NYSERNet White Pages Pilot Project: Administrator's Guide

Marshall T. Rose
NYSERNet, Inc.
`mrose@nisc.nyser.net`

March 2, 1990

Abstract

The need for a comprehensive white pages service increases in relation to the size of the user community. The early Internet was served well by a relatively simple facility. Today's rapidly expanding Internet has outstripped the capabilities of the existing system. In order to meet new requirements, NYSERNet, Inc. is sponsoring a pilot project to provide white pages service based on the OSI Directory.

This document describes the pilot project from an administrator's perspective and provides operational reference for the installation and maintenance of a local OSI Directory capability.

This[0] work was partially supported by the U.S. Defense Advanced Research Projects Agency and the Rome Air Development Center of the U.S. Air Force Systems Command under contract number F30602–88–C–0016. The content of the information contained herein does not necessarily reflect the position or the policy of the U.S. Government, and no official endorsement should be inferred.

Contents

This document (version #2.70) was LaTeXset March 2, 1990 with lplain v2.09-10/29/85.

List of Figures

List of Tables

Chapter 1

Introduction

This document is *The Administrator's Guide* for the NYSERNet White Pages Pilot. The goal of *The Guide* is to provide a site administrator with enough information to be able to participate in the White Pages Pilot. In practical terms, this means that *The Guide* provides information on how to install and maintain an OSI Directory at your site.

The OSI Directory is used to provide the white pages service. *The Guide* is not intended as a tutorial nor a detailed description of the OSI Directory. However, as administrative responsibilities and procedures are outlined, the appropriate Directory concepts will be introduced.[1]

Your comments are welcome! The OSI Directory is a new, complex technology. Although *The Guide* attempts to be straight-forward it probably doesn't succeed all the time. If you have comments on this document, send them to the Internet mailbox

```
wpp-camayocs@nisc.nyser.net
```

so that *The Guide* can be improved.

[1]Actually this paragraph is a lie: you can't administer the white pages without knowing what you're doing, and you don't know what you're doing unless you understand both the basics of the OSI Directory and how the pilot project software implements the Directory Service. As such, this entire chapter is really a thinly-veiled primer on these two topics.

Related Documentation

The white paper *An Introduction to a NYSERNet White Pages Pilot Project* introduces the goals and phases of the pilot project. Administrators should familiarize themselves with this document before proceeding.

There is also a document which you should provide to the user community at your organization. It is called *NYSERNet White Pages Pilot Project: User's Handbook*, or simply *The Handbook*.

The OSI Directory standard is defined in [ISO88, CCITT88].

The research note *The Design of QUIPU* [SKill89a] describes the design of the software used for the pilot project, whilst *Volume Five* of the ISODE User's Manual [SKill89b], henceforth termed *Volume Five*, describes the implementation of the software. *The Guide* is intended to replace these later two documents as the primary reference for administrators of the pilot project.

PostScript versions of these documents are available via anonymous FTP from host `nisc.nyser.net` (`[192.33.4.10]`):

```
% cd /usr/src/local/
% ftp nisc.nyser.net
Connected to nisc.nyser.net.
220 nisc.nyser.net FTP server ready
Name (nisc.nyser.net:user): anonymous
Password (nisc.nyser.net:anonymous): guest
331 Guest login ok, send ident as password.
230 Guest login ok, access restrictions apply.
ftp> binary
200 Type set to I.
ftp> cd pub/isode
ftp> get isode-ps.tar.Z
ftp> get pilot-ps.tar.Z
ftp> quit
% uncompress < isode-ps.tar.Z | tar xf -
% uncompress < pilot-ps.tar.Z | tar xf -
% rm isode-ps.tar.Z pilot-ps.tar.Z
```

1.1 Informational Model

From a theoretical standpoint, the internals of the white pages service can be viewed from four different perspectives. We begin with the Informational Model which describes the service in terms of information objects. The Directory's representation of an information object, typically called an *entry*, contains information about a person, a place, an organization, etc. Each entry consists of one or more attributes.

Each attribute consists of a type, indicating what kind of attribute it is, and one or more values (one of which is termed the *distinguished value*). Attribute values are structured using a data definition language called Abstract Syntax Notation One (ASN.1). As such, different programs using the Directory will interpret information in the same way. In addition, the Directory will perform type-checking on the values in order to keep things consistent.

1.1.1 Naming

One of the attributes of an entry is particularly special: it is referred to as the *Relative Distinguished Name* (RDN) of the entry. The RDN is formed by taking the name of the attribute and its distinguished value. For example, if the attribute in question was called `countryName` and it had a distinguished value of `US`, then we might say that the RDN for the entry was `countryName=US`. Of course, this is strictly a "user-friendly" notation: the Directory uses a concise binary format for representing an RDN. Fortunately, the pilot project software allows simple textual strings to be used in their place and converts back and forth accordingly.

In the OSI Directory, information is primarily organized according to a hierarchical tree structure. The top of the tree is termed the *root*, and has no explicit name. To find the name of an object, termed its *Distinguished Name* (DN), one concatenates the RDNs found when traversing the tree by starting at the root and proceeding directly to the object's entry.

For purposes of discussion, we write a Distinguished Name as an ordered series of RDNs separated by an '`@`'-sign with the most significant RDN appearing at the left; e.g.,

```
countryName=US@organizationName=NYSERNet Inc.
```

refers to an entry with an RDN of `organizationName=NYSERNet Inc.` whose parent has an RDN of `countryName=US`. In turn, this parent entry is an immediate child of the root.

To avoid any potential ambiguity when using an interface to the Directory such as *fred*(1c) or *dish*(1c), one prefixes a '`@`'-sign to a string when referring to a fully qualified Distinguished Name; e.g.,

```
@countryName=US@organizationName=NYSERNet Inc.
```

always refers to the same entry regardless of context. Note that this is a convention only for interface programs such as these.

As a rule, unless searching, text before the '`=`'-sign is not case sensitive, neither is text after the '`=`'-sign.

Of course, names like `countryName`, while more friendly than binary strings, are still rather long, so there are several abbreviations that the pilot project software permits. Table 1.1 lists the attribute types supported by the pilot project, any abbreviations, and the syntax associated with each attribute's value. Throughout *The Guide*, the semantics of these attributes will be introduced. When using this document later on, refer to the Index to quickly find a particular attribute you are interested in.

The entities which access the Directory on behalf of a user are termed *Directory User Agents* (DUAs). The entities which provide the Directory service are termed *Directory System Agents* (DSAs). Both kinds of entities are identified by their Distinguished Name. That is, the DSAs and DUAs are not only network processes, they are also objects which have entries in the Directory.

1.1.2 How Entries and Objects are Described

It is now time to explain the notation used to describe entries. This notation is termed the "EDB format", and is named after the disk-resident data structures used by the pilot project software.

An entry is textually described in an ASCII formatted file. The format of the file is simple:

- Each entry consists of two or more lines followed by a blank line.
- Each line consists of an attribute type/value pair using the familiar

  ```
  name=value
  ```

  notation, such as:

  ```
  c=US
  ```

Attribute Name	Abbrev.	Syntax
accessControlList	acl	special
aliasedObjectName		Distinguished Name
associatedDomain		string
businessCategory		string
commonName	cn	string
countryName	c	string
description		string
eDBinfo		special
facsimileTelephoneNumber		string
favouriteDrink	drink	string
friendlyCountryName	co	string
homePhone		string
homepostalAddress		special
info		string
lastModifiedBy		Distinguished Name
lastModifiedTime		Universal Time
localityName	l	string
manager		Distinguished Name
masterDSA		Distinguished Name
mobileTelephoneNumber	mobile	string
objectClass		object class
organizationName	o	string
organizationalUnitName	ou	string
otherMailbox		special
pagerTelephoneNumber	pager	string
photo		ASN.1 string
physicalDeliveryOfficeName		string
postOfficeBox		string
postalAddress		special
postalCode		string
presentationAddress		special
quipuVersion		string
registeredAddress		special
rfc822Mailbox	mail	string
roleOccupant		Distinguished Name
roomNumber		string
secretary		Distinguished Name
seeAlso		Distinguished Name
slaveDSA		Distinguished Name
stateOrProvinceName		string
streetAddress		string
surname	sn	string
telephoneNumber		string
telexNumber		special
title		string
treeStructure		object class

- If multiple values are present for an entry, then either multiple lines may be used or the values may be separated by the '`&`'-sign. The first value occurring is the distinguished value. Hence,

  ```
  co=US
  co=United States of America
  ```

  and

  ```
  co=US & United States of America
  ```

  are identical: both say that the attribute `friendlyCountryName` has two values: the distinguished value is `US` and an alternate value is `United States of America`.

- Most attributes take textual strings for their value. However, for a large number of attributes, the actual characters used are limited to a subset of the "printable" characters:

  `A` through `Z`
  `a` through `z`
  `0` through `9`
  `'` (apostrophe)
  `(` (left parenthesis)
  `)` (right parenthesis)
  `+` (plus-sign)
  `,` (comma)
  `-` (hyphen)
  `.` (period)
  `/` (solidus)
  `:` (colon)
  `?` (question-mark)
  space

  In particular, note that the characters '`=`', '`&`', '`$`' and '`#`' are not allowed.

  If it makes sense to use a particular character, e.g., '`@`' in a mail address, then this usage is usually permitted.

  If it is necessary to characters from the restricted set, proceed the value with `{T.61}`, e.g.,

```
{T.61}whatever$you$want
```

Always use the appropriate case (upper or lower) when entering values. Although the Directory supports imprecise matching for searching, direct lookup is made on the basis of an exact match.

When strings are read by the pilot project software, it strips leading and trailing blanks and also compresses multiple blanks to a single blank space.

- Another common kind of attribute value is a Distinguished Name, which is specified using the DN notation described earlier. Note that a DN is always interpreted in absolute form when it is written in an EDB file. Hence, the leading '`@`'-sign is never written.

- Of course, there are a few attributes with very special syntaxes, when these attributes are introduced, the associated notation will be discussed.

Note that the first line for an entry gives the object's RDN. This line should be repeated later on in the entry, this causes it to be included in the attributes for the object. (The pilot software includes this attribute automatically, but it is a good practice regardless.)

An EDB file contains all of the child objects of a node in the tree. The first line of the file consists of a single keyword, one of:

MASTER: indicating that this EDB file is the original source of the information;

SLAVE: indicating that this EDB file is a copy of the information from an authoritative source; and,

CACHE: indicating that this EDB file is a partial, unauthoriative copy of the information.

The second line of the file contains the creation date of the information in the file. This is expressed as a `UniversalTime`. Put simply, the format used is:

```
yymmddhhmmssZ
```

as in:

```
890509213614Z
```

```
SLAVE
19890509120000Z
cn=Manager
acl=
cn= Manager
aliasedObjectName= c=US@o=NYSERNet Inc.@ou=Development@cn=Wengyik Yeong#
objectClass= top & quipuObject
objectClass= alias

ou=Research and Development
masterDSA= c=US@cn=Spectacled Bear#
acl= others # read # entry
acl= others # read # default
acl= others # compare # attributes # accessControlList$userPassword
ou= Research and Development & Development
postalAddress= NYSERNet Inc. $ 165 Jordan Road $ Troy, NY 12180
streetAddress= 165 Jordan Road
physicalDeliveryOfficeName= Troy
stateOrProvinceName= New York
postalCode= 12180
telephoneNumber= +1 518-283-8860
facsimileTelephoneNumber= +1 518-283-8904
treeStructure= quipuNonLeafObject & organizationalUnit
treeStructure= quipuDSA & pilotPerson & organizationalRole
objectClass= top & quipuObject & quipuNonLeafObject
objectClass= organizationalUnit
```

Figure 1.1: An Example of an EDB file

which is May 9th, 1989, 21:36:14 hours at UT.

Hence, the format of an EDB file is:

```
<keyword>
yymmddhhmmssZ
<entries>
```

(remember that each entry ends with a blank line). Figure 1.1 shows an example of a simple EDB file containing two entries.

1.1.3 Object Classes

One attribute of an entry, its `objectClass`, determines what kind of object this entry corresponds to (e.g., a person). The value of this attribute indicates what types of attributes the entry *must* and *may* contain. For example, if the value of `objectClass` indicates that the entry corresponds to a person, then it would make sense for that entry to have a `surName` attribute. On the other hand, if the value of `objectClass` indicates that the entry corresponds to an organization, then a `surName` attribute would be inappropriate.

The OSI Directory is flexible in that it defines several common types of objects, and then allows users to define their own objects. Further, object definition is based on the notion of *class inheritance*. This means that an object class can be defined as a "subclass" of a previously defined object class with additional refinements. As a subclass, the newly defined object "inherits" all the semantics of its superclass, in addition to having additional semantics.

For example, the Directory defines an object class called `person`. This object class defines the attributes which a person in the real world might have. It may be useful to refine this somewhat to talk about persons who have Internet access. So, we need a new object class, e.g., `internetPerson`. This can be defined in a straight-forward fashion:

> The object class `internetPerson` is a subclass of the object class `person` which *may* contain an additional attribute, `internetMailbox`.
>
> The syntax of an `internetMailbox` is a simple string of printable characters which is not case sensitive when performing comparisons.

Thus, when entries are defined with an `objectClass` attribute which contains the value `internetPerson`:

- they *must* contain whatever attributes an entry with `objectClass person` *must* contain;
- they *may* contain whatever attributes an entry with `objectClass person` *may* contain; and,
- they *may* contain the `internetMailbox` attribute.

As was seen in Figure 1.1 on page 8, the `objectClass` attribute is multi-valued. For our purposes, object class values fall into one of two categories:

- the primary class, of which each object has one (this is the class that inherited all the properties from the superclasses); and,
- one or more superclasses.

In order to limit the focus of the pilot project, only a few primary classes are supported. Further, of those classes, not all optional attributes are supported.[2] Before discussing the primary classes and their class-specific attributes, there are a few attributes which are required for every object. These are introduced by looking at a few key superclasses.

The top objectClass

The Directory requires that every object have `top` as a super class. Only one attribute is required for members of this class:

objectClass: which defines all the classes that this object belongs to.

The quipuObject objectClass

In addition, the pilot project software requires that every object also have the object class `quipuObject` as a superclass. Only one attribute is required for members of this class:

accessControlList: which defines how users of the Directory may access the entry.

The `accessControlList` attribute uses a special notation to enter its value in the EDB file:

```
<syntax> ::= <who> "#" <action> "#" <what>

<who> ::=       "group" "#" DN
            |   "prefix" "#" DN
            |   "others"
            |   "self"

<action> ::=    "none"
            |   "detect"
            |   "compare"
            |   "read"
            |   "add"
```

[2]This limitation is specific to the pilot project; the pilot project software can support the entire range of object classes and attribute types.

```
                 |   "write"

<what> ::=           "child"
                 |   "entry"
                 |   "default"
                 |   "attributes" "#" <attribute-list>

<attribute-list> ::=
                     attribute-name
                 |   attribute-name "$" <attribute-list>
```

The `<who>` part indicates who the rule applies to. Recall that when a user connects to the Directory, the user *binds* on behalf of a Distinguished Name. This DN is used to determine which rules apply to the user:

group: indicates a DN which has an `objectClass` attribute of `groupOfNames` or `organizationalRole`, or it indicates a particular DN, which is the most common case.

prefix: indicates all DNs which start with the indicated prefix. Thus, access may be granted to all users at or below a particular part of the tree.

self: indicates the DN which corresponds to this object.

others: the default case.

The `<what>` part indicates what part of the entry the rule applies to:

child: the access control rule applies to children immediately below the entry.

entry: the access control rule applies to the entry itself.

attributes: the access control rule applies to exactly those attributes listed. Multiple attributes pertaining to the same rule are separated by the '`$`'-sign.

default: the default rule for attributes.

The `<action>` part indicates the actions which are permitted. The semantics of this part vary, depending on the `<what>` part. However, the `<action>` values are ordered from least- to most-significant; e.g., having "read" permission implies both "compare" and "detect" permission.

none:	for entries:	any knowledge of the entry is hidden
	for attributes:	any knowledge of the attribute is hidden
	for children:	downwards progress is blocked
detect:	for entries:	determination of the entry's existence is permitted
	for attributes:	determination of the attribute's existence is permitted
	for children:	the existence of children is admitted, but downwards progress is still blocked
compare:	for entries:	the RDN may be compared
	for attributes:	the values of the attribute may be compared
	for children:	exactly specified RDNs may be matched
read:	for entries:	the RDN may be read
	for attributes:	the attribute values may be read
	for children:	the child information may be listed, and downward searches are permitted
add:	for entries:	new attributes may be added
	for attributes:	new attribute values may be added
	for children:	new children may be added
write:	for entries:	the RDN may be changed and existing attributes may be added
	for attributes:	values may be modified or removed
	for children:	children may be removed

The only exception to these rules is the `accessControlList` attribute itself: access control rules may be added only to the extent access is allowed by the user.

The default access control list is

```
acl=
```

which simply means "read" access for everything and also

```
self # write # entry
self # write # default
```

There are two optional attributes. These are maintained automatically by the Directory:

lastModifiedBy: identifies the Directory entity which last modified this entry. The value is a Distinguished Name.

lastModifiedTime: identifies the time at which this entry was last modified. The value is a `UniversalTime`, the syntax of which was described earlier on page 7.

The quipuNonLeafObject objectClass

If an object is permitted children, the pilot project software requires that that object also have `quipuNonLeafObject` as a superclass.

This class has one mandatory attribute:

masterDSA: identifies the Directory entity which is responsible for maintaining the MASTER EDB for the children of this entry. The value is a Distinguished Name.

There is typically a single MASTER for a particular entry in the tree. Hence, this value is usually single-valued. When an entry is to be modified, the Directory must contact the entity responsible for the MASTER EDB for that entry in order to perform the modification.

This class has two optional attributes:

slaveDSA: identifies any Directory entities which have authoritative copies of the EDB for the children of this entry. The value is one or more Distinguished Names.

treeStructure: identifies the object classes which may exist immediately below this entry. The value is one or more object classes.

Since a fundamental assumption of the Directory is that reads (queries) occur much more frequently than writes (updates), it is common to have several entities containing authoritative copies of an EDB. By keeping copies locally, queries can be answered with less latency.

The domainRelatedObject objectClass

If an object has some relationship to the Internet Domain Name System (DNS), the pilot project software requires that that object also have `domainRelatedObject` as a superclass.

This class has one mandatory attribute:

associatedDomain: identifies the domain which corresponds to this object. The value is a domain string, e.g.,

```
nyser.net
```

There is typically a single MASTER for a particular entry in the tree. Hence, this value is usually single-valued. When an entry is to be modified, the Directory must contact the entity responsible for the MASTER EDB for that entry in order to perform the modification.

Now we look at the primary object classes. Keep in mind that these all have both `top` and `quipuObject` as superclasses. Further, the primary classes which are permitted children all have `quipuNonLeafObject` as a superclass.

The friendlyCountry objectClass

Objects of this class represent a sovereign nation. For the purposes of the NYSERNet White Pages Pilot, there is only one of these, although the pilot project is attached to a world-wide Directory service.

Table 1.2 summarizes the attributes types for a `friendlyCountry` object. There are two mandatory attributes:

countryName: which gives the name of the country. The value of this attribute is a two-letter ISO 3166 code; e.g.,

```
US
```

friendlyCountryName: which gives a user-friendly rendition of the country's name (hence the term `friendlyCountry`). The value of this attribute is a string; e.g.,

```
United States of America
```

In addition, there is one attribute that may be present:

Superclass Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`top & quipuObject`
		`& quipuNonLeafObject`
		`& domainRelatedObject`
`acl`	M	
`masterDSA`	M	
`slaveDSA`	O	
`treeStructure`	O	`quipuNonLeafObject`
		`& organization & locality`
		`& quipuDSA`
		`& alias & organizationalRole`
		`& domainRelatedObject`
Primary Class Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`country & friendlyCountry`
`countryName`	RDN	
`friendlyCountryName`	M	
`associatedDomain`	O	
`description`	O	

Legend

M:	mandatory
O:	optional, but recommended
RDN:	mandatory, must form the RDN for object

Table 1.2: Attribute Types for the friendlyCountry Object Class

description: which is a simple textual description; e.g.,

```
the land of the free and the home of the brave
```

Putting everything together, one could describe an entry for an object of class `friendlyCountry` thusly:

```
c=US
masterDSA= cn=Alpaca#
slaveDSA= cn=Fruit Bat#
slaveDSA= cn=Giant Tortoise#
acl= others # read # entry
acl= group # c=US@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@cn=Manager # write # default
treeStructure= quipuNonLeafObject & organization & locality
treeStructure= quipuDSA & alias & organizationalRole
c= US
co= USA & US & United States of America
objectClass= top & quipuObject & quipuNonLeafObject
objectClass= country & friendlyCountry
```

These lines indicate that:

- The RDN for this object is `c=US` (read "country name is US").
- The user-friendly name for this country takes on one of three values

  ```
  USA
  US
  United States of America
  ```

- There are two DSAs which have SLAVE copies of the EDB for this object, one is called `cn=Giant Tortoise` and the other is called `cn=Fruit Bat`. (We'll discuss how these colorful names are chosen later on.)
- The DSA responsible for maintaining the MASTER copy of the EDB is called `cn=Alpaca`. Note that all three of these DSAs live directly under the ROOT of the Directory tree.
- Everyone is allowed to read the entry and all of its attributes, and only `c=US@cn=Manager` is allowed to change anything.

- The `treeStructure` attribute says that the children immediately below this object:
  - may have children themselves (`quipuNonLeafObject`), in which case, the objects under `c=US` are either `organization` or `locality` objects; or,
  - needn't have children, being a DSA (of object class `quipuDSA`) or an `organizationalRole`; or,
  - may be an `alias`.
- The official name of this country is `US`.
- This object belongs to five object classes:
  - `top` and `quipuObject` because membership is mandatory;
  - `quipuNonLeafObject` so that children may reside under the entry;
  - `friendlyCountry` which is the primary object class; and,
  - `country` which is a superclass of `friendlyCountry`.

The organization objectClass

Objects of this class represent a top-level organizational entity, such as a corporation, university, government entity, and so on. There is an `organization` object for each organization participating in the NYSERNet White Pages Pilot.

Table 1.3 summarizes the attributes types for an `organization` object. There is one mandatory attribute:

organizationName: which gives the name of the organization. The value of this attribute is a string; e.g.,

```
NYSERNet Inc.
```

There are several attributes that may be present, which are divided into three groups: physical address, telecommunications address, and miscellaneous information.

The first group describes the physical address of the object:

Superclass Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`top & quipuObject`
		`& quipuNonLeafObject`
		`& domainRelatedObject`
`acl`	M	
`masterDSA`	M	
`slaveDSA`	O	
`treeStructure`	O	`quipuNonLeafObject`
		`& organizationalUnit`
		`& quipuDSA`
		`& alias & organizationalRole`
		`& domainRelatedObject`

Primary Class Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`organization`
`organizationName`	RDN	
`associatedDomain`	O	
`postalAddress`	O	
`registeredAddress`	O	
`streetAddress`	O	
`physicalDeliveryOfficeName`	O	
`stateOrProvinceName`	O	
`postalCode`	O	
`localityName`	O	
`telephoneNumber`	O	
`facsimileTelephoneNumber`	O	
`telexNumber`	O	
`description`	O	

Legend

M:	mandatory
O:	optional, but recommended
RDN:	mandatory, must form the RDN for object

Table 1.3: Attribute Types for the organization Object Class

postalAddress: which describes how physical mail is addressed to the object. The syntax of this attribute's value is special: it consists of 1 to 6 fields, seperated by the "`$`"-sign, each field being from 1 to 30 characters long; e.g.

```
NYSERNet Inc. $ 165 Jordan Road $ Troy, NY 12180
```

registeredAddress: which defines how registered physical mail is addressed to the object. The syntax is identical to that of the `postalAddress` attribute.

streetAddress: which is a string describing where the object physically resides (i.e., the street name, place, avenue, and building number); e.g., object

```
165 Jordan Road
```

This need have no relationship to the object's postal address.

postOfficeBox: which is a string describing the box at which the object will receive physical postal delivery; e.g.,

```
1010
```

physicalDeliveryOfficeName: which is a string describing the geographical location of the physical delivery office which services the postal address of this object; e.g.,

```
Troy
```

stateOrProvinceName: which is a string describing the state in which the `locality` is found; e.g.,

```
New York
```

postalCode: which is a string containing the ZIP code; e.g.,

```
12180
```

or

```
94043-2112
```

localityName: which is a string describing the geographical area containing the `streetAddress`; e.g.,

```
Troy, New York
```

The second optional group describes telecommunications addressing information for the object.

telephoneNumber: which is a string describing the phone number of the object using the international notation; e.g.,

```
+1 518-283-8860
```

or

```
+1 518-283-8860 x1234
```

In general, the syntax is:

```
+<<country code>> <<national number>> [x<<extension>>]
```

facsimileTelephoneNumber: which is a string describing the fax number of the object using the international notation; e.g.,

```
+1 518-283-8904
```

telexNumber: which is defines the TELEX address of the object in a three-field format:

```
number $ country $ answerback
```

e.g.,

```
650 103 7390 $ US $ MCI UW
```

The final optional group contains one miscellaneous attribute:

description: which is a simple textual description; e.g.,

```
Not-for-profit organization providing network services
```

Figure 1.2 shows an entry for an `organization` object.

```
o=NYSERNet Inc.
masterDSA= c=US@cn=Spectacled Bear#
acl= others # read # entry
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # attributes # accessControlList$userPassword
o= NYSERNet Inc.
postalAddress= NYSERNet Inc. $ 165 Jordan Road $ Troy, NY 12180
streetAddress= 165 Jordan Road
physicalDeliveryOfficeName= Troy
stateOrProvinceName= New York
postalCode= 12180
telephoneNumber= +1 518-283-8860
facsimileTelephoneNumber= +1 518-283-8904
l= Troy, New York
description= Not-for-profit organization providing network services and software
treeStructure= quipuNonLeafObject & organizationalUnit
treeStructure= quipuDSA & alias & organizationalRole & thornPerson
objectClass= top & quipuObject & quipuNonLeafObject
objectClass= organization
```

Figure 1.2: An Example of an Entry for the organization Object Class

The organizationalUnit objectClass

Objects of this class represent a unit within an organization, such as a division or department. Each organization participating in the NYSERNet White Pages Pilot decides how many organizational units it wishes to maintain as subordinates to its organization entry.

Table 1.4 summarizes the attributes types for an `organizationalUnit` object. There is one mandatory attribute:

organizationalUnitName: which gives the name of the organizational unit. The value of this attribute is a string; e.g.,

```
Research and Development
```

There are several attributes that may be present. They belong to the physical address, telecommunications address, and miscellaneous information groups described earlier.

Figure 1.3 shows an entry for an `organizationalUnit` object.

Superclass Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`top & quipuObject`
		`& quipuNonLeafObject`
	O	`& domainRelatedObject`
`acl`	M	
`masterDSA`	M	
`slaveDSA`	O	
`treeStructure`	O	`quipuNonLeafObject`
		`& organizationalUnit`
		`& alias & pilotPerson`
		`& organizationalRole`
		`& domainRelatedObject`

Primary Class Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`organizationalUnit`
`organizationalUnitName`	RDN	
`associatedDomain`	O	
`postalAddress`	O	
`registeredAddress`	O	
`streetAddress`	O	
`physicalDeliveryOfficeName`	O	
`stateOrProvinceName`	O	
`postalCode`	O	
`localityName`	O	
`telephoneNumber`	O	
`facsimileTelephoneNumber`	O	
`telexNumber`	O	
`description`	O	

Legend

M:	mandatory
O:	optional, but recommended
RDN:	mandatory, must form the RDN for object

Table 1.4: Attribute Types for the organizationalUnit Object Class

```
ou=Research and Development
masterDSA= c=US@o=NYSERNet Inc.@cn=Toucan#
slaveDSA= c=US@cn=Spectacled Bear#
acl= others # read # entry
acl= others # read # default
acl= others # compare # attributes # accessControlList$userPassword
ou= Research and Development & Development
postalAddress= NYSERNet Inc. $ 165 Jordan Road $ Troy, NY 12180
streetAddress= 165 Jordan Road
physicalDeliveryOfficeName= Troy
stateOrProvinceName= New York
postalCode= 12180
telephoneNumber= +1 518-283-8860
facsimileTelephoneNumber= +1 518-283-8904
treeStructure= quipuNonLeafObject & organizationalUnit
treeStructure= alias & pilotPerson & organizationalRole
objectClass= top & quipuObject & quipuNonLeafObject
objectClass= organizationalUnit
```

Figure 1.3: An Example of an Entry for the organizationalUnit Object Class

The locality objectClass

Objects of this class represent a geographical area, such as a state, county, or city. The sponsors of the NYSERNet White Pages Pilot maintain such an entry for the State of New York. This is used to contain objects corresponding to individuals who are associated with organizations that are not participating in the pilot project.

Table 1.5 summarizes the attributes types for a `locality` object. There is one mandatory attribute:

localityName: which gives the name of the locality. The value of this attribute is a string; e.g.,

```
NY
```

or

```
Rensselaer County
```

There is only one attribute that may be present, it belongs to the miscellaneous information group described earlier.

Figure 1.4 shows the entry for the State of New York, whilst Figure 1.5 shows an entry for a subordinate `locality` object.

Superclass Attributes		
Attribute Name	**Status**	**Value**
objectClass	M	top & quipuObject
		& quipuNonLeafObject
	O	& domainRelatedObject
acl	M	
masterDSA	M	
slaveDSA	O	
treeStructure	O	quipuNonLeafObject
		& locality
		& quipuDSA
		& alias & pilotPerson
		& domainRelatedObject

Primary Class Attributes		
Attribute Name	**Status**	**Value**
objectClass	M	locality
localityName	RDN	
associatedDomain	O	
description	O	

Legend

M:	mandatory
O:	optional, but recommended
RDN:	mandatory, must form the RDN for object

Table 1.5: Attribute Types for the locality Object Class

```
l=NY
masterDSA= cn=Fruit Bat#
slaveDSA= cn=Alpaca#
acl= others # read # entry
acl= group # c=US@l=NY@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@l=NY@cn=Manager # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@l=NY@cn=Manager # write # attributes # accessControlList$userPassword
l= NY
description= State of New York
treeStructure= quipuNonLeafObject & locality
treeStructure= quipuDSA & alias & organizationalRole
objectClass= top & quipuObject & quipuNonLeafObject
objectClass= locality
```

Figure 1.4: The Entry for The State of New York

```
l=Rensselaer County
masterDSA= cn=Fruit Bat#
slaveDSA= cn=Alpaca#
acl= others # read # entry
acl= group # c=US@l=NY@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@l=NY@cn=Manager # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@l=NY@cn=Manager # write # attributes # accessControlList$userPassword
l= Rensselaer County & County of Rensselaer
description= A County in the State of New York
treeStructure= quipuNonLeafObject & locality
treeStructure= alias & pilotPerson & organizationalRole
objectClass= top & quipuObject & quipuNonLeafObject
objectClass= locality
```

Figure 1.5: An Example of an Entry for the locality Object Class

The pilotPerson objectClass

Objects of this class represent a person participating in the pilot project.

Table 1.6 summarizes the attributes types for a `pilotPerson` object. There are two mandatory attributes:

commonName: which gives a (potentially ambiguous) name for the person. The value of this attribute is a string usually containing the person's first and last names; e.g.,

```
Marshall Rose
```

This attribute is usually multi-valued, containing variations on the first, middle, and last names; e.g.,

```
Colin Robbins
Colin John Robbins
Colin J. Robbins
```

For purposes of the pilot project, the distinguished value of the attribute usually contains only the person's first and last names. (See Section 3.6.4 on page 93 for a discussion of this.)

surName: which gives the person's last name. The value of this attribute is a string; e.g.,

```
Rose
```

There are several attributes that may be present, which are divided into five groups: physical address, telecommunication information, computer environment, miscellaneous information, and home information. The first two groups have been previously described.

The third group describes information relating to the person's computer environment:

rfc822Mailbox: which is the user's computer mail address, e.g.,

```
mrose@nisc.nyser.net
```

otherMailbox: which is the user's computer mail address in various domains. The syntax of this attribute's value is special:

Superclass Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`top & quipuObject`
`acl`	M	

Primary Class Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`person & thornPerson & pilotPerson`
`commonName`	RDN	
`surName`	M	
`postalAddress`	O	
`registeredAddress`	O	
`roomNumber`	O	
`streetAddress`	L	
`postOfficeBox`	L	
`physicalDeliveryOfficeName`	L	
`stateOrProvinceName`	L	
`postalCode`	L	
`localityName`	L	
`telephoneNumber`	O	
`facsimileTelephoneNumber`	L	
`telexNumber`	L	
`rfc822Mailbox`	O	
`otherMailbox`	X	
`userid`	O	
`userClass`	O	
`description`	O	
`info`	O	
`businessCategory`	O	
`title`	O	
`userPassword`	O	
`mobileTelephoneNumber`	X	
`pagerTelephoneNumber`	X	
`favouriteDrink`	O	
`secretary`	X	
`seeAlso`	X	
`photo`	X	
`homePostalAddress`	O	
`homePhone`	O	

Legend

M:	mandatory
O:	optional, but recommended
L:	optional, recommended if person is under a locality or geographically distant from organization
X:	optional, use only if applicable
RDN:	mandatory, must form the RDN for object

```
<domain> $ <mailbox>
```

e.g.,

```
internet $ mrose@nisc.nyser.net
```

The current list of mail domains are:

applelink
bitnet
compuserve
genie
internet
mcimail
nasamail
preferred
uucp

userid: which is the user's login name; e.g.,

```
mrose
```

userClass: which describe's the user's classification; e.g.,

```
staff
```

The next optional group contains a few miscellaneous attributes:

description: which is a simple textual description; e.g.,

```
Principal Implementor of the ISO Development Environment
```

info: which is additional information about the object; e.g.,

```
It's nearly as good as BIND
```

businessCategory: which describes the person's business, e.g.,

```
networking
```

title: which is the person's job title, e.g.,

```
Senior Scientist
```

roomNumber: which is a string describing where the object resides at the location, e.g.,

```
Building T-30
```

userPassword: which is a string containing the object's password in the Directory. This is used, for example, when the user wants to update the entry. The password is kept in the clear (the security ramifications of this are discussed later on); e.g.,

```
secret
```

mobileTelephoneNumber: which is a string describing the user's mobile number (e.g., for a cellular phone).

pagerTelephoneNumber: which is a string describing the user's pager number.

favouriteDrink: which is a string describing the user's favorite drink.

secretary: which is the Distinguished Name of the user's secretary.

seeAlso: which is a Distinguished Name pointing to another entry related to the user (perhaps in a different role).

photo: which is a facsimile bitmap of the user's face.

The final optional group contains a few attributes about the person at home:

homePostalAddress: which describes how physical mail is addressed to the person's home. The syntax of this attribute's value is special: it consists of 1 to 6 fields, seperated by the "`$`"-sign, each field being from 1 to 30 characters long; e.g.

```
NYSERNet Inc. $ 165 Jordan Road $ Troy, NY 12180
```

homePhone: which is a string describing the phone number of the object using the international notation; e.g.,

```
+1 518-283-8860
```

Figure 1.6 shows an entry for a `pilotPerson` object.

```
cn=Marshall Rose
acl= self # write # entry
acl= others # read # entry
acl= self # write # default
acl= others # read # default
acl= self # write # attributes # accessControlList$userPassword
acl= others # compare # attributes # accessControlList$userPassword
cn= Marshall Rose & Marshall T. Rose
surname= Rose
telephoneNumber= +1 415-961-3380
facsimileTelephoneNumber= +1 415-961-3282
rfc822Mailbox= mrose@nisc.nyser.net
otherMailbox= internet $ mrose@nisc.nyser.net
otherMailbox= uucp $ nyser!mrose
userid= mrose
title= Senior Scientist
description= Principal Implementor of the ISO Development Environment
userPassword= secret
objectClass= top & quipuObject
objectClass= person & thornPerson & pilotPerson
```

Figure 1.6: An Example of an Entry for the pilotPerson Object Class

The alias objectClass

Objects of this class represent an alias to some other entry in the NYSERNet White Pages Pilot. Each organization participating in the pilot must define at least two of these:

- a manager responsible for the organization's portion of the Directory tree; and,
- a postmaster responsible for the organization's electronic mail.

Table 1.7 summarizes the attributes types for an `alias`. There are two mandatory attributes:

commonName: which gives the name of the alias. The value of this attribute is a string; e.g.,

```
PostMaster
```

aliasedObjectName: which is a pointer to another object in the Directory; e.g.,

Superclass Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`top & quipuObject`
`acl`	M	

Primary Class Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`alias & organizationalRole`
`commonName`	RDN	
`aliasedObjectName`	M	

Legend

M:	mandatory
RDN:	mandatory, must form the RDN for object

Table 1.7: Attribute Types for the alias Object Class

```
c=US@o=NYSERNet Inc.@ou=Operations@cn=Chris Kolb
```

NOTE:	The value of this attribute must not be another alias: each alias must point to a non-aliased entry!

There are no optional attributes for this object class. Figure 1.7 shows an entry for an `alias` object.

The organizationalRole objectClass

Objects of this class represent a position filled by a person.

Table 1.8 summarizes the attributes types for a `organizationalRole`. There is one mandatory attribute:

commonName: which gives the name of the role. The value of this attribute is a string; e.g.,

```
Manager
```

```
cn=Manager
acl=
cn= Manager
aliasedObjectName= c=US@o=NYSERNet Inc.@ou=Development@cn=Wengyik Yeong#
objectClass= top & quipuObject
objectClass= alias
```

Figure 1.7: An Example of an Entry for the alias Object Class

```
cn=Office Manager
acl= others # read # entry
acl= group # c=US@o=Yoyodyne@ou=Corporate@cn=Kimberly Brown # write # entry
acl= others # read # default
acl= group # c=US@o=Yoyodyne@ou=Corporate@cn=Kimberly Brown # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@o=Yoyodyne@ou=Corporate@cn=Kimberly Brown # write # attributes # accessControlList$userPassword
cn= Office manager
telephoneNumber= +1 518-283-8860
facsimileTelephoneNumber= +1 518-283-8904
description= Manager for the Troy Office
roleOccupant= c=US@o=Yoyodyne@ou=Corporate@cn=Kimberly Brown#
objectClass= top & quipuObject
objectClass= organizationalRole
```

Figure 1.8: An Example of an Entry for the organizationalRole Object Class

There are several attributes that may be present, which are divided into three groups: physical address, telecommunication information, and miscellaneous information. The first two groups have been previously described. The final optional group contains two miscellaneous attributes:

description: which is a simple textual description; e.g.,

```
Manager of the Troy Office
```

roleOccupant: which is a pointer to another object in the Directory, referring to the person who currently fills the role. e.g.,

```
c=US@o=NYSERNet Inc.@ou=Administration@cn=Kimberly Brown
```

Figure 1.8 shows an entry for a `organizationalRole` object.

Superclass Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`top & quipuObject`
`acl`	M	

Primary Class Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`organizationalRole`
`commonName`	RDN	
`postalAddress`	O	
`registeredAddress`	O	
`streetAddress`	N	
`physicalDeliveryOfficeName`	N	
`stateOrProvinceName`	N	
`postalCode`	N	
`localityName`	O	
`telephoneNumber`	O	
`facsimileTelephoneNumber`	O	
`telexNumber`	O	
`description`	O	
`roleOccupant`	O	

Legend

M:	mandatory
O:	optional, but recommended
N:	optional, not recommended
RDN:	mandatory, must form the RDN for object

Table 1.8: Attribute Types for the organizationalRole Object Class

The quipuDSA objectClass

Objects of this class represent a Directory System Agent (DSA) running the pilot project software. Each organization participating in the NYSERNet White Pages Pilot typically runs at least one DSA.

Table 1.9 summarizes the attributes types for a `quipuDSA`. There are several mandatory attributes:

commonName: which gives the name of the DSA, by convention this is the name of an endangered species in South America; e.g.,

```
Fruit Bat
```

eDBinfo: which indicates how the DSA participates when propagating authoritative EDB information (this is the dual of the `masterDSA` and `slaveDSA` attribute types discussed earlier). The syntax of this attribute's value is special:

```
<name> # <upstream> # <downstream>
```

The `<name>` field is a Distinguished Name referring to the object containing the EDB. By convention, if absent this refers to the ROOT (unnamed) EDB. The `<upstream>` field is the Distinguished Name of a DSA which is considered to be "closer" to the MASTER of this EDB. It may be the name of the actual DSA containing the MASTER EDB, but it might also be the name of an intermediate DSA (containing a SLAVE copy). If this field is empty, then this DSA does not pull the EDB from another DSA (this needn't mean that it holds the MASTER EDB, though usually this is the case). The `<downstream>` field is the Distinguished Name of a DSA which is allowed to copy the EDB from this DSA. If this field is empty, then this DSA does not allow other DSAs to copy the EDB. If the `<downstream>` field is to contain more than one DSA, either enter multiple `eDBinfo` lines or separate the Distinguished Names of the downstream DSAs with the '`$`'-sign.

A few examples:

```
# cn=Giant Tortoise # cn=Fruit Bat
c=US # # cn=Fruit Bat
c=US # # c=US@cn=Spectacled Bear
c=US@l=NY # cn=Fruit Bat #
```

Note that there is no harm to using multiple `eDBinfo` lines, even if they refer to the same EDB. These lines indicate that:

- The ROOT EDB is read from the `cn=Giant Tortoise` DSA, further the `cn=Fruit Bat` DSA is allowed to read the ROOT EDB from this DSA.
- The `cn=Fruit Bat` DSA and the `c=US@cn=Spectacled Bear` DSA are allowed to read the EDB for `c=US`. Note that the second and third line could be combined as:

  ```
  c=US # # cn=Fruit Bat$c=US@cn=Spectacled Bear
  ```

  While more concise, this format is not recommended as it is probably more difficult to read.
- The DSA named `cn=Fruit Bat` supplies the EDB for `c=US@l=NY` to this DSA.

presentationAddress: which is the OSI presentation address that the DSA resides at. The syntax of this attribute's value is special:

```
'0101'H/Internet=<quad>+<port>
```

or

```
'0101'H/Internet=<quad-1>+<port>|Internet=<quad-2>+<port>
```

if the DSA is capable of listing on multiple IP addresses.[3] The `<quad>` field is a 32-bit IP address expressed in the usual quad notation (e.g., `130.117.128.2`). The `<port>` field is a decimal TCP port which isn't likely to be used on your system. The value `17003` is recommended for the purposes of the pilot project.

A simple example:

```
'0101'H/Internet=130.117.128.2+17003
```

Note that multiple values for the `presentationAddress` attribute type should not be used: multiple IP addresses (up to 4) can be specified using a single attribute value.

[3]Actually, the rules for forming OSI presentation addresses are *much* more complicated; however, for the purposes of the pilot project, this should suffice.

manager: which is a pointer to another object in the Directory of the person who manages this DSA; e.g.,

```
c=US@o=NYSERNet Inc.@cn=Manager
```

userPassword: which is a string containing the DSA's password in the Directory. This is used when the manager of the DSA wants to change the entry for the DSA itself (as opposed to the entries held by the DSA). The password is kept in the clear (the security ramifications of this are discussed later on); e.g.,

```
secret
```

quipuVersion: which is a simple string relating the version of the pilot project software being run by this DSA; e.g.,

```
Version 5.0
```

There are two optional attributes:

description: which, by convention, is multi-valued. The distinguished value is the wildlife description of the `commonName` attribute whilst the other values describe the role that the DSA plays in the pilot project; e.g.,

```
The Mischevous Varmint
A MASTER DSA for l=NY under c=US
A SLAVE DSA for ROOT and c=US
```

supportedApplicationContext: which defines the application ports offerred by this process. The current values are:

```
x500DSP & x500DAP & quipuDSP
```

Figure 1.9 shows an entry for a `quipuDSA` object.

1.2 Functional Model

The Functional Model describes the service in terms of entities. As noted earlier, two kinds of entities exist: *Directory User Agents* (DUAs), which operate on behalf of a user, and *Directory System Agents* (DSAs), which provide the service.

Superclass Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`top & quipuObject`
`acl`	M	

Primary Class Attributes		
Attribute Name	**Status**	**Value**
`objectClass`	M	`dSA & quipuDSA`
`commonName`	RDN	
`eDBinfo`	M	
`presentationAddress`	M	
`manager`	M	
`userPassword`	M	
`quipuVersion`	M	
`supportedApplicationContext`	O	`x500DSP & x500DAP & quipuDSP`
`description`	O	

Legend

M:	mandatory
O:	optional, but recommended
RDN:	mandatory, must form the RDN for object

Table 1.9: Attribute Types for the quipuDSA Object Class

```
cn=Spectacled Bear
acl= others # read # entry
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # attributes # accessControlList$userPassword
cn= Spectacled Bear
eDBinfo= # cn=Fruit Bat #
eDBinfo= c=US # cn=Alpaca #
eDBinfo= c=US@o=NYSERNet Inc. # # cn=Fruit Bat
eDBinfo= c=US@o=NYSERNet Inc. # # cn=Alpaca
presentationAddress= '0101'H/Internet=130.117.128.3+17003
manager= c=US@o=NYSERNet Inc.@cn=Manager#
manager= c=US@cn=Manager#
userPassword= spectacledbear
quipuVersion= 5.0 Distribution
description= The South American spectacled bear
description= Master DSA for NYSERNet Inc.
objectClass= top & quipuObject
objectClass= appliationEntity & dSA & quipuDSA
supportedApplicationContext= x500DSP & x500DAP & quipuDSP
```

Figure 1.9: An Example of an Entry for the quipuDSA Object Class

The key aspect of the functional model is deciding how information is transferred through the system. When a DUA requests some action from a DSA (e.g., to read an entry), the DSA may not have that information resident. In this case, the DSA has a choice: it may either contact another DSA which is "closer" to the information and propagate the request (this is called *chaining*), or it may return information about this "closer" DSA to the DUA, and let the DUA re-issue its request (this is called *referral*). WHen DSAs communicate between themselves, they may also chain or refer requests.

For the purposes of the pilot, it is important to understand what the term "resident" means with respect to information. This residency requirement is:

Operation Requested	**EDB Required for Residency**
read, compare	MASTER, SLAVE, or CACHE
list, search	MASTER, or SLAVE
update	MASTER

Note that after an update occurs, the DSA actually performing the update (the one holding the MASTER EDB) will write the in-core EDB structure to disk prior to returning the result of the operation. Of course, once a MASTER DSA updates its copy of the EDB, there is still the question of how DSAs with SLAVE copies receive this new information. The MASTER DSA does not explicitly inform them. A polling model is used instead.

At present, a SLAVE DSA will reload its copy of an EDB when it starts, or when its `manager` issues a management command directing it to do so. Later on, a script will be discussed which performs this task. The frequency at which the script runs (or how often a DSA reboots) determines how out of date a SLAVE copy of an EDB can become.

Prior to requesting a copy of an EDB, a DSA checks to see if the timestamp has changed. If so, the EDB is updated. As noted earlier, the timestamp is the second line of an EDB file. Whenever a modification is made to an entry in an EDB file, the MASTER DSA for that EDB will update the timestamp accordingly. Similarly, whenever a text editor is used to modify an EDB file, the user should update the timestamp. The actual value is unimportant: when a DSA examines the timestamp, it looks only for a change; it does not consider whether the EDB is newer or older.

Thus, while replication in the pilot is important to speed queries, updates still rely on a centralized entity being available. (This is the best compromise which can be taken without making the system tremendously more complex.)

1.3 Organizational Model

The Organizational Model describes the service in terms of the policy defining interactions between entities and the information they hold. That is, this model describes how portions of the Directory tree map onto the DSAs. This includes issues of replication and access control. A *Directory Management Domain* (DMD) defines a portion of the Directory Tree and how it is managed.

For the purposes of the pilot project, DSAs are divided into three categories:

Level-0: These DSAs are run by the sponsors of the NYSERNet White Pages Pilot. Their goal is to be highly-available authoritative servers.

Level-1: Each organization participating in the pilot project runs a Level-1 DSA.[4] The Level-1 DSA for a participating organization holds authoritative information about that organization object.

[4]The pilot project sponsors may run the Level-1 DSA on behalf of an organization so requested. This is discouraged, but may be necessary for resource constrained sites.

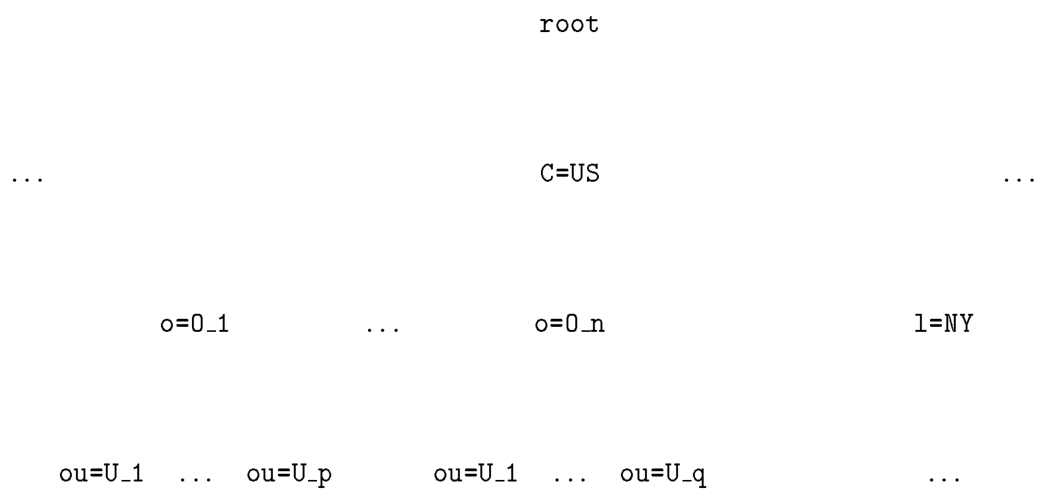

Figure 1.10: Topology of the Pilot Project Directory Management Domain

Level-2: As the size of the organization's subtree requires, an organization runs one or more Level-2 DSA to hold a portion of that subtree. Typically, a Level-2 DSA is run for each top-level organizational unit in the participating organization.

Figure 1.10 shows the top-level topology of the pilot project DMD. Each participating organization is represented by an `organization` object (O_1 through O_n) under the `c=US` tree. Underneath each `c=US@o=O_i` object, one or more objects of class `organizationalUnit` are placed. The structure beneath each `c=US@o=O_i@ou=U_j` object is decided by the participating organization. The pilot project sponsors suggest that only `pilotPerson` objects be placed underneath the organizational units.

Now that the general structure is outlined, how does this portion of the Directory tree map onto DSAs?

1.3.1 Level-0 DSAs

There are two Level-0 DSAs. Both reside directly under the ROOT.

The first Level-0 DSA, `cn=Alpaca`, contains a copy of the ROOT EDB which it obtains directly from the DSA holding the MASTER copy of the ROOT EDB.

```
cn=Alpaca
acl= others # read # entry
acl= group # c=US@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@cn=Manager # write # default
cn= Alpaca
eDBinfo= # cn=Giant Tortoise #
eDBinfo= # # cn=Fruit Bat
eDBinfo= # # c=US@cn=Spectacled Bear
eDBinfo= c=US # # cn=Giant Tortoise
eDBinfo= c=US # # cn=Fruit Bat
eDBinfo= c=US # # c=US@cn=Spectacled Bear
eDBinfo= c=US@l=NY # cn=Fruit Bat #
eDBinfo= c=US@o=NYSERNet Inc. # c=US@cn=Spectacled Bear #
presentationAddress= '0101'H/Internet=130.117.128.2+17007
manager= c=US@cn=Manager
userPassword= alpaca
quipuVersion= 5.0 Distribution
description= The Wily Alpaca
description= a second MASTER DSA for c=US
objectClass= top & quipuObject
objectClass= applicationEntity & dSA & quipuDSA
supportedApplicationContext= x500DSP & x500DAP & quipuDSP
```

Figure 1.11: The Entry for the Level-0 DSA, cn=Alpaca

In addition, `cn=Alpaca` is a MASTER DSA for `c=US`. It propagates this EDB to the second Level-0 DSA, and to the Level-1 DSAs of all participating organizations (e.g., each DSA holding the MASTER of the `c=US@o=O_i` EDB). In addition, these Level-1 DSAs propagate a copy of their EDB for `c=US@o=O_i` to this Level-0 DSA. Finally, this DSA receives a copy of the EDB for `c=US@l=NY`.

Figure 1.11 shows the entry corresponding to this DSA. A single participating organization is shown in this entry.

The second Level-0 DSA, `cn=Fruit Bat`, contains a copy of the ROOT and `c=US` EDBs which it obtains from `cn=Alpaca`. It may propagate these EDBs to any Level-1 DSA (in case `cn=Alpaca` is unavailable for an extended period of time). In addition, these Level-1 DSAs propagate a copy of their MASTER EDB to this Level-0 DSA. Finally, `cn=Fruit Bat` is a MASTER DSA for `c=US@l=NY`, which it propagates to `cn=Alpaca`.

```
cn=Fruit Bat
acl= others # read # entry
acl= group # c=US@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@cn=Manager # write # default
cn= Fruit Bat
eDBinfo= # cn=Alpaca #
eDBinfo= # # c=US@cn=Spectacled Bear
eDBinfo= c=US # cn= Alpaca #
eDBinfo= c=US # # c=US@cn=Spectacled Bear
eDBinfo= c=US@l=NY # # cn=Alpaca
eDBinfo= c=US@o=NYSERNet Inc. # c=US@cn=Spectacled Bear #
presentationAddress= '0101'H/Internet=130.117.128.3+17007
manager= c=US@cn=Manager
userPassword= fruitbat
quipuVersion= 5.0 Distribution
description= The Mischievous Varmint
description= a MASTER DSA for l=NY under c=US & a SLAVE DSA for c=US
objectClass= top & quipuObject
objectClass= appliationEntity & dSA & quipuDSA
supportedApplicationContext= x500DSP & x500DAP & quipuDSP
```

Figure 1.12: The Entry for the Level-0 DSA, cn=Fruit Bat

Figure 1.12 shows the entry corresponding to this DSA. As with the `cn=Alpaca` DSA, a single participating organization is shown.

Level-0 DSAs perform chaining when handling requests from other DSAs.

1.3.2 Level-1 DSAs

Each participating organization runs a DSA which contains the MASTER EDB for that organization. This DSA resides directly under `c=US`.

This DSA receives a copy of the ROOT and `c=US` EDBs from one of the Level-0 DSAs (usually `cn=Alpaca`). It also propagates its MASTER copy of the `c=US@o=O_i` EDB to both Level-0 DSAs.

In addition, this DSA propagates its EDB and a copy of the ROOT and `c=US` EDBs to all Level-2 DSAs run by this organization. Similarly, each of these Level-2 DSAs propagate *all* of the EDBs they MASTER (e.g., `c=US@o=O_i@ou=U_j`) to this Level-1 DSA.

```
cn=Spectacled Bear
acl= others # read # entry
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # attributes # accessControlList$userPassword
cn= Spectacled Bear
eDBinfo= # cn=Fruit Bat #
eDBinfo= # # c=US@o=NYSERNet Inc.@cn=Toucan
eDBinfo= c=US # cn=Alpaca #
eDBinfo= c=US # # c=US@o=NYSERNet Inc.@cn=Toucan
eDBinfo= c=US@o=NYSERNet Inc. # # cn=Fruit Bat
eDBinfo= c=US@o=NYSERNet Inc. # # cn=Alpaca
eDBinfo= c=US@o=NYSERNet Inc. # # c=US@o=NYSERNet Inc.@cn=Toucan
eDBinfo= c=US@o=NYSERNet Inc.@ou=Example Unit # c=US@o=NYSERNet Inc.@cn=Toucan #
presentationAddress= '0101'H/Internet=130.117.128.3+17003
manager= c=US@o=NYSERNet Inc.@cn=Manager#
manager= c=US@cn=Manager#
userPassword= spectacledbear
quipuVersion= 5.0 Distribution
description= The South American spectacled bear
description= MASTER DSA for NYSERNet Inc.
objectClass= top & quipuObject
objectClass= appliationEntity & dSA & quipuDSA
supportedApplicationContext= x500DSP & x500DAP & quipuDSP
```

Figure 1.13: The Entry for a Level-1 DSA

Level-1 DSAs perform chaining when handling requests from other DSAs.

Figure 1.13 shows the entry corresponding to a Level-1 DSA, which might be named `c=US@cn=Spectacled Bear`. A single subordinate Level-2 DSA is shown in this entry.

1.3.3 Level-2 DSAs

Each participating organization runs zero or more Level-2 DSAs as the size of their organizational units require. These DSAs reside directly under `c=US@o=O_i`.

```
cn=Toucan
acl= others # read # entry
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@o=NYSERNet Inc.@cn=Manager # write # attributes # accessControlList$userPassword
cn= Toucan
eDBinfo= # c=US@cn=Spectacled Bear #
eDBinfo= c=US # c=US@cn=Spectacled Bear #
eDBinfo= c=US@o=NYSERNet Inc. # c=US@cn=Spectacled Bear #
eDBinfo= c=US@o=NYSERNet Inc.@ou=Example Unit # # c=US@cn=Spectacled Bear
presentationAddress= '0101'H/Internet=130.117.128.4+17003
manager= c=US@o=NYSERNet Inc.@cn=Manager#
manager= c=US@cn=Manager#
userPassword= toucan
quipuVersion= 5.0 Distribution
description= An Example of Endangered South American Wildlife
description= Master DSA for an Example Unit in NYSERNet Inc.
objectClass= top & quipuObject
objectClass= appliationEntity & dSA & quipuDSA
supportedApplicationContext= x500DSP & x500DAP & quipuDSP
```

Figure 1.14: The Entry for a Level-2 DSA

NOTE: At present, the pilot sponsors discourage the use of Level-2 DSAs. Nevertheless, it is important to structure an organization so that it contains organizational units. In this fashion, when a Level-1 DSA is too large to run on an available system, a Level-2 DSA can be used to reduce the memory requirements on the system running the Level-1 DSA.

Level-2 DSAs receive a copy of the `c=US@o=O_i` EDB from their superior Level-1 DSA, along with SLAVE copies of the ROOT and `c=US` EDBs. They also propagate all MASTER copies of the EDBs they hold to this Level-1 DSA.

Level-1 DSAs perform referrals when handling requests from other DSAs.

Figure 1.14 shows the entry corresponding to a Level-2 DSA, which might be named `c=US@o=NYSERNet Inc.@cn=Toucan`.

1.3.4 DUAs

In terms of the pilot project, DUAs conceptually exist at the same level as Level-2 DSAs. When a DUA associates with a DSA to access the Directory service, chaining is preferred over referrals, so that the associated DSA may cache the results to any query made by the DUA. This will slightly enhance the performance for other DUAs which later associate with this particular DSA.

The preference to be used when associating with a DSA is:

- "closest" Level-2 DSA; followed by
- Level-1 DSA; followed by
- `cn=Fruit Bat`; followed by
- `cn=Alpaca`.

1.4 Security Model

The Security Model describes the service in terms of authentication and authorization. For the purposes of the pilot project, "simple" authentication is performed (i.e., passwords). The pilot project supports an anonymous DN, termed the NULL DN, which may be used for public, read-only access. Otherwise, users are required to bind to the Distinguished Name which corresponds to them.

Note that an important feature of the DMD topology presented in the Organizational Model is that information on users (and hence their passwords and any other sensitive information) is kept only on Level-1 and Level-2 DSAs. Since, in general, these DSAs are run by each participating organization, each organization may safeguard the disk-resident EDB files accordingly.

Because the ROOT, `c=US`, and `c=US@o=O_i` EDBs are widely replicated, entries contained therein should not rely on `self` access control for modification. That is, the entries contained in this EDBs should:

- if of object class `quipuDSA`, have a `manager` attribute which resolves to an entry which is not propagated outside of the associated organization; e.g.,

  ```
  manager= c=US@o=O_i@cn=Manager
  ```

  which is an object of classe `alias` to

```
c=US@o=O_i@ou=Development@cn=FirstName LastName
```

and,

- regardless of the object class, always use `group` access to specify an entry which is not propagated outside of the associated organization; e.g.,

```
acl= others # read # entry
acl= group # c=US@o=O_i@cn=Manager # write # entry
acl= others # read # default
acl= group # c=US@o=O_i@cn=Manager # write # default
acl= others # compare # attributes # accessControlList$userPassword
acl= group # c=US@o=O_i@cn=Manager # write # attributes # accessControlList$userPassword
```

Note that the object associated with the `manager` attribute of a DSA is effectively the "super user" for all EDBs which that DSA MASTERs. As such, the `userPassword` associated with the corresponding object should be protected to the greatest extent practical.

Since there is no hope of protecting the passwords for a DSA or any object residing in the ROOT, `c=US`, and possibly `c=US@o=O_i` EDBs, this policy effectively revokes any privileges assocated with the passwords of the entries contained in these EDBs.

1.5 Running your DMD

The pilot sponsors prefer that you administer your own DMD by running the pilot software on one of your own systems, doing maintenance, and so on. You should now continue with Chapter 2.

However, if your site is a member of NYSERNet, Inc., then the pilot sponsors will administer the DMD for you, if you lack the facilities to do so. In this case, you will be required to provide EDB files to the pilot sponsors. Contact

```
wpp-manager@nisc.nyser.net
```

to find out the full details.

For now, note that entries are formatted as described above, but with a few read-only modifications:

- The `accessControlList` attribute is simply:

```
acl=
```

- The `userPassword` attribute is not present.

Chapter 2

Installation and Configuration

This chapter describes how to install, configure, and test the pilot project software. Upon completion, your Level-1 DSA will have successfully joined the pilot project DMD!

2.1 Software

The software supporting the pilot project is the QUIPU implementation of the OSI Directory, which is a part of the ISO Development Environment (ISODE).

QUIPU was originally developed as a part of the INCA project (under the auspices of the ESPRIT initiative of the EEC). 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. Such a device was known as a *Quipu* (pronounced *kwip-ooo*). The encoding was obscure, and could only be read by selected trained people: the *Quipucamayocs*. The Quipu was a key component of Inca society, as it contained information about property and locations throughout the extensive Inca empire.

The pilot sponsors and the administrators of the participating organizations form the *camayocs* for the pilot project. There is a special mailing list

```
wpp-camayocs@nisc.nyser.net
```

which each *Camayoc* belongs to.

NOTE: This is the official mailing list for administrators of the pilot project. If you have a problem, send mail!

2.1.1 Selecting a Machine

The machine you run the software on should support the Berkeley variant of UNIX[1]. Although the software is known to run on other variants of UNIX, such as those from AT&T and HP, these configurations are not supported by the pilot sponsors.

You should plan on the DSA consuming 2Kbytes of memory *per* entry kept in your DMD This is called the QUIPU *2K rule*. Make sure that your machine has available memory for the DSA. A memory-constrained (or worse yet, paging DSA) performs extremely poorly. Find an unloaded system to run your DSA. This investment might seem large, just think of it as a sunk cost.

2.1.2 Files

Regardless of how you install QUIPU and the ISODE, the number of files needed for the pilot project are quite small.

In ISODE's `BINDIR` directory, typically */usr/local/bin/*, there are a few programs of interest:

dish: The DIrectory SHell
This is a interface to the OSI Directory which can be used to access the full functionality of the Directory service. This program is not directly run by the "users" of the pilot project, rather various front-ends access *dish*. In addition, the administrators employ this program directly to perform routine maintenance and diagnostics.

bind: Shell interface to *dish*
There are actually several links to a program called *bind*. These act to export the *dish* interface to the UNIX shell. As such, you can issue commands to *dish* from the shell, rather than running *dish* directly.

[1]UNIX is a trademark of AT&T Bell Laboratories.

add
compare
delete
dsacontrol
list
modify
modifyrdn
moveto
search
showentry
showname
squid

fred: A white pages user interface
This is the interface for the users of the pilot project.

editentry: Edit a Directory entry
This is a simple shell script that *dish* invokes when you ask *dish* to edit an entry in the Directory.

unbind: Unbind from *dish*
This command is used to terminate *dish*.

In ISODE's `SBINDIR` directory, typically */usr/etc/*, the DSA resides:

ros.quipu: The QUIPU DSA
This program will be started once, for each DSA you are running, from rc.local. A script is provided to invoke this program, in case you need to restart it.

dsaconfig: a configurator for Level-1 DSAs.

In ISODE's `ETCDIR` directory, also typically */usr/etc/*, there are a few programs and files of interest:

oidtable.at, oidtable.gen, oidtable.oc: These define the attribute types, generic object identifiers, and object classes known to the system. (An object identifier is a method used to unambiguously encode, among other things, the names of attributes and object classes.) These files you never deal with unless they are accidently corrupted.

dsaptailor: This is the run-time tailor file for the DUAs. You will configure this file initially and then probably leave it alone.

isoaliases, isobjects, isoentities, isomacros, isoservices: These are various databases used by the ISODE. These files you never deal with unless they are accidently corrupted.

isologs: This script runs nightly under *cron*(8) to trim the ISODE log files, kept in ISODE's `LOGDIR` directory, typically */usr/tmp/*. This file you never deal with unless it is accidently corrupted.

isotailor: This is the run-time tailor file for the ISODE. You will configure this file initially and then probably leave it alone.

There are a few other things kept in a directory called *quipu/* which resides in ISODE's `ETCDIR` directory. The *scripts/* directory contains various scripts which may be used for maintenance purposes. The *templates/* directory contains templates of entries.

2.1.3 Source Release

You should retrieve a copy of the pilot software. It is available via anonymous FTP from from host `nisc.nyser.net` (`[192.33.4.10]`):

```
% cd /usr/src/local/
% ftp nisc.nyser.net
Connected to nisc.nyser.net.
220 nisc.nyser.net FTP server ready
Name (nisc.nyser.net:user): anonymous
Password (nisc.nyser.net:anonymous): guest
331 Guest login ok, send ident as password.
230 Guest login ok, access restrictions apply.
ftp> binary
200 Type set to I.
ftp> cd pilot/src
ftp> get isode-pilot.tar.Z
ftp> quit
% uncompress < isode-pilot.tar.Z | tar xfp -
% rm isode-pilot.tar.Z
```

Note that this is a interim release of the ISODE. All of the changes are upwards compatible, but you must run this version rather than the standard ISODE 5.0 release.

Now change to the top-level directory.

```
% cd isode-5.9/
```

The file *READ-ME* in this directory contains instructions on how to configure, generate, and install the ISODE.

NOTE: The source installation instructions are intended for sites which are not already running QUIPU. If you are, care should be taken to see that your existing Directory environment is not compromised. For example, the DSAs run under the pilot do not provide the "higher performance" name service, nor is it intended that they support application entities other than DSAs.

Configuration of the ISODE

As described in the *READ-ME* file, configuration is straight-forward. Here's an example of how it's done on a Sun workstation running SunOS 3.5:

```
% cp config/sunos3.h h/config.h
% cp config/sunos3.make config/CONFIG.make
% cp config/*.local support/
```

For other systems, consult the *READ-ME* file to see which templates to use. For all systems, take a look at

h/config.h
config/CONFIG.make

to see if things like `BINDIR`, `SBINDIR`, `ETCDIR`, and `LOGDIR` are set to your liking.

Generation of the ISODE

Once properly configured, generate both the ISODE and QUIPU using this command from the top-level directory:

```
% ./make once-only all all-quipu
```

This takes a while, so its best to redirect the output to a file and then go do something else for a few hours:

```
% ./make once-only all all-quipu >& make.out &
```

If the make command finishes with a non-zero exit status, or if you see a line with:

```
*** Error code 1
```

or perhaps

```
Exit
```

Then something has gone wrong. Try and determine what the problem is. If you are unable to find the problem, drop a note to the Internet Mailbox

```
Bug-ISODE@NISC.NYSER.NET
```

Your message should contain your two configuration files, the output of the make command, and a description of the hardware/software platform you are using.

Keep in mind that the ISODE runs "flawlessly" worldwide on many thousands of machines running numerous variants of UNIX. Failure to complete generation usually indicates a broken configuration file, or a non-standard system (i.e., one with a broken compiler or a different *make* command).

Now generate the additional commands needed for the pilot project:

```
% (cd others/quipu; ./make pilot)
```

Installation of the ISODE

Once you are satisfied that the generation was successful, become the super-user and install both the ISODE and QUIPU. First make sure that the directories `BINDIR`, `SBINDIR`, `ETCDIR`, `LOGDIR`, as defined in *config/CONFIG.make* exist. If not, create them using *mkdir*. The protection on the `LOGDIR` directory should be 0777.

Next, install the programs, databases, and libraries:

```
# ./make inst-all inst-quipu
# (cd others/quipu; ./make inst-pilot)
```

Note the use of trailing slashes in the last command.

Do **not** perform any other installation instructions for QUIPU as given in the *READ-ME* file. The remainder of this chapter supercede those instructions.

If you are not planning on doing program development (probably a safe bet), you can remove the various libraries and program development tools by using

```
# ./make zap
```

from the *isode-5.9/* directory as the super-user.

2.2 Starting the ISODE

In order to aid debugging, you should keep the ISODE system around in an operational state. So, you need to do a few more things:

1. Verify that the file */etc/rc.local* has an entry like this:

   ```
   if [ -f $(SBINDIR)tsapd ]; then
      $(SBINDIR)tsapd -t & (echo -n ' tsap') > /dev/console
   fi
   ```

   in the section where the network servers are started (e.g., where *sendmail*(8) is started).

2. Verify that the file */etc/services* has an entry like this:

   ```
   tsap    102/tcp
   ```

3. Verify that the file */usr/lib/crontab* has an entry similar to this:

   ```
   0 4 * * * su daemon < $(SBINDIR)isologs
   ```

   or this

   ```
   0 4 * * * su daemon -c "/bin/sh $(SBINDIR)isologs"
   ```

   This runs a script to clean-up the logfiles at 4:00am each morning.

4. Start the *tsapd*(8c) daemon by hand:

   ```
   # $(SBINDIR)tsapd -t >& /dev/null
   ```

The daemon will automatically detach. Instead, if its diagnostic output is associated with a terminal, then the daemon will not detach and will log informative messages to the terminal.

5. Now verify that the ISODE is installed and tailored correctly. See Sections 2.3 and 2.4.

2.3 ISODE Tailoring

Before any DSAs or DUAs are installed, the file *isotailor* in the ISODE `ETCDIR` directory should be properly configured.

Typically source installations have ISODE's `ETCDIR` and `LOGDIR` directories hardwired in during configuration of the ISODE. As such, the source installation procedures do not install a *isotailor* file. If a future need arises, the *isotailor* file may be modified accordingly. Initially it is recommended that you use a tailoring file which contains no commands; copy it from the *quipu/scripts/* area:

```
# cp $(ETCDIR)quipu/scripts/isotailor.empty $(ETCDIR)isotailor
```

2.4 Testing of the ISODE

The easiest way to test the ISODE system is to run the *isode-test* script in the directory *quipu/scripts/* under ISODE's `ETCDIR` directory.

```
% isode-test
```

Prior to termination, the script will indicate the number of errors encountered. If an error occurs, try and determine what the problem is. If you are unable to find the problem, drop a note to the Internet Mailbox `Bug-ISODE@NISC.NYSER.NET`. Your message should contain your two configuration files, the output of *isode-test*, and a description of the hardware/software platform you are using.

2.5 Configuration of DUAs

The first step will be to have the *dish*(1c) program try to connect to one of the Level-0 DSAs in the pilot project:

```
% dish -c "Alpaca"
Welcome to Dish (DIrectory SHell)
Dish ->
```

indicates that you are able to connect. You might now try some simple commands. Figure 2.1 shows a small interaction. Chapter 4 of *Volume Five* describes *dish* in considerable detail.

2.5.1 DUA Failure

There are a few reasons why the DUA might fail. First verify that the ISODE is operating correctly (see Section 2.4 on page 54.) If the ISODE is working, then here is the most common error encountered:

If instead of the `Dish ->` prompt, you see:

```
*** Service error : Unable to contact DSA ***
```

then it is likely that there is some problem with the network, or you have mistyped the name of the DSA. If you see any other diagnostic, contact a *Camayoc* for help.

Otherwise, first check the spelling of the name you are providing. If this is correct, then look at the file *dsaptailor* in the ISODE `ETCDIR` directory for the line which starts with:

```
dsa_address "Alpaca"
```

You will find an Internet address and TCP port number on the rest of the line.

1. Use the *ping*(8c) utility to see if you can reach the IP address.

2. If not, then try contacting one of the other DSAs, e.g., `Fruit Bat`, listed in the *dsaptailor* file.

   Repeat this process until *dish* successfully contacts a DSA.

3. If *ping* can reach the IP address, then use *telnet* to try to connect to the TCP port.

4. If you can connect, immediately close the connection. Something is drastically wrong since *telnet* can reach the TCP address of the DSA, but *dish* can't. Contact a *Camayoc* for assistance.

```
% dish -c "Alpaca"
Dish -> list
1   commonName=Spectacled Bear
2   organizationName=NYSERNet Inc.
3   commonName=Manager
4   localityName=NY
5   organizationName=Navy
6   organizationName=NASA
7   organizationName=The Wollongong Group
8   commonName=Turkey
9   organizationName=Columbia University
10  commonName=Golden-headed Lion Tamarin
11  commonName=iguana
12  organizationName=DCA
13  commonName=Rhea
14  organizationName=CONTEL ASD
15  commonName=Cayman
Dish -> show 3
otherMailbox          - uucp: nyser!mrose
otherMailbox          - internet: mrose@nisc.nyser.net
rfc822Mailbox         - mrose@nisc.nyser.net
userid                - mrose
facsimileTelephoneNumber - +1 415-961-3282
telephoneNumber       - +1 415-961-3380
physicalDeliveryOfficeName - Mountain View
postalCode            - 94043-2112
description           - Principal Implementor of the ISO Development Environment
title                 - Senior Scientist
postalAddress         - NYSERNet Inc.
                        Western Development Office
                        420 Whisman Court
                        Mountain View, CA  94043-2112
streetAddress         - 420 Whisman Court
stateOrProvinceName   - California
localityName          - Mountain View, California
surname               - Rose
commonName            - Marshall T. Rose
commonName            - Marshall Rose
objectClass           - top & person & pilotPerson & quipuObject ...
Dish -> quit
```

Figure 2.1: A Small DISH Interaction

5. If *telnet* reports an error, usually

   ```
   Connection refused
   ```

   then this indicates that the DSA isn't listening at that TCP address. Either it is temporarily unavailable or it has moved to a new address. In either event, notify a *Camayoc* of the situation. Then go back to Step 2 above.

If you exhaust all DSAs listed in the *dsaptailor* file, contact a *Camayoc* and give up.

2.6 An Important Note

The configuration instructions contained herein are for sites in the United States. Depending on the policies of the administrators for other national DMDs, these instructions may, or may not, be useful. If your DSA is joining a DMD other than that of c=US, then you should contact your national administrator to see what turn-key procedures exist for joining your national DMD.

Note that the *dsaconfig*(8) program can be used to configure Level-1 DSAs residing outside of the c=US DMD. At present, the current list of supported countries is:

AU
DE
ES
FI
GB
IS
NO
SE

In most cases, for the countries listed above, you can simply follow the directions below substituting your country code for any occurrence of `c=US`.

In all cases, for DSAs residing outside of the c=US DMD, you should always contact your national administrator before proceeding.

2.7 Configuring your Level-1 DSA

The steps towards configuring a Level-1 DSA are straight-forward. For the text that follows, any example that includes `O_i` means that the name of your organization

Agouti	Alpaca	Anaconda
Giant Anteater	Armadillo	Brazilian Three-banded Armadillo
Giant Armadillo	Axolotl	Fruit Bat
Vampire Bat	Spectacled Bear	Boa Constrictor
Capybara	Andean Cat	Little Spotted Cat
Cayman	Long-tailed Chinchilla	Short-tailed Chinchilla
Greater Pichi Ciego	Lesser Pichi Ciego	Condor
Andean Condor	Argentinian Pampas Deer	Marsh Deer
Bush Dog	Small-eared Dog	Electric Eel
Guanaco	North Andean Huemul	South Andean Huemul
Hornero	Hummingbird	Iguana
Jaguar	Jaguarundi	Llama
Macaw	Amazonian Manatee	Caribbean Manatee
Margay	Buffy-headed Marmoset	Buffy-tufted-ear Marmoset
Goeldi's Marmoset	Tassel-eared Marmoset	White Marmoset
Brown Howler Monkey	Black Spider Monkey	Brown-headed Spider Monkey
Geoffroy's Spider Monkey	Long-haired Spider Monkey	Woolly Spider Monkey
Woolly Monkey	Yellow-tailed Woolly Monkey	Ocelot
Giant Brazilian Otter	La Plata Otter	Marine Otter
Southern River Otter	Pangolin	Chacoan Peccary
Brown Pelican	Galápagos Penguin	Rockhopper Penguin
Piranah	Thin-spined Porcupine	Nothern Pudu
Puma	Rhea	Southern Bearded Saki
White-nosed Saki	Elephant Seal	Galápagos Fur Seal
Juan Fernandez Fur Seal	Brazilian three-toed Sloth	Maned Sloth
Bare-face Tamarin	Cotton-top Tamarin	Emperor Tamarin
Golden Lion Tamarin	Golden-headed Lion Tamarin	Golden-rumped Lion Tamarin
White-footed Tamarin	Central American Tapir	Mountain Tapir
Masked Titi	Giant Tortoise	Toucan
Black-headed Uakari	Red Uakari	Red and White Uakari
Vicuna	Maned Wolf	

Table 2.1: Endangered South American Wildlife

should be substituted. Similarly, any occurrance of `U_j` refers to the name of your organizational unit.

2.7.1 Choosing a DSA Name

First, you must choose a name for your DSA. It is a QUIPU convention that DSAs should be named after endangered South American wildlife, and that the entry for the DSA should contain a description of the animal or plant in question. Table 2.1, taken from *Volume Five* of the ISODE User's Manual, contains a partial list of these species. You might also want to consult IUCN's *Red Book* [IUCN82].

For the purposes of the pilot project, the name of each Level-1 DSA takes the form:

```
c=US@cn=wildlife name
```

So, go to Table 2.1 and pick an unused name. Alternately, find a name which sounds plausible enough to fool a *Camayoc*. Prepend `c=US@cn=`, and this is the Distinguished name of your Level-1 DSA.

To find out what names are already taken, do the following:

```
% dish -c "Alpaca"
Welcome to Dish (DIrectory SHell)
Dish -> search @c=US -filter objectClass=dsa -nosizelimit
```

This will print out the list of names already in use.

Since some of the names in Table 2.1 are long, it is acceptable to use only part of the wildlife name as the common name of your Level-1 DSA. For example, if the wildlife name is something like `Southern Bearded Saki`, then it's probably okay to use `Saki` as the name of the DSA, e.g., `c=US@cn=Saki`.

Now decide upon a "sanitized" name that will be used for the UNIX directory which will contain the database for your Level-1 DSA. Using all lower case letters and substituting hyphens for spaces is a good rule. So, if the name of your DSA is `c=US@cn=Spectacled Bear`, then *spectacled-bear/* or just *bear/* would be fine. For the purpose of discussion, let's call this directory *wildlife/*.

The owner of the directory should be some maintenance account (e.g., `daemon`) or `root`. When the DSA executes, it will set its user- and group-IDs based on the owner and group of this directory.

2.7.2 Editing the DSA configuration file

Go to the directory *quipu/* under ISODE's `ETCDIR` directory:

```
# cd $(ETCDIR)quipu/
```

Now copy the configuration tempate file, e.g.,

```
# cp wildlife.dsa spectacled-bear.dsa
# chmod 0600 spectacled-bear.dsa
```

Now edit the new file to define the initial configuration for your Level-1 DSA. Figure 2.2 shows an example DSA configuration file.

```
###########################################################
# 1. Information about your DSA
###########################################################

dsa             "Spectacled Bear"


###########################################################
# 2. Information about your organization
###########################################################

organization    "NYSERNet Inc."
domain          nyser.net
street          "165 Jordan Road"
town            "Troy"
state           "New York"
zipcode         "12180"
telephone       "+1 518-283-8860"

fax             "+1 518-283-8904"

unit            "Research and Development"

description     "Not-for-profit organization providing network services"


###########################################################
# 3. Information about you
###########################################################

firstname       "Marshall"
lastname        "Rose"

middleinitial   "T"

mailbox         "mrose@nisc.nyser.net"
title           "Senior Scientist"
password        "secret"

extension       "x1234"
```

Figure 2.2: Example DSA configuration file

Note that when defining information such as `streetaddress`, `telephone`, and `description`, you should use the values for your organization, not for your own department. If you like, *after* your DSA is running, you can edit *c=US/o=O_i/EDB* to insert the corresponding information for your own department.

2.7.3 Running the DSA configuration program

Once the file is edited, run the DSA configuration program, *dsaconfig*(8):

```
# $(SBINDIR)dsaconfig wildlife
```

e.g.,

```
# $(SBINDIR)dsaconfig spectacled-bear
```

This will create the database directory and all requisite files. Finally, it will set the owner and group of the files appropriately.

It is necessary to explain a little bit about what *dsaconfig* does. Way back in Section 1.1.2 on page 4, the "EDB format" of the database was described. All that is now missing is an explanation how different nodes in the Directory tree are structured. This is done using the UNIX directory hierarchy.

Starting at the ROOT, at every level in the Directory tree for which your Level-1 DSA holds data, there is a single file called *EDB*. If an entry defined in an *EDB* file has children, the *EDB* file for the children will be found in a subdirectory whose name is the string encoded Relative Distinguