entry of the entry to be added or deleted and the DSA where the entry will be stored or removed. This means for the following operations:
- 'Add-Entry': the DSA of the superior entry has to ensure the uniqueness of the Relative Distinguished Name of the entry to be added and its knowledge references must be extended by a subordinate reference. A new naming context must be inserted into the knowledge information of the DSA where the entry will be stored.
- 'Remove-Entry': deleting an entry may affect knowledge information of two DSAs. The DSA holding the superior entry of the entry to be removed has to remove the associated subordinate reference. The DSA holding the entry has to remove its corresponding naming context.

There is another problem: representation of knowledge information is not standardized. Even if implemented, automatic modification and exchange of knowledge information might not work when systems of different vendors are involved.

Thus the task of distributing the information base of the Directory will be within the responsibility of the administrators of the affected DSAs.

Replication

The standard does not include provision for replication, the 'reliable' management of copies.

Running a distributed Directory without replication may be possible if only human users access the Directory. Getting the requested information is of more interest than the response time, which will be depending on the numbers of DSAs involved.

Response time within a distributed Directory may be inacceptable if the Directory is used by applications (e.g. a user which runs FTAM with access to the Directory).

Caching (storing information that has been received from other DSAs and later accessing this information) will not be the solution for this problem. Cached information does not reveal access control restrictions. Thus, cached information should be returned only to the user which initially got the original information. Consequently, cached information has to be stored on a 'per user' base.

Access Control

Access control will not be based on a standardized model, it is left to DSAs as a local matter. This maybe of minor importance if all DSAs within a management domain are supplied by the same manufacturer. But there could be problems if interworking of different systems has to be considered (e.g. incompatible access policies).

Schema

Some suggestions for naming practice and DIT structures are given in Annex B of [X.521-88], but there are problems when DSAs with different DIT structures need to interwork. The Directory schema of a particular DSA may not be communicated to other DSAs, because there is no provision in the protocol for communicating the Directory definitions to other DSAs. As a consequence, a DSA do not know the DIT structure of other DSAs, and cannot help the user in querying the Directory in a more efficient way when remote DSAs are involved.

Knowledge

The current version of the X.500 standard does not include global concepts for the acquisition and management of knowledge information within a widely distributed Directory. Operations which affect consistency of knowledge information across DSA boundaries are permitted only by bilateral agreement.

Charging and Accounting

X.500 makes no provision for charging and accounting.

All services of the Directory should be accountable, because the Directory System providers may want to receive payment for Directory use.

If payment is necessary, the user should be informed about the costs and the user should be able to restrict the search for information to limit his costs.

9 The X.500 Directory as a Database System

Up to now, the Directory Service has been considered mainly as an application in the context of communication systems architecture. This chapter takes a different view and considers the Directory in the context of a database theory. Main ideas are picked up from [Len-6.5/M].

The X.500 Directory Service manages information about both persons, groups, organizations, etc and communication services. This information is distributed among several DSAs, each of which is generally managed by an organization. In most of cases, such information is already stored and maintained in databases within the organizations. It can be assumed that the Directory Service implementation will not often be based on a special purpose DataBase (DB) and DataBase Management System (DBMS) that are created and maintained for exclusive use as part of the Directory. Rather, existing DB and DBMSs will be made available for Directory Service use, in order to minimize data maintenance costs. These systems will continue to be used for local non Directory specific purposes as well.

X.500 mentions that the Directory Service is not a general purpose DBMS: transient inconsistencies of the DB are acceptable, which simplifies the design and implementation of such systems. On the other hand, the Directory Service exhibits properties that classifies it as a database management system, e.g. it manages structured data and manipulates data via predefined interfaces and protocols. As the Directory System is composed of autonomous functional units (DSAs, each of which administrates a part of the Directory Information Base), possibly managed with different DBMSs, one should consider the Directory as a distributed and heterogeneous database system.

In X.500, entries in the Directory are described in a very general way, and are embedded in a hierarchical structure. Such a description is suitable if we consider the multitude of applications that may use the Directory Service. X.500 provides flexibility as far as data structure of entries is concerned (e.g. it allows definitions of new object classes, new attribute types, ...). This may lead to a rather confusing variety of data instances which could make it difficult for users to know about all information they can get. Moreover, X.500 does not define any provision for access control on entries (who has the right to read, modify, remove, add, etc... entries).

The data contained in the Directory are here described on a conceptual level. An abstract description of the database is obtained by restricting the generality permitted by X.500.

9.1. Database Modelisation of The Directory.

As the Directory can be regarded as a DB system, it can be described under this aspect. This description of the Directory is based on the architecture proposed by the ANSI/SPARC Study Group. The ANSI/SPARC architecture defines three schemata (figure 9.1) which are briefly characterized below:

- the 'conceptual schema' is a logical description of the whole database. It involves neither aspects of its physical organization nor external representation of data (e.g. : an Entity/Relationship diagram (E/R diagram));
- the user's views of the database are defined by the 'external schema'. Each external schema defines a partial view of the database for a class of users. Several external schemata may coexist which provides different data views for various user classes. External schemata describe data visibility and structure, but not the user interface. It thus helps to define access rights for users being granted to work on such external schemata;
- the 'internal schema' describes the physical data representation (e.g. tables in a relational DBMS).

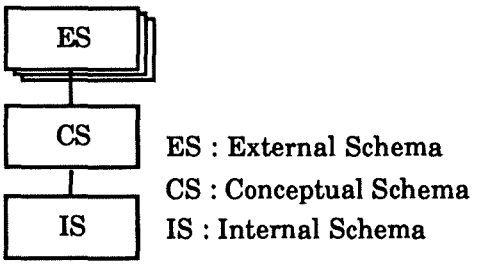


Figure 9.1 : The ANSI/SPARC Architecture

The Directory, considered as a distributed and heterogeneous database system, can be described in terms of an ANSI/SPARC architecture.

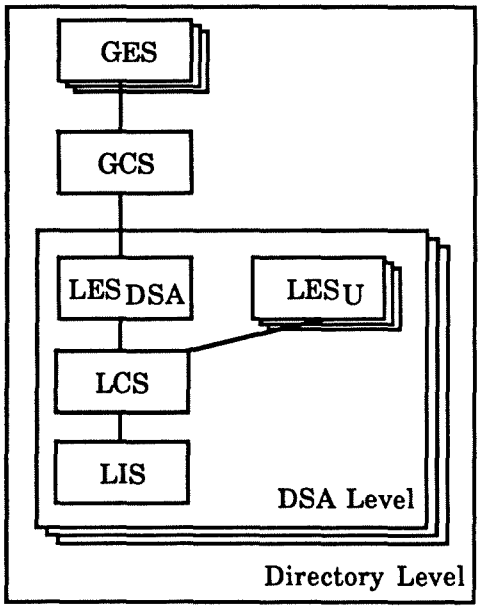


Figure 9.2 : ANSI/SPARC Architecture of a distributed Directory

Two levels of data are distinguished (figure 9.2): the global level, which defines the schema that is related to the Directory as a whole, and the local level with a set of schemata, each of which valid for a particular DSA composing the Directory.

The Global Conceptual Schema (GCS) directly corresponds to the global description of the DIB, i.e. to the Directory Schema (note that the GCS of the Directory is itself distributed, as a private or national Directory administrative authority can define its own Directory definitions, and in particular its own DIT structure). As the DIB allows the application of access control, a policy for defining access rights needs to be devised. Defining a specific Global External Schema (GES) corresponds to defining such a policy.

X.500 does not regulate the schemata on the local level, since this is an implementation dependent issue. Thus in a distributed Directory, many schemata may coexist at this level.

9.2. The Global Conceptual Schema.

Defining a Global Conceptual Schema for the Directory implies a description of the DIB on a conceptual level. In a first approach, entries and associations between entries in the DIB is considered as a conceptual description. This results in the hierarchical description which was introduced in the DIT. Figure 9.3 shows this structure as an Entity/Relationship diagram (in addition to this diagram, it must be stated that the structure has exactly one root). Since this approach results in a poor description, it is not followed any longer.

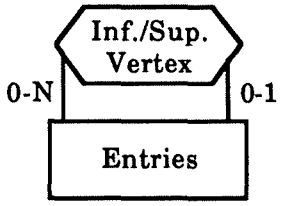


Figure 9.3 : E/R Diagram for the structure of the DIT.

Another approach is to analyze the internal structure of the entries and then to figure out the characteristics within the data description. X.500 is not very restrictive: an entry may have an arbitrary number of multi-valued attributes of different types and access control may be applied on four different levels of information (entire subtree, entry, attribute and attribute value). A first step towards a conceptual description of the DIB is possible by restricting the set of valid data structure for the entries.

Actually, X.500 is going in that direction too, as it defines a number of standard object classes (object class definitions). Each object belongs to a given class and comprises a minimal fixed number of mandatory named attributes. In addition, the relative position in the DIT is defined for each object belonging to a given class (DIT structure definitions). If we concentrate on these object classes, we are able to present an Entity/Relationship diagram of the GCS by:

- mapping object classes onto entity types and
- mapping hierarchical relations between superior and inferior vertices in the DIT onto associations between the corresponding entities.

Figure 9.4 corresponds to the DIT structure definitions of figure 5.8. Names of associations and roles have been omitted for clarity. The arrows in the associations indicate the hierarchical relation among entities: an arrow points to one of the possible superior object class.

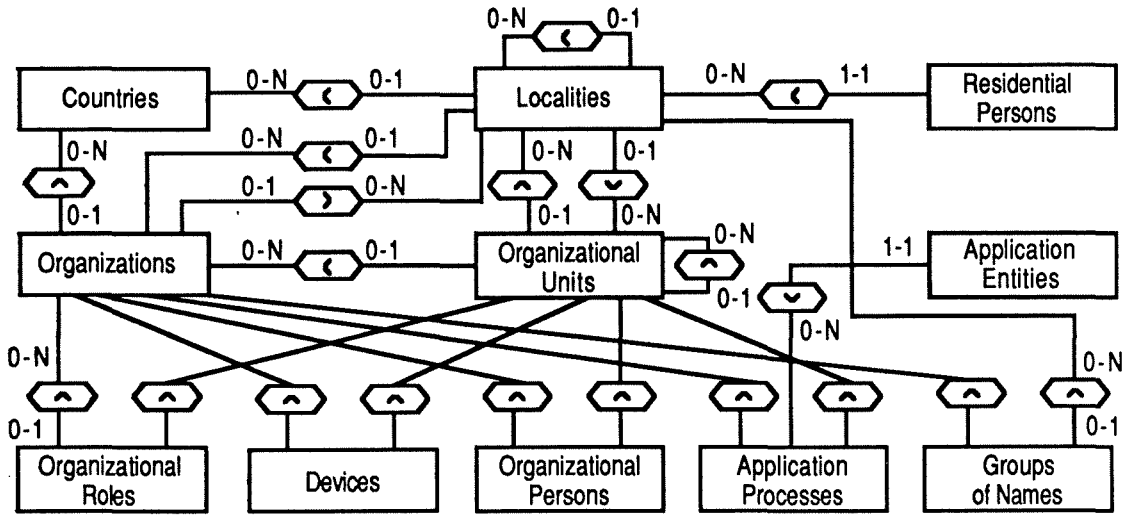


Figure 9.4 : E/R Diagram of the Global Conceptual Schema.

Each entity type covers the set of properties which are defined in X.500 as attribute types of an object class. As an example, the entity type 'Organizational Person' contains the properties 'Common Name', 'Locality', 'Telephone Number', 'Title', etc... Because there are only hierarchical associations among entity types, the mapping of the DIT onto an Entity/Relationship diagram is straightforward. What has to be specified in this case is only the following: every object must have one and only one superior entry, and objects of the class 'Countries' are mapped onto direct subnodes of the root. The same may be done with objects of the class 'Organizations' and 'Localities', whereas this mapping is (in contrast to the mapping of 'Countries') not obligatory. As the diagram shows, objects of the class 'Organizations' may also depend on objects of the class 'Countries' or 'Localities', and can therefore also be mapped onto according subnodes. Whilst objects of the class 'Localities' can also be mapped onto subnodes representing objects of the classes 'Localities', 'Organizational Units', 'Organizations' and 'Countries'.

9.3. The Local Level Schemata.

It is assumed that DSAs access local data by means of local database systems which are not exclusively used for the Directory Service. Here is an example: a company manages information about its organizational structure, employees, etc... In this case, the Local Conceptual Schema (LCS) defines the conceptual schema of the local database of the company. The Local External Schema of the DSA (LES_{DSA}) defines the data view offered to the Directory Service, while the Local External Schemata of the Users (LES_{U}) represent local user views which are not dependent of the Directory, and used for local purposes. This implies that data manipulation and maintenance could be done exclusively via these later interfaces, and not by means of the Directory Service protocols. I.e. the LES_{DSA} could prevent any update of the information held in the common database (common to the Directory DSA and the organization) to be performed via the Directory Service protocols.

This thus allow the local part of the DIB held by the DSA to be a restricted view of the existing database of the organization. There is a single database, used both for the Directory Service and for the own purposes of the organization. Access to the database information of the organization by the Directory Service is controlled by the LES_{DSA}.

No problems arise in the case where the LCS and the GCS are identical, i.e. when all information specified in the Directory Schema is stored in the local database, and just this information. In this case, the structure of the information managed by the DSA and the local database system is identical to the GCS given in figure 9.4, and corresponds to the Directory Schema. The LES_{DSA} is trivial: it can be reduced to the LCS (except if access control has to be specified).

In a more complex situation, certain entity types or properties of entity types defined in the GCS of the Directory may be missing in the LCS, i.e. the Directory defines information which is not managed by the local DB. This applies in the example where the local database holds no information about 'Residential Persons', or 'E-Mail Addresses', and consequently, the DSA cannot provide these informations for such entities or properties. When mandatory attribute types defined in the GCS of the Directory are missing in the LCS, they can be simulated by the DSA by providing default values. Entries are never mandatory.

On the other hand, the LCS may be more detailed than the basic GCS (i.e. the one defined by the X.500 standard Directory Service). It is the case if the local database of an organization manages information about 'Buildings', or if the LCS contains associations not present in the GCS of the Directory.

This can be managed by defining additional local Directory definitions (defining new object classes, DIT structure definitions, ...), and extending thus the GCS locally. But it could also be decided not to offer any access to these informations by the Directory Service, this being performed and controlled by a LES_{DSA} restricting the view of the LCS for the DSA to, for example, what is defined in the basic GCS of the Directory. A solution being a compromise between these two approaches is obviously possible, i.e. giving access to some information not defined in the basic GCS of the Directory Service, and prohibiting access to some other information held in the LCS.

9.4. Mapping.

The question is how to map a Local Conceptual Schema onto data structures supported by the X.500 Directory Service. It is proposed that entities are mapped onto object entries of the DIB, and that associations between entities are represented by hierarchical references between entries of the DIT. As an example, let's consider the conceptual schema of a database system which holds information about members of an organization, organizational units, buildings, etc... as shown in figure 9.5.

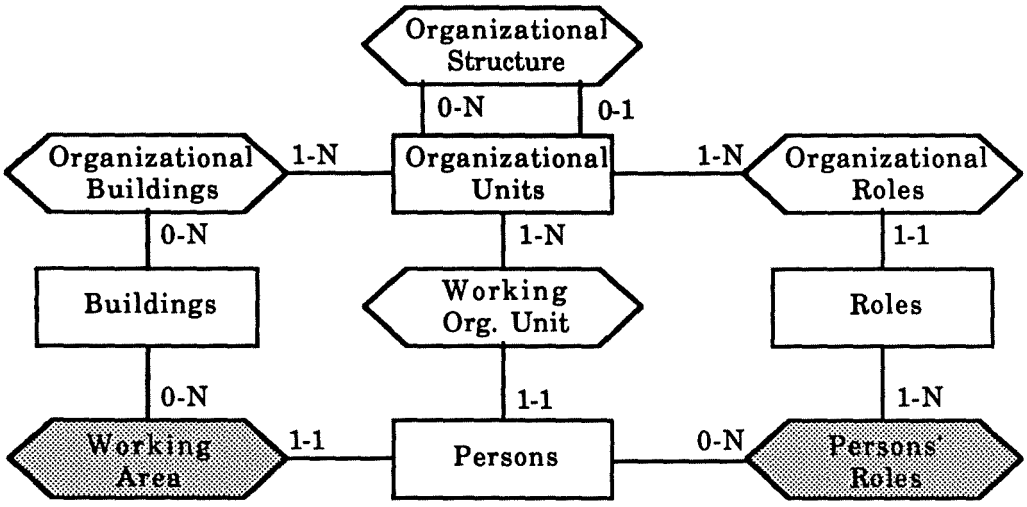


Figure 9.5 : Conceptual Schema of a DataBase.

If we do not care about the shaded associations, the mapping is trivial: we resolve the recursive association of the entity type 'Organizational-Unit' and get a hierarchical structure whose mapping onto the DIT is straightforward (under the assumption that all entities are known as object classes in the Directory definitions).

The mapping of the shaded associations is somewhat more complex, but there is a relatively elegant way to realize it: X.500 allows alias entries in the DIB, which do not hold information about objects but point to other objects of the DIB.

Assuming the figure 9.5 represents the conceptual schema of a database system, figure 9.6 shows a possible corresponding part of the resulting DIT.

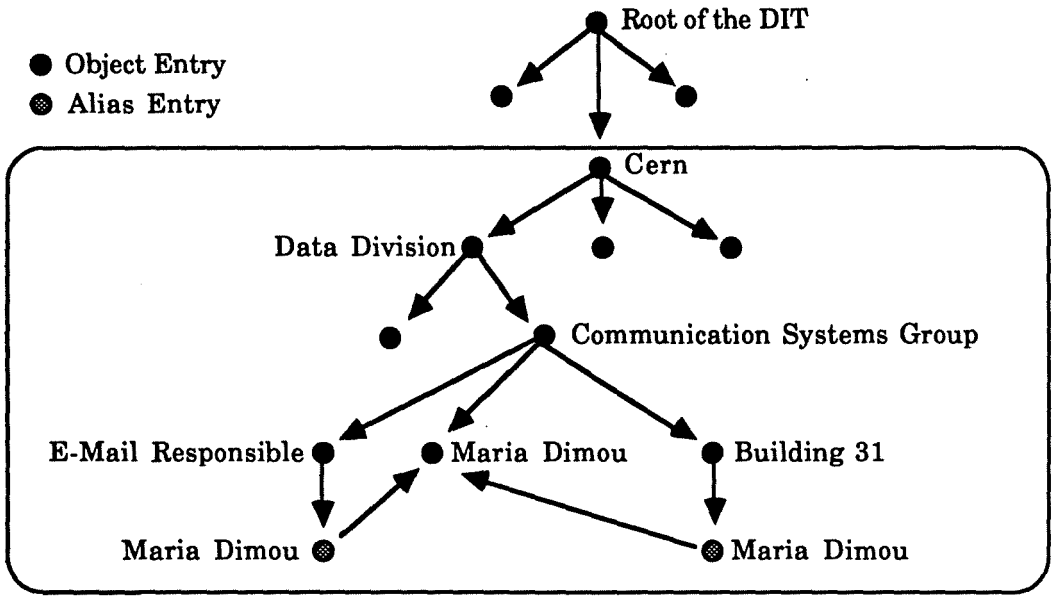


Figure 9.6 : Part of the DIT for the Schema of figure 11.5.

9.5. MetaInformation.

MetaInformation is the information about the structure of a database and express the 'semantic' of the information held in this structure. In the Directory Service database system, MetaInformation consists of the Directory schema and its definitions.

Access to the MetaInformation of the Directory by users would allow them to know all about the information they can get and could help them to query the Directory in an efficient way, thus often preventing them from starting costly Directory exploration activities.

X.500 does not provide standardization for the management and exchange of MetaInformation. Administration and distribution of this MetaInformation could be done by a service independent of the Directory Service defined in the standards. However, it would be more elegant and advantageous to handle MetaInformation like any other information managed by the Directory Service.

10. The RARE Directory

RARE (Réseaux Associés pour la Recherche Européenne) is the European association for research in networking and telecommunications. RARE has defined its own position concerning Directory needs in [Huit-88]. This chapter aims to present the main points raised in this paper.

10.1. The need for Directory Services.

Three OSI applications are envisaged for early use by RARE: terminal access through X.28 PADs, electronic mail via X.400 and file transfer using FTAM. There is also a critical need for distribution of a wide range of information about projects, facilities and people. The need of a directory service to support the operations of these applications was recognized very early, and RARE set up a working group (RARE-WG3) to provide a solution in both of these areas. The RARE-WG3 now works at the definition of a Directory Service which will be supported by the X.500 standard.

It has been realized that there are two aspects of Directory services, of a rather different nature:

- the first form is generally described as a 'user information service', i.e. services oriented towards interactive access by humans, which will progressively focus their queries and eventually find the relevant information. Terminal access, as provided in France by the MINITEL is adequate.
- the second approach is generally described by the term 'name server'. It aims at providing information in real time to processes. A typical usage is that of a file transfer program, which will use a name server to solve the external name provided by the user and derive from this user friendly name, the address of the remote file store. The important characteristic of this second form is the real time nature. This is usually coupled with queries of a much simpler nature (i.e. direct lookups as opposed to browsing).

RARE-WG3 worked primarily on the first form of directory service, but the need of a real time name server was soon recognized as urgent. The analysis shows that at least two problems have to be solve in order to provide such a service:

- a naming strategy has to be adopted. Names are the keys by which the data will be accessed, and it is important to keep these keys uniform in all the networks unified by RARE;
- a format has to be specified for the data that will be stored in the directory, so that the data can be manipulated by various applications.

As already mentioned, three main OSI applications are envisaged by RARE: terminal access through X.28 PADs, electronic mail via X.400 and file transfer using FTAM. A number of objects are manipulated by these applications, and these are clearly important things to name. These objects include:

- People, who have to be identified for E-Mail addressing, and for other information (e.g. postal address);
- Organizations and components of Organizations;
- Hosts, which are the physical machines that support applications (e.g terminal access);
- Filestores, the virtual file store of FTAM (a logical entity composed of a real file server and of presentation and access programs);
- Projects;
- Electronic Distribution Lists.

Other applications may be developed later, e.g. full screen terminal access or remote job submission, which will have their own specific needs. It is hoped that these needs will naturally fit into the proposed structure.

10.2. Network Addresses.

One of the important requirements of a Directory is that it should hold the information necessary to connect to remote hosts, in particular their network address.

The usage of the term 'network address' is hiding an important difference between the networks that use the standard OSI network service, and those which only use X.25. The OSI network addresses, or NSAP addresses, are independent from the X.121 addressing scheme used in the X.25 Public Data Networks, although X.121 addresses may be used as a subset of the global OSI addressing scheme (see Annex A).

The usage of subnetwork independent network addresses is particularly useful when a machine or a local subnetwork is connected to several networks or with several attachments to the same public network (see section 2.2):

- if the destination is connected to several networks, e.g. satellite and packet switched, the gateway will choose the network onto a particular call should be relayed on the basis of that call 'quality of service' parameters;
- if the regular attachment to the Public Data Network of the destination is out of order, the gateway will be able to derive from the NSAP the X.121 address of a back-up attachment, and can automatically retry the call.

It is thus very sensible to store in the Directory the network addresses (i.e. the NSAP addresses) of the applications entities, and not only their X.121 addresses (plus possibly an information to identify the particular destination entity on the local subnetwork, the X.121 address(es) being the point(s) of attachment of the local subnetwork to which the entity is attached to the PDN). If only the X.121 addresses were stored, the list of all public network attachments to a local subnetwork would have to be stored for every application entity on this local subnetwork. On the contrary, if we register only one NSAP address for each application entity, we only have to be able to associate this list of X.121 addresses to the set of NSAPs allocated to this particular local subnetwork.

So the main concern addressed by RARE here is to be able to derive, from a NSAP address, the X.121 (PDN) address by which this NSAP address can be reached. This X.121 address may either be the destination address, if the entity we want to reach is connected to a Public Data Network, or the address of a gateway relaying information between PDNs and, for example, a Local Area Network. The way by which the gateway interpret the NSAP address in order to get the next (non PDN) subnetwork to reach is not addressed by RARE.

It is the feeling of the WG-3 that the RARE name server should be used for this function, and that a table giving the list of X.121 addresses suitable for reaching a range of NSAP addresses (e.g the range NSAP addresses allocated and used in a Local Area Network) should be maintained and distributed. However, this choice poses two problems:

- few systems already used the NSAP addresses. A frequently encountered alternative is to only use the X.121 address. An even worse alternative is to complete the X.121 address by some addressing information passed in the X.25 'Call User Data Field' (CUDF); this very questionable practice will have to be accommodated, at least for an interim period, by the RARE community;
- the syntax of the NSAP addresses is so compact that it is difficult to derive from an NSAP a key to a X.500 Directory, as explained below.

In the case of the Data Country Code (DCC), we may well find a solution where the IDI '208' (France) is administrated by AFNOR, which only allocates ranges of numbers to group of users, but does not run a Directory Service. Within this hypothetical initial domains:

- the DSP between '000...' and '011...' could be administrated by the ministry of defense;
- the DSP between '012...' and '025...' could be administrated by the ministry of education and research;
- different governments agencies could share the DSP between '026...' and '099...';
- DSP from '100000...' to '899999...' could be allocated to commercial companies on a first come first served basis;
- DSP starting with a 9 could be reserved for future use.

Indeed, in this hypothetical initial domain each branch of the various ministries, universities and private companies runs its own directory service, some of which being interconnected, while some others are strictly secret. One can well imagine that even finding the relevant directory for a particular NSAP in such an initial domain could require an expert system, or would at least quite time consuming.

The problem should not be so severe when the IDP is a subnetwork address, e.g. a X.121, ISDN or Telex address. The idea is that it is easier to derive a Directory key when a subnetwork address is used in the IDP. As a subnetwork address is likely to be particular to a local network, if not to a host, it should be possible then to use the IDP to build up a single component directory name, under which is stored the list of X.121 addresses. For precisely that reason, the use of the X.121 form of IDP has already been recommended in some European networks: if the list cannot be extracted from a directory, the gateway can at least try to set up a call to this X.121 address.

This analysis of the NSAP addresses leads RARE to a rather drastic conclusion: a 'standard' Directory Service is by no way the proper tool for providing the 'network level directory service' that is required if it is intended to route on any type of NSAP (i.e. all the types described in the table of section A.1 in Annex A). If this service is really required, RARE will have to study another solution. In between, both the NSAP and an X.121 address that will permit access from the European X.25 services to a point where the NSAP can be understood have to be stored, or the hypothesis that NSAPs are in line with the ECMA-117 recommendations has to be made.

10.3. Application Entity Titles.

Another information that is available during the set up of associations, e.g. by FTAM or between MTAs, is the Application Entity Title (AET). AETs are carried by the Association Control protocol (ACSE). They are supported to uniquely identify both ends of an association. Identifying a server or an entity may be somewhat redundant with addressing it, except maybe in some configurations where a single process, listening on a single access point, may act as different entities depending on the client request. On the other hand, it is very useful for the servers to get a precise identification of their clients, so that they can perform access control.

A particular problem with AETs is that their format is not specified in the ACSE draft standard, although their semantic appears to be in line with that of Directory Names.

A proposition of the RARE-WG3 was to adopt, within RARE, the RARE Names (which are Directory Names), as specified section 10.4, as value for AETs. Thus, it would be extremely easy both for the caller to use the called AET as a key to the Directory in order to get the value of the requested 'connection data', and for the called server to verify that the actual calling presentation address corresponds to the value stored in the Directory for the calling AET. However, it is not certain that this convention will be adopted. Directory Names are designed for user-friendliness, but are regarded as bulky by the designers of the ACSE protocol. As the first version of the session protocol only allows 512 bytes of connection data, they prefer to use a more compact 'object identifier' syntax (the limitation of the connection data will only be removed in the next version of the session protocol).

Object identifiers are another form of names, designed this time for machine processing rather than user friendliness. They consist of a sequence of numeric strings, each one identifying one level of naming authority. Their syntax is specified in the ISO standard 8824, which also guarantees the uniqueness of object identifiers. The standard also specifies a user friendly display format for these object identifiers.

> E.g.: ISO Member-Body AFNOR(208)INRIA(1035)Sophia-Antipolis(3)Main-MTA(17), which means that the ISO member AFNOR numbered 208 has been delegated a naming space. It has given a unique number to INRIA, which in turns has given some of the addressing space to each of its branches. In each branch, various kinds of objects can be registered in a catalog, like the main MTA of the campus. This results in the compact encoding 202,1035,3,17.

RARE cannot predict now which of the solutions will be chosen. If, contrary to RARE's wishes, the standard or the recommended profiles were to enforce the use of object identifiers for AETs, it would imply two requirements for the RARE Directory systems: one will have to be able to find the object identifier given a Directory Name, and vice versa. Obviously, it will be easy to store the object identifier as attribute of the application entity entries, together with the PSAP address. The reverse path can also be achieved , e.g. by using the aliasing facility provided by X.500: each numeric string could be the label of an aliasing arc. E.g., reading the child number '3' of 'INRIA' would point to 'Sophia-Antipolis'. But the maintenance of this cross references will be cumbersome, and could be summed up as a great waste of efforts.

10.4. The Proposed Interim Name Structure.

The proposed RARE name structure is a subset of the name forms recommended by X.500. The proposed form of names tries to reconcile two constraints:

- it should be in line with the X.500 standard;
- it should be possible to represent any RFC 822 names (the currently most used names) in this structure.

The approach of RARE is to use the X.500 name structure. That is to consider that the name of any object starts with a country name, continues with the name of an organization and an indefinite number of organizational unit names, and finishes with the common name of the object, when indeed that object is neither a country, nor an organization, nor an organizational unit.

However, RARE recognizes three other forms of names that are commonly encountered. Some of the RARE members, e.g CERN and ESA, qualified as international organizations, should not include a final country component. Then, there is the problem of non-RARE members who have their own naming strategy: it would not be reasonable to rename them, and RARE has accepted the fact that in some cases the top of the hierarchy will be a Well-Known Domain (WKD) like 'EDU' or 'CDN' instead of a two letter country name. The last case is the one of network names which will still be used for a limited period of time, although they should be considered as undergoing resorption: 'MCVAX.UUCP' should sooner or later be renamed 'VAX.CWI.NL'. This gives us a fourth form of name, where the top domain is a Domain Under Resorption (DUR).

The RARE recommended name structure can be summarized in the following table.

Object Class	RDN	Root	C.	DUR	WKD	Org.	O.U.
Country	CountryName	X					
DUR	DurName	X					
WKD	WkdName	X					
Organization	OrganizationName	X	X	X	X		
Organizational Unit	OrganizationalUnitName					X	X
Organizational Person	CommonName					X	X
Organizational Role	CommonName					X	X
Group of Names	CommonName					X	X
Application Entity	CommonName					X	X
DSA	CommonName					X	X
Device	CommonName					X	X

Figure 10.1 : RARE recommended DIT Structure Definition.

RARE will use the 'Group of Names' object class for Distribution Lists, the 'Application Entity' object class for Filestores and MTAs, and the 'Organizational Unit' object class for Organizational Units, but also for Hosts and Projects.

10.5. Some Restrictions to the general X.500 name Structure.

The name structure described above makes some restrictions to that recommended in X.500: the geographical name forms are absent, and the syntax of some of the naming attributes is more constrained than in the X.500 specifications. These restrictions are motivated by the desire of homogeneity in the academic networks' naming strategies, which should result in more user friendliness. However, it should be very clear that other X.500 systems, e.g. those provided by the PTTs, will not apply any such restrictions. Hence, the X.500 user interfaces provided to the academic communities shall indeed have the capability to enter any regular form of X.500 names.

The geographical name forms are intended for use by a public service, where a private user could be named by common name, city of residence and country. It can also be used for local companies, named by an organization name, a city and a country. The inclusion of a city component is not a common practice among the RARE members. It is not applicable for the large organizations like INRIA or GMD, which have branches in several cities, and it may lead to confusion when organizations are located in unexpected or not so well known places

In addition to these restrictions, RARE defines a text format for names of entries (applying only to the distinguished values of attributes involved in RDNs) in order to facilitate the parsing of names, and to improve their readability in an international environment. Among other things, RARE limits the set of characters which may be used for the names.

In order to support its particular needs, RARE has defined its own set of object classes. In particular, it defines the WKD and DUR new object classes. For other object classes, it completes the X.500 definitions by adding attribute types (e.g. an O/R Name to an organizational person). It also lists the attribute types defined in the used object classes and defines some new attribute syntaxes.

The Presentation-Address syntax is defined within X.500, but that definition has to be changed to accommodate the needs of RARE mentioned above. The Presentation-Address is defined in X.500 as a sequence of Presentation-Selector, Session-Selector, Transport-Selector and a Network-Address, where all selectors are optional octet strings, whilst the Network-Address is a sequence of octet strings. The only point that has to be changed is the definition of the Network-Address, replacing the sequence of octet strings by a choice between a sequence of octet strings (the NSAP-Add) and a sequence of a numeric string (the X.121 address) and two optional octet strings: the Call User Data Field, and the NSAP-Add.

X.500 PSAP-Add attribute syntax definition:

```
PSAP-Add      = {   P-Selector : optional octet string ;
                    S-Selector : optional octet string ;
                    T-Selector : optional octet string ;
                    N-Add : sequence of octet string ;
                }
```

RARE PSAP-Add attribute syntax definition:

```
PSAP-Add      = {   P-Selector : optional octet string ;
                    S-Selector : optional octet string ;
                    T-Selector : optional octet string ;
                    N-Add = {   NSAP : sequence of octet string }
                          or {  X.121 : sequence of numeric string ;
                                CUDF  : optional octet string ;
                                NSAP : optional octet string }
                }
```

11. Directory Service Implementations

This chapter aims to present some Directory Service implementations. The two first of them are oriented toward X.500, and are in fact two large scale pilot exercises in order to test the X.500 standard. These are QUIPU and THORN . The three next ones are non-X.500 Directory Service systems, but are actually operational. These are the Janet Name Registration Scheme (NRS), the ARPA Domain Name Service and the EARN NetServ.

11.1. QUIPU.

The X.500 standard has a much more complex functionality than existing directory services. Whilst there have been directory services which have tackled some of the problems of scale and distribution, these have used much simpler name structures, and have not provided the powerful searching facilities of the X.500 Directory. At the other end of the spectrum, many existing database systems deal with the searching problems, but this is usually done in a very centralized manner. Thus despite the massive requirement for the OSI Directory, the specification has been made without real practical experience of the problem being tackled. It is therefore important to experiment with the OSI Directory prior to the provision of a large scale service. QUIPU and THORN have been developed to facilitate such experimentation [QUI-88a].

The QUIPU Directory Service was developed under the INCA (Integrated Network Communication Architecture) project, which was an ESPRIT research project, and has subsequently been released in the public domain. It is intended to be a lightweight prototype, which will provide an environment for early experimentation with standardized Directory Services.

The ISODE system has been developed as an openly available implementation of the upper layers of OSI. ISODE was used by the QUIPU system from the start to provide the OSI services required by the OSI Directory. QUIPU is now being issued as a part of ISODE, and ISODE applications will be adapted to use QUIPU for their Directory requirements.

To understand the etymology of QUIPU, it must be remembered that QUIPU was originally developed as part of the INCA project. The Inca of Peru did not have writing. Instead, they stored information on strings, carefully knotted in a specific manner, with colored thread, and attached to a larger rope. These devices were known as 'Quipus'. The Quipu was a key component of the Inca society, as it contained information about property and locations throughout the extensive Inca empire.

Design Principles

The design of QUIPU is simple and flexible. These two characteristics facilitate implementation and allow experimentation with a variety of problems associated with the provision of Directory services, and in particular, with replication.

QUIPU is implemented in the 'C' programming language, to run initially under the UNIX operating system.

An application, which might be a user interface or a process, accesses the Directory through a DUA. In QUIPU, the DUA is provided as a 'C' procedural interface, which is structured in a manner aligned to the X.500 Directory Service (i.e. one routine per Directory operation). This interface gives the application access to the full Directory Service.

The DSA database is directly mapped onto central memory, using 'C' structures which correspond to the form of the DIT. The master data files are read from disk and loaded into memory at machine start time, and updated back to the disk when modify operations occur.

This approach is straightforward to implement and gives high performance for small volumes of data.

The use of a virtual memory and paging algorithms should allow extensions on the volume of data a DSA can hold.

DSA database

The master database for each DSA is held on disk. As the data are only accessed by the DSA at startup, the format of data has been chosen to be convenient from a management standpoint, and are only converted into 'C' structures at load time. A text format has been elected because it can easily be created and inspected without the use of special tools. A text editor is quite suitable.

The DIT hierarchical structure is mapped onto the UNIX file structure (directories in a tree structure). Every Unix directory represents a non-leaf entry. All children entries of an entry are stored in a single file located in this Unix directory. To every non-leaf entry stored in this file, corresponds a Unix sub-directory whose name is the RDN of the entry.

This structure has proved easy to generate and manipulate. It also allows replication of parts of the DIT to be accomplished in a straightforward manner.

Access Control

QUIPU provides for user access control to the information held in the database. The QUIPU Directory handles this by specifying an Access Control List for each entry, and which is stored as an attribute of the entry. The Directory knows about this attribute, and so can make choice based on it. It is defined in a manner which also allows specification of rights to update the Access Control List itself.

Six level of access categories are defined, which are related either to a particular attribute, the entry itself, or the children of an entry. QUIPU allows four possibilities for specifying who is being controlled (identified by a Directory Name) on access to a particular piece of information: the entry itself, a whole subtree, a list of names or public access.

This approach of storing access control information within an entry provides a correct solution for caching (see section 8.3, replication). When caching information, a DSA always choose to read the 'Access Control List' information, in such a way that access control can also be performed on cached information.

Schema

There is a need to understand attribute syntaxes in order to perform correct matching. The QUIPU DSA 'knows' about a limited number of attribute syntaxes. Any other structures are mapped to one of the known syntaxes.

QUIPU understands object classes. An entry may belong to one or more object classes. All object classes must be known to the QUIPU DSA. The definitions of the object classes allow a DSA to derive a list of mandatory and optional attributes for a given entry. Later versions of QUIPU will access this information from the Directory (instead from a local table at the moment).

A mechanism for controlling and obtaining information about the structure of the tree has been set up. QUIPU defines a 'Tree Structure List' attribute which specifies the potential type of child entries for a particular entry. The object classes of entries below a particular node entry must be listed in the 'Tree Structure List' attribute of that entry, and the Directory which knows about this attribute, ensures consistency. This mechanism allows the Directory manager to control the shape of the tree, and the user to determine the potential shape of the tree by reading this attribute.

Distributed operations

QUIPU provides for Directory Access Protocol interworking with other X.500 systems. There is not any Directory System Protocol interworking with other DSAs at the moment (i.e. a DSA acts as a DUA to access another DSA). It is intended that in the next versions of QUIPU, the DSP between QUIPU DSAs, and as far as practical, interworking with other Directory systems will be achieved.

Note that in QUIPU, chaining is performed using the DAP, and that DUAs do not support the referral mechanism.

Other characteristics

QUIPU provides for a large set of searching and matching facilities. Among other things, phonetic searching is implemented.

QUIPU is not intended for very large scale systems nor high performances. No data back-up technique is provided, nor security features as digital signature or strong authentification.

11.2. THORN.

The THORN project (THe Obviously Required Nameserver) is an ESPRIT project to develop a Directory Service aligned onto the emerging standards.

The aim is to develop a pre-competitive implementation of a Directory Service, with the intent that this work shall contribute to the development of commercial products. It should also offer feedback of the experience gained to the international standard definition activity thanks to a real usage of Directory services.

The first phase of the project has produced an implementation according to the ECMA TR/32 Directory specification, with simple replication facilities.

The second phase has two components:
- The Large Scale Pilot eXercice (LSPX). This utilizes the system produced in the first phase of the project, and aims to gain real experience with usage of Directory Services;
- The implementation of a new system conforming to X.500.

Future versions of THORN should include the full implementation of X.500.

Design issues

THORN is being implemented in the 'C' programming language, under the UNIX operating system.

THORN has developed a DUA and a DSA.

A DUA library is available which offers to upper layer applications the access to the Directory Service. The DSA supports both the DAP and the DSP.

THORN includes a proprietary special purpose database. The database is based on a hashed access method which virtually guarantees two disk accesses per record only. This database puts no constraints on the size and number of the attributes of a Directory object.

The THORN design has been driven towards obtaining a high performance Directory. DSA processes are not spawned when an incoming call arrives, but are configured in a static way. A spawn-on-request configuration can however easily be obtained if the use of the Directory Service is very limited and no requirements exist on response time.

Due to lack of documentation about THORN, design and implementation issues are not further discussed here.

THORN features

THORN supports the concept of a Directory schema which defines allowed object classes, attribute types and name structures.

THORN also supports access control and knowledge representation. Partial replication of the global Directory Information Base is an extension to X.500.

Although the project is not primarily considering user interfaces, a number of interfaces have been developed to explore the problems of accessing the Directory Service.

In particular, THORN contains an end-user interface that allows to have a friendly interactive access to the Directory system. This interface includes facilities for local caching of information and nicknames. It allows to browse, list and search the Directory base, and makes an intelligent use of the Directory schema for presenting data in a friendly way, and for requiring minimal information from the user to perform a query.

THORN has also been experimented with access by processes.

Interactive management utilities are available to perform basic Directory data management functions such as creating, modifying and removing entries, and administering passwords and access control. Other utilities offer tools to manage the schema and the knowledge information.

Due again to lack of documentation about THORN, for which the second phase (full X.500 implementation) is currently under way, it is not possible to give further details on the characteristics of THORN.

11.3. The Janet NRS.

This section starts the description of non-X.500 Directory Services.

The Joint Academic NETwork (JANET - UK) uses a centralized Directory System, known as the Name Registration Scheme (NRS).

The NRS is a database of mappings. The mappings relate 'user-friendly' names to addressing information to be used to access the service named. Mappings go from name to addressing information and vice-versa [NRS-88].

Names

Names follow a hierarchical structure. Domains are managed by 'institutions'. A name is constructed from a succession of fields separated by the character ".". There are usually two forms of names: the required standard form, and the optional abbreviated form.

E.g.: Standard form : UK.AC.OXFORD.PHYSICS.VAX1
Abbreviated form : UK.AC.OX.PH.V1

Information update and distribution

There is a single central database. The database is held in a set of files by institution. Each institution has the responsibility of keeping the information in its corresponding files correct and up to date. The files located on the central NRS service machine can be accessed by the institutions for update by remote login on this machine or using a file transfer protocol to pull information for the institution back to a local machine, to carry out updates there, and to transfer the modified data back to the central machine again. Institutions must thus supply the information needed to translate a name of a communication entity by which it is known for computer access on the network into its address or addresses.

Every night, a derived database file is produced from the files held in the central machine. This file is made available for transfer to other sites. Remote sites access the central machine to get a copy of the file and format it in a suitable format for local use. It is this formatted copy of the master file, held locally, which is used to provide the Directory Service. The file transfer can be done automatically on a periodic basis. If so requested, such transfers may be issued by the central NRS machine.

A NRS Lookup Protocol (NLP) permits small machines which have not enough space capacity to hold the whole file to keep a local cache of commonly used names, and when asked for one information not in cache, to make a call to a suitable NLP responder which holds the whole file.

Mappings

There are many different kinds of mappings between names and addressing information held in the NRS database. One major grouping of these mappings is according to their direction.

The 'forward' mappings take a name as argument and return the addressing information. It is used by a caller wishing to make a network connection to a called service. Note that several possible sets of addressing information can be returned if the target service has more than one connection to a network, or is connected to more than one network.

The 'reverse' mappings are used by a called service to derive, from the addressing information of the caller provided by the underlying network service, the caller's NRS name (if it is registered in the NRS database).

It is possible to check if a name is registered in the database but without requiring the associated addressing information, or to obtain a description of the service associated with a name in a form suitable for human consumption.

The addresses are used by the network users to access services for different types of networking activities. Some examples are File Transfer, Job Transfer and Manipulation, Electronic Mail and Terminal Access.

Characteristics

The characteristics of the NRS are:
- centralized and widely replicated Directory System;
- hierarchical domain scheme;
- no access control facilities (all information is public).

The usage of the NRS is free of charge. The NRS provides software for both local and remote access to the directories (although a particular institution may implement its own method of interrogating information).

11.4. The ARPA Domain Name Service.

A dedicated name service has been specified for the ARPA Internet: the ARPA Domain Name Service. It runs on a widely heterogeneous collection of machines running a variety of operating systems.

The Domain Name Service is intended to help users (Clients) to locate servers for common directory service, and to locate objects managed by such servers.

Names

The domains are hierarchically structured and typically reflect an administrative or geographical grouping. An administrative authority controls which names are introduced within a domain.

Names follow a hierarchical structure. A name is constructed from a succession of fields separated by the character ".".

Information distribution and update

The name service functions are divided between two classes of servers: 'resolvers' (similar to DUAs), and 'name server' (similar to DSAs).

Clients make requests to resolvers, which in turn make requests to the name servers.

Typically, a name server will not query another name server in order to resolve a name. Instead, it will instruct the resolver which name server, if any, to query next.

Every domain or subdomain has the responsibility for maintaining up to date one or more 'master files', which are the basis for name resolution. Every master file should have at least one copy held by a name server. Several name servers may acquire copies of the same master file. A name server has the responsibility for acquiring parts or all of the master files on which its service is based.

Modifications in the master files are not echoed in copies. A name server periodically checks to make sure that its data are up to date, and if not, obtains a new copy of updated data from master files stored locally or in another name server.

A second kind of data is cached data which is acquired by a local resolver. This data may be incomplete but improves the performance of the retrieval process when non-local data is repeatedly accessed. Cached data is eventually discarded by a time-out mechanism.

Functions

There are three functions offered by a name server:
- name to resource information mapping;
- resource information to name mapping (note that this mapping does not guarantee uniqueness or completeness);
- name completion, which allows to ask a name server to complete a partial domain name and returns a set of names which are 'close' to the partial input.

Characteristics

The characteristics of the Domain Name Service are:
- distributed directory system with replication facilities;
- hierarchical domain scheme;
- no access control facilities.

[RFC-883] is the reference paper which discusses the Domain Name Service. It specifies the format of transactions between resolvers and name servers, and suggests some implementation issues.

11.5. The EARN NetServ.

NetServ is a network server for the academic networks EARN, BITNET and NETNORTH [NETS-88].

In NetServ, a distributed server concept has been realized, which provides for balanced load of the network, faster response time to user requests and better availability of the server. This is achieved by the installation of many servers in the network so that each user finds one nearby (usually only a few number of hops via intermediate nodes). All servers communicate with each other to distribute updated information and make it available in each installation.

In general, one NetServ is installed in each country. But for some countries, it was necessary to install more than one server due to the size of the network inside the country. And for a few countries, it was not feasible to have their own server. People of these countries use the services of a NetServ in a nearby country.

There are three classes of services: File Server, Node Management and User Directory Service.

File Server

The File Server service provides a repository of files (information files and programs). One can obtain the list of the available files, retrieve a file (transfer a copy of a file on your own machine), and store a file in a NetServ. A facility exists which allows you to be informed when a particular file is updated in a NetServ. Another function sends automatically updated files when they are changed.

Typically, the way NetServs communicate with each other to distribute updated information is performed thanks to this last function.

Except if your NetServ maintains lists of files available in other NetServs, there is no way to know which files can be obtained on another NetServ, nor easy way to locate the particular NetServ where a given file is located.

Node Management

Node management is the administration of an individual node with regard to the network, as well as the administration of information and the coordination of changes for all nodes in a network.

NetServ provides support for both aspects of node management by:
- providing data files and programs for node administration and network coordination (available thanks to the File Server service of NetServ);
- supporting registration of new nodes and changes in the network.

User Directory Service

A User Directory Service is available by which network users can create and update their own entry in a directory, containing their name, user identifier, node identifier, mailing address, phone, profession, etc... This Directory can then be searched by all users to find communication partners.

The User Directory information is not replicated. NetServ provide no 'hints' to locate a particular user's entry. You have to query the proper NetServ in order to get your information.

Access to NetServ

NetServs may be accessed in two ways:
- by sending an 'interactive' message to a NetServ in which the message text is a NetServ command (e.g.: Tell NetServ AT NetServNode CommandLine);
- by sending a file whose first record contains the NetServ command, the next records optionally containing data.

Characteristics

The characteristics of NetServ are:
- distributed Directory system with replication facilities;
- simple domain scheme;
- no access control facilities.

It has to be noted that the NetServ service is not suitable for a real time usage, and does not provide for address resolution mechanisms to be compared with the previous described Directory Services. Therefore, if should not be categorized with the X.500, NRS or the ARPA Domain Scheme Service Directory systems.

Part 4 Directory Services at CERN

Cern has begun to develop its own set of directory services on computer systems far before the X.500 standard was published.

Although directory services are offered or being developed in other fields at Cern (as the electronic phonebook, or the X.29 directory), these last chapters will be devoted to the Cern Electronic-Mail directory services.

There are currently two major services available at Cern.

The first (EmNodes) is about nodes, and allows to retrieve information about E-Mail nodes and hosts.

The second (EmDir) is about E-Mail users in the High Energy Physics community (HEP community) and essentially allows to retrieve information about users' E-Mail addresses.

These two services have been recently upgraded, and a new one (Automatic Router) has been set up, based on EmDir, which allows to use more user-friendly addresses.

12. EMNODES

12.1. The database and the provided service.

The EmNodes project is based on the NetNodes DataBase (DB). NetNodes is an Oracle DB currently installed on CERNVM (IBM/XA).

This DB is composed of a single table containing the following information about E-Mail nodes of the EARN, BITNET, NETNORTH, JANET, UUCP and DECNET networks: node name, network name, site, address, country, contact person, operating system, network software. These fields are not all fulfilled for each node. But the node name, network name, site, address and country information is at least present for all nodes.

The DB contains at the moment information for about 5000 nodes. It is not an exhaustive list. The following nodes are described in the DB:
- all EARN, BITNET and NETNORTH nodes;
- European UUCP nodes only;
- JANET nodes accessible from Cern only;
- Cern DECNET nodes only;
- all Cern nodes.

The DB information for EARN, BITNET, NETNORTH and JANET nodes is picked up from routing tables in backbone MTAs or Cern gateways, and updated on a periodic basis. Information about Cern DECNET nodes and other Cern nodes is derived from a file manually updated by the Cern gateway managers.

The DB content is updated by DB loaders, i.e. programs reading routing table files and updating the DB.

The DB was accessible in two ways:
- the usual SQL language interface (select node, network from netnodes where site like '%Namur%'), rather designed for expert users or DB management;
- an SQL-Forms program called NetNodes. SQL-Forms is an Oracle DB interface manager which is powerful but not easy to use properly by novice users.

The DB was mainly used for the gateway management, in order to have information about nodes, to look whether a given institute has an E-Mail node or not, to derive E-Mail addresses, and so on...

It was also used by the User Consultancy Office (UCO: the Cern computer center user service) to solve E-Mail addresses problems. But quite often, they were so reluctant to use the NetNodes program that they preferred to phone to the E-Mail responsibles to get the needed information.

12.2. Definition of the new service.

The objective was to create a more user-friendly interface for this database, which could easily be used by a large set of users, and to offer facilities to query the DB on selection items as the town or the institute where the node is located (these informations being scattered in the site and address fields).

Moreover, the program should compute an E-Mail address proposition for the selected nodes from the information contained in the database, as it does not explicitly contain this information.

The program, which will firstly have to run on the IBM, should easily be ported onto other machines (PCs, VAXes, WorkStations, ...), because this service could be made available to the large community of Cern E-Mail users.

The program has to allow to save results on file and to send them by mail. The latter facility is mainly designed for UCO people, who have to be able to send query results rather than transmit them orally by phone.

12.3. Technical analysis and adopted solution.

Programming Language

The first considered solution was to improve the current SQL-Forms interface, in better using its capabilities. But SQL-Forms proved not to be flexible enough, and we could not have obtained a really user-friendly interface.

This is an important point, because this application will be used occasionally by users. So, the interface has to be extremely simple, close from what they already know, in order not to impose them a specific learning for a rarely used tool.

Besides, the address computing would have imposed 'User's Exits' (routine calls written in a traditional language), as SQL-Forms is not a programming language. But on one hand, these 'User's Exits' do not allow to receive results from the called user exits, and on the other hand, the only accepted languages for these 'User's Exits' were C, Cobol and Fortran.

Cobol and Fortran were rejected because a change in the routines would have involved problems as very few people at Cern are using those languages. C was excluded because Oracle is written in C but compiled with an Oracle specific compiler for IBM machines. The linkage conventions of this compiler are incompatible with the C compilers used at Cern on the same machine. And the program had to run in priority on the IBM machine.

The advantage of SQL-Forms was that it is portable on most of the Cern systems.

The second solution was to write a program in a traditional language, while using the SQL data manipulation language to access the DB. The program has to be precompiled in order to translate the SQL-Statements in source code. C, Fortran and Cobol were excluded for the reasons mentioned here above, and there were no SQL precompiler available at Cern for other languages (as Pascal).

The third type of solution was to write the program in a traditional language, while calling directly the source database manipulation routines. Again these routines were only available in C and Fortran, and this solution had to be given up.

The fourth and adopted solution has been to write a Pascal program, with a compiler allowing to perform calls to routines defined in Fortran, in such a way that we can use the Fortran database manipulation routines. But we had to take care of the way to call these routines, because Pascal and Fortran types are not directly compatible. The interface realized for this purpose is very simple, as the application only manipulates strings. Pascal is also a relatively portable language.

Level of Service

The next problem to solve was to define which level of service we wanted to offer, and the resources Cern was ready to allocate in order to provide this service. The human resource is very critical in the group responsible for the project.

The main problem was due to the DB content. We wanted to permit a selection on the node, network, country, town and institute names.

There were no problem for the node and network selection items, as this information is explicitly contained in the DB.

For the country, the DB is only containing the two letter ISO country code. We have chosen to accept only this two letter country code, and not the full country names, in English or in French. This would have imposed a conversion table, which would had to be fulfilled and updated manually. Moreover, these two letter codes are more and more used, and an exhaustive list of the codes with the corresponding country names in English, will be available from the program.

For the town and institute names, the problem was a little bit more complex. These informations are contained somewhere in the site and/or address fields of the DB, and quite often in the local language ('Brussels' has been found in English, Dutch and French !). For institutes, names are sometimes complete, incomplete, abbreviated or just the official initial letters.

To create a new specific DB containing appropriate clean data for this project, in which we would have tables for all possible synonyms for the town and the institute names would have been too heavy a burden. Such a DB would have had to be filled manually from the information currently contained in NetNodes, and a permanent maintenance would have been necessary. This could have been reduced thanks to update programs, but they should have been written. Anyway, the initial burden was too heavy. This solution had then to be rejected.

We have decided to do simple pattern matching in the site and address fields to find the node and institute names given as selection items. The help file gives some hints for more efficiency. We have thus given up to offer a reliable and efficient service. But the program will be relatively simple, and the project will not involve further maintenance resources than the one required at the moment for the maintenance of the NetNodes DB.

E-Mail Address

An other problem was to know if we were going to add an E-Mail address field in the DB, or if the address had to be computed thanks to the information retrieved from the DB after the query.

Adding a new field in the DB was not very difficult: loader programs had just to be changed. But the NetNodes DB and loaders are not under the responsibility of the group involved in the EmNodes project, and political reasons prevent us to intervene in other groups' activities.

This solution had the advantage of being more efficient if the address computing had to be changed due to changes in the routing table files formats. Maintenance would only to be done for loader programs, and not also for the EmNodes program which, in the adopted solution, computes the address. Besides, it could have been interesting for other applications using the NetNodes DB to have the node address in it.

The imposed solution was to compute the address after the query. But this involves to modify the address computing routine, recompile and reinstall the program every time a modification in the address computing is needed.

Note that the address provided by the program is an address valid when sending mail messages from Cern, and not from everywhere in the world. This dependency is due to the fact that some systems require that routing information to be explicitly indicated in the address, which makes addresses dependent from the originator location (e.g.: in Uucp, Host1!Host2!Host3!UserId like addresses). Most of addresses used at Cern are free from routing information, except when messages send from the DecNet mail system have to pass through a gateway, i.e. send to another non-DecNet host.

User Interface

A last problem was the full-screen terminal mode on IBM machines. Terminals are running the block-mode, i.e the screen is managed by the terminal, and its content is only send to the central machine when the 'return' or 'pf' keys are pressed. The appropriate action are then taken according to the content of the screen, or to the modifications with the last screen sent.

Besides, with a normal program on this kind of machine, only one line on the screen can be used to enter data, which is, after having pressed 'return', displayed on the current line on the screen. It is also only possible to display data on the screen line per line, and not wherever you want on the screen. And there is no automatic scrolling: the user has to press on a key to see the next screen. These constraints were not aiding to design a pleasant user-interface.

On the other hand, the program has to be portable. It has thus been decided to entirely pull apart the user interface and the DB access modules, in such a way that a specific interface for each kind of machine can be designed, being close from the usual user interface philosophy of the machine, taking advantage of their own possibilities, and dealing with their specific problems. On the other side, a rough user interface will be written in Pascal to be able to easily install the program on other machines. So the DB access module can be separately tested on a machine before developing a specific user interface for this machine.

The DB access module just receives the search criteria entered by the user from the user interface. It does some checking and processing on these criteria before accessing the DB in order to improve efficiency and reliability of the retrieval. After having accessed the DB, it computes the E-Mail addresses of the retrieved nodes, and paginate the results. A file is given back to the user interface.

The user interface is responsible for receiving search criteria from the user and for displaying of the resulting file.

Implementation

As previously stated, the access module to the database has been written in Pascal. To be portable, it will only be necessary to redefine some predefined routines specific to the compiler. Moreover, it will maybe be necessary to change the database manipulation routines if the compiler do not accept Fortran routines, or if these routines are not available in the Oracle library of this machine.

A first program has been written in Pascal (DB access module and user interface). It is running properly, but the interface is not quite pleasant.

A special interface for the IBM machine has thus been written. This program is written in REXX, the IBM system language. It uses IOS3270, a screen manager utility, which allows to have a very pleasant user interface. To display the results of queries, the usual XEDIT editor specially profiled for this application is used. This program uses the DB access module written in Pascal.

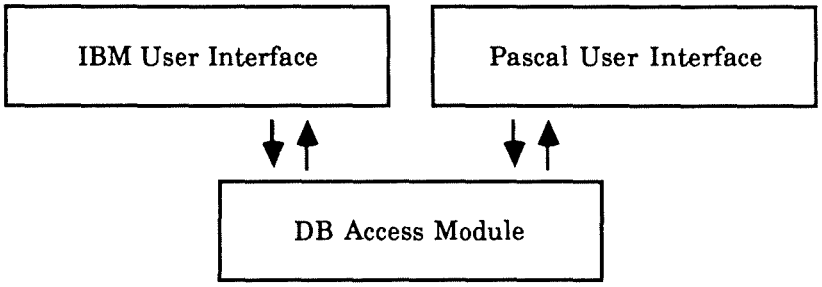


Figure 12.1 : The EmNodes Architecture

It is worth noting that the access module is not completely independent from the interface, and this for performance reasons:

- The DB access module for the IBM specific interface had to be implemented as a separate program. So the database opening and closure are executed at each call of this module, e.g. at each query. These operations are really time consuming. For the program completely written in Pascal, the DB access module is implemented as a procedure and the database opening is done at the beginning and the closure at the end of the program. Successive queries are thus quite faster.
- For the Pascal program, the DB access module is implemented as a procedure, and the selection items could be shared and accessed in global variables. Whilst for the DB access program on the IBM, the selections items had to be passed via a communication file, what is far less efficient.

But apart this, the code is common for the two access modules. And the specific parts to an application are isolated in a file.

All the specific features linked to the compiler are explained and gathered in a file.

A maintenance documentation and program have been written. The maintenance program is a REXX program which, from the source files, builds libraries, compiles, loads and generates executable modules.

12.4. The future of EmNodes.

After a two month testing period, the program has underwent minor changes, mainly to ensure a more efficient pattern matching, and to control and manage DB access errors. Some changes due to operating system upgrade had also later to be performed in the user interface (IOS3270 was not further fully compatible).

Five months after the definition phase, the program was installed on public disks, and has been made available to all IBM users. The feedback has been very positive, and it has been asked to install the program on other machines, for people not having access to the IBM machines. But lack of human resources prevents us to further develop the project.

Another service which could be provided, is a 'remote DB consultation' possibility for every E-Mail user in the world. The principle is the following: a user wishing to query the DB sends an E-Mail message to a Cern mailbox. The message describes a request in a special purpose syntax. Messages are received by a program, analyzed, the query is performed and the results are sent back to the source address.

If the service is expected to be largely extended to other machines, it could proved more efficient to have a solution close to the EmDir implementation. In EmDir, the DB access module is only located on one machine and is accessed from all other machines through Remote Procedure Calls (RPCs). As there is at the moment no RPC system on the IBM machine, this kind of solution for EmNodes could be implemented in three ways:

- to install a RPC system on IBM and using it normally;
- to move the DB from the IBM to the VAX which supports a RPC system;
- to use the VAX RPC system, and the routines on the VAX access the DB on IBM thanks to SQLNet.

The first two solutions are probably very expensive and long to develop. In particular, the second solution requires the loader programs to be adapted. The third solution is a compromise, but SQLNet has to be reliable.

13. The EMDIR NameServer

13.1. The database and the provided service.

The E-Mail EmDir NameServer is based on the EmDir database. EmDir is an Oracle DB currently installed on VXCERN (VAX/VMS).

This DB is an E-Mail addresses directory for and about CERN users and the HEP community. It is composed of a single table containing the following fields: name, firstname, division, experiment, internal CERN phone number, institute, institute phone number, E-Mail address and comment.

Additional hidden fields are a password (for update control), and other identifiers used for DB management.

The DB contains information about more or less 7000 people. Most of entries are incomplete or mention unreliable information because information relative to a person is freely fulfilled and updated by this person. But few people know this service or use it. So, the interest for maintaining a reliable information in the directory is limited.

The database, is accessible in two ways:
- by using the EmDir program, which allows users to query the DB and to update their own entry.
- for Bitnet users, by using the tell command (query mode only; e.g.: tell EmDir at CernVM query Dimou Maria).

People who whish to have their entry updated but having no access to the EmDir program can send a message to a special purpose mailbox regularly opened to manually satisfy entry update requests.

The DB is mainly used by E-Mail users to find the (preferred) E-Mail address of somebody, if any. Most of the time, E-Mail users have several mailboxes, either on the same account (e.g.: UUCP mail and EAN on a Unix machine), either on different accounts. Even if they set an auto-forward in every non-used mailbox toward the preferred mailbox, it is more efficient to send directly the message to this mailbox.

The EmDir program is a C program running on CERNVM (IBM/XA), VXCERN (VAX/VMS), CERNVAX (VAX/UNIX) and several kinds of workstations. Exactly the same code runs on the different machines, the source code being tailored by compiler directives when the code has to vary according to the machine. Remote DB access on VXCERN is done thanks to Remote Procedure Calls (RPCs).

13.2. Definition of the new service.

The NameServer project is an extension to the EmDir service. The new service has to allow any E-Mail user to query the DB by mean of E-Mail messages. So the service could potentially be used by every E-Mail user in the world.

A message with a special purpose syntax describing a query to the DB is sent to the NameServer mailbox. The NameServer is a process which receives a message, analyzes it to detect the queries, accesses the DB to satisfy the request and sends back the results to the 'from' address of the incoming message.

The service should be flexible enough to accept several queries in the same message, and to detect queries which do not exactly correspond to the syntax which is defined in the help file. Moreover, two syntaxes have to be accepted: one in the EmDir style (query Dimou Maria ...), and the one defined by the 'M&D MHS Service' (Find Dimou:CERN).

The service should easily be extended to other syntaxes (e.g. close to Directory Names or X.400 O/R addresses). The service should not fail. The service has to detect help requests and to provide a Help service by sending back the Help file.

The service has to trace its activity, i.e. a log file has to be appended with the received messages, the originator of these messages and the performed requests.

13.3. Technical analysis and adopted solution.

Remote Procedure Calls

The NameServer has to run on CERNVAX, a Unix machine. As at Cern Oracle is not yet available under Unix, the only way to access the DB is to use RPCs.

An application using RPCs can roughly be divided in two parts:
- the client program, using RPCs, and
- the client/server which execute RPCs.

The client is the interface between the client program and the remote routines installed in the server. The client communicates with the server through the network. The server is the process which runs on the remote machine, listening to the requests of the client, and executing RPCs.

A problem to consider was whether a new set of RPCs had to be written, i.e. a special purpose client/server for the NameServer application, or if the existing one for the EmDir program would be used.

The existing set was able to satisfy the needs of the NameServer, although in an inefficient way (explained below). As the DB was not reliable, as the DB structure could change in a near future because of a Cern reorganization, and as the RPC system was not yet in a stable state (upgrades were foreseen), it has been decided not to invest in the design and implementation of a new RPC set.

In a first step, the service will be offered as a prototype. If the NameServer is used enough to justify an optimization of the program, more efficient access methods to the DB will be studied.

Message Syntax

The accepted syntaxes for the incoming messages (which specify therein a request to the NameServer) had to be chosen. Two syntaxes have been considered: the one defined by the 'M&D MHS Service' (find <name>:<organization>, where <..> indicate parameters) and a syntax close to the one used for the EmDir program (query <name> [<firstname> [<division> [<experiment> [<institute> [<E-Mail address>]]]]] , where [...] indicate optional parameters). These syntaxes have been extended in order to be more flexible. In particular, in our 'M&D MHS service' syntax, <name> may denote a name or a firstname, and <organization> a division, an experiment or an institute. They allow more than one query per message, and requests may be in the subject field of the message, the body, or both.

Implementation

To analyze the incoming messages, the LEX lexical analyzer is used. A file describes patterns to detect and actions to take when these patterns are detected. The file is precompiled, and a C program is generated: the message analyzer.

The message analyzer analyzes a message and produces a diagnostic file. This diagnostic file contains the syntax used, the parameters of the request or the help request, the address where the results or help file have to be sent back, or if it must be refused to answer to this message (messages sent by daemons, returned mails, ...).

A C program using RPCs performs a request, and produces a resulting file in a defined syntax. Another LEX program consumes this file to paginate the results.

A Unix Shell script (the NameServer) uses all these programs. It first runs the message analyzer, analyses the produced diagnostic file to decide the appropriate actions.

It sends the help file if the help service has been asked or if no valid query was detected. It performs requests and pagination while managing error messages if valid requests have been detected. It sends then back the resulting file, and appends a log file. It does nothing special if it must be refused to answer to the message, except appending the log file.

Figure 13.1 shows the architecture of the NameServer.

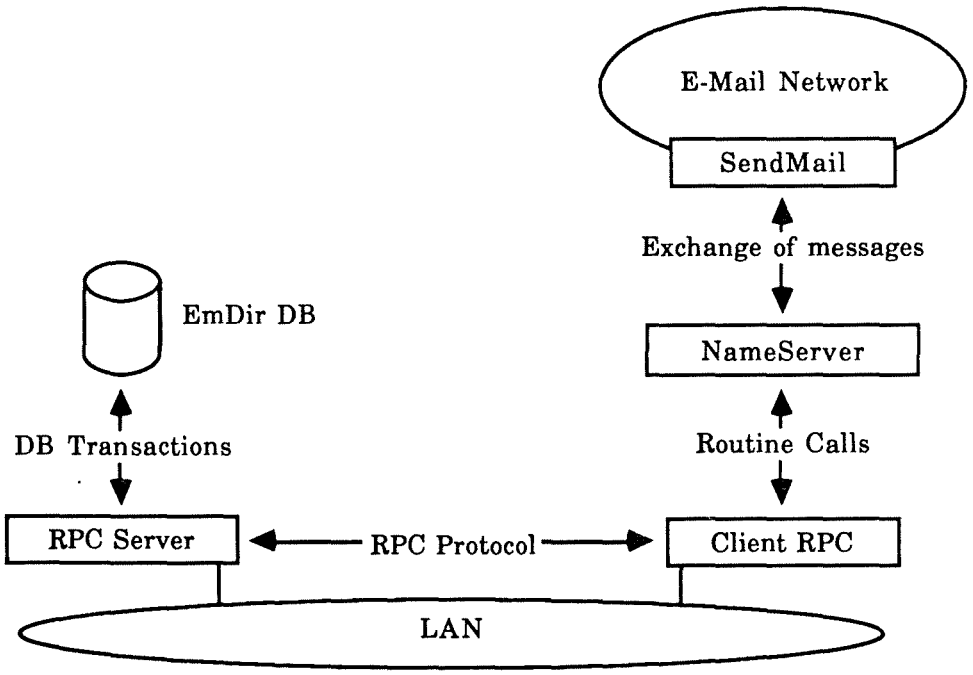


Figure 13.1 : The NameServer Architecture.

Evaluation

The solution is not quite efficient, but has the advantage of being simple and modular.

In particular, the set of RPCs allows only to query the DB on an 'and' composition of selection criteria, while the NameServer application requires queries on an 'and/or' composition of selection criteria (when the 'M&D MHS service' syntax is used). In order to offer proper functionalities, the NameServer performs 6 RPCs per 'Find' query detected in the message, among which three or four are almost always useless. And it is impossible to determine beforehand which ones will fail. The program thus uses a lot of DB and network resources.

Another problem was to detect the availability of the remote RPC server. The permanent process which runs the server must be killed during the Oracle back-up (3-4 hours a night), and must be restarted automatically when Oracle is available again. The client must also face network failure.

When the NameServer was first developed, the RPC system was managing such situations by aborting the client program. To detect such problems, the NameServer was calling the querying program with a dummy request. If at the end of the execution, no result file was produced, it was concluded that the server was not reachable at the moment. The NameServer then sends an error return code to 'SendMail' (the MTA process which calls the NameServer) on termination. This error code is such that 'SendMail' will submit the message again to the NameServer a few time later.

But this dummy request also consumed needlessly resources. The problem has been submitted to the RPC system manager, and the system has been changed in such a way that the RPC system sends back an error code rather than aborting when the client is not reachable.

The NameServer was then integrated to 'SendMail' on a WorkStation to be tested. It has later been installed on Cernvax, the VAX/Unix machine which runs one of the Cern E-Mail gateways.

A short documentation and a 'makefile' have been written for maintenance purposes.

13.4. The future of the NameServer.

After a short testing period, the code of the NameServer has deeply been changed to improve efficiency, code clarity and modularity.

This program was a prototype aiming to show the feasibility of such a NameServer. Future developments could be done in order to improve code efficiency, and in particular, to study other ways to access the DB (new RPC set, SQLNet, ...).

Another service related to the NameServer would be to offer remote update facilities using the same principle, i.e. by sending an E-Mail message to the NameServer specifying an update request.

It could also be desirable to accept query syntaxes which look like X.500 Directory Names or X.400 O/R addresses (e.g.: Search <FN=Maria; OU=DD; C=CH>).

14. AutoRouter

14.1. Definition of the new service.

This project is related to E-Mail addressing for people registered in the EmDir DB.

The usual Cern E-Mail address syntax is 'user@host.domain', where the host part denotes the name of a Cern E-Mail host, the user part is the user identifier on this machine, and the domain part is an E-Mail network name (e.g. Uucp or Bitnet) or the string 'cern.ch'.

Such addresses are not quite user-friendly, mainly because one cannot easily remember or guess them (how do you guess the machine on which somebody has an account, and his or her user identifier on this machine). The aim of the AutoRouter is to provide an addressing scheme close to the Cern internal organization, and allowing to use the real person's name and firstname in the user part. The scheme is based on the RFC 987 address format.

The defined address syntax is:

E-Mail-Address	::=	User'@'Domain
User	::=	[Firstname'.'][Initial'.'][5][Name]
Domain	::=	[Experiment'.'][Division'.'['cern.ch'] |
		[Division'.'][Experiment'.'['cern.ch']

Where :
- [...] denotes an optional string;
- Firstname is a string of at least two characters denoting a firstname;
- Initial is a single character denoting an initial (5 initials maximum);
- Name denotes a name;
- Division denotes a Cern organizational division;
- Experiment denotes a Cern organizational experiment;

When addressing somebody at Cern, one may use its personal knowledge of the addressee, rather than some machine dependent information.

14.2. Technical analysis and adopted solution.

Principle of the Solution

The principle of the solution is to have a program (the AutoRouter) which receives all messages addressed with such addresses, queries the EmDir DB to get the preferred E-Mail address of the addressee, and forwards the message to this address.

From non-Cern MTAs, routing is decided upon the upper part of the domain part of the address (i.e. 'cern.ch'). When the message arrives in a Cern MTA, the address is further analyzed and routed to the AutoRouter when a division or experiment name is detected as first Cern subdomain.

The AutoRouter is a program running on the Vax/Unix Cernvax MTA. It receives the messages from 'SendMail' (the MTA process) on its standard input, and the recipient and originator addresses on its command line. When receiving a message, it analyses the address and performs a request to the EmDir DB using the RPC system described in the previous chapter. If one and only one entry matches the request and if a valid E-Mail address is specified in the E-Mail address field of this entry, the message is forwarded to this address using SendMail. If no entry or more than one entry match, or if no valid E-Mail address is specified in the matching entry, the message is sent back to the originator with appropriate comments and error messages.

The AutoRouter traces its activity and appends a log file. When a message is successfully forwarded, a line is added to the message header to indicate that it has been routed according to the content of the EmDir DB.

Implementation

The AutoRouter is a short C program. A solution similar to the one adopted for the NameServer has been used to detect and manage unreachability of the RPC server.

Figure 14.1 shows the architecture of the AutoRouter, very close to the one of the NameServer.

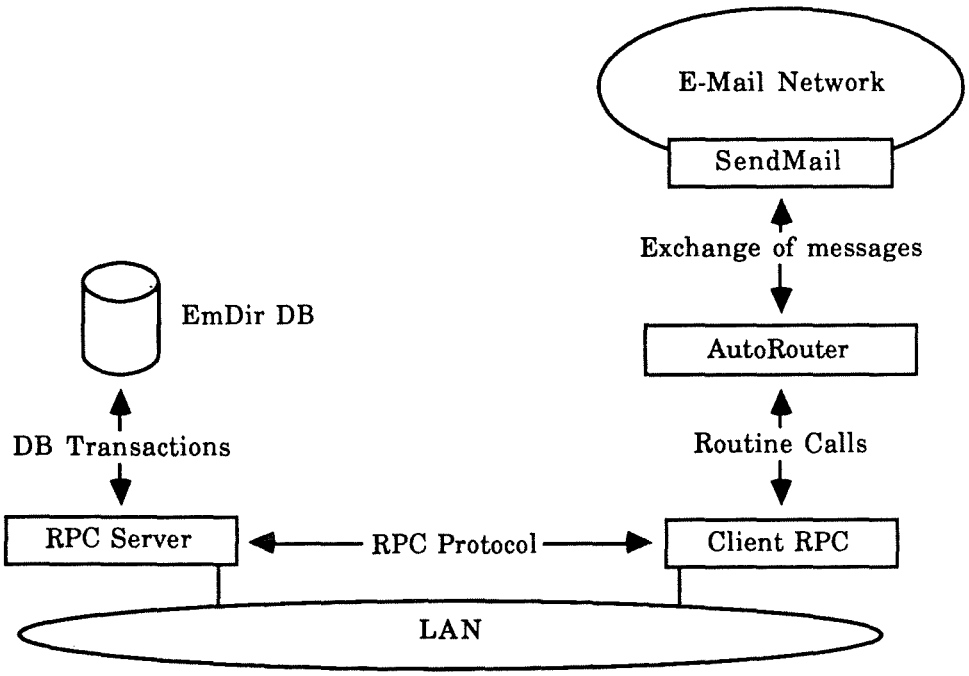


Figure 14.1 : The AutoRouter Architecture.

The AutoRouter has been integrated to SendMail and successfully tested.

This program is a prototype aiming to show the feasibility of such a router. The service is at the moment limited to a reduced set of recipients.

14.3. EmDir issues.

It is felt that EmDir could take a more and more important role in the future for some E-Mail functions as Directory Services or routing. And to offer reliable NameServer and AutoRouter services, it is absolutely necessary to have reliable and complete information in the EmDir DB.

Therefore, an important effort has been decided in order to define new requirements for the DB, and for the ways to query it.

Some Decisions

Decisions were made in the following fields:

- Definition a new RPC set allowing a large but restricted set of SQL queries to the DB;
- Provision of information about the origin of the information contained in an EmDir entry, and an information allowing to determine whether the person described in an entry has an account on one of the Cern machines;
- Reliability of the information, achieved by regularly cross-checking the EmDir content with the 'Personnel' and 'Centre de Calcul' DBs, and by a validity control on update of information by users;
- Common file list of Cern divisions and experiments for validity control and routing.

These decisions have impacts on both the NameServer and AutoRouter applications. For the NameServer, the first decision should allow more efficiency of the program (although this is not an important requirement as it is not an on-line service). But mainly, it will have an impact on the consumption of DB and network resources. Reliability of data is an important requirement for a NameServer service.

For the AutoRouter, the new RPC set should allow more flexibility on the selection of DB entries, and requests closer to the address syntax and content. Information about the origin is essential: it will allow to work up a routing policy. Cern wants to be able to select people who will benefit this service (e.g.: people having an account on one of the Cern machines, or people who belong to the HEP community) in order to avoid abuses of the Cern E-Mail facilities. Reliability should allow to successfully route messages in most of the cases, and toward valid addresses. The use of a common file list for divisions and experiments will avoid differences between information stored in EmDir (valid division and experiment names) and information used to route messages toward the AutoRouter.

Security and Reliability

The use of data stored in EmDir for automatic routing requires a high level of security and reliability for the information stored in the DB. The security aspect is particularly important here. A person wishing to misappropriate messages addressed to somebody else can do it by replacing the aimed person's address by its own address in the corresponding EmDir entry. EmDir update protection based on a password is not considered as being secure enough for some very sensitive Cern people.

As far as the reliability of the information in EmDir is concerned, a first step is done by regularly updating entries according to the contents of the 'Personnel' and 'Centre de Calcul' DBs. For E-Mail addresses, a verification routine has been written.

This routine aims three objectives:

- to transform an address in an Uucp (host1!...!hostn!user), Decnet (host::user), 'user@host', 'user AT host' (and so on...) syntax in a 'user@host.domain' syntax, while accepting Janet addresses (even inversed) and more hierarchical addresses. X.400 addresses are not yet accepted. This aims to have a single address syntax in EmDir. Address having invalid syntaxes are rejected;
- if the user does not specify any domain, to try to find it by a lookup in the NetNodes DB, and if no valid domain has been derived either from the address syntax, either from the query to NetNodes, to reject the address;
- to check if the specified address is not an address which would route the message toward the AutoRouter again, which would induce message looping. This control is performed by checking if a Cern subdomain is specified (if not, the address is rejected) and if this subdomain is listed in the file of divisions and experiments (in which case the address is also rejected).

In this way, all possible local check is done, and a certain number of invalid addresses are detected, mainly due to syntactical errors.

This routine will be used when an update of an E-Mail address in EmDir is performed. But as there is invalid E-Mail addresses in EmDir at the moment, the AutoRouter also uses this routine to check whether an address is valid or not before forwarding the message, or sending it back with an error message to the originator.

Matching Emdir entries and E-Mail Addresses

An important issue for automatic routing is to perform the mapping between the defined addressing scheme and the content of the EmDir entries to find the corresponding recipient's E-Mail address, if any. Problems arise when people have composed names or firstnames (with or without dashes, as Jean-Luc or John Peter), when the preferred well-known firstname is not the first listed in the official forenames, when initials are used in the DB firstname field (real initials or initials of complementary forenames), when the words 'de', 'd'', 'van', 'van den', 'von', 'di', ... are used before a name and the fact that E-Mail addresses do not accept blanks in it (as a consequence, names are quite often concatenated, or blanks are replaced by dashes or underscores), for married women known by their husband's name and/or by their own native name, ...

A flexible syntax had thus to be defined. This syntax has not to be too much constraining for the format and order of data stored in EmDir. It attempts to harmonize the new addressing scheme and the content of the name and firstname fields of the DB in order to achieve an efficient matching algorithm. As the address verification routine, this syntax should guide the EmDir managers in such a way that the content of the DB will conform to our matching rules, and is thus a guide to design the EmDir user interface.

14.4. The future of the AutoRouter.

As previously stated, this program is only a prototype. In order to be fully operational, the program and the protection of EmDir should be further improved and tested. Mainly for security and efficiency reasons.

In particular, the existing mapping algorithm performs simple exact pattern matching between the different elements of the address and the name, firstname, division and experiment fields of the DB. A more intelligent matching algorithm should be designed in order to deal with the difficulties quoted here above.

Before allowing a large usage of the new addressing scheme, it should be proceeded to a burden testing. All messages using this new addressing scheme will converge toward the AutoRouter and its associated MTA. The AutoRouter also consumes DB and LAN resources on very loaded machines and media. Tests should be performed in order to determine how the global system will react, in particular during peaks of machine and network load.

Another useful improvement of the AutoRouter service would be the possibility of sending E-Mail messages to any Cern people, even those having no E-Mail account. As the name, firstname, and division are the required informations for the Cern internal 'Paper-Mail' system to send a document to a person working at Cern, as this information is registered in the EmDir DB, and as every people working at Cern is registered in the EmDir DB, it is easy to imagine the AutoRouter working in the following way: when receiving a message, the AutoRouter analyses the address and queries EmDir. If only one entry is retrieved and an E-Mail address is specified in it, the message is forwarded to this E-Mail address. If no address is specified, it is checked whether the person corresponding to the entry is working at Cern thanks to the indication about the origin of the information stored in the entry. If it is the case, the message is printed on paper with a header page indicating the name, firstname and division of the person to whom the message has to be delivered to, and is introduced in the Cern internal 'Paper-Mail' system. In every other case, an error message is sent back to the originator.

Conclusions

Directory needs arise mainly for four reasons:

- user-friendly naming facilities of objects for human users;
- facilities for searching and browsing a large and distributed database of information about people, services and facilities, and in particular to provide a mean to get the necessary information to communicate with these people, facilities and service providers. This way of using the Directory is rather designed for human use;
- name to address mapping functions. This need arises from the wish to isolate users (either human users or processes) from changes in the location of objects;
- routing functions. Requirements for routing functions arise both in the higher level communication systems as the Mail Handling System, and in the lower level of the OSI network layer.

The X.500 Directory Service system allows for an access to information based on a user-friendly, hierarchically structured key (Directory Names). It is designed to support a large and distributed database. In addition to key access, facilities are provided to browse, search and update information in the Directory Information Base. These characteristics of the X.500 Directory make it meet the two above quoted requirements.

To provide a name to address mapping function thanks to the X.500 Directory Service is straightforward if Directory Names are used as names: the address or addresses associated to a name are stored in the entry representing the named object.

However, the use of the 'bulky' Directory Names (rather designed for human use) as names may not be suitable for efficient machine processing or for use in some applications or communication protocols.

If there is no need of user-friendly naming facilities, and that hierarchical naming is appropriate, then the use of some form of Directory Names (and hence of the X.500 Directory Service) may be acceptable to implement a compact form of names (as Object Identifiers described in 10.3).

Otherwise, if a flat naming scheme is adopted, or if one wishes to have both a user-friendly naming scheme and a naming scheme suitable for machine processing, then the X.500 Directory Name structure is not convenient.

In fact, a major drawback of the X.500 Directory Service is its lack of an efficient mechanism to manage multiple key access, and the absence of mechanism for the use of non-hierarchical key to access information.

The alias mechanism provides a mean to build a simple multiple key access mechanism based on hierarchical keys. It is thus possible to use several naming schemes (e.g. one user-friendly, and a more compact one rather designed for machine processing), although there is no facilities in the Directory Service to help to manage cross-references in the Directory Information Base.

There is no facilities in X.500 for a key access based on non-hierarchical keys. This makes the address to name mapping functions (to know to what corresponds a particular address) impracticable in a number of cases (e.g. if Ethernet addresses are considered), or for some form of NSAP to SNPA address mapping functions (as discussed in 10.2).

The Directory may be unsuitable to provide a global and direct support for a majority of routing functions. On the one side, the routing decision is dependent on both the current location and the destination of the information being routed. This makes a routing function specific to a switching point in a network. But the retrieval of information held in the Directory is not dependent on the location nor the identity of the inquirer.

On the other side, the routing decisions are quite often based on the address where the information has to be forwarded, this address being used as argument of the routing function. The use of a flat addressing scheme leads to a compact but non-hierarchical access key for a routing table, which is held in some kind of directory. These considerations makes the Directory inapt to provide a global and direct support for some kind of routing functions.

If a hierarchical addressing scheme is used (as in the X.400 Mail Handling System), then the hierarchical structure of addresses may be used to build a routing function based on the Directory. A sub-structure of the Directory Information Tree may be mapped on the hierarchical structure of addresses (i.e. to every top-domain corresponds a first-level entry, and to every subdomain corresponds a directly subordinate entry). Every entry represents a domain or a subdomain and is associated with a number of switching points representing entry points in this domain or subdomain, which are reachable from outside the domain or subdomain and have to be used to enter the domain. When having to take a routing decision, a switching point derives from the address a key to the Directory which corresponds to a domain or subdomain name. With this name as key, it then accesses the Directory to get the addresses of the entry points to this domain or subdomain, and sends the information to one of these entry points. Access control in the Directory may be used to restrict the use of entry points to a subdomain by foreign switching points ('by hiding entry points'), thus obliging them to access this subdomain via entry points of a superior domain or subdomain. Such kind of routing mechanism requires a rather good connectivity between switching points and entry points.

Otherwise, the Directory may be used as support to build routing tables. E.g. maps of the network topology may be stored and distributed thanks to the Directory. Local switching points could access these maps to derive the locally needed routing information in order to build routing tables.

The X.500 Directory as a database system has to prove its performance, i.e. its capacity to provide fast responses to queries. It is felt that the performance of the Directory will be an important requirement if it is intended to be used by processes. One can doubt of acceptable response times by the Directory in its actual definition, as a Directory User Agent has first to set up a connection with a Directory Service Agent, and the Directory Service Agent has then to search in a database. The situation is even worst when Directory Service Agents start chaining, possibly establishing connections between each other to process a request.

It should maybe desirable to provide, in addition to the current service, a faster, transaction oriented querying mechanism (just a query and a response, opposed to the current session oriented service).

On the other side, taking into account the performance issue and the existence of the current databases and database management systems, it should be discussed whether special purpose databases and database management systems for the Directory have to be designed, or if tools should be developed to interface the Directory with these existing databases and database management systems.

X.500 itself has shortcomings which should be removed in the next standard release: provision for a standardized way to exchange the Directory Schema (i.e. mainly the Directory Information Tree structure definitions and the object classes definitions) and the knowledge information should be considered. Decisions should also be made to facilitate the management of the Directory distribution and replication, and to harmonize the access control mechanisms.

Annexes

Annex A The Network Address Format

This annex summarizes the main points developed in [ECMA-117].

A.1. ISO 8348/DAD 2.

This clause presents the major points contained in ISO 8348/DAD 2 which defines the nature and form of NSAP addresses.

The standard assigns several properties to NSAP addresses amongst which are [WG3-88]:

- it must be globally unique;
- it can not be used for routing by the network service user;
- it should be used by the network service provider in particular for routing through interconnected subnetworks;
- it should be constructed to facilitate use by network service provider.

Network addresses are defined to be hierarchical. An authority may either assign a complete address, or else may identify a subdomain of its own addressing domain within which addresses may be further assigned by an identified authority for the subdomain. This is done in such a way that all addresses are unique. When an authority identifies a subdomain, this creates in effect a prefix which applies to all addresses assigned within the subdomain.

Certain methods of assigning authority are recognized within the body of ISO 8348/DAD 2. These take account of existing addressing standards such as CCITT Rec. X.121. The abstract structure of a NSAP address is shown here above:

Authority & Format Identifier (AFI)	Initial Domain Identifier (IDI)	Domain Specific Part (DSP)

An NSAP address is represented as a string of either decimal or binary digits, composed of three fields. Together, the AFI and the IDI form the Initial Domain Part (IDP). The content of the IDP is entirely standardized. The AFI indicates both the type of the IDI, and the format (decimal or binary) of the DSP. It also identifies the authority responsible for allocating IDI values. The IDI identifies the subdomain from which DSP values are allocated, and the authority responsible for allocating these values [Huit-88].

The total length of the NSAP address is at most 40 digits, i.e. up to 20 octets. As the AFI is represented in two digits, the maximum length of the DSP will vary with the type of the IDI.

IDI formats specified in ISO 8348/DAD 2 include a number of authorities for IDI allocation. The following table gives a list of possible IDIs, their length, the value of the AFI, and the corresponding syntax and maximum length of the DSP [Huit-88].

IDI type	IDI length	AFI value	DSP length
Data Network Address (X.121) (Public Data Network Numbering)	14	36 37	24 digits 9 octets
Data Country Code (ISO-DCC) (Geographic Address Assignment)	3	38 39	35 digits 14 octets
International Organizations (ISO-ICD) (Non-Geographic Address Assignment)	4	46 47	34 digits 13 octets
Telex Address (F.69) (Telex Numbering)	8	40 41	30 digits 12 octets
Telephone Number (E.163) (Telephone Numbering)	12	42 43	26 digits 10 octets
ISDN address (E.164) (ISDN Numbering)	15	44 45	23 digits 9 octets
Local (The IDI is null and the entire address is contained in the DSP)	0	48 49 50	38 digits 15 octets 19 characters

The structure of the DSP is not specified in ISO 8348/DAD 2. It is normally only known within the domain identified by the IDP, except for the format (binary or decimal) indicated by the AFI.

A.2. Topological example.

As an example, the IDI and DSP may be illustrated in term of an addressing topography comprising a central domain, representing a global public network, surrounded by many satellite domains (e.g.: Local Area Networks).

The AFI identifies the global network to which the IDI is relevant. The IDI identifies the point of attachment of the satellite network to the global network, i.e. a relay. The DSP identifies a particular host within the satellite network.

The following example shows an NSAP address composed of an X.121 address for the global network (X.25 Public Data Network), and a Ethernet address for the satellite network. This address could correspond to the topology shown in figure A.1, when Host 1 wants to address Host 2.

AFI	IDI	DSP
37	(X.121 address)	(Ethernet address)

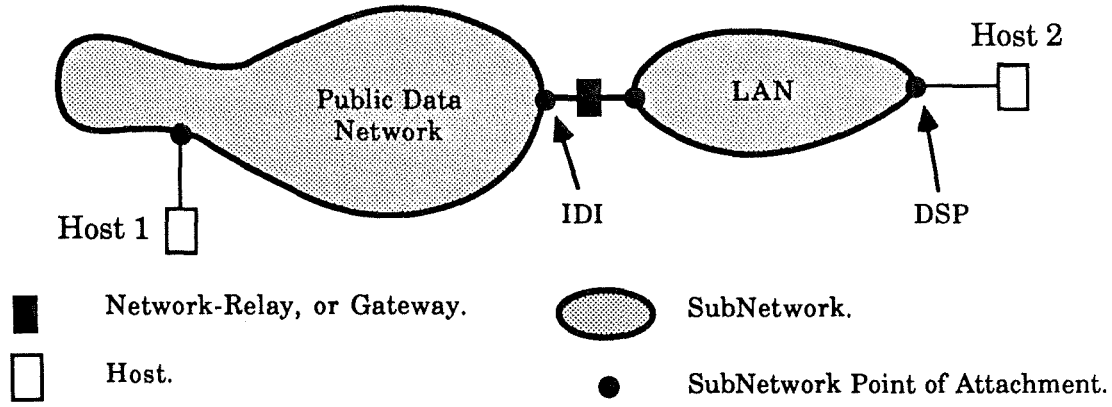


Figure A.1 : Topological Example.

If the destination and originating hosts are both connected to the global network, the DSP is null. If the originating host on a particular satellite network has to address another host on the same satellite network, the IDI is null, and the AFI has the 'local' value corresponding to the DSP syntax.

The private satellite domain may itself be complex. It is not restricted, for example, to a single LAN. It may consist of several LANs interconnected, or even comprise an entire global private network. The complexity is only limited by the addressing capability restricted by the maximum DSP length in ISO 8348/DAD 2.

It should be noted that although an IDP may correspond to a particular point of attachment to a public network, it is not constrained to do so. A private domain may have several points of attachment to the public network. The IDP could thus be a logical point of attachment which corresponds to a set of physical SubNetwork Points of Attachment (SNPAs), and which has to be mapped into a SNPA address before proceeding to the connection.

Equally, private domains may exist which are not attached to public networks at all.

A.3. ECMA-117.

The example and the following considerations are the base of the ECMA-117 standard.

The DSP should satisfy the following requirements [ECMA-117]:
- It must enable multiple NSAPs to be separately identified in a single computer system which is directly attached to a public network;
- It must enable multiple computer systems attached to the private domain to be identified;
- It must enable multiple NSAPs to be separately identified in a computer system which is attached to a private domain.

ECMA-117 further normalize the NSAP address format by defining a DSP structure with three components:

1) Subnetwork identifier.

The internal structure of a private domain may be complex. It may include, for example, several interconnected LANs. To facilitate operation of the internal gateways, it may be necessary for addresses to identify a particular subnetwork within the domain.

The subnetwork identified by the subnetwork identifier may, but is not constrained to, coincide with a physical subnetwork.

The subnetwork identifier may be null, if the topology of the private network is such that routing can be done using only the subnetwork address.

2) Subnetwork address.

The subnetwork address provides the way in which a particular point of attachment to the subnetwork may be addressed. To allow for efficient and easily managed use of local area networks, it must be possible for this to be the real address used by the subnetwork.

Alternatively, the subnetwork address component may not, in fact, contain a real subnetwork address. Synonyms may be used to represent subnetwork points of attachment in some cases, including:

- several SNPAs exist and any one may be used;
- no suitable SNPA exists;
- for some other reasons, a mapping of a logical representation of the subnetwork address to a real subnetwork address is required.

The means by which synonyms are interpreted to find an actual subnetwork address are outside the scope of ECMA-117.

The subnetwork address may be absent in the case of an end-system the subnetwork point of attachment is completely identified by the IDP. In this case, the subnetwork identifier would have the null value.

3) NSAP selector.

This identifies a particular NSAP which can be accessed through the particular subnetwork point of attachment.

The standard permits the DSP to be expressed in either binary or decimal syntax. The particular syntax being used is identified by the AFI value.

The permitted maximum length of the DSP for the various address syntaxes permitted in ISO 8348/DAD 2 are varying according to the AFI. The highest permissible length, common to all formats, are nine octets for binary syntax, and 23 digits for decimal syntax. Therefore, these are the maximum lengths the use of which is described by the standard.

The DSP formats in ECMA-117 are :

Syntax	Subnet. ID.	Subnet. Add.	NSAP-selector
Binary	2 octets	0 - 6 octets	1 octet
Decimal	5 digits	0 - 15 digits	3 digits

It is important to note that routing information can potentially directly be derived from the NSAP-address content.

A.4. The X.121 Addressing Scheme.

The addressing of the network entities on public X.25 networks is the object of the CCITT recommendation X.121. An international numbering scheme provides an internetwork addressing.

Every Public Data Network (PDN) received a Data Network Identifier Code (DNIC), which is a number of 4 digits. The 3 first ones identify a particular country, and the fourth identifies a PDN in this country.

An internetwork address is composed of a DNIC, followed by a Network Terminal Number (NTN) which specifies the address of a Data Terminal Equipment (DTE) in the PDN. The NTN is a number of 1 to 10 digits.

ISO 8348/DAD 2 permits IDIs being X.121 addresses. The X.121 address space allows PDNs to be viewed as a single global network.

A.5. The ISO-DCC Addressing Scheme.

In the ISO Data Country Code (DCC), the country code is a three digits number identifying a country and an authority allocating further numbers to users or group of users.

ISO 8348/DAD 2 permits IDIs to be DCCs, further number are part of the DSP. This scheme considers a country as being a global logical network to which every user is directly connected. DCC-addresses are independent from the particular physical subnetwork to which users are attached, and its particular physical address. When using the DCC, a mapping has to take place in order to get the subnetwork to which the user is attached and the physical subnetwork address of the user's point of attachment (SNPA) on this subnetwork. Such a scheme is said to be subnetwork independent.

Annex B Distributed Operation Procedures

This annex roughly outlines the way distributed operations can be performed as described in [X.518-88]. A detailed description of the procedures is exposed in [X.518-88]. It should be noted that this description is included for expositional purposes only and is not intended to constrain or govern the implementation of an actual DSA.

B.1. DSA Behavior.

Each DSA is equipped with procedures capable of completely fulfilling all Directory operations. In the case that a DSA contains the entire DIB, all operations are in fact completely carried out within that DSA. In the case that the DIB is distributed across multiple DSAs, the completion of a typical operation is fragmented, with just a portion of that operation carried out in each of potentially many cooperating DSAs.

In the distributed environment, the typical DSA sees each operation as a transitory event: the operation is invoked by a DUA or some other DSA. The DSA carries out processing on the object and then directs it toward another DSA for further processing.

An alternate view considers the total processing experienced by an operation during its fulfillment by multiple, cooperating DSAs. This perspective reveals the common processing phases that apply to all operations.

Every Directory operation may be thought of as comprising three phases:
- the 'Name Resolution' phase, in which the name of the object on whose entry a particular operation is to be performed is used to locate the DSA which holds the entry;
- the 'Evaluation' phase, in which the operation specified by a particular Directory request (e.g. Read) is actually performed;
- the 'Results Merging' phase, in which the results of a specified operation are returned to the requesting DUA. If a chaining mode of interaction was chosen, the Result Merging phase may involve several DSAs, each of which chained the original request or sub-request to another DSA during either or both of the preceding phases.

The Name Resolution is the process of sequentially matching each RDN in a name to a vertex of the DIT, beginning logically at the root, and progressing downwards in the DIT. However, because the DIT is distributed between arbitrarily many DSAs, each DSA may only be able to perform a fraction of the Name Resolution process. A given DSA performs its part of the Name Resolution process by traversing its local knowledge. When a DSA reaches the border of its naming context, it will know from the knowledge information contained therein, whether the resolution can be continued by another DSA, or whether the name is erroneous.

In the case of the operations Read, Compare, List, Search and Modify-Entry, name resolution takes place on the object name provided in the argument of the operation. In the case of Add-Entry, Remove-Entry and Modify-RDN, name resolution takes place on the name of the immediately superior object (derived by removing the final RDN from the name provided in the operation argument).

When the Name Resolution phase is completed, the actual operation required (e.g., Read or Search) is performed. It is the Evaluation phase.

Operations that involve a single entry (Read, Compare, Add-Entry, Remove-Entry, Modify-RDN and Modify-Entry) can be carried out entirely within the DSA in which that entry has been located.

Operations that involve multiple entries (List and Search) need to locate subordinates of the target, which may or may not reside in the same DSA. If they do not all reside in the same DSA, operations need to be directed to the DSAs specified in the subordinate references to complete the evaluation process.

The Result Merging phase is entered once some of the results of the evaluation phase are available.

In those cases where the operation affected only a single entry, the result of the operation can simply be returned to the requesting DUA. In those cases where the operation has affected multiple DSAs, results returned by the other DSAs need to be combined with those generated locally to form a consolidated set of results.

The permissible responses returned to a requestor after result merging include:
- a complete result of the operation;
- a result which is not complete because some parts of the DIT remain unexplored (applies to List and Search only). Such a partial result may include continuation references for those parts of the DIT not explored;
- an error (a referral being a special case).

An operation on a particular entry may initially be directed at any DSA in the Directory. That DSA uses its knowledge, possibly in conjunction with other DSAs to process the operation through the three phases.

Each individual DSA performs one or more of the three phases. The collective action of all DSAs produces the full set of services provided to users by the Directory.

B.2. Managing Distributed Operations.

Information is included in the arguments of each operation a DSA is asked to perform indicating the progress of the operation as it traverses a DSA. This makes possible for each DSA to perform the appropriate aspect of the processing required, and to record the completion of that aspect before directing the operation outward further DSAs.

A DSA that receives a request can check the progress of that request using the Operation Progress parameter. This will determine whether the operation is still in the name resolution phase or has reached the evaluation phase, and what portion of the operation the DSA should attempt to satisfy. If the DSA cannot fully satisfy the request it must either pass the operation on to one or more DSAs which can help to fulfill the request (by chaining or multicasting), or return a referral to another DSA, or terminate the request with an error.

Additional procedures are included in the DSA to physically distribute the operations and support other needs arising from their distribution.

Request Decomposition is a process performed internally by a DSA prior to communication with one or more other DSAs. A request is decomposed into several sub-requests in such a way that each of the latter accomplishes a part of the original task. Request decomposition can be used, for example, in the Search operation, after that the base object has been found. After decomposition, each of the sub-requests may then be chained or multicasted to other DSAs to continue the task.

Each DSA that has initiated an operation or propagated an operation to one or more DSAs must keep track of that operation's existence until each of the other DSAs has returned a result or an error, or the operation's maximum time limit has expired. This requirement applies to all operations, propagation modes and processing phases. It ensures the orderly closing down of distributed operations that have been propagated out into the Directory.

B.3. DSA Procedures.

The behavior of the distributed Directory as a whole is the sum of the behavior of its cooperating DSAs. Each of these DSAs can be viewed as a process, supported internally by a set of procedures.

Figure B.1 illustrates the internal view of the DSA behavior.

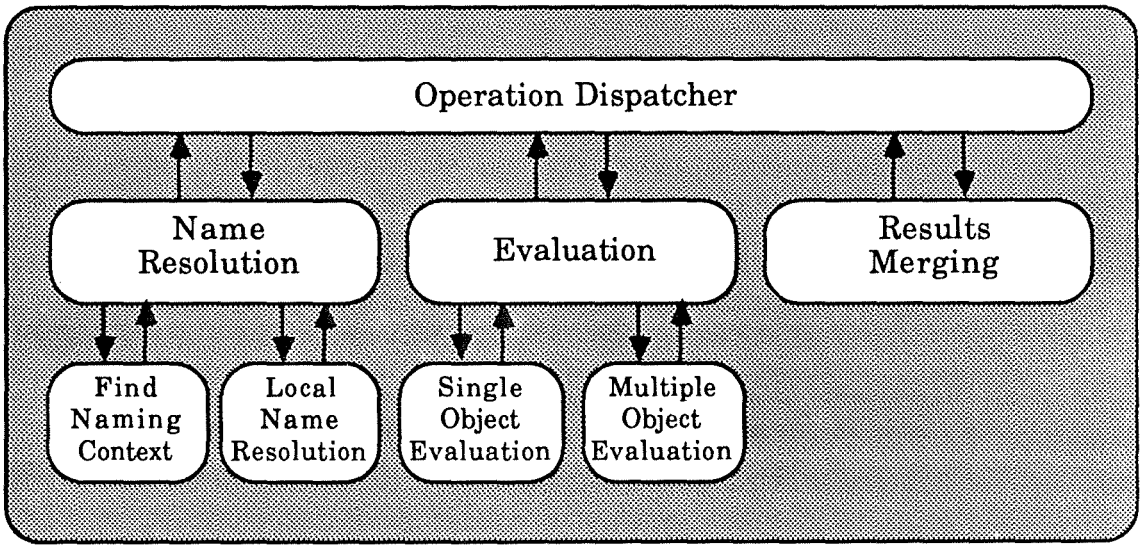


Figure B.1 : DSA Procedures

The Operation Dispatcher

Upon initially receiving an operation, the Operation Dispatcher takes it in charge. It validates it, checking for loops (if it is a distributed operation) thanks to the trace information, time limit or authentification errors. If none is found, it calls Name Resolution, which either returns a Found, a Reference or an error indication. References are handled by a referral or by a chain or multicast action. A found indication is handled by calling the Evaluation procedure, which actually performs the intended operation. Once returned, internal or external results are collated by Results Merging, and, in the absence of errors, returned to the calling DUA or DSA.

Name Resolution

Name Resolution calls Find Naming Context. If the returned context is local, then Local Name Resolution is called, otherwise Name Resolution returns a reference and terminates. If Local Name Resolution encounters an alias, it is dereferenced and Name Resolution repeats the analysis from the beginning. Otherwise, Local Name Resolution returns a Found or a reference, which is passed back to the Operation Dispatcher.

Find Naming Context attempts to match the name on which name resolution takes place against context prefixes of naming contexts held locally. If a context prefix of a naming context held locally matches, an indication that a suitable naming context was found locally is returned. If none matches, then Find Naming Context attempts to identify a cross reference. If a context prefix is matched, Find naming context returns the corresponding cross reference. In the case neither a suitable naming context is found locally, nor a cross reference identified, the superior reference is returned.

The Local Name Resolution procedure attempts to match RDNs in the name on which name resolution takes place against internal references until it can return a Found indication. If unable to match all RDNs internally, it attempts to identify first specific, then non-specific subordinate references, and return these to name resolution. If an alias is encountered, and dereferencing is allowed by the service controls, a dereferenced alias indication is returned. Otherwise a Found indication is returned if and only if all RDNs had matched at the time the alias was encountered.

Evaluation

The Evaluation procedure actually executes the requested Directory operation against the target object. Depending on the type of the operation, Single Object Evaluation or Multiple Object Evaluation is invoked.

Single Object Evaluation is invoked for Read, Compare, Add-Entry, Remove-Entry, Modify-Entry and Modify-RDN operations. It is in this procedure that attributes are actually retrieved, checked or changed.

Multiple Object Evaluation is invoked for the Search and List operations to check filters, retrieve results, and if necessary, dispatch sub-requests.

Results Merging

The Results Merging procedure collates results or errors received from other DSAs in the case of a multicast or a request decomposition. Locally retrieved results are collated in the collation.

B.4. Specific Operations.

The operations fall in three categories (in each case an operation and its distributed counterpart are both in the same category):

- Single-Object operations : Read, Compare, Modify-Entry, Add-Entry, Remove-Entry, Modify-RDN;
- Multiple-Object operations : List and Search;
- Abandon operation : Abandon.

The handling of these categories are described in the following sections. Since there is considerable similarity between the way that a DSA behaves in performing an operation of a service port and in performing its counterpart operation of a distributed service-port, there is a single description applying to both, with exceptions to this rule being noted.

Single-Object Operations

Single-Object operations are those which affect a single entry, and which therefore can be carried out entirely within the DSA which contains the entry on which the operation is to be performed. Such operations can be commonly described by the following sequence of operations:

1- Activate the Operation Dispatcher;

2- Perform Name Resolution to locate the object whose name was specified as the argument of the operation;

3- Perform Single-Object operation Evaluation procedure;

4- Service controls, such as time limit, should be checked during the course of the operation to enforce the constraints specified by the user;

5- Return the results to the DUA or DSA which forwarded the request.

Multiple-Object Operations

Multiple-Object operations are those which affect several entries which may or may not be co-located in the same DSA. Such operations may thus entail a cooperative effort by several DSAs to locate and operate on all the entries affected by the requested operation. The common behavior of such operations can be summarized as follow:

1- Activate the Operation Dispatcher;

2- Perform the Name Resolution procedure to locate the object whose name was specified as the argument of the operation;

3- Once the target object of the operation has been located, perform the Multiple-Object evaluation procedures;

4- If request decomposition has taken place in one of the Multiple-Object evaluation procedures and sub-requests have been chained or multicasted, the Operation Dispatcher maintains the current local results, waits for distributed operations responses, and activates result merger.

5- Service controls, such as time limit, size limit should be checked during the course of the operation to enforce the constraints specified in the common arguments;

6- Return the results or errors to the DUA or DSA which forwarded the request.

Abandon Operation

On receipt of an abandon operation, a DSA determines whether it can abandon the specified operation, and, if so, abandons it and returns a result (the operation that was abandoned returns an 'Abandoned' error). If it cannot abandon the specified operation, it returns an 'Abandoned-Failed' error.

The following specifies the procedure specific to the Abandon operation:

1- Locate the operation whose invoke identifier is specified as the argument of the Abandoned operation;

2- Optionally compose request(s) with the proper invoke identifiers to abandon any outstanding chained or multicasted operations to other DSAs.

3- If 2 succeeds, the abandon operation is performed:

4- Return result or error to the DUA or DSA which forwarded the request.

Annex C EmNodes

Annex C is the Help file of the EmNodes Service described in chapter 12.

The last page is the screen of the user interface, plus the corresponding retrieved results.

```
.cm CAT: CMS
.cm NAM: EMNODES
.cm EXP: Search for mail node addresses.
.cm DAT: 88.12.12
.cm A/R: B.Heuse/M.Dimou
.cm KEY: MAIL ADDRESS NODE HOST DOMAIN E-MAIL EMDIR
.cm ABS:
.cm ABS: EmNodes is an electronic mail directory service, allowing
.cm ABS: mail node addresses to be search for and retrieved.
.cm ABS:
.cm END:
```

EmNodes is an electronic mail directory service, allowing mail node addresses to be searched for and retrieved.

Use the EMNODES command to query the directory. The syntax is:

```
+---------+-------------------------------------------------------------------+
|         |                                                                   |
| EMNODES | [node|* [inst|* [town|* [country|* [network|* ]]]]][ ( fn ft fm ] |
|         |                                                                   |
+---------+-------------------------------------------------------------------+
```

Typing EMNODES without any parameters will start an interactive display.

If parameters are given, the DataBase will be accessed to retrieve any corresponding entries. An asterisk (*) may be used to replace any missing parameter, or as a wild card at either end of, or within, any of the parameters. An asterisk (*) replaces zero, one or more characters, whilst question mark (?) replaces one and only one character.

node — is the name of the node (host) generally forming part of the full node address (e.g. CERNVM, VXCERN, ...).

institute — is the Research institute (or University) to which the node is located.

town — is the town/city where the node is located (see usage note 2).

country — is the 2 letter country code where the node is located (see usage note 1).

network — is the network which this node is on (e.g. EARN, BITNET, NETNORTH, UUCP, JANET, DECNET, CERN. See usage note 4).

fn ft fm — are filename, filetype and filemode of a file for storing the result of the query (if required).

The result of the query is a file on which is written the selection criteria and a number of records. A record contains the following information:

- Country of the node,
- Node name,
- Network name,
- Site and Address,

FILE: EMNODES HELPCMS A1 VM/XA CERN CMS 5.5 v.8

- Example of an Electronic Mail Address.

Usage Notes :

1. Country : Enter ISO two letter Country Codes only. A table is available by pressing PF4 (ISO CC).
2. Town : Some town names are stored in English others are in local language. Several trials in different languages may be necessary to obtain an exhaustive list of what you wish to retrieve. Example: for nodes in Brussels, town may be either 'Bruxelles' (French), 'Brussel' (Dutch) or 'Brussels' (English). For town or city names composed of several parts as 'Ann Arbor', just give the longest part as search criterion, or use '*' as part separator.
3. Institute : If using abbreviations, use '*' between each letter in place of '.'.
4. Network : This selection criteria should only be used if you are sure about your network information. Note that BITNET = EARN = NETNORTH and that UUCP = EUNET. The only networks known are EARN, BITNET, NETNORTH, JANET, UUCP and DECNET. Specifying one of the BITNET, EARN or NETNORTH network will trigger a search in these three network, as they are very similar, and as a novice user could not know the name used in each part of the world. The CERN network is the collection of the Cern E-Mail nodes.
5. E-Mail Address : The provided E-Mail Address is a suggestion and is not 100% reliable.
6. Mailing results : Note the possibility, in interactive mode only, to send the result of a query to a remote correspondent, provided that you know his E-Mail address.
7. Printing results : To print the resulting file, just issue 'XPRINT (PRINTER printername' from the command line of the display.
8. DataBase content : The DataBase contains JANET entries only for nodes with which Cern users can exchange mail. The DataBase contains DECNET entries only for Cern's nodes, and UUCP entries for European nodes only.
9. Selection criterion : Entering minimal information is often sufficient (e.g. only name of institute or town) and searching the resulting file with an editor may give best results. Restricting the search too narrowly can sometimes hide the address you are really looking for.
10. Going faster : Specifying the country will accelerate the retrieval. Specifying the network will have the same effect, however, this should only be used if you are certain that your network information is correct.
11. Oracle Errors : When database access fails, an Error Report should be made in the resulting file, specifying the Oracle Error. In such a case, no search is done in the database.

Examples :

- To get the address of the node named SLACVM, issue 'EMNODES SLACVM'. In interactive mode, fill the node field with the value SLACVM and press on PF5 (search). This will retreive the results :

```
EmNodes - Results For Request
  Node        : SLACVM
  Institute   : *
  Town        : *
```

FILE: EMNODES HELPCMS A1 VM/XA CERN CMS 5.5 v.8

```
  Country     : *
  Network(s) : *
--------------------------------------------------------------------------------
Country : US     Node : SLACVM          Network : BITNET
Site: Stanford Linear Accelerator Center - SLAC Computer Center
Stanford University;Stanford, CA 94305 US
-> E-Mail : User@SLACVM.BITNET

   1 node(s) found.
--------------------------------------------------------------------------------
```

- To obtain the list of the nodes located at CERN, issue the command 'EMNODES * CERN' or, in interactive mode, fill the field institute with the value CERN and press on the PF5 (search).

EmNodes is only available on Cern's IBM/VM-CMS.

Report problems to M. Dimou (Dimou@priam.cern.ch).

Screen of the EmNodes user Interface.

```
              EmNodes : E-Mail Nodes Addresses
             ----------------------------------

Enter search criteria with '*' or/and '?'. Each criterion is optional.

   Node          :
   Institute     :
   Town          : LEON
   Country       : ES
   Network       :

Options :
   Save results on File        (fn ft fm) >
   Mail results          (E-Mail Address) >
-------------------------------------------------------------------------

-------------------------------------------------------------------------
   Institute   : If using abbreviations, enter '*' instead of '.' !
   Town        : Enter in English or in local language.
   Country     : Enter ISO two letter Country Code only.
   Network     : Bitnet, Earn, Netnorth, Decnet, Uucp, Janet or Cern.
-------------------------------------------------------------------------
PF1=Help                  PF3=QUIT    PF4=ISO CC  PF5=Search
PF7=Mail      PF8=Save    PF9=Clear
```

Result of the query

```
-------------------------------------------------------------------------
EmNodes - Results For Request
  Node          : *
  Institute     : *
  Town          : LEON
  Country       : ES
  Network(s)    : *
-------------------------------------------------------------------------
Country : ES      Node : ELEULE11         Network     : BITNET
Site : Univesidad de Leon, Spain
Univesidad de Leon;Centro de Proceso de Datos;Campus de Vegazana
-> E-Mail : User@ELEULE11.BITNET

  1 node(s) found.
-------------------------------------------------------------------------
```

Annex D Code of the NameServer

Annex D is the code of the NameServer described in Chapter 13.

The following files are given:
- the **Makefile**
- **MED.help** is the Help file;
- **med.server** is the NameServer itself;
- **med.c** accesses the EmDir database and outputs results in a special format;
- **getadd.c** performs the Remote Procedure Call after having processed the arguments of the query;
- **MED.msganalyser.lex** analyses the incoming E-mail messages in order to extract the requests and the originator's address;
- **MED.mep.lex** paginates the results produced by med.c.

Emdir.h (included in getadd.c) is not joined for space reasons. It consists of all of the precompiled RPC system type and routine definitions.

The last page displays an example of message sent to the NameServer and the corresponding response.

```
PRG        = med
RPC        = emdir
CRBASE     = /usera/heuse/emdirtmp
CFLAGS     = -O -I$(CRBASE)/h -I.
USEROBJS   = $(PRG).o $(RPC)_client.o
LIBS       = $(CRBASE)/lib/libcr.a $(CRBASE)/lib/libnet.a
INCS       = $(RPC).h
COURIERCC  = $(CRBASE)/compiler/courier
GETADD     = getadd

MSGMEPLEX  = MED.mep.lex
MSGMEP     = MED.mep
MSGANALLEX = MED.msganalyser.lex
MSGANAL    = MED.msganalyser
SERVER     = med.server

all: $(SERVER)

$(SERVER): $(MSGANAL) $(PRG) $(MSGMEP)

$(MSGMEP): $(MSGMEPLEX)
        lex -t $(MSGMEPLEX) > mep.c
        cc -o $(MSGMEP) mep.c -ll

$(MSGANAL): $(MSGANALLEX)
        lex -t $(MSGANALLEX) > anal.c
        cc -o $(MSGANAL) anal.c -ll

$(GETADD).c : $(RPC).h

$(PRG): $(USEROBJS) $(PRG).c $(GETADD).c
        cc -o $(PRG) $(USEROBJS) $(LIBS)

$(USEROBJS) : $(LIBS) $(INCS)

$(RPC).h: $(GETADD).cr
        rm -f $(RPC).h
        $(COURIERCC) $(GETADD).cr

clean:
        rm -f *.o mep.c anal.c ,* emdir_*.c emdir.h *.old

receive:
        mv $(GETADD).cr $(GETADD).cr.old
        cp "../libc/$(GETADD).cr" .
        mv $(GETADD).c $(GETADD).c.old
        cp "../libc/$(GETADD).c" .
```

CERN Name Server - Help file

The CERN Name Server is a service provided by the Cern Institute.

It allows to access to a database containing all Cern's users, and to obtain their E-Mail addresses. Some other users are also registered in this DB. As the DB is freely filled and updated by the people registered in the DB, the information is not always reliable.

To use this service, you have to send a mail in a specific format. This mail has to be send to (E-Mail address):

nameserver@cernvax.cern.ch

The subject field or the body must be filled in one of the following syntaxes (angle brackets <> indicate request parameters, square brackets [] indicate optional parameters):

1) EAN find syntax :

find [<name|*>]:[<organization|*>]

where <name> is the name or the firstname of the person you wish to obtain the E-Mail address, and <organization> the institute, division or experiment where this person is working.
Wild card characters '*' and '?' may be used in order to replace respectively 0, 1 or more characters and one and only one character (blanks have to be replaced by '?').
Missing parameters will be considered as '*' wild card.
Note the a 'find *:*' will be rejected, as such a request is not allowed in the DB.
The 'find' keyword may be alone in the subject field, and the other part of the request in the body, according to the real EAN syntax.
Several requests (Maximum 2) may be put in the same message. In this case, the find must be seperated by " ;". The 'find' keyword has not to be repeated for each new find. So "find smith ; *:delphi" is a valid request.

2) EmDir query syntax :

query <name|*> [<firstname|*> [<division|*> [<experiment|*>
[<institute|*> [<E-Mail Address|*>]]]]]

where the parameters are respectively the name and firstname of the person who above you wish to obtain information, the division, experiment, and institute where she is working, and her E-Mail address.
Wild card characters '*' and '?' may be used in order to replace respectively 0, 1 or more characters and one and only one character (blanks have to be replaced by '?').
Missing parameters will be considered as '*' wild card.
Note the a 'query * * * * * *' will be rejected, as such a request is not allowed in the DB.
Several queries (Maximum 6) may be asked in the same message. In this case, the queries must be seperated by " ;" and the 'query' keyword repeated for each new query. So "query smith ;query * john" is a valid request.
The queries may be distributed in the subject field and the body. In this case, the end of the subject field is supposed to be a query separator.

In each case, a limited number of entries will be retrieved.

A syntax error will send this file.

The help file may also be obtained by filling the subject field with the 'help' or 'info' keywords.

The syntax of the reply message is the following :

```
<Request>

[<Error messages>]

<Name> <FirstName>
  Div: <Division>   Exp: <Experiment>   Cern phone (ext): <CERN phone number>
  Home Institue: <Institute>   Phone: <Institute phone number>
  E-Mail Address: <E-Mail Address>
  Comment: <Comment>

Usual error messages are:
ERR 03 - No match for your query.
ERR 06 - Query returns too many entries.
ERR 11 - No valid query detected.

            Comments may be sent to <bothner@priam.cern.ch>.
```

```
#!/bin/sh

# VERSION 2

###############################################################################
###                                                                         ###
### CERN Name Server                                                        ###
###                                                                         ###
###      Input (stdin) : Message send to 'nameserver@vscstb.cern',          ###
###                      written according to EAN syntax or                 ###
###                      syntax described in the $HELPFILE. This            ###
###                      message constitutes a query to the CERN            ###
###                      Name Server. The message is received on            ###
###                      the standard input file.                           ###
###      Result (mail message) : Message sent back to the sender.           ###
###                      This message is the answer of the server           ###
###                      to the query of the sender.                        ###
###                                                                         ###
###      This shell uses an EMDIR modified program, written by             ###
###           F. Hemmer, modified by myself and a specific message          ###
###           analyser. The message analyser produces a file containing     ###
###           the address of the sender, a diagnostic and a set of          ###
###           queries in accordance to the EMDIR syntax. This               ###
###           file is analysed by the shell to build the reply              ###
###           message and appended to the log file.                         ###
###                                                                         ###
###      More information in the MED.doc file.                              ###
###                                                                         ###
###  Written by Bernard Heuse, CERN technical student, DD/CS/EN, SEPT 88    ###
###############################################################################

DIR="/usera/heuse/nameserver"
cd $DIR

# Adding paths and exporting them to child processes
PATH=".:$PATH:/bin:/usr/ucb:/usr/local/unix"
export PATH

# Message Analyser, RPC caller, Display prg
ANALYSE=MED.msganalyser
RPCC=med
MEPPRG=MED.mep

# Log and Help files
LOGFILE=MED.log
HELPFILE=MED.help

# Temporary files : reply and message analyse results
REMSG=/tmp/MED.rep.$$
TEMP=/tmp/MED.mar.$$
TMPF=/tmp/MED.tmp.$$
ERRF=/tmp/MED.err.$$
MEPF=/tmp/MED.mep.$$
ERRFB=/tmp/MED.errb.$$
CHECK=/tmp/MED.check.$$
MAIL=/tmp/MED.mail.$$
ALLFILES=/tmp/MED.*.$$

# msg sender (from) and errors-to
ADDFROM="CERN EmDir Name Server <nameserver@cernvax.cern.ch>"
ERRTO="<bothner@priam.cern.ch>"
NAMESERVER="<nameserver@cernvax.cern.ch>"

#Line of
```

```
DASHES="--------------------------------------------------------------------------------"
BLANKS=""

#Exit codes
RETRY=75
ALLOK=00

#Server Ok
SEROK=00

# Taking date
DATE=`date "+%a, %d %h %y %T MET (GVA)`

# Analysing message
$ANALYSE > $TEMP

##############################################################################

# If Reject => exit (pattern TR- means 'type : reject')
if grep -s "^TR-" $TEMP
then
   cat $TEMP >> $LOGFILE
   echo "      Message rejected." >> $LOGFILE
   echo "      $DATE" >> $LOGFILE
   echo "      " >> $LOGFILE
   rm -f $ALLFILES
   exit $ALLOK
fi

##############################################################################

# If Help file must be sent (pattern TH- means 'type : Help')
if grep -s "^TH-" $TEMP
then
   # Building the reply message
   SUBJECT="EmDir Help File."
   #=== Syntax Error======================
   if (grep -s -i "^TH- [0-9]* *.*syntax" $TEMP)
   then
      echo $BLANKS >> $REMSG
      echo "ERR 11 - No valid request detected." >> $REMSG
      echo $BLANKS >> $REMSG
      echo $DASHES >> $REMSG
   fi
   cat $HELPFILE >> $REMSG
   LOGMSG="Help File sent."
fi

##############################################################################

# Queries detected (pattern TQ- means 'type : query')
if grep -s "^TQ-" $TEMP
then
   # Building the reply message
   SUBJECT="Query: EmDir Request Answer."
   cat - << +++EOF+++ >> $REMSG
Query Syntax :
 Query <Name|*> [<FirstName|*> [<Division|*> [<Experiment|*>
       [<Institute|*> [<E-Mail Address|*>]]]]]
-------------------------------------------------------------------------------
+++EOF+++
   # Extraction of the number of queries (Number following 'TQ- ')
   NB=`grep "^TQ- " $TEMP | sed -e "s/TQ- //" -e "s/ .*//"`
   II=0
```

```
   # for each request
   while (test "$NB" != "$II")
     do
       II=`expr $II + 1`
       HEAD="Q"$II"- "
       # Parameter extraction
       PAR=`grep "^$HEAD" $TEMP | sed "s/^$HEAD//"`
       PART=`grep "^$HEAD" $TEMP | sed "s/^$HEAD//" | tr \% \* | tr \_ \?`
       echo "Query $PART" >> $REMSG
       echo $BLANKS >> $REMSG
       # EmDir call
       $RPCC $PAR > $TMPF
       if (test $? -ne 0) then SEROK=1 ; fi
       fgrep ERR < $TMPF > $ERRF
       $MEPPRG < $TMPF > $MEPF
       if grep -s ".*" $ERRF
       then
          cat $ERRF >> $REMSG
          echo $BLANKS >> $REMSG
       fi
       cat $MEPF >> $REMSG
       echo $DASHES >> $REMSG
     done
   LOGMSG="Query: Reply Message sent."
fi

##############################################################################

# Find detected (pattern TF- means 'type : find')
if grep -s "^TF-" $TEMP
then
   # Building the reply message
   SUBJECT="Find: EmDir Request Answer."
   cat - << +++EOF+++ >> $REMSG
Find Syntax :
 Find [<string1|*>]:[<string2|*>]
      where <string1> is a name or a firstname,
            <string2> is an expriment, a division or an institute.
------------------------------------------------------------------------------
+++EOF+++
   # Extraction of the number of find (Number following 'TF- ')
   NF=`grep "^TF- " $TEMP | sed "s/TF- \([0-9]*\) .*/\1/"`
   IF=0
   # for each find
   while (test "$NF" != "$IF")
     do
       IF=`expr $IF + 1`
       #Extraction of the nber the find request
       HEADF="F$IF-"
       P1=`grep "^$HEADF" $TEMP | sed -e "s/^$HEADF[^ ]* *//" -e "s/:.*$//" `
       P2=`grep "^$HEADF" $TEMP | sed -e "s/^$HEADF[^:]*://" -e "s/ *$//" `
       Q1="$P1 % $P2"   ; Q2="$P1 % % $P2"   ; Q3="$P1 % % % $P2"
       Q4="% $P1 % $P2" ; Q5="% $P1 % % $P2" ; Q6="% $P1 % % % $P2"
       NO="%%"
       if (test "$P1" = "%") then Q4=$NO ; Q5=$NO ; Q4=$NO ; fi
       if (test "$P2" = "%") then Q2=$NO ; Q3=$NO ; Q5=$NO ; Q6=$NO ;fi
       # for each query du find
       rm -f $ERRF
       rm -f $MEPF
       for PAR in "$Q1" "$Q2" "$Q3" "$Q4" "$Q5" "$Q6"
         do
           if (test "$PAR" != "$NO") then
              # EmDir call
              $RPCC $PAR > $TMPF
```

```
            if (test $? -ne 0) then SEROK=1 ; fi
            fgrep ERR < $TMPF >> $ERRF
            $MEPPRG < $TMPF >> $MEPF
          fi
        done
      FREQ=`grep "^$HEADF" $TEMP | sed -e "s/^$HEADF //" | tr \% \* | tr \_ \?`
      echo "Find $FREQ" >> $REMSG
      echo $BLANKS >> $REMSG
      # Extraction of the "No Match" ERR msg and only 1 ERR 06 msg if any
      fgrep -v "ERR 03" < $ERRF | fgrep -v "ERR 06" > $ERRFB
      fgrep "ERR 06" < $ERRF | sed -e "2,10 d" >> $ERRFB
      if grep -s ".*" $ERRFB
      then
         cat $ERRFB >> $REMSG
         echo $BLANKS >> $REMSG
      fi
      # Si MEPF vide => No Match
      if grep -s ".*" $MEPF
      then
         cat $MEPF >> $REMSG
         echo $DASHES >> $REMSG
      else
         echo "ERR 03 - No match for your query" >> $REMSG
         echo $BLANKS >> $REMSG
         echo $DASHES >> $REMSG
      fi
    done
   LOGMSG="Find: Reply Message sent."
fi

##############################################################################

if (test $SEROK -ne 0)
then
   cat $TEMP >> $LOGFILE
   echo "      Server not available. Will retry later." >> $LOGFILE
   echo "      $DATE" >> $LOGFILE
   echo "      " >> $LOGFILE
   rm -f $ALLFILES
   exit $RETRY
else
   # Taking the address.
   ADD=`grep "^AD-" $TEMP | sed -e "s/^AD- //" -e "s/^[ \t]*//"`
   # Building the message
   cat - << +++EOF+++ $REMSG > $MAIL
To:         $ADD
From:       $ADDFROM
Errors-to: $ERRTO
Subject:    $SUBJECT
Date:       $DATE

$DASHES
Send a mail to $NAMESERVER with the keyword 'Help' as subject
  to receive more information about this service and the syntax of the
  EmDir Request Answer.
------------------------------------------------------------------------------
+++EOF+++
   # Sending the message
   /usr/lib/sendmail -t -f "$ERRTO" < $MAIL
#  cat $MAIL
   # Completing log file
   cat $TEMP >> $LOGFILE
   echo "      $LOGMSG" >> $LOGFILE
   echo "      $DATE  " >> $LOGFILE
```

```
   echo "              " >> $LOGFILE
   # Deleting temporary files
   rm -f $ALLFILES
   exit $ALLOK
fi

# END OF FILE
```

```
/* ************************************************************************
   Produce a file with the following syntax :
   ('^' indicates the beginning of a ligne et <var> indique la valeur
 de la variable var)

 - First line :   ^##<nb>    where nb is the number of entries retrieved.
 - For i = 1 to nb
                  ^
                  ^$#<i>
                  ^NM- <name>
                  ^FN- <firstname>
                  ^CD- <cern division>
                  ^CP- <cern phone>
                  ^CE- <cern experiment>
                  ^HI- <home institute>
                  ^IP- <intitute phone>
                  ^MA- <mail address>
                  ^CT- <comment>

  Return Code : 75 si serveur distant non-disponible, 0 sinon.

************************************************************************ */

#include <stdio.h>

#include <getadd.c>  /* RPC Access */

/* RPC Authorization an return codes */
#define USAGE                    -9
#define BIND_FAILED               1

/* Emdir errors definition */
#define EMDIR_BAD_QID        50000  /* Bad query id - should not happen */
#define EMDIR_NO_MEM         50001  /* Host unable to malloc            */
#define EMDIR_TOO_MANY_ROWS  50002  /* too many rows retrieved          */
#define EMDIR_UKN_RPC        50003  /* UNKOWN RPC - SHOULD NOT HAPPEN   */
#define EMDIR_BAD_PSW        50004  /* Bad password given for update    */
#define EMDIR_BAD_VER        50005  /* Emdir server and client have     */

/* Maximum 10000 rows will be returned */
#define MAX_QUERY_LIMIT      10000

#define ROW addresses.return_rows.sequence

extern Addresses     addresses;
       char          *name,*fname,*div,*exp,*inst,*maddr;

/* ************************************************************
   *** MAIN                                                 ***
   ************************************************************
*/
main(argc,argv)
int argc;
char *argv[];
{
int garc,arnu;

  arnu=getarg(argc,argv);
  if (arnu > 0)
  { garc=GetAddresses(name,fname,div,exp,inst,maddr,USAGE);
    if (garc<=0) d_rows();
    else switch(garc) {
         case 1 : fprintf(stdout,"ERR 20 - Cern Network Failure : try again later !\n");
```

```
                break ;
        case 2 : fprintf(stdout,"ERR 01 - No valid query was issued\n");
                break ;
    }
  }
  else fprintf(stdout,"ERR 00 - No arguments given in command line\n");
  if (garc==BIND_FAILED) exit(75);
  exit(0);
}

/* Receive the arguments of the command line, set the query bits */
getarg(argc,argv)
int  argc;
char *argv[];
{ static char nullstring[]={"\0"};
  char *arg;
  int rc,i;

  name = fname = div = exp = inst = maddr = nullstring;
  rc = argc-1; /* rc > 0 if some arg given */
  while (--argc > 0)
  { arg = argv[argc] ;
    switch(argc) {
    case 1: name = arg;
            break;
    case 2: fname = arg;
            break;
    case 3: div = arg;
            break;
    case 4: exp = arg;
            break;
    case 5: inst = arg;
            break;
    case 6: maddr = arg;
            break;
    }
  }
}

d_row(n)
int n;
{
  fprintf(stdout,"\n");
  fprintf(stdout,"$#%d\n",n+1);
  fprintf(stdout,"NM- %s\n",ROW[n].name);
  fprintf(stdout,"FN- %s\n",ROW[n].first_name);
  fprintf(stdout,"CD- %s\n",ROW[n].cern_div);
  fprintf(stdout,"CP- %s\n",ROW[n].cern_tel);
  fprintf(stdout,"CE- %s\n",ROW[n].exp);
  fprintf(stdout,"HI- %s\n",ROW[n].inst_mnemo);
  fprintf(stdout,"IP- %s\n",ROW[n].inst_tel);
  fprintf(stdout,"MA- %s\n",ROW[n].mint_addr);
  fprintf(stdout,"CT- %s\n",ROW[n].comments);
}


d_rows()
{ int i;
  Cardinal count;

  if (addresses.status) d_error(addresses.status);
  if ((addresses.status==0) ||
      (addresses.status==1) ||
      (addresses.status== EMDIR_TOO_MANY_ROWS)) /* status ok */
```

```
      { if (addresses.status == EMDIR_TOO_MANY_ROWS)
                              count = MAX_QUERY_LIMIT; /* request too long */
                         else count = addresses.return_rows.length;
        if (count > 0) { fprintf(stdout,"###%d entries retrieved.\n",count);
                         for(i=0;i<=count-1;i++)  d_row(i);
                       }
        else fprintf(stdout, "ERR 03 - No match for your query.\n");
      }
}

d_error(e_no)
int e_no;
{
  switch (e_no) {
  case 0:
          return; /* OK */
  case 1:
          return; /* OK */
  case EMDIR_BAD_QID:
          fprintf(stdout,"ERR 04 - Incorrect query_id, contact system manager\n");
          break;
  case EMDIR_NO_MEM:
          fprintf(stdout,"ERR 05 - Insufficient memory on host side, contact system mana
          break;
  case EMDIR_TOO_MANY_ROWS:
          fprintf(stdout,"ERR 06 - Query returns too many entries. Issue a more restrict
          break;
  case EMDIR_UKN_RPC:
          fprintf(stdout,"ERR 07 - Unknown remote procedure, contact system manager\n");
          break;
  case EMDIR_BAD_PSW:
          fprintf(stdout,"ERR 08 - Incorrect password\n");
          break;
  case EMDIR_BAD_VER:
          fprintf(stdout,"ERR 09 - Obsolete Emdir version, contact system manager\n");
  default:
       if (e_no < 0) fprintf(stdout,"ERR 10 - Oracle error %d on host, contact system ma
       else fprintf(stdout,"ERR 11 - Unknown error %d, contact system manager\n",e_no);
  }
}
```

```
# include <stdio.h>
# include <ctype.h>
# include <time.h>
# include <pwd.h>              /* To get unix username */

# include "emdir.h"            /* Courier typedefs */

/* Connection Parameters */
#define MACHINE          "vxcrna"
#define TCP_PORT             2152
#define BIND_FAILURE           -1

/* Authorization Parameters */
/* Defined usage */
#define USAGE_NAMESERVER        9
#define USAGE_AUTOROUTE         8
/* current version, release and fix */
#define VERSION    1
#define RELEASE    1
#define FIX_LEVEL  2
#define RPC_SYSTEM 3
#define AUTH_ID    0

/* Return Codes */
#define QUERY_NOUSER       -1
#define QUERY_OK            0
#define BIND_FAIL           1
#define QUERY_FULL          2

/* Oracle Specific */
#define ORAWCHAR          '%'

static Addresses addresses;

/* Usage specifies the origin of the query for the Authorization       */
/* Full queries are rejected                                           */

GetAddresses(name,fname,div,exp,inst,add,usage)
char *name,*fname,*div,*exp,*inst,*add;
int usage;
{ int             qid,rc;
  Query_row       qrow;
  Authorization   auth;
  rc=GA_CptQid(name,fname,div,exp,inst,add,&qrow,&qid);
  if ( ((usage<0) || (rc==QUERY_OK)) && (rc!=QUERY_FULL))
     { GA_FillAuth(&auth,usage);
       if ((BindemdirToMachine(MACHINE,TCP_PORT)) != BIND_FAILURE)
          {  addresses = get_address(auth,qid,qrow);
             Unbindemdir();
          }
       else rc = BIND_FAIL ;
     }
  return(rc);
}

char *GA_toora(st)
/*  '*'=>'%'  and  '?'=>'_'  */
/*  strip redundant wildchar */
char *st;
{ char *i,*cc,*sw;
  for(i=st;*i;i++) if (*i=='*') *i='%';
  for(i=st;*i;i++) if (*i=='?') *i='_';
  for(i=cc=sw=st;*i;)
     if ((*i!='%')&&(*i!='_')) *cc++ = *i++ ;
```

```
     else if (*i=='%') { *cc++ = *i++ ;
                         for(;((*i=='%')||(*i=='_'));i++);
                       }
     else { sw = cc ; *cc++ = *i++ ;
            for(;*i=='_';*cc++ = *i++);
            if (*i=='%') cc=sw;
          }
  *cc='\0';
  return(st);
}

char *GA_toup(st)
/* to lower string */
char *st;
{ char *i;
  for(i=st;*i;i++) *i = toupper(*i);
  return(st);
}


GA_CptQid(name,firstname,div,exp,inst,maddr,qrow,qid)
char *name,*firstname,*div,*exp,*inst,*maddr;
Query_row *qrow;
int *qid;
/*
   Compute the query id and set qrow

   rc = if ((name = '*') and (fname = '*')) QUERY_NOUSER
        else if ((name = '*') and (fname = '*') and (div = '*")
                and (exp = '*') and (inst = '*') and (maddr = '*'))
                QUERY_FULL
        else if "Invalid Usage" AUTH_DENIED
        else QUERY_OK
*/

{
    static struct {
      unsigned name : 1;
      unsigned first_name : 1;
      unsigned cern_div : 1;
      unsigned exp : 1;
      unsigned inst_mnemo : 1;
      unsigned mint_addr : 1;
    } qbit = { 0, 0, 0, 0, 0, 0 };

    int len,rc;

    *qid = 0;
    qrow->name       = name;
    qrow->first_name = firstname;
    qrow->cern_div   = div;
    qrow->exp        = exp;
    qrow->inst_mnemo = inst;
    qrow->mint_addr  = maddr;

    /* set appropriate bits */
    GA_toora(GA_toup(qrow->name));
    if ((len = strlen(qrow->name)) > 0) {
     if (len > 1) qbit.name = 1;
     else if (*qrow->name == ORAWCHAR) qbit.name = 0;
     else qbit.name = 1;
    }
    GA_toora(GA_toup(qrow->first_name));
    if ((len = strlen(qrow->first_name)) > 0) {
```

```
     if (len > 1) qbit.first_name = 1;
     else if (*qrow->first_name == ORAWCHAR) qbit.first_name = 0;
     else qbit.first_name = 1;
    }
    GA_toora(GA_toup(qrow->cern_div));
    if ((len = strlen(qrow->cern_div)) > 0) {
     if (len > 1) qbit.cern_div = 1;
     else if (*qrow->cern_div == ORAWCHAR) qbit.cern_div = 0;
     else qbit.cern_div = 1;
    }
    GA_toora(GA_toup(qrow->exp));
    if ((len = strlen(qrow->exp)) > 0) {
     if (len > 1) qbit.exp = 1;
     else if (*qrow->exp == ORAWCHAR) qbit.exp = 0;
     else qbit.exp = 1;
    }
    GA_toora(GA_toup(qrow->inst_mnemo));
    if ((len = strlen(qrow->inst_mnemo)) > 0) {
     if (len > 1) qbit.inst_mnemo = 1;
     else if (*qrow->inst_mnemo == ORAWCHAR) qbit.inst_mnemo = 0;
     else qbit.inst_mnemo = 1;
    }
    GA_toora(qrow->mint_addr);
    if ((len = strlen(qrow->mint_addr)) > 0) {
     if (len > 1) qbit.mint_addr = 1;
     else if (*qrow->mint_addr == ORAWCHAR) qbit.mint_addr = 0;
     else qbit.mint_addr = 1;
    }

    if ((qbit.name==0) && (qbit.first_name==0)) rc = QUERY_NOUSER ;
    else rc = QUERY_OK ;

    *qid = (qbit.mint_addr +
           (qbit.inst_mnemo << 1) +
           (qbit.exp << 2) +
           (qbit.cern_div << 3) +
           (qbit.first_name << 4) +
           (qbit.name << 5)
         ) & 0x0000003F ;

    if (*qid==0) rc = QUERY_FULL;

    if (*qid==0) { *qid = 32 ; strcpy(qrow->name,"%%"); }

    return(rc);

}

GA_FillAuth(auth,usage)
Authorization *auth;
int usage;
{
struct passwd *getpwuid();
struct passwd *pd;
struct tm     *timestr;
char   loc_userid[256];
char   loc_station[100];
char   loc_date[10];
char   loc_time[10];
int    btime;
static char *months[12] = {"Jan", "Feb", "Mar",
                           "Apr", "May", "Jun",
                           "Jul", "Aug", "Sep",
                           "Oct", "Nov", "Dec"};
```

```
    strcpy(loc_userid,"");
    pd = getpwuid(getuid());
    sprintf(loc_userid,"%s",pd->pw_name);
    if (strlen(loc_userid)==0) sprintf(loc_userid,"%s","Unkown");

    time(&btime);
    timestr = localtime(&btime);
    sprintf(loc_date,"%d-%s-%d",timestr->tm_mday,
                                months[timestr->tm_mon],
                                timestr->tm_year);
    sprintf(loc_time,"%d:%d:%d",timestr->tm_hour,
                                timestr->tm_min,
                                timestr->tm_sec);

    gethostname(loc_station,100);

    auth->emdir_version     = VERSION;
    auth->emdir_release     = RELEASE;
    auth->emdir_fix_level   = FIX_LEVEL;
    auth->rpc_system        = RPC_SYSTEM;
    auth->auth_id           = AUTH_ID;
    auth->date              = loc_date;
    auth->time              = loc_time;
    auth->userid            = loc_userid;
    auth->station           = loc_station;
    auth->operation         = abs(usage);
}
```

```
%{
/* Written by Bernard Heuse, CERN technical student, DD/CS/EN, SEPT 88 */

/* This LEX program is designed to analyse a E-Mail message (written
   following the RFC 822 standard) received by the CERN Name Server.
   It extracts the E-Mail address of the sender, and the request
   according to the syntax described in the MED.help file. It
   produces a file with the following informations (one per line):
   - Address of the sender (1 line);
        Syntax : 'AD- <address>'.
         - address is the address of the sender.
   - Request type or reject diagnostic (1 line);
        Syntax : 'T<c>- <n> <diag> '
         - c = 'Q' for query request => n = number of queries
           diag = 'Query'
         - c = 'F' for find request => n = number of finds
           diag = 'Find'
         - c = 'H' => n = 0
           diag = 'Syntax Error' if syntax error ; 'Help' if help file asked.
         - c = 'R' => n = 0
           diag = 'rejected on address' if addresses on 'the black list'
             are detected, if no address field is detected, or if no
             address is specified, or 'rejected on subject' if returned mail
             or rejected mail are detected in the subject field.
   - Valid Find Requests Detected (n lines if c = 'F', 0 else).
        Syntax : 'F<j>- <param1>:<param2>'
         - j = 1 to n where f = number of find accepted.
         - param1 and param2 are the parameters of the find detected.
   - Valid Query Requests Detected (n lines if c = 'Q', 0 else).
        Syntax : 'Q<i>- <param>'
         - i = 1 to n where n = number of queries accepted.
         - param is the parameter string for Emdir.
*  It reads the message on the standard input file, and write the file on
   the standard output file.
*/
%}

%START BEG HDR BODY
%START ADDR ADDRNEXT SUBJ
%START HEMDIR HEAN1 HEAN2 HEAN3
%START BEMDIR BEAN1 BEAN2 BEAN3

%{

# define bool int
# define TRUE 1
# define FALSE 0

# define MaxReq  9     /* Max Nber of query requests accepted */
# define MaxFind 2     /* Max Nber of find requests accepted  */

# define MaxPar 6      /* Max number of param for a query request */

# define AdrLen 200    /* Max Lgth of an E-Mail address */
# define ReqLen 500    /* Max Lgth of a request */

# define WCHAR  "%"    /* Oracle Wild Card */
# define WJOKER "_"    /* Oracle Wild Card */

/* Addr Field Priorities (highest is winner ; <= 0 is rejected ) */
#define P_SENDER              10
#define P_RESENT_SENDER        9
#define P_FROM                 8
#define P_RESENT_FROM          7
```

```
#define P_REPLY_TO              6
#define P_RESENT_REPLY_TO       5

 /* NB: - Unput('\n') is used to reject the CR/LF and allow next
        pattern matching to detect field header beginning always
        at a new line, possibly with blanks and tabs before
        - If a Help or Info request is detected, no query are accepted.
        - Priority is given among the possible resent field.
        - Several subject fields may occur.
        - Only one syntax per message is accepted (either EAN, either EMDIR).
        - Syntax EmDir :
        In the body or the subject, a (new) query is detected when the
        KeyWord "query" with blanks or tabs or NL before and after
        are matched. The following words are considered as being
        parameters until a EOF or a " ;" or a new header mail field is
        detected. The parameters exceeding the MaxPar first ones and the words
        following a " ;"are ignored, except if a new query is detected.
        Several queries may occur in a file, in the header, in the body
        or both. The strings preceeding a 'query' are ignored.
        Queries without any parameters or with only wild card are
        rejected.
        Syntax : query <par1> [<par2..6>] ; query <par1> [<par2..6>] ; ...
        - Syntax EAN :
        In the body or the subject, a Find is detected when the
        KeyWord "find" with blanks or tabs or NL before and after
        are matched. Several queries may be seperated by a " ;".
        The strings preceeding a 'find' are ignored. Only the
        last string before the ':' and the first string after the ':'
        are considered. The others are ignored. Only one 'find'
        per message is necessary. Several queries may occur in a file,
        in the header, in the body or both. All string are optional,
        missing strings are considered as being wild card.
        Queries with only wild card are rejected.
        Syntax : "find" [<str1>]":"[<str2>] ; [[<str1>]":"[<str2>]] ; ...
 */
%}

 int  reqtype;                  /* 0=SynError, 1=Help, 2=Query, (request type)
                                   3=Find, 5=Reject on addr, 6=Rej on subj */
 int  syntype;                  /* 0=?, 1=EmDir, 2=Ean (syntax type) */
 char addresse[AdrLen];         /* E-Mail Address */
 char str1[ReqLen];             /* Str1 syntax Find EAN */
 char str2[ReqLen];             /* Str2 syntax Find EAN */
 int  nfind;                    /* Nber of find */
 int  nreq;                     /* Nber of queries */
 char *req[MaxReq+1];           /* Query Request storage Array */
 char *find[MaxFind+1];         /* Find  Request storage Array */
 int  state;                    /* 1=BEAN1 or BEAN2, 3=BEAN, 0=? */
 bool helpreq;                  /* True = help request */
 bool reject;                   /* True = message rejected */

%%
 int prior;                     /* Current Address Priority */
 int fieldprior;                /* Field Address Priority */
 int percentq;                  /* All WCHAR in query => 0 */
 int parcnt;                    /* Parameter Counter    */
 int i;
 char *ireq;
 extern int  nfind;
 extern int  nreq;
 extern char *req[];
 extern char addresse[];
 extern int  reqtype;
 extern int  syntype;
```

```
 extern char str1[];
 extern char str2[];
 extern int  state;
 extern bool helpreq;
 extern bool reject;

 nfind=nreq=fieldprior=prior=state=reqtype=syntype=0;
 addresse[0]=str1[0]=str2[0]='\0';
 helpreq=reject=FALSE;

 BEGIN BEG;

%{
/* ****************************************************************
   *** BEG beginning of message                                 ***
   ****************************************************************
*/
%}
%{
/* Skipping first white lines, rejecting first character detected
   on the first non-white line and a CR/LF.
   Beginning the header message analyse.
*/
%}
<BEG>^[ \t]*\n                  ;
<BEG>.                          { unput(yytext[0]);
                                  unput('\n');
                                  BEGIN HDR;
                                }

%{
/* ****************************************************************
   *** HDR header analysis                                      ***
   ****************************************************************
*/
%}
%{
/* Detecting Address fields to send back the message.
   Assigning a priority (the higest is preferred).
   Beginning address data field reading.
*/
%}
<HDR>^[ \t]*[fF][rR][oO][mM]":"  { fieldprior=P_FROM;
                                  BEGIN ADDR;
                                }
<HDR>^[ \t]*[sS][eE][nN][dD][eE][rR]":" {
                                  fieldprior=P_SENDER;
                                  BEGIN ADDR;
                                }
<HDR>^[ \t]*[rR][eE][sS][eE][nN][tT]"-"[sS][eE][nE][dD][eE][rR]":" {
                                  fieldprior=P_RESENT_SENDER;
                                  BEGIN ADDR;
                                }
<HDR>^[ \t]*[rR][eE][pP][lL][yY]"-"[tT][oO]":" {
                                  fieldprior=P_REPLY_TO;
                                  BEGIN ADDR;
                                }
<HDR>^[ \t]*[rR][eE][sS][eE][nN][tT]"-"[fF][rR][oO][mM]":" {
                                  fieldprior=P_RESENT_FROM;
                                  BEGIN ADDR;
                                }
<HDR>^[ \t]*[rR][eE][sS][eE][nN][tT]"-"[rR][eE][pP][lL][yY]"-"[tT][oO]":" {
                                  fieldprior=P_RESENT_REPLY_TO;
                                  BEGIN ADDR;
```

```
                                    }
%{
/* Detecting Subject field where a request could be.
   Starting analyse of the data part of the field.
*/
%}
<HDR>^[ \t]*[sS][uU][bB][jJ][eE][cC][tT]":" {
                                      BEGIN SUBJ;
                                    }
%{
/* Detecting End of Header (A blank line after the header)
   If a 'Find' as been detected, syntype=2, and we must
   immediately begin the EAN syntax analyse, because the find
   may be the only word in the subject, and the request being
   in the body. For the Emdir syntax, the end of the subject
   field is considered as a separator just like the ' ;' and
   the 'query' keyword (mandatory for each new Emdir request)
   will be detected during the body parsing.
*/
%}
<HDR>\n[ \t]*\n                     { unput('\n');
                                      if (syntype==2) BEGIN BEAN1;
                                                 else BEGIN BODY;
                                    }
%{
/* Skipping other characters and CR/LF.
*/
%}
<HDR>.                              ;
<HDR>\n                             ;

%{
/* ****************************************************************
   *** ADDR ADDRNEXT address field analysis                      ***
   ****************************************************************
*/
%}
%{
/* If the address field priority is higher than the current one,
   copying the first line of the address in the 'addresse' variable.
   Beginning the next lines address data field reading.
*/
%}
<ADDR>.*                            { if (fieldprior>prior) {
                                        strcpy(addresse,yytext);
                                        prior=fieldprior;
                                        fieldprior=0;
                                        BEGIN ADDRNEXT;
                                      }
                                     else BEGIN HDR;
                                    }
<ADDR>\n                            { if (fieldprior>prior) {
                                        strcpy(addresse,"");
                                        prior=fieldprior;
                                        fieldprior=0;
                                        BEGIN ADDRNEXT;
                                      }
                                     else BEGIN HDR;
                                     unput('\n');
                                    }

%{
/* Adding the next lines of the address field to the 'addresse' variable.
*/
```

```
%}
<ADDRNEXT>.*                    { strcat(addresse,yytext);
                                }
%{
/* Detecting Non Address field end. Skipping.
*/
%}
<ADDRNEXT>\n[" "\t]             { unput(' ');
                                }
%{
/* Else : Address field end. Returning to header analysis.
*/
%}
<ADDRNEXT>\n                    { unput('\n');
                                  BEGIN HDR;
                                }
%{
/* ****************************************************************
   *** SUBJ subject field analysis                                ***
   ****************************************************************
*/
%}

%{
/* Skipping 'tell' and 'emdir' patterns.
*/
%}
<SUBJ>[ \t]*[tT][eE][lL][lL][ \t\n]      |
<SUBJ>[ \t]*[eE][mM][dD][iI][rR][ \t\n]  { if (yytext[yyleng-1]=='\n')
                                             unput('\n');
                                         }
%{
/* Detecting 'query' pattern. If the EAN syntax has not been first detected,
   and if we can accept any more requests, initilization of the reading
   of the parameters. Request type is query. Starting Emdir request
   analyse.
*/
%}
<SUBJ>[ \t]*[qQ][uU][eE][rR][yY][ \t\n]  { if (syntype!=2) {
                                             syntype=1;
                                             if (nreq<MaxReq) {
                                               ireq=(char*)malloc(ReqLen);
                                               reqtype=2;
                                               parcnt=0;
                                               percentq=0;
                                               if (ireq!=NULL) {
                                                  nreq++;
                                                  req[nreq]=ireq;
                                                  strcpy(req[nreq],'\0');
                                                  BEGIN HEMDIR;
                                                  unput(yytext[yyleng-1]);
                                           } } }
                                         }
%{
/* Detecting 'help' or 'info' patterns. Setting the help flag.
*/
%}
<SUBJ>[ \t]*[hH][eE][lL][pP][ \t\n]      |
<SUBJ>[ \t]*[iI][nN][fF][oO][ \t\n]      { helpreq=TRUE;
                                         }
%{
/* Detecting 'find' pattern. If the EmDir syntax has not been first detected,
   initilization of the reading of the parameters. Request type is find.
   Starting the EAN syntax analyse.
```

```
*/
%}
<SUBJ>[ \t]*[fF][iI][nN][dD][ \t\n]        { if (syntype!=1) {
                                               syntype=2;
                                               reqtype=3;
                                               strcpy(str1,"");
                                               strcpy(str2,"");
                                               unput(yytext[yyleng-1]);
                                               BEGIN HEAN1;
                                             }
                                           }
%{
/* If 'returned mail' or 'rejected mail' is detected, rejecting the request.
   Setting the reject flag.
*/
%}
<SUBJ>[rR][eE][a-zA-Z]*[eE][dD][ \t"\n ""\n\t"]*[mM][aA][iI][lL] {
                                             reject = TRUE;
                                           }
%{
/* Skipping other characters.
*/
%}
<SUBJ>.                                    ;
%{
/* Detecting Non Address field end. Skipping.
*/
%}
<SUBJ>\n[" "\t]                            { unput(' ');
                                           }
%{
/* Else : Address field end. Returning to header analysis.
*/
%}
<SUBJ>\n                                   { unput('\n');
                                             BEGIN HDR;
                                           }

%{
/* ****************************************************************
   *** HEMDIR  Emdir syntax analysis                              ***
   ****************************************************************
*/
%}

%{
/* Skipping blanks.
*/
%}
<HEMDIR>[ \t]*              ;
%{
/* Detecting strings, adding them to the parameter string, after
   conversion of the wild card char, going back to 'subject' if more
   than 'MaxPar' are parameters detected. Rejecting request with all
   wild card char.
   As the keyword 'query' is mandatory to start an Emdir request,
   it will be detected in 'subject' and 'subject will initiate the
   next parameter analysis.
*/
%}
<HEMDIR>[^ \t\n]*           { WCDEANTOORA(yytext);
                              strcat(req[nreq],yytext);
                              strcat(req[nreq]," ");
                              parcnt++;
```

```
                              if (strcmp(WCHAR,yytext)!=0) percentq++;
                              if (strcmp("*",yytext)!=0) percentq++;
                              if (parcnt>=MaxPar) { if (percentq==0)
                                                      { free(req[nreq]);
                                                        nreq--;
                                                      }
                                                    BEGIN SUBJ;
                                                  }
                            }
%{
/* Detecting end of query request. Rejecting request with only
   wild card char or with no parameter.
   As the keyword 'query' is mandatory to start an Emdir request,
   it will be detected in 'subject' and 'subject will initiate the
   next parameter analysis.
*/
%}
<HEMDIR>[ \t]*[;]           { if ((parcnt==0) || (percentq==0))
                                                      { free(req[nreq]);
                                                        nreq--;
                                                      }
                              BEGIN SUBJ;
                            }
%{
/* Detecting Non Subject field end. Skipping.
*/
%}
<HEMDIR>\n[ \t]             { unput(' ');
                            }
%{
/* Else : Subject field end. Returning to header analysis.
*/
%}
<HEMDIR>\n                  { if ((parcnt==0) || (percentq==0)) nreq-- ;
                              unput('\n');
                              BEGIN HDR;
                            }

%{
/* ************************************************************
   *** HEAN1 2 3 EAN syntax analysis                         ***
   *** 1 picks the first string                              ***
   *** 2 the second one                                      ***
   *** 3 detects end of request                              ***
   ************************************************************
*/
%}
%{
/* Skipping blanks.
*/
%}
<HEAN1>[ \t]*               ;
%{
/* Detecting string 1, only the last one before the ':' is considered.
   The others are skipped. Conversion of the wild card.
*/
%}
<HEAN1>[^ \t\n:]*           { strcpy(str1,yytext);
                              WCDEANTOORA(str1);
                            }
%{
/* ':' detected, scanning string 2.
*/
%}
```

```
<HEAN1>[:]                  { BEGIN HEAN2;
                            }
%{
/* Detecting end of find request. No string 2 => WCHAR.
   Building the request if string 1 non-empty.
   Back to detection of string 1
*/
%}
<HEAN1>[ \t]*[;]            { strcpy(str2,WCHAR);
                              if (strcmp(str1,"")!=0) BuildReq();
                              BEGIN HEAN1;
                            }
%{
/* End of subject field detected => end of find request.
   No string 2 => WCHAR.
   Building the request if string 1 non-empty.
   Back to Header analysis.
*/
%}
<HEAN1>\n                   { strcpy(str2,WCHAR);
                              if (strcmp(str1,"")!=0) BuildReq();
                              unput('\n');
                              BEGIN HDR;
                            }
%{
/* Non end of subject field detected : skipping.
*/
%}
<HEAN1>\n[ \t]              { unput(' ');
                            }

%{
/* Skipping blanks.
*/
%}
<HEAN2>[ \t]*               ;
%{
/* Detecting string 2, only the first one after the ':' is considered.
   The others are skipped. Conversion of the wild card.
   If no string 1 => string 1 = Wchar. Building the request.
   Beginning HEAN3 to skip until end of request detected.
*/
%}
<HEAN2>[^ \t\n:]*           { strcpy(str2,yytext);
                              WCDEANTOORA(str2);
                              if (strcmp(str1,"")==0) strcpy(str1,WCHAR);
                              BuildReq();
                              BEGIN HEAN3;
                            }
%{
/* Detecting end of find request. No string 2 => WCHAR.
   Building the request if string 1 non-empty.
   Back to detection of string 1
*/
%}
<HEAN2>[ \t]*[;]            { strcpy(str2,WCHAR);
                              if (strcmp(str1,"")!=0) BuildReq();
                              BEGIN HEAN1;
                            }
%{
/* End of subject field detected => end of find request.
   No string 2 => WCHAR.
   Building the request if string 1 non-empty.
   Back to Header analysis.
```

```
*/
%}
<HEAN2>\n                    { strcpy(str2,WCHAR);
                               if (strcmp(str1,"")!=0) BuildReq();
                               unput('\n');
                               BEGIN HDR;
                             }
%{
/* Non end of subject field detected : skipping.
*/
%}
<HEAN2>\n[ \t]               { unput(' ');
                             }

%{
/* Skipping other caracters.
*/
%}
<HEAN3>.                     ;
%{
/* End of request detected, back to detection of string 1.
*/
%}
<HEAN3>[ \t]*[;]             { BEGIN HEAN1;
                             }
%{
/* End of subject field detected : back to header analysis.
*/
%}
<HEAN3>\n                    { unput('\n');
                               BEGIN HDR;
                             }
%{
/* Non end of subject field detected : skipping.
*/
%}
<HEAN3>\n[ \t]               { unput(' ');
                             }

%{
/* ****************************************************************
   *** The following is the body analysis, similar to the       ***
   *** subject analysis except that BEGIN intructions are       ***
   *** directed elsewhere, and that no end of fields has to     ***
   *** be detected.                                             ***
   ****************************************************************
*/
%}
%{
/* ****************************************************************
   *** BODY body analysis                                       ***
   ****************************************************************
*/
%}
<BODY>.                                    ;
<BODY>\n                                   ;
<BODY>[ \t]*[tT][eE][lL][lL][ \t\n]        ;
<BODY>[ \t]*[eE][mM][dD][iI][rR][ \t\n]    ;
<BODY>[ \t]*[qQ][uU][eE][rR][yY][ \t\n]    { if (syntype!=2) {
                                               syntype=1;
                                               if (nreq<MaxReq) {
                                                 ireq=(char*)malloc(ReqLen);
                                                 reqtype=2;
                                                 parcnt=0;
```

```
                                                    percentq=0;
                                                    if (ireq!=NULL) {
                                                       nreq++;
                                                       req[nreq]=ireq;
                                                       strcpy(req[nreq],'\0');
                                                       BEGIN BEMDIR;
                                                       unput(yytext[yyleng-1]);
                                                  } } }
                                                }
<BODY>[ \t]*[iI][nN][fF][oO][ \t\n]               |
<BODY>[ \t]*[hH][eE][lL][pP][ \t\n]               { helpreq=TRUE;
                                                }
<BODY>[ \t]*[fF][iI][nN][dD][ \t\n]               { if (syntype!=1) {
                                                    syntype=2;
                                                    reqtype=3;
                                                    strcpy(str1,"");
                                                    strcpy(str2,"");
                                                    unput(yytext[yyleng-1]);
                                                    BEGIN BEAN1;
                                                  }
                                                }

%{
/* ****************************************************************
   *** BEMDIR emdir syntax analysis                               ***
   ****************************************************************
*/
%}
<BEMDIR>\n                      ;
<BEMDIR>[ \t]*                  ;
<BEMDIR>[ \t]*[;]               { if ((parcnt==0) || (percentq==0))
                                                  { free(req[nreq]);
                                                    nreq--;
                                                  }
                                  BEGIN BODY;
                                }
<BEMDIR>[^ \t\n]*               { WCDEANTOORA(yytext);
                                  strcat(req[nreq],yytext);
                                  strcat(req[nreq]," ");
                                  parcnt++;
                                  if (strcmp(WCHAR,yytext)!=0) percentq++;
                                  if (parcnt>=MaxPar) { if (percentq==0)
                                                          { free(req[nreq]);
                                                            nreq--;
                                                          }
                                                        BEGIN BODY;
                                                      }
                                }

%{
/* ****************************************************************
   *** BEAN1 2 3 EAN syntax analysis                              ***
   *** 1 picks the first string                                   ***
   *** 2 the second one                                           ***
   *** 3 detects end of request                                   ***
   *** state is the last state when an EOF was detected           ***
   ****************************************************************
*/
%}
<BEAN1>\n                       { state=1;
                                }
<BEAN1>[ \t]*                   { state=1;
                                }
<BEAN1>[^ \t\n:]*               { state=1;
```

```
                          strcpy(str1,yytext);
                          WCDEANTOORA(str1);
                         }
<BEAN1>[:]                { state=1;
                          BEGIN BEAN2;
                         }
<BEAN1>[ \t\n]*[;]        { state=1;
                          strcpy(str2,WCHAR);
                          if (strcmp(str1,"")==0) strcpy(str1,WCHAR);
                          BuildReq();
                          BEGIN BEAN1;
                         }

<BEAN2>\n                 { state=1;
                         }
<BEAN2>[ \t]*             { state=1;
                         }
<BEAN2>[^ \t\n:]*         { strcpy(str2,yytext);
                          WCDEANTOORA(str2);
                          if (strcmp(str1,"")==0) strcpy(str1,WCHAR);
                          BuildReq();
                          BEGIN BEAN3;
                          state=3;
                         }
<BEAN2>[ \t\n]*[;]        { strcpy(str2,WCHAR);
                          if (strcmp(str1,"")==0) strcpy(str1,WCHAR);
                          BuildReq();
                          BEGIN BEAN1;
                          state=1;
                         }
<BEAN3>.                  { state=3;
                         }
<BEAN3>[ \t\n]*[;]        { BEGIN BEAN1;
                          state=1;
                         }
<BEAN3>\n                 { state=3;
                         }


%%

yywrap()
{extern int  nfind;
 extern int  nreq;
 extern char *req[];
 extern char addresse[];
 extern int  reqtype;
 extern int  state;
 extern char str1[];
 extern char str2[];
 int i;
 char ad[AdrLen];

 /* If we left in state 1, this means that no end of request has been
    detected before EOF in find, so the request has not been build and this
    has to be done. No string 2 detected.
 */
 if (state==1) {
    strcpy(str2,WCHAR);
    if (strcmp(str1,"")!=0) BuildReq();
    }


 if (helpreq) reqtype=1;
```

```
 /* Address to reject */
 strcpy(ad,addresse);
 STRTOLOCASE(ad);
 if (STRMATCH(ad,"daemon")) reqtype=5 ;
 if (STRMATCH(ad,"uucp@")) reqtype=5 ;
 if (STRMATCH(ad,"mailer")) reqtype=5 ;
 if (STRMATCH(ad,"listserv")) reqtype=5 ;
 if (STRMATCH(ad,"kermsrv")) reqtype=5 ;
 if (strcmp(ad,"")==0) reqtype=5;

 if (reject) reqtype=6;

 /* Writing the address */
 printf("AD- %s\n",addresse);

 /* Writing the request type or error diagnostic */
 if ((reqtype==2) && (nreq!=0)) printf("TQ- %d Query \n",nreq);
    else if ((reqtype==3) && (nfind!=0)) printf("TF- %d Find \n",nfind);
    else if  (reqtype==5) { nreq=0; printf("TR- %d Rejected on address \n",nreq);}
    else if  (reqtype==6) { nreq=0; printf("TR- %d Rejected on subject \n",nreq);}
    else if  (reqtype==1) { nreq=0; printf("TH- %d Help \n",nreq);}
    else { nreq=0; printf("TH- %d Syntax Error \n",nreq);};

 /* Writing the find requests */
 for (i=1;i<=nfind;i++) printf("F%d- %s\n",i,find[i]);

 /* Writing the query requests */
 for (i=1;i<=nreq;i++) printf("Q%d- %s\n",i,req[i]);

 /* LEX return code */
 return(1);
}

WCDEANTOORA(st)
/*  '*'=>'%'  and  '?'=>'_'  */
/*  strip redundant wildchar */
char *st;
{ char *i,*cc,*sw;
  for(i=st;*i;i++) if (*i=='*') *i='%';
  for(i=st;*i;i++) if (*i=='?') *i='_';
  for(i=cc=sw=st;*i;)
     if ((*i!='%')&&(*i!='_')) *cc++ = *i++ ;
     else if (*i=='%') { *cc++ = *i++ ;
                         for(;((*i=='%')||(*i=='_'));i++);
                       }
     else { sw = cc ; *cc++ = *i++ ;
            for(;*i=='_';*cc++ = *i++);
            if (*i=='%') cc=sw;
          }
  *cc='\0';
}

STRMATCH(ls,ss)
 /* Check if string 'ss' is a part of 'ls' */
char ls[];
char ss[];
{ int i,j;
  int ite1,ite2;
  int ok;

  ite1=strlen(ls)-strlen(ss);
  ite2=strlen(ss)-1;
  for (i=0;i<=ite1;i++)
```

```
    { ok=TRUE;
      for (j=0;j<=ite2;j++) if (ls[i+j]!=ss[j]) {ok=FALSE;break;}
      if (ok==TRUE) return(TRUE);
    }
  return(FALSE);
}

STRTOLOCASE(s)
 /* Convert a string to lowercase */
char s[];
{ int i;
  for (i=0;i<=strlen(s)-1;i++) s[i]=tolower(s[i]);
}

BuildReq()
{extern int  nfind;
 extern char str1[];
 extern char str2[];
 char *ireq ;

 if (nfind>=MaxFind) return;
 if ((strcmp(str1,WCHAR)==0) && (strcmp(str2,WCHAR)==0)) return;
 ireq=(char*)malloc(ReqLen);
 nfind++;
 if (ireq!=NULL) find[nfind]=sprintf(ireq,"%s:%s",str1,str2);
 strcpy(str1,"");strcpy(str2,"");
}
```

```
%START BEG NBER NO NAME FNAME DIV PHONE EXP INST INSTPH MAIL CMT END

%{
/* ************************************************************************
   Met en page les donnees contenues dans le fichier produit par MED.emdir.
   ************************************************************************
*/
%}

%{
# define MAXENTRIES        20 /* Max Number of Entries Mise En Page */
# define MAXSTR           100 /* Max Lgth of a string */
%}

 int nber;           /* Nber of records */
 int nent;           /* Number of Entries Mise En Page */
 int no  ;           /* Current record number */

 char name[MAXSTR];
 char firstname[MAXSTR];
 char cdiv[MAXSTR];
 char cphone[MAXSTR];
 char cexp[MAXSTR];
 char inst[MAXSTR];
 char iphone[MAXSTR];
 char mail[MAXSTR];
 char comment[MAXSTR];

%%

 extern int nber;
 extern int no  ;

 extern char name[MAXSTR];
 extern char firstname[MAXSTR];
 extern char cdiv[MAXSTR];
 extern char cphone[MAXSTR];
 extern char cexp[MAXSTR];
 extern char inst[MAXSTR];
 extern char iphone[MAXSTR];
 extern char mail[MAXSTR];
 extern char comment[MAXSTR];

 nber=no=nent=0;
 name[0]=firstname[0]=cdiv[0]=cphone[0]=cexp[0]='\0';
 inst[0]=iphone[0]=mail[0]=comment[0]='\0';

 BEGIN BEG;

^"ERR".*             ;

<BEG>.               ;
<BEG>\n              ;
<BEG>##              { BEGIN NBER; }

<NBER>[0123456789]   { nber=nber*10+(yytext[0]-'0');}
<NBER>.              ;
<NBER>\n             ;
<NBER>$#             { BEGIN NO; }

<NO>[0123456789]     { no=no*10+(yytext[0]-'0');}
<NO>.                ;
<NO>\n               ;
<NO>"NM- "           { BEGIN NAME; }
```

```
<NAME>.                 { strcat(name,yytext);}
<NAME>\n                ;
<NAME>"FN- "            { BEGIN FNAME; }

<FNAME>.                { strcat(firstname,yytext); }
<FNAME>\n               ;
<FNAME>"CD- "           { BEGIN DIV; }

<DIV>.                  { strcat(cdiv,yytext); }
<DIV>\n                 ;
<DIV>"CP- "             { BEGIN PHONE;}

<PHONE>.                { strcat(cphone,yytext); }
<PHONE>\n               ;
<PHONE>"CE- "           { BEGIN EXP; }

<EXP>.                  { strcat(cexp,yytext); }
<EXP>\n                 ;
<EXP>"HI- "             { BEGIN INST; }

<INST>.                 { strcat(inst,yytext); }
<INST>\n                ;
<INST>"IP- "            { BEGIN INSTPH;}

<INSTPH>.               { strcat(iphone,yytext); }
<INSTPH>\n              ;
<INSTPH>"MA- "          { BEGIN MAIL;}

<MAIL>.                 { strcat(mail,yytext); }
<MAIL>\n                ;
<MAIL>"CT- "            { BEGIN CMT;}

<CMT>.                  { strcat(comment,yytext); }
<CMT>\n                 ;
<CMT>$#                 { mep();
                          if (nent < MAXENTRIES) BEGIN NO;
                                                 else BEGIN END;
                        }

<END>.                  ;
<END>\n                 ;

%%

yywrap()
{ if (no!=0) mep();
  if (nber>0) printf("%d out of %d matching entries.\n",nent,nber);
  return(1);
};

mep()
{
 extern int nber;
 extern int nent;
 extern int no  ;

 extern char name[MAXSTR];
 extern char firstname[MAXSTR];
 extern char cdiv[MAXSTR];
 extern char cphone[MAXSTR];
 extern char cexp[MAXSTR];
 extern char inst[MAXSTR];
 extern char iphone[MAXSTR];
```

```
 extern char mail[MAXSTR];
 extern char comment[MAXSTR];

 if (strcmp(name,"")==0) strcpy(name,"\\");
 if (strcmp(firstname,"")==0) strcpy(firstname,"\\");
 if (strcmp(cdiv,"")==0) strcpy(cdiv,"\\");
 if (strcmp(cexp,"")==0) strcpy(cexp,"\\");
 if (strcmp(cphone,"")==0) strcpy(cphone,"\\");
 if (strcmp(inst,"")==0) strcpy(inst,"\\");
 if (strcmp(iphone,"")==0) strcpy(iphone,"\\");
 if (strcmp(mail,"")==0) strcpy(mail,"\\");
 if (strcmp(comment,"")==0) strcpy(comment,"\\");

 printf("%s %s\n",name,firstname);
 printf("  Div: %s   Exp: %s  Cern phone (ext): %s\n",cdiv,cexp,cphone);
 printf("  Home Institute: %s   Phone: %s\n",inst,iphone);
 printf("  E-Mail Address: %s\n",mail);
 printf("  Comment: %s\n",comment);
 printf("\n");

 no=0;nent++;
 name[0]=firstname[0]=cdiv[0]=cphone[0]=cexp[0]='\0';
 inst[0]=iphone[0]=mail[0]=comment[0]='\0';

}
```

Example of message request sent to the NameServer

```
Message inbox:4 - Sent
From :    Bernard Heuse <S=heuse;OU=ts;O=info;P=fundp;A=rtt;C=be>
To:       Nameserver <S=nameserver;O=cernvax;P=cern;A=arcom;C=ch>
Subject:  Find heuse

find dimou:dd
```

NameServer reply message

```
Message inbox:5 - Read
From :    CERN EmDir Name Server
          <S=cernvax!nameserver;O=prlb2;P=uucp;A=rtt;C=be>
To:       Bernard Heuse <S=heuse;OU=ts;O=info;P=fundp;A=rtt;C=be>
Subject:  Find: EmDir Request Answer.

>Errors-To: <cernvax!priam.CERN!bothner@prlb2.UUCP>

-------------------------------------------------------------------
Send a mail to <namesever@cernvax.cern.ch> with the keyword 'Help'
  as subject to receive more information about this service and the
  syntax of the EmDir Request Answer.
-------------------------------------------------------------------
Find Syntax :
   Find [<string1|*>]:[<string2|*>]
     where <string1> is a name or a firstname,
           <string2> is an experiment, a division or an institute.
-------------------------------------------------------------------
Find HEUSE
HEUSE Bernard
  Div: DD            Exp: -            Cern phone (ext): -
  Home Institute: Namur Univ.    Phone: -
  E-Mail Address: heuse@ts.info.fundp.rtt.be
  Comment: -

1 out of 1 matching entrie(s).
-------------------------------------------------------------------
Find DIMOU:DD
DIMOU Maria
  Div: DD            Exp: -            Cern phone (ext): 3356
  Home Institute: CERN          Phone: -
  E-Mail Address: dimou@priam.cern.ch
  Comment: Communication Systems Group - External Networking Section

1 out of 1 matching entrie(s).
-------------------------------------------------------------------
```

Annex E Code of the AutoRouter

Annex E is the code of the AutoRouter described in Chapter 14.

The following files are given:
- the **Makefile**;
- **ar.h** and **ar.c** are the AutoRouter program;
- **addcheck.c** is the address checker routine;
- **emnodes.c** accesses the NetNodes database to determine the E-Mail address domain;
- **ora.h** manages Oracle errors.

The file getadd.c is given in Annex D.

Emdir.pol describes the syntax defined to facilitate matching between E-Mail addresses and Emdir entries.

The last page shows an example of message rejected by the AutoRouter, and an example of message routed by the AutoRouter.

```
PRG        = ar
RPC        = emdir
CRBASE     = /usera/heuse/emdirtmp
CFLAGS     = -O -I$(CRBASE)/h -I.
USEROBJS   = $(PRG).o $(RPC)_client.o
LIBS       = $(CRBASE)/lib/libcr.a $(CRBASE)/lib/libnet.a
INCS       = $(PRG).h $(RPC).h
COURIERCC  = $(CRBASE)/compiler/courier
GETADD     = getadd
ADDCHECK   = addcheck
AC         = ac

all: $(PRG)

$(USEROBJS): $(LIBS) $(INCS)

$(RPC).h: $(GETADD).cr
        rm -f $(RPC).h
        $(COURIERCC) $(GETADD).cr

$(PRG): $(USEROBJS) $(GETADD).c $(ADDCHECK).c
        cc -o $(PRG) $(USEROBJS) $(LIBS)

clean:
        -rm -f *.o ,* emdir_*.c emdir.h *.old

ac: $(AC).c $(ADDCHECK).c
        cc -o $(AC) $(AC).c

receive:
        mv $(GETADD).cr $(GETADD).cr.old
        cp "../libc/$(GETADD).cr" .
        mv $(ADDCHECK).c $(ADDCHECK).c.old
        dcp "vxcern/heuse/[redacted]::[heuse.addcheck]$(ADDCHECK).c" $(ADDCHECK).c
        mv $(GETADD).c $(GETADD).c.old
        cp "../libc/$(GETADD).c" .
        mv strings.c strings.c.old
        dcp "vxcern/heuse/[redacted]::[heuse.addcheck]strings.c" strings.c
        mv $(AC).c $(AC).c.old
        dcp "vxcern/heuse/[redacted]::[heuse.addcheck]$(AC).c" $(AC).c
```

```
/* Configuration File */

/* Status Codes */
# define STNULL         0
# define STERRDIVERS    1
# define STNOADDR       2
# define STONEADDR      3
# define STNOTOKADDR    4
# define STMOREADDR     5
# define STNOUSER       6
# define STTRYLATER    75

/* SendMail Return codes */
#define SMOK            0
#define SMUSAGE        64
#define SMNOUSER       67
#define SMSOFTERR      70
#define SMRETRY        75

/* Emdir errors definition */
#define EMDIR_BAD_QID         50000  /* Bad query id - should not happen */
#define EMDIR_NO_MEM          50001  /* Host unable to malloc            */
#define EMDIR_TOO_MANY_ROWS   50002  /* too many rows retrieved          */
#define EMDIR_UKN_RPC         50003  /* UNKOWN RPC - SHOULD NOT HAPPEN   */
#define EMDIR_BAD_PSW         50004  /* Bad password given for update    */
#define EMDIR_BAD_VER         50005  /* Emdir server and client have     */

/* Error codes */
#define ERRNULL               00000  /* No error */
#define ERRIVDREQ             10001  /* Invalid request */
#define ERRNOADDR             10002  /* No address */
#define ERRNOOKADDR           10003  /* No in list */
#define ERRMOREADDR           10004  /* More than 1 address */
#define ERRNOSERVER           10005  /* No server available*/

/* Boolean */
#define bln int
#define NO   0
#define YES  1

/* Oracle wild card */
#define ORAWC '%'          /* Oracle's wild card character */

/* variables lengths */
# define MaxAddr 150       /* Max length of an address                */
# define MaxCtry   3       /* Max length of a country name            */
# define MaxInst  10       /* Max length of an institute name         */
# define MaxDiv    5       /* Max length of a division name           */
# define MaxExp   10       /* Max length of an experiment name        */
# define MaxName  30       /* Max length of a name                    */
# define MaxFN    40       /* Max length of a firstname               */
# define MaxInit   5       /* Max nber of accepted initials           */

/* Directory for execution */
# define DIRECTORY          "/usera/heuse/autoroute"

# define LDIV "divisions"   /* File containing the division list      */
# define LEXP "experiments" /* File containing the experiment list    */
# define LOGFILE "ar.log"   /* Log file                               */

# define INST "CERN"        /* Accepted Institute                     */
# define CTRY "CH"          /* Accepted Country                       */

/* Maximum 20 rows will be given in case of ambiguous name */
```

```
#define MAX_ADDR_BACK              20

/* Name Server address */
#define NSADDR "<nameserver@cernvax.cern>"

#define NO_MAIL_ADDRESS_DEFINED_YET "(No Mail address)"
```

```
# include <stdio.h>
# include <time.h>

# include "addcheck.c"          /* Address test */

# include "getadd.c"            /* RPC Access */

# include "ar.h"                /* Configuration File */

static char *months[12] = {"Jan", "Feb", "Mar",
                           "Apr", "May", "Jun",
                           "Jul", "Aug", "Sep",
                           "Oct", "Nov", "Dec"};

  /* RPC DECLARATION */
extern Addresses     addresses;
  /* Maximum 10000 rows will be returned by RPC */
#define MAX_QUERY_LIMIT          10000
  /* RPC Codes : Actions definitions */
#define USAGE             8
#define BIND_FAILED      -1
#define ROW addresses.return_rows.sequence



/* Global Variables */
char *address;
char *from;
char routedto      [MaxAddr];
char initials      [MaxInit];
char firstname     [MaxFN];
char name          [MaxName];
char experiment    [MaxExp];
char division      [MaxDiv];
char institute     [MaxInst];
char country       [MaxCtry];
char *qname;
char *qfirstname;
char *qcern_div;
char *qexp;
char *qinst_mnemo;
char *qmint_addr;


int exitc  = 0        ;  /* exit code */
int status = STNULL   ;  /* no value */
int error  = ERRNULL  ;  /* no value */

bln qok    = YES      ;  /* valid query */
bln serok  = YES      ;  /* server available */
int garc              ;  /* getaddress ret code */

bln smex   = NO       ;  /* Sendmail not executed */
int smrc   = 0        ;  /* sendmail return code */

int count  = 0        ;  /* Nber of queries retrieved */
bln tmrow  = NO       ;  /* too many rows */


GetAddr()
/*
   Realise le RPC et remplit 'addresses'
*/
{
```

```
  garc=GetAddresses(qname,qfirstname,qcern_div,qexp,
                    qinst_mnemo,qmint_addr,USAGE);
  if (garc==BIND_FAIL) serok = NO;
                       serok = YES;
  if ((garc==QUERY_NOUSER) || (garc==QUERY_FULL)) qok = NO;
                                                  else qok = YES;
}

Dispatch()
/*
   Fait ce qu'il faut faire en fonction du 'addresses.status' en retour,
   de la disponibilite du serveur et du qid.
*/
{
strcpy(routedto,"(Not Forwarded)");
     if (qok == NO)   processnouser() ;
else if (serok == NO) processnoserver();
else if ((addresses.status==0) ||
         (addresses.status==1) ||
         (addresses.status==EMDIR_TOO_MANY_ROWS))  /* Status OK */
       { if (addresses.status==EMDIR_TOO_MANY_ROWS)
                    { count = MAX_QUERY_LIMIT ;
                      error = addresses.status;
                      tmrow = YES ;
                    }
                  else count = addresses.return_rows.length;
         if (count == 0 ) processnoaddr();
         if (count == 1 ) processoneaddr();
         if (count >  1 ) processmoreaddr();
       }
else switch(addresses.status)
    { case 0: break; /* OK */
      case 1: break; /* OK */
      case EMDIR_NO_MEM:
          status = STTRYLATER  ; /* Insufficient memory on host side */
          error  = addresses.status ;
          strcpy(routedto,"(Not Yet Routed)");
          break;
      default:
       error  = addresses.status ;
       strcpy(routedto,"(Not Forwarded)");
       if (addresses.status < 0)
          status = STERRDIVERS ; /* Oracle error on host */
       else
          status = STERRDIVERS ; /* Unknown error */
    }
}

processnoserver()
/*
   Pas de serveur => Try Later
*/
{   status = STTRYLATER;
    error  = ERRNOSERVER;
    strcpy(routedto,"(Not Yet Routed)");
}

processnouser()
/*
   Qid <= 0 ou pas de nom et prenom => No recipient
*/
{ status = STNOUSER    ;
  error  = ERRIVDREQ   ;
  strcpy(routedto,"(Not Forwarded)");
```

```
  perr_header("No recipient specified");
  perr_footer();
}

processnoaddr()
/*
   Pas d'entree trouvee => Renvoi un msg d'erreur
*/
{ status = STNOADDR ;
  error  = ERRNOADDR ;
  strcpy(routedto,"(Not Forwarded)");
  perr_header("Unknown recipient");
  perr_footer();
}

processoneaddr()
/*
   Une d'entree trouvee => Si l'E-Mail address est valide => Forward
                           Sinon, msg d'erreur.
*/
{ int rnber = 0 ;
  char str[1024];
  char adok[512];

  /* Checking if the recipient has a valid address */
  { int result,syn,net;

    result=addcheck(ROW[rnber].mint_addr,adok,&syn,&net);
    if ((result != 0) && (result != 4 /* No Domain */)) {
        perr_header("recipient has no valid address in the Emdir DB");
        fprintf(stdout,"\n Here follow the information retrieved about the recipient:");
        perr_rec(rnber);
        perr_footer();
        status = STNOTOKADDR ;
        error  = ERRNOOKADDR ;
        strcpy(routedto,"(Not Forwarded)");
        return;
    }
  }

  status = STONEADDR ;
  strcpy(routedto,adok);
  sprintf(str,"   echo    \"X-Rerouted-To: %s by CERN Automatic Router (Emdir DB).\" > /
  sprintf(str,"%s cat >> /tmp/ar$$ \n",str);
  sprintf(str,"%s /usr/lib/sendmail -f %s %s < /tmp/ar$$ \n",str,from,routedto);
  sprintf(str,"%s EXT=$? \n rm /tmp/ar$$ \n exit $EXT \n", str);
/*printf("%s",str); */
  smrc = system(str);
  smex = YES;
}

processmoreaddr()
/*
   Plus d'une entree trouvee => Msg d'erreur.
*/
{ int rnber;

  perr_header("ambiguous recipient");
  fprintf(stdout,"\n Here follow the information retrieved about the matching recipients
  if ((tmrow == YES) || (count > MAX_ADDR_BACK))
    fprintf(stdout,"\n (the list is not exhaustive)");
  for (rnber=0;rnber< ((count>MAX_ADDR_BACK)?MAX_ADDR_BACK:count) ;rnber++)
    perr_rec(rnber);
  fprintf(stdout,"\n");
```

```
  fprintf(stdout,"\n %d out of %d matching entries.",rnber,count);
  perr_footer();

  strcpy(routedto,"(Not Forwarded)");
  status = STMOREADDR ;
  error  = ERRMOREADDR ;
}

perr_header(errmsg)
char *errmsg;
{
fprintf(stdout,"\n CERN Emdir Automatic Router (Emdir DB).  ",address);
fprintf(stdout,"\n Automatic router failed. \"%s\" : %s. ",address,errmsg);
}

perr_footer()
{
fprintf(stdout,"\n");
fprintf(stdout,"\n Use the nameserver to retrieve a CERN user's E-Mail address.");
fprintf(stdout,"\n Send a mail to %s with the keyword ",NSADDR);
fprintf(stdout,"\n 'Help' as subject to receive more information about this service.");
fprintf(stdout,"\n\n");
}

perr_rec(rnber)
int rnber;
{
if (strcmp(ROW[rnber].name,"")==0) strcpy(ROW[rnber].name,"\\");
if (strcmp(ROW[rnber].exp,"")==0) strcpy(ROW[rnber].exp,"\\");
if (strcmp(ROW[rnber].cern_div,"")==0) strcpy(ROW[rnber].cern_div,"\\");
if (strcmp(ROW[rnber].cern_tel,"")==0) strcpy(ROW[rnber].cern_tel,"\\");
if (strcmp(ROW[rnber].inst_mnemo,"")==0) strcpy(ROW[rnber].inst_mnemo,"\\");
if (strcmp(ROW[rnber].inst_tel,"")==0) strcpy(ROW[rnber].inst_tel,"\\");
if (strcmp(ROW[rnber].comments,"")==0) strcpy(ROW[rnber].comments,"\\");

fprintf(stdout,"\n");
fprintf(stdout,"\n   %s %s",ROW[rnber].name,ROW[rnber].first_name);
fprintf(stdout,"\n    Div: %s   Exp: %s  Cern phone (ext): %s",ROW[rnber].cern_div,ROW[r
fprintf(stdout,"\n    Home Institute: %s   Phone: %s",ROW[rnber].inst_mnemo,ROW[rnber].i
fprintf(stdout,"\n    E-Mail Address: %s",ROW[rnber].mint_addr);
fprintf(stdout,"\n    Comment: %s",ROW[rnber].comments);
}

CptExitCode()
/*
   Calcul du code de retour (destine a SendMail).
*/
{ char str[1024];

  switch(status)
  { case STONEADDR :
      exitc = smrc / 256 ;
      break;
    case STTRYLATER :
      exitc = SMRETRY;
      break;
    case STNOUSER :
      exitc = SMUSAGE;
      break;
    case STNOTOKADDR :
      exitc = SMUSAGE;
      break;
    case STMOREADDR :
      exitc = SMUSAGE;
```

```
      break;
    case STNOADDR :
      exitc = SMNOUSER;
      break;
    default :
      exitc = SMSOFTERR;
      break;
  }
}

AnalyseAddress()
{
/* Analyse the address 'address' and put results into 'country',
   'institute', 'division', 'experiment', 'name', 'initials' and
   'firstname'.
*/

/* ************************************************************************
   Syntax accepted for the address :
   addressfield = address | [string]'<'address'>'[string]
   address = name['@'organization]
   organization = [string'.'][experiment'.'][division'.']['CERN.']['CH'] |
                  [string'.'][division'.'][experiment'.']['CERN.']['CH']
       where experiment in LEXP
         and division   in LDIV
   name = [firstname'.']MaxInit*[initial'.'][surname]
       where firstname does not contain full stop '.' and is
                at least two characters long.
             initial contains only one letter.
             if surname contains full stop, then it may not be in the
                first two characters, and either initials or given
                name or both are present.
   string are ignored.
   ************************************************************************ */

int i,j;
char *b,*e,*c,*an,*ao;
char add[MaxAddr];
char tmp[MaxAddr];
char a_name[MaxAddr];
char a_orga[MaxAddr];
FILE *f;
bln  okdom;

tmp[0]=initials[0]=firstname[0]=name[0]='\0';
experiment[0]=division[0]=institute[0]=country[0]='\0';
strcpy(add,address);

/* taking the address out of the address field (into <> if any) */
   if (((b=rindex(add,'<'))!=0) && ((e=index(add,'>'))!=0) && (b<e))
        { strncpy(add,b+1,e-b-1);
          add[e-b-1]='\0';
        }

/* to uppercase and suppressing leading and trailing blanks */
   for (b=add;*b==' ';b++);
   for (i=strlen(b)-1;((i>=0) && (b[i]==' '));b[i--]='\0');
   for (i=0;i<strlen(b);b[i++]=toupper(b[i]));
   strcpy(add,b);

/* separating Name and Organization in Name@Organisation */
if ((b=index(add,'@'))==0) { strcpy(a_name,add);
                             a_orga[0]='\0';
                           }
```

```
                    else { *b='\0';
                           strcpy (a_name,add);
                           strcpy (a_orga,b+1);
                         }

/* backward scanning of the organization */
   /* Only CH as country is searched */
   if (strlen(a_orga)!=0) {
      if (a_orga[strlen(a_orga)-1]=='.') a_orga[strlen(a_orga)-1]='\0';
      if ((c = rindex(a_orga,'.') )==0) c=a_orga; else c++;
      if (strcmp(c,CTRY)==0) { strcpy(country,c); *c='\0'; }
   }
   /* Only CERN as institute is searched */
   if (strlen(a_orga)!=0) {
      if (a_orga[strlen(a_orga)-1]=='.') a_orga[strlen(a_orga)-1]='\0';
      if ((c = rindex(a_orga,'.') )==0) c=a_orga; else c++;
      if (strcmp(c,INST)==0) { strcpy(institute,c); *c='\0'; }
   }
   /* Scanning Division and Experiment */
      /* First String */
   if (strlen(a_orga)!=0) {
      if (a_orga[strlen(a_orga)-1]=='.') a_orga[strlen(a_orga)-1]='\0';
      if ((c = rindex(a_orga,'.') )==0) c=a_orga; else c++;

      f=fopen(LDIV,"r");okdom=NO;
      if (f==NULL) perror("Cannot open DIVISION file");
      while ((fgets(tmp,100,f))!=NULL)
        { tmp[strlen(tmp)-1]='\0';
          rmltmbl(tmp);
          if(strcmp(tmp,c)==0) okdom=YES;
        }
      fclose(f);

      if (okdom==YES) { strcpy(division,c); *c='\0'; }
      else { f=fopen(LEXP,"r");okdom=NO;
             if (f==NULL) perror("Cannot open EXPERIMENT file");
             while ((fgets(tmp,100,f))!=NULL)
             { tmp[strlen(tmp)-1]='\0';
               rmltmbl(tmp);
               if(strcmp(tmp,c)==0) okdom=YES;
             }
             fclose(f);
             if (okdom) { strcpy(experiment,c); *c='\0'; }
           }
   }
      /* Second string */
   if (strlen(a_orga)!=0) {
      if (a_orga[strlen(a_orga)-1]=='.') a_orga[strlen(a_orga)-1]='\0';
      if ((c = rindex(a_orga,'.') )==0) c=a_orga; else c++;

      f=fopen(LDIV,"r");okdom=NO;
      if (f==NULL) perror("Cannot open DIVISION file");
      while ((fgets(tmp,100,f))!=NULL)
        { tmp[strlen(tmp)-1]='\0';
          rmltmbl(tmp);
          if(strcmp(tmp,c)==0) okdom=YES;
        }
      fclose(f);

      if (okdom==YES) { strcpy(division,c); *c='\0'; }
      else { f=fopen(LEXP,"r");okdom=NO;
             if (f==NULL) perror("Cannot open EXPERIMENT file");
             while ((fgets(tmp,100,f))!=NULL)
             { tmp[strlen(tmp)-1]='\0';
```

```
                rmltmbl(tmp);
                if(strcmp(tmp,c)==0) okdom=YES;
              }
              fclose(f);
              if (okdom) { strcpy(experiment,c); *c='\0'; }
            }
    }

/* forward scanning of the name */
an = a_name ;
/* first string */
if (strlen(an)!=0)
    {
      c = index(an,'.');
      if (c==0) { strcpy(name,an); *an = '\0' ; }
      else if (c-an>1)  { *c = '\0';
                          strcpy(firstname,an);
                          an = c+1;
                        }
      else if (c-an==1) { *c = '\0';
                          strcat(initials,an);
                          an = c+1;
                        }
      else { strcpy(name,an); *an = '\0' ; }
    }
/* initials */
if (strlen(an)!=0)
    {
     c = index(an,'.');
     if (c==0) { strcpy(name,an); *an = '\0' ; }
     else while (c-an==1) { *c = '\0';
                            if (strlen(initials)<MaxInit) strcat(initials,an);
                            an = c+1;
                            c = index(an,'.');
                            if (c==0) { strcpy(name,an); *an = '\0' ; break ; }
                          }
    }
/* name */
if (strlen(an)!=0) { strcpy(name,an); *an = '\0' ; }

}

BuildRequest()
{
/* Build the request for the RPC call, initialization of 'query_row'
   according to the rules described hereafter and the content of
   the variables 'country', 'institute', 'division', 'experiment',
   'name', 'initials' and 'firstname'.
   Also converts wildchar and compute qid.
*/

/* ************************************************************************
   qname = if (name<>'') then name
                         else ''
   qfirstname = if (firstname<>'') then firstname
                                   else if (initials<>'') then initials[0]%
                                                          else ''
   qcern_div = if (((qname='') and (qfirstname='')) or (division=''))
                  then ''
                  else division
   qexp = if (((qname='') and (qfirstname='')) or (experiment=''))
                  then ''
                  else experiment
   qinst_memo = ''
```

```
   qmint_addr = ''

   ********************************************************************** */

static char nullstring[]={"\0"};

if (strlen(name)!=0) qname = name;
                else qname = nullstring;

if (strlen(firstname)!=0)     qfirstname = firstname;
else if (strlen(initials)!=0) qfirstname = sprintf(malloc(MaxFN),"%c%c",initials[0],ORAW
                         else qfirstname = nullstring;

if (((strlen(qname)==0) && (strlen(qfirstname)==0)) || (strlen(division)==0))
                         qcern_div = nullstring;
                    else qcern_div = division;

if (((strlen(qname)==0) && (strlen(qfirstname)==0)) || (strlen(experiment)==0))
                         qexp = nullstring;
                    else qexp = experiment;

qinst_mnemo = nullstring;

qmint_addr = nullstring;

}


LogFile()
{
char query[MaxAddr];
char parsing[MaxAddr];
char errmsg[100];
char smst[100];
char rpcst[100];
char vdate[10];
char vtime[10];
int  btime;
struct tm *tstr;
FILE * logFile = NULL;

time(&btime);
tstr = localtime(&btime);
sprintf(vdate,"%d-%s-%d",tstr->tm_mday,
                         months[tstr->tm_mon],
                         tstr->tm_year);
sprintf(vtime,"%d:%d:%d",tstr->tm_hour,
                         tstr->tm_min,
                         tstr->tm_sec);

sprintf(parsing,"- %s - %s - %s -@- %s - %s - %s - %s -",
                firstname,initials,name,
                experiment,division,institute,country);
sprintf(query,"- %s - %s - %s - %s - %s - %s -",
                qname,qfirstname,
                qcern_div,qexp,
                qinst_mnemo,qmint_addr);
switch(status)
{ case STNULL :
                strcpy(errmsg,"Null Error");
                break ;
  case STERRDIVERS :
                strcpy(errmsg,"Error : msg not forward");
                break ;
```

```
  case STNOADDR :
                strcpy(errmsg,"No address found");
                break ;
  case STNOUSER :
                strcpy(errmsg,"No recipient specified");
                break ;
  case STONEADDR :
                strcpy(errmsg,"Msg forward");
                break ;
  case STNOTOKADDR :
                strcpy(errmsg,"No valid address in Emdir");
                break ;
  case STMOREADDR :
                strcpy(errmsg,"More than one address found");
                break ;
  case STTRYLATER :
                strcpy(errmsg,"Server machine not available, will retry");
                strcpy(routedto,"Not Yet Routed");
                break ;
  default :
                strcpy(errmsg,"Unknown status number");
                break ;
}

logFile = fopen (LOGFILE, "a+");
if (logFile==NULL) perror("Cannot open log file");

if ((garc==QUERY_NOUSER) || (garc==QUERY_FULL)) strcpy(rpcst,"(Invalid Query)");
else if (garc==BIND_FAIL) strcpy(rpcst,"(Server unavailable)");
else strcpy(rpcst,"");

if (smex == NO)  strcpy(smst,"(Not Executed)");
           else strcpy(smst,"");

fprintf(logFile,"- %s ; %s : F=%s T=%s FT=%s \n      P=%s Q=%s C=%d\n      S=%d(%s) E=%d
                  vdate,vtime,
                  from,address,routedto,
                  parsing,query,count,
                  status,errmsg,error,
                  smrc,smst,exitc,garc,rpcst);

close (logFile);
}

GetArg(argc,argv)
int argc;
char *argv[];
{
address = argv[2];
from    = argv[1];
}

main(argc,argv)
int argc;
char *argv[];
{
if (chdir(DIRECTORY)) perror("Cannot change dir");
GetArg(argc,argv);
AnalyseAddress();
BuildRequest();
GetAddr();
Dispatch();
CptExitCode();
LogFile();
```

```
exit(exitc);
}
```

```
From heuse@vxcern.decnet  Thu Jan 19 13:44:25 1989
Date: Thu, 19 Jan 89 13:44:25 +0100
Message-Id: <8901191244.AA17265@cernvax.uucp>
From: heuse@vxcern.decnet
To: printer@priam.decnet
Subject:


/* Compiler Directives */

#define DB-ACCESS               0
/* #define DEBUG                   0  */

/* Local Definitions */

/* File of Divisions and experiments */
#define CDFILE          "cern.dom"

/* String size */
#define MAXSIZE               256

/* Illegal char set in addresses */
#define ILLCHARSET ",$~'; {}=^*<>"

/* Diagnostic bits */
#define DG_CHANGEDUUCP          2
#define DG_CHANGEDDEC           4
#define DG_ANGLEUNSYM           8
#define DG_CHANGEDAT           16
#define DG_INVJANET            32
#define DG_ADOM                64
#define DG_ORA_ERROR          128
#define DG_ILLCHAR            256

/* Network codes */
#define IS_NOT_DET            000
#define IS_UNKNOWN            100
#define IS_NETNORTH           200
#define IS_EARN               300
#define IS_DECNET             400
#define IS_CERN               500
#define IS_JANET              600
#define IS_BITNET             700
#define IS_UUCP               800
#define IS_ADDED_DB             1
#define IS_ADDED_SY             2
#define IS_INADD                0

/* Return codes and diagnostic */
#define RC_OK                   0
#define RC_ILLEGALCHAR          1
#define RC_NOHOST               2
#define RC_NOUSER               3
#define RC_NODOM                4
#define RC_NOADD                5
#define RC_ILLCD                6

/* DB acces routines */
#ifdef DB-ACCESS
#include "emnodes.c"
#endif DB-ACCESS
/* String routines */
#include "strings.c"
```

```
#include <stdio.h>


printdiag(addin,outstring,result,net,rc)
char addin[],outstring[];int *result,*net,*rc;
{ int diag;
    printf ("\nAddress Test Diagnostic :\n");
    printf ("    IN  : '%s'\n    OUT : '%s' \n",addin,outstring);
    printf ("    Add=%d - Net=%d - Err=%d \n    ",*result,*net,*rc);

    if (*result == 0 )                printf ("Add Syntax OK");
    if (*result & DG_CHANGEDDEC)      printf ("Ch_Decnet ");
    if (*result & DG_CHANGEDUUCP)     printf ("Ch_Uucp ");
    if (*result & DG_ANGLEUNSYM)      printf ("<>_Unsym ");
    if (*result & DG_INVJANET)        printf ("Inv_Janet ");
    if (*result & DG_ADOM)            printf ("Add_Domain ");
    if (*result & DG_CHANGEDAT)       printf ("Ch_AT_@ ");
    if (*result & DG_ORA_ERROR)       printf ("Ora_Error ");
    if (*result & DG_ILLCHAR)         printf ("Ill_Char ");
    printf ("\n    ");

    if (*result & DG_ORA_ERROR)  printf ("Oracle Error %d",*net);
    else {
    diag = *net % 100 ; *net = (*net / 100) * 100;
    if (*net == IS_NOT_DET   ) printf ("Network not determined ");
    if (*net == IS_UNKNOWN   ) printf ("Network UNKNOWN ");
    if (*net == IS_EARN      ) printf ("Network EARN    ");
    if (*net == IS_BITNET    ) printf ("Network BITNET  ");
    if (*net == IS_NETNORTH  ) printf ("Network NETNORTH");
    if (*net == IS_UUCP      ) printf ("Network UUCP    ");
    if (*net == IS_DECNET    ) printf ("Network DECNET  ");
    if (*net == IS_JANET     ) printf ("Network JANET   ");
    if (*net == IS_CERN      ) printf ("Network CERN    ");
    if (diag == IS_INADD    ) printf ("(In address) ");
    if (diag == IS_ADDED_SY ) printf ("(Derived from syntax) ");
    if (diag == IS_ADDED_DB ) printf ("(Found in Netnodes) ");
    }
    printf ("\n");

    printf("> ");
    if (*rc == RC_OK          ) printf ("Address OK");
    if (*rc == RC_ILLEGALCHAR) printf ("Rejected: Illegal char %s",ILLCHARSET);
    if (*rc == RC_NODOM       ) printf ("Rejected: No Domain specified");
    if (*rc == RC_NOHOST      ) printf ("Rejected: No Host specified");
    if (*rc == RC_NOUSER      ) printf ("Rejected: No User specified");
    if (*rc == RC_NOADD       ) printf ("Rejected: No Address specified");
    if (*rc == RC_ILLCD       ) printf ("Rejected: Illegal Cern Sub-Domain");
    printf ("\n");
}

addcheck (instring,outstring,add_diag,net_diag)
char *instring,*outstring;
int  *add_diag,*net_diag;
{
int  i;
char *b,*e,
     *start;
char add     [MAXSIZE],
     user    [MAXSIZE],
     node    [MAXSIZE], adnode   [MAXSIZE],
     domain  [MAXSIZE], addomain [MAXSIZE], cerndom[MAXSIZE];
     int  net,adnet,oraerr,rc;

   rc = RC_OK ;
```

```
   *add_diag = *net_diag = adnet = oraerr = 0;
   strcpy(add,instring);
   node[0]=user[0]=domain[0]='\0';

#ifdef DEBUG
printf("IN    : %s\n",add);
#endif DEBUG

   /* taking the address out of the address field (into <> if any) */
   b=rindex(add,'<');
   e=index(add,'>');
   if ((b!=0) && (e!=0) && (b<e))  { strncpy(add,b+1,e-b-1);
                                     add[e-b-1]='\0';
                                   }
   else if ((b!=0) || (e!=0)) *add_diag += DG_ANGLEUNSYM;

#ifdef DEBUG
printf("<>    : %s\n",add);
#endif DEBUG

   /* suppressing useless blanks */
   rmltmbl(add);

#ifdef DEBUG
printf("BL-   : #%s#\n",add);
#endif DEBUG

   /* from "user AT host" or "user @ host" to "user@host" */
   if (((e=strmatch(add," AT "))!=0)
      || ((e=strmatch(add," At "))!=0)
      || ((e=strmatch(add," at "))!=0)
      || ((e=strmatch(add," aT "))!=0)
      || ((e=strmatch(add," @ "))!=0))
      { *e='\0';strcpy(user,add);strcpy (node,rmltmbl(e+3));
        *add_diag += DG_CHANGEDAT;
        sprintf (add,"%s@%s",user,node);
      }

#ifdef DEBUG
printf("AT@-  : %s\n",add);
#endif DEBUG

   /* detecting illegal characters */
   if (strpbrk (add,ILLCHARSET)) { rc = RC_ILLEGALCHAR;
                                   *add_diag += DG_ILLCHAR;
                                 }

#ifdef DEBUG
printf("ICS   : %s\n",add);
#endif DEBUG

   /* UUCP : from host!user to user@host.uucp */
   if ((e=index(add,'!'))!=0)
     {  *e = '\0'; strcpy(node,add); strcpy(user,e+1);
        sprintf (add, "%s@%s.uucp" ,user,node);
        *add_diag += DG_CHANGEDUUCP;
        adnet = IS_ADDED_SY;
     }

#ifdef DEBUG
printf("UUCP  : %s\n",add);
#endif DEBUG

   /* Decnet : from host::user to user@host.decnet.cern.ch */
```

```
   if ((e=strmatch(add,"::"))!=0)
     {  *e = '\0'; strcpy(node,add); strcpy(user,e+2);
        sprintf (add, "%s@%s.decnet.cern.ch" ,user,node);
        *add_diag += DG_CHANGEDDEC;
        adnet = IS_ADDED_SY;
     }

#ifdef DEBUG
printf("DEC  : %s\n",add);
#endif DEBUG

   /* No Address */
   if (strlen(add)==0) rc = RC_NOADD;

   /* Seperating user@host.domain */
   start = &add[0] ;
   if ((e=index(start,'@'))!=0)
     {  *e = '\0';
        strcpy(user,start);
        start = e+1;
     }
   else { strcpy(user,start);
          *start = '\0' ;
        }
   if ((e=index(start,'.'))!=0)
     {  *e = '\0';
        strcpy(domain,e+1);
        strcpy(node,start);
     }
   else strcpy(node,start);

#ifdef DEBUG
printf("SEP  : U=%s  H=%s  D=%s  \n",user,node,domain);
#endif DEBUG

   strcpy(addomain,domain);
   strcpy(adnode,node);

   tolo(domain);
   tolo(node);

   /* Checking JANET */
   if ((strcmp(node,"uk")==0)
       && ((strmatch(domain,"ac.")==&domain[0])
       ||  (strmatch(domain,"co.")==&domain[0])))
       { char tmp[MAXSIZE],t[MAXSIZE];
         *add_diag += DG_INVJANET;
         /* Inversion */
         sprintf(tmp,"%s.%s",node,domain);
         if ((e=rindex(tmp,'.'))!=0)
          { *e='\0'; strcpy(node,e+1); strcpy(adnode,e+1); }
         domain[0]='\0';
         while ((e=rindex(tmp,'.'))!=0)
          { *e='\0'; sprintf(t,"%s.%s",domain,e+1); strcpy(domain,t); }
         sprintf(t,"%s.%s",domain,tmp); strcpy(domain,&t[1]);
       }

#ifdef DEBUG
printf("Jan  : U=%s  H=%s  D=%s  \n",user,node,domain);
#endif DEBUG

   /* Network determination */
   net = IS_UNKNOWN ;
   if (strmatch(domain,"co.uk"  )==&domain[0])              net = IS_JANET  ;
```

```
   if (strmatch(domain,"ac.uk"  )==&domain[0])              net = IS_JANET  ;
   if (strmatch(domain,"cern"   )==&domain[0])              net = IS_CERN   ;
   if (strmatch(domain,"decnet" )==&domain[0])              net = IS_DECNET ;
   if (strcmp(domain,"uucp")==0)                            net = IS_UUCP   ;
   if (strcmp(domain,"bitnet")==0)                          net = IS_BITNET ;
   if (strcmp(domain,"netnorth")==0)                        net = IS_NETNORTH ;
   if (strcmp(domain,"earn")==0)                            net = IS_EARN   ;


    /* Domain: decnet => decnet.cern.ch */
    if (strmatch(domain,"decnet" )==&domain[0])
      strcpy(domain,"decnet.cern.ch");

    /* Checking Cern domain sub-domains */
    if (strmatch(domain,"cern")==&domain[0])
      { *add_diag += DG_ADOM;
        strcpy (domain, "cern.ch");
      }
    if (net==IS_CERN)
      { char tmp[MAXSIZE];
        FILE *cd;
        sprintf(tmp,"%s.%s",node,domain);
        if ((e=strmatch(tmp,".cern"))!=0) *e = '\0';
        if ((e=strmatch(tmp,"cern."))!=0) *e = '\0';
        if ((b=rindex(tmp,'.'))==0) b = &tmp[0] - 1;
        strcpy(cerndom,b+1);
        if (strlen(cerndom)==0) rc = RC_ILLCD;
        cd=fopen(CDFILE,"r");
        while ((fgets(tmp,100,cd))!=NULL)
          { tmp[strlen(tmp)-1]='\0';
            rmltmbl(tolo(tmp));
            if (strcmp(tmp,cerndom)==0) rc = RC_ILLCD;
          }
        fclose(cd);
      }

#ifdef DEBUG
printf("Cern : U=%s  H=%s  D=%s CD=%s \n",user,node,domain,cerndom);
#endif DEBUG


#ifdef DB-ACCESS
   /* Getting the absent domain */
   if ((strlen (domain) == 0) && (strlen(node)!=0) && (net==IS_UNKNOWN))
           {  if ((oraerr = (EMNODES(node,domain))) ==0)
                 {  if (strcmp(domain,"cern.ch")==0)        net = IS_CERN  ;
                    if (strcmp(domain,"uucp")==0)           net = IS_UUCP ;
                    if (strcmp(domain,"bitnet")==0)         net = IS_BITNET ;
                    if (strcmp(domain,"netnorth")==0)       net = IS_NETNORTH ;
                    if (strcmp(domain,"earn")==0)           net = IS_EARN ;
                    if (strcmp(domain,"decnet.cern.ch")==0) net = IS_DECNET ;
                    adnet = IS_ADDED_DB;
                 }  else *add_diag += DG_ORA_ERROR;
           }
#endif DB-ACCESS


#ifdef DEBUG
{ int err,ec;char msg[300],st[300];
if (sqlreport(st,&msg,&err))
   printf("Oracle Error %d on %s \n%s\n",err,st,msg);
   else printf("Oracle Access succeed or Not done\n");
}
#endif DEBUG
```

```
   strcpy(node,adnode);

   /* Ending stuff */
   if (!(rc))
   { if (strlen (domain) == 0) rc = RC_NODOM;
     if (strlen (user)   == 0) rc = RC_NOUSER;
     if (strlen (node)   == 0) rc = RC_NOHOST;
   }
   if (!(rc)) sprintf (outstring, "%s@%s.%s" ,user,node,domain);
   else if (rc==RC_NODOM) sprintf (outstring, "%s@%s" ,user,node);
   else strcpy(outstring,"");
   if (oraerr!=0) *net_diag = oraerr ; else *net_diag = net + adnet ;

#ifdef DEBUG
printf("OUT  : %s \n-Add=%d-Net=%d-Err=%d\n",outstring,*add_diag,*net_diag,rc);
#endif DEBUG

   return  (rc);

}
```

```
From heuse@vxcern.decnet  Thu Jan 19 13:45:53 1989
Date: Thu, 19 Jan 89 13:45:53 +0100
Message-Id: <8901191245.AA17380@cernvax.uucp>
From: heuse@vxcern.decnet
To: printer@priam.decnet
Subject:

/* Compiler directive */
/* #define EMNODES_DEBUG 0 */

/* Domain Priority (higher is winner) */
#define   P_CERN      10
#define   P_DECNET     9
#define   P_BITNET     8
#define   P_EARN       7
#define   P_NETNORTH   6
#define   P_UUCP       5
#define   P_JANET      4

/* BD Control routines */
#include "ora.h"

static short nlda[32];
static short ncursor[32];
static short nhda[128];
static int   ndbopen = 0 ;

#define EMNODES_DBUSERandPW "[redacted]"
#define LGNETW          08

EMNODES_DBOPEN()
{

#ifdef EMNODES_DEBUG
printf("DBOpen-EMNODES : Orlon\n");
#endif EMNODES_DEBUG

  orainit();
  if (oraok()) orlon(nlda,nhda,EMNODES_DBUSERandPW,strlen(EMNODES_DBUSERandPW),
                     (char *)-1,-1,0);
    if (!(sqlok(nlda)))      oraerror("DBopen-Orlon",nlda);

#ifdef EMNODES_DEBUG
printf("DBOpen-EMNODES : Oopen\n");
#endif EMNODES_DEBUG

  if (oraok())               oopen(ncursor,nlda,
                                   (char *)-1,-1,-1,(char *)-1,-1);
    if (!(sqlok(ncursor)))   oraerror("BDopen-Oopen",ncursor);
  ndbopen = 1 ;

#ifdef EMNODES_DEBUG
printf("DBOpen-EMNODES : Done\n");
#endif EMNODES_DEBUG

}

EMNODES_DBCLOSE()
{

#ifdef EMNODES_DEBUG
printf("DBClose-EMNODES : Oclose\n");
#endif EMNODES_DEBUG
```

```
  oclose(ncursor);
    if (!(sqlok(ncursor)))   oraerror("DBclose-Oclose",ncursor);

#ifdef EMNODES_DEBUG
printf("DBClose-EMNODES : Ologof\n");
#endif EMNODES_DEBUG

  ologof(nlda);
    if (!(sqlok(nlda)))      oraerror("DBclose-Ologof",nlda);

#ifdef EMNODES_DEBUG
printf("DBClose-EMNODES : Done\n");
#endif EMNODES_DEBUG

}

EMNODES(node,domain)
char *node,*domain;
{ char req[500];
  char host[500],dom[500],st[500],msg[500],*b,*e;
  int i,p,priority,err,errc;

#ifdef EMNODES_DEBUG
printf("EMNODES : Begin\n");
#endif EMNODES_DEBUG

  p = priority = 0 ;
  if (ndbopen == 0) {

#ifdef EMNODES_DEBUG
printf("EMNODES : Opening DB\n");
#endif EMNODES_DEBUG

                      EMNODES_DBOPEN(); ndbopen = 2 ; }
  strcpy(host,node);
  for(b=host;*b;b++) *b=toupper(*b);
  sprintf(req,"SELECT UPPNET FROM NETNODES WHERE UPPNODE = '%s'",host);

#ifdef EMNODES_DEBUG
printf("EMNODES : Osql3\n%s\n",req);
#endif EMNODES_DEBUG

  if (oraok())   osql3(ncursor,req,strlen(req));
    if (!(sqlok(ncursor)))   oraerror("EMnodes-Osql3",ncursor);

#ifdef EMNODES_DEBUG
printf("EMNODES : Odefin\n");
#endif EMNODES_DEBUG

  if (oraok())   odefin(ncursor,1,dom,499,5,-1,
                        (short *)-1,(char*)-1,-1,-1,(short *)-1,(short *)-1);
    if (!(sqlok(ncursor)))   oraerror("EMnodes-Odefin",ncursor);

#ifdef EMNODES_DEBUG
printf("EMNODES : Oexec\n");
#endif EMNODES_DEBUG

  if (oraok())   oexec(ncursor);
    if (!(sqlok(ncursor)))   oraerror("EMnodes-Oexec",ncursor);

#ifdef EMNODES_DEBUG
printf("EMNODES : Ofetch\n");
#endif EMNODES_DEBUG
```

```
  if (oraok())

#ifdef EMNODES_DEBUG
{ printf("EMNODES : Avt Ofetch\n");
#endif EMNODES_DEBUG

                    ofetch(ncursor);

#ifdef EMNODES_DEBUG
printf("EMNODES : Apr Ofetch\n"); }
#endif EMNODES_DEBUG

  while (!(eofetch(ncursor)))
    { if (!(sqlok(ncursor)))   oraerror("EMnodes-Ofetch",ncursor);
      for (b=dom; *b==' '; b++);
      for (i=strlen(b)-1;((i>=0) && (b[i]==' '));b[i--]='\0');
      if ((strcmp(b,"CERN")==0))      p = P_CERN ;
      if ((strcmp(b,"UUCP")==0))      p = P_UUCP ;
      if ((strcmp(b,"EARN")==0))      p = P_EARN ;
      if ((strcmp(b,"DECNET")==0))    p = P_DECNET ;
      if ((strcmp(b,"NETNORTH")==0)) p = P_NETNORTH ;
      if ((strcmp(b,"BITNET")==0))    p = P_BITNET ;
      if ((strcmp(b,"JANET")==0))     p = P_JANET ;
      if (p > priority)
         { strcpy(domain,b); priority = p;
           if (strcmp(domain,"CERN")==0) strcpy(domain,"CERN.CH");
           if (strcmp(domain,"DECNET")==0) strcpy(domain,"DECNET.CERN.CH");
         }

#ifdef EMNODES_DEBUG
printf("EMNODES : Ofetch bcle\n");
#endif EMNODES_DEBUG

      if (oraok())   ofetch(ncursor);
    }
  if (ndbopen == 2) {

#ifdef EMNODES_DEBUG
printf("EMNODES : Closing DB\n");
#endif EMNODES_DEBUG

                     EMNODES_DBCLOSE(); ndbopen = 0 ; }

#ifdef EMNODES_DEBUG
printf("EMNODES : Close done\n");
#endif EMNODES_DEBUG

  if (!(oraok())) strcpy(domain,"");
  for(b=domain;*b;b++) *b=tolower(*b);
  sqlreport(st,msg,&err);

#ifdef EMNODES_DEBUG
printf("EMNODES : End => %d (Oracle Error)\n",err);
#endif EMNODES_DEBUG

  return(err);
}
```

```
From heuse@vxcern.decnet  Thu Jan 19 13:46:31 1989
Date: Thu, 19 Jan 89 13:46:31 +0100
Message-Id: <8901191246.AA17440@cernvax.uucp>
From: heuse@vxcern.decnet
To: printer@priam.decnet
Subject:

/* Compiler directives */
/* #define ORA_DEBUG */

# define SQLSTR    005
# define SQLSTRL   300
# define SQLBUFL   500
# define SQLSTATL  700

static int    SQLERROR;
static short  SQLERRNU;
static short  SQLERRCO;
static char  *SQLERRST[SQLSTRL];

eofetch(area)
short *area;
{

#ifdef ORA_DEBUG
if ( (((abs(*area))==4) || (SQLERROR))) printf("End Of Fetch\n");
#endif ORA_DEBUG

  return( (((abs(*area))==4) || (SQLERROR)) ); }

sqlok(area)
short *area;
{

#ifdef ORA_DEBUG
printf("Ret Err Code : %d\n",*area);
#endif ORA_DEBUG

  return( (*area==0) ); }

oraok()
{

#ifdef ORA_DEBUG
if (SQLERROR) printf("N- ");
        else printf("Y- ");
#endif ORA_DEBUG

  return( !(SQLERROR) ); }

orainit()
{ SQLERRNU=SQLERRCO=SQLERROR=0;
  SQLERRST[0]='\0';
}

sqlreport(st,erst,err)
char *st,*erst;
int  *err;
{ oermsg(SQLERRNU,erst);
  strcpy(st,SQLERRST);
  *err = SQLERRNU;
  return( SQLERROR );
}
```

```
oraerror(st,area)
char *st;
short *area;
{ if (!(SQLERROR))
  { SQLERROR = SQLERRNU = abs( SQLERRCO = *area );
    strcpy(SQLERRST,st);
  }
}
```

Emdir.pol

Proposed Politic For Names and Forenames In the EmDir DB and Automatic Routing.

Bernard Heuse, 04/01/89.

The problem is how to deal with names and forenames in the routing of electronic mail based on the EmDir database with addresses in the syntax:

[firstname '.'] [initial '.']5 [name] @ ...

The main difficulties come from :
- for forenames :
  - composed firstnames with or without dashes as 'Jean-Luc' or 'John Peter';
  - the preferred firstname is not the first one listed in the official forenames;
  - initials which can be registered as real initials or which can be the initials of the complementary forenames;
  - dots which are used in the DB firstname field as 'J.-L.', 'H. Angeles', 'David R.', 'C. J.', 'M.J.', 'N.';
  - undescores which are used in the DB firstname field as 'Bernard_Andre'.
- for names :
  - names with particles ('de', 'd' ', 'van', 'von', 'van den', 'di', ...) and the fact that E-Mail addresses usually do not accept blanks inside;
  - composed names with or without dashes as 'Ruhet Froissard' or 'Durand-Dupond';
  - for married women, they are known by their husband's name and/or by their own native name.

Let's define 'separator' (sep) a sequence of one or more of the following characters : dash (-), underscore (_), blank ( ), dot (.) and kot (').

Let's define 'illegal address separator (ias) a sequence of one or more of the following characters : dash (-), underscore (_), blank ( ) and kot (').

An 'initial' (I) is a single character allowed as first character of a name or a firstname, excluding sep.

A 'string' (St) is 0, 1 or more characters allowed to compose parts of names or firstnames, excluding sep, each part being separated by sep.

So

sepc	::=	' ' | '-' | '_' | '.' | '"
sep	::=	sepc | sep sepc
iasc	::=	' ' | '-' | '_' | '.' | '"
ias	::=	iasc | ias iasc
I	::=	"a single letter allowed as first character of a name or a firstname, at the exclusion of the characters of sepc"
St	::=	"0, 1 or more characters allowed to compose parts of names or firstnames, at the exclusion of the characters of sepc, each part being separated by a character a sep"
part	::=	I | I St
Ename	::=	part | Ename sep part
Efname	::=	part | Efname sep part
Aname	::=	part | Aname ias part
Afname	::=	part | Afname ias part

where Ename and Efname could be the syntax of respectively the name and the firstname contained in the EmDir DB,

and Aname and Afname are the syntax of respectively the name and the firstname plus initials when replacing dots by blanks in an E-Mail address.

Moreover, parts of the name and forenames should be ordered by preference of usage.

With such a syntax, an E-Mail address can be defined as:

```
[[I St ias]* I St '.'][I '.']5[[I St ias]* I St]@...
|<--- firstname ---->||< I ->||<---- name ---->|
|<--------- Afname --------->||<---- Aname --->|
|<--------- Efname --------->||<---- Ename --->|
```

The main drawback of this syntax is that it do not provide a mean to distinguish whether two or more parts of a name or a firstname form in fact a single name or firstname (e.g. "Jean-Luc" as firstname is equivalent to "Jean Luc" where "Jean" is the firstname, and "Luc" and complementary forename). An other consequence of this is that particles should not be concatenated in the E-Mail address if they are not in the name and firstname fields of the EmDir DB. In this case, a ias should be used in the E-Mail address (as underscore) to allow successful matching.

Such a common syntax should not be a big constraint for the content of the EmDir DB. And a matching rule between E-Mail addresses and the fields NAME and FIRSTNAME of the EmDir DB can be easily defined and implemented, by defining matching rules between Afname and Efname, and Aname and Ename. This matching algorithm can be from very simple (exact matching) to very complex and refined (based on the defined syntax, with order unimportant, ...), but this is relevant to the EN section. The concern of this document is to provide a base for the EmDir managers in such a way that the content of the DB will conform to our matching rules, and is thus a guide to design the EmDir user interface.

Example of message rejected by the AutoRouter

```
Message inbox:6 - Read
From:     <S=cernvax!MAILER-DEAMON;0=prlb2;P=uucp;A=rtt;C=be>
To:       <S=heuse;OU=ts;O=info;P=fundp;A=rtt;C=be>
Subject:  Returned mail: User unknown

   ----- Transcript of session follows -----

   CERN Emdir Automatic Router (Emdir DB).
   Automatic Router failed. "dummy@dd.cern.ch" : Unknown recipient.

   Use the nameserver to retrieve a CERN user's E-Mail address.
   Send a mail to <nameserver@cernvax.cern.ch> with the keyword
   'Help' as subject to receive more information about this service.

550 dummy@dd.cern.ch ... User unknown

   ----- Unsent message follows -----

Date      : 26 Apr 89  20:05
From      : Bernard Heuse <heuse@ts.info.fundp.rtt.be>
To        : <dd!dummy>
Message-Id : <120*heuse@ts.info.fundp.rtt.be>
Subject   : Test of DUMMY@DD.CERN.CH

Body is here. This is a test message.
```

Example of message routed by the AutoRouter

```
Message inbox:7 - Read
From      : Bernard Heuse
  <S=info.fundp.rtt.be!heuse;G=cernvax!ts;0=prlb2;P=uucp;A=rtt;C=be>
To        : B. Heuse <S=heuse;I=b;O=dd;P=uucp;A=rtt;C=be>
Subject   : Test of B.HEUSE@DD.CERN.CH

>X-Rerouted-To : heuse@ts.info.fundp.rtt.be by CERN Automatic Router
  (Emdir DB).

Body is here. This is a test message.
```

Annex F List of Abbreviations

ACSE	Association Control Service Element
ADMD	ADministration Management Domain
AET	Application Entity Title
AFI	Authority and Format Identifier
AFNOR	Association Francaise de NORmalisation
AMF	Address Mapping Function
ANSI	American National Standards Institute
ARPA	Advanced Research Project Agency
ARPANET	Advanced Research Project Agency NETwork
ASE	Application Service Element
ASN1	Abstract Syntax Notation 1
AVA	Attribute Value Assertion
BITNET	Because It's Time NETwork
BSI	British Standards Institution
CCITT	Comité Consultatif International des Télégraphes et Téléphones
CERN	Comité Européen pour la Recherche Nucléaire
CUDF	Call User Data Field
CS	Conceptual Schema
DAP	Directory Access Protocol
DB	DataBase
DBMS	DataBase Management System
DCC	Data Country Code
DECNET	Digital Equipment Corporation Network
DF	Directory Function
DIB	Directory Information Base
DIN	Deutshe Industrie Normen
DIT	Directory Information Tree
DL	Distribution List
DNIC	Data Network Identifier Code
DMD	Directory Management Domain
DS	Directory Service
DSA	Directory Service Agent
DSP	Domain Specific Part
DSP	Directory Service Protocol
DTE	Data Terminal Equipment
DUA	Directory User Agent
DUR	Domain Under Resorption
EAN	Electronic Access Network
EARN	European Academic Research Network
ECMA	European Computer Manufacturers Association
E-Mail	Electronic Mail
ES	External Schema
ESA	European Space Agency
ESPRIT	European Stategic Program for Research and devlopment in Information Technologies
E/R	Entity / Relationship
FTAM	File Transfer and Access Management

FTP	File Transfer Protocol
FTS	File Transfer Service
GCS	Global Conceptual Schema
GES	Global External Schema
GMD	Gessellschaft fur Mathmatik and Datenverarbeitung
GT	Global Title
HEP	High Energy Physics
IBM	International Business Machines
IDI	Initial Domain Identifier
IDP	Initial Domain Part
INCA	Integrated Network Communication Architecture
INRIA	Institut National de Recherche en Informatique et en Automatique
IP	Internetwork Protocol
IPMS	InterPersonal Messaging Service
IS	Internal Schema
ISDN	Integrated Service Data Network
ISO	International Standardization Organization
ISODE	ISO Devlopment Environment
JANET	Joint Academic NETwork
JTAM	Job Transfer and Access Management
LAN	Local Area Network
LCS	Local Conceptual Schema
LES_{DSA}	Local External Schema of the DSA
LES_{U}	Local External Schema of the Users
LIS	Local Internal Schema
LSPX	Large Scale Pilot eXercice
LT	Local Title
MHS	Mail Handling System
MTA	Mail Transfer Agent
MTS	Mail Transfer System
(N)Add	(N-Entity) Service Access Point Address
NetNorth	North American Network
NetServ	Network Server
NLP	NRS Lookup Protocol
(N)SAP	(N-Entity) Service Access Point
(N)SPA	(N-Entity) Service Access Point Address
NTN	Network Terminal Number
NRS	Network Registration Scheme
O/R	Originator / Recipient
OSI	Open System Interconnection
PAD	Packet Assembler Disassembler
PC	Personal Computer
PDN	Public Data Network
PTDN	Primary Title Domain Name
PTT	Post Telegraph Telephone
PRMD	PRivate Management Domain
PSAP	Presentation-SAP
RARE	Réseaux Associés pour la Recherche Européenne
RDN	Relative Distinguished Name
RFC	Request For Comments

ROSE	Remote Operation Service Element
RPC	Remote Procedure Call
SNPA	SubNetwork Point of Attachment
SQL	Simple Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
THORN	THe Obviously Required Nameserver
TR/32	(ECMA) Technical Report 32
UA	User Agent
UCO	User Consultancy Office
UK	United Kingdom
US	United States
USSR	Union of Soviet Socialist Republics
UUCP	Unix to Unix CoPy
VFS	Virtual FileStore
VT	Virtual Terminal
WKD	Well Known Domain

Bibliography

[Ben-6.5/CM] Steve Benford, "Navigation and knowledge management within a distributed directory system", IFIP WG 6.5 conference papers : MHS and distributed Applications, Costa Mera, October 1988.

[Ben-6.5/M] Steve Benford, Julian Onions, "Pilot Distribution Lists, Agents and Directories", IFIP WG 6.5 conference papers : Message Handling Systems, Munich, October 1988.

[COS-88] Albert Biber, Johann Forster, "Cosine Specification Phase, Study on Directories" (draft version), Soflab Gmbh, Munich, April 1988.

[Der-88] Eric Derruine, "Electronic Mail Gateways", FNDP Namur, June 1988.

[Dimou-88] Maria Dimou, "Naming Structures for Directory Services", CERN/DD, July 1988.

[ECMA-117] ECMA, STANDARD ECMA-117: "Domain Specific Part of Network Layer Addresses", June 1986.

[FTAM-DS] E. Loevdal, W. Black, F. Fluckiger, K. Truoel, "Directory Requirements for the Rare Cosine FFTAM service" (draft version), CERN/DD, November 1986.

[HandBook] Keith Bartlett, "Handbook", Department of Trade and Industry, USA, 1988.

[Huit-6.5/M] Christian Huitema, Juan Antonio Saras, "The use of distribution lists in MHS", IFIP WG 6.5 conference papers : Message Handling Systems, Munich, October 1988.

[Huit-88] Christian Huitema, "A proposal for a naming, data structure, and name data distribution in RARE", May 1988.

[ISO-7498] ISO/DIS 7498, "Information Processing Systems - Open System Interconnection - Basis reference model".

[IES-19/1] Steve E. Kille, "Experience with the use of THORN", IESNEWS No 19, December 1988.

[IES-19/2] Franco Sirovich, Massino Antonellini, "The THORN X.500 distributed Directory Environment", IESNEWS No 19, December 1988.

[Kille-6.5/CM] Steve E. Kille, "The Quipu Directory Service", IFIP WG 6.5 conference papers : MHS and distributed Applications, Costa Mera, October 1988.

[Kille-6.5/M] Steve E. Kille, "MHS use of Directory Services for routing", IFIP WG 6.5 conference papers : Message Handling Systems, Munich, October 1988.

[Kille-88] Steve E. Kille, "An Overview of Directory Services", University College London, March 1988.

[Len-6.5/M] Thomas Lenggenhoger, Bernhard Plattner, Rolf Stadler, "The ISO/CCITT Directory as a distributed database: data models", IFIP WG 6.5 conference papers : Message Handling Systems, Munich, October 1988.

[NETS-88] B. Pasch, "NetServ", NetServ Help File, February 1988.

[NRS-88] W. Black, "The Janet Name Registration Scheme", February 1987.

[OSN-87] "Directory Standardization rethink results in a more limited service", The Open Systems Newsletter, Vol 1 Issue 2, April 1987.

[QUI-88a] Steve E. Kille, "The Design of Quipu", Department of Computer Sciences, University College London, July 1988.

[QUI-88b] Steve E. Kille, "The Quipu Directory", Department of Computer Sciences, University College London, August 1988.

[RFC-883] P. Mockapetris, "Domain Name Implementation and Specifications", RFC-883, Nov. 1983.

[Shock-78] John F. Shock, "Inter-Network Naming, Addressing and routing", Xerox-PARC 1978.

[Sim-88] Serge Simonet, "Le service de directory", FNDP Namur, 1988.

[Sol-6.5/M] Karen R. Sollins, David D. Clark, "Distributed Name Management", IFIP WG 6.5 conference papers : Message Handling Systems, Munich, October 1988.

[Stal-88] William Stallings, "Data and Computer Communications", Collier MacMillan Publishers, London 1988.

[Tan-81] Andrew S. Tanenbaum, "Computer Networks", Prentice/Hall International Editions, 1981.

[TR/32] ECMA, ECMA Technical Report 32 : "Directory Access Service and Protocol", December 1985.

[WG3-88] RARE-WG3 : Directory services, Distribution List 'RARE-WG3', 1988.

[X.400-88] CCITT X.400/ISO 10021-1, "MOTIS : System and Service Overview", Version 5.6, April 1988.

[X.402-88] CCITT X.402/ISO 10021-2, "MOTIS : Overall Architecture", Version 6.5, April 1988.

[X.500-88] CCITT X.500/ISO 9594-1, "The Directory - Part 1: Overview of Concepts, Models and Services", 1988.

[X.501-88] CCITT X.501/ISO 9594-2, "The Directory - Part 2: Models", 1988.

[X.509-88] CCITT X.509/ISO 9594-8, "The Directory - Part 8: Authentification Framework", 1988.

[X.511-88] CCITT X.511/ISO 9594-3, "The Directory - Part 3: Abstract Service Definition", 1988.

[X.518-88] CCITT X.518/ISO 9594-4, "The Directory - Part 4: Procedures for Distributed Operations", 1988.

[X.519-88] CCITT X.519/ISO 9594-5, "The Directory - Part 5: Protocol Specifications", 1988.

[X.520-88] CCITT X.520/ISO 9594-6, "The Directory - Part 6: Selected Attribute Types", 1988.

[X.521-88] CCITT X.521/ISO 9594-7, "The Directory - Part 7: Selected Object Classes", 1988.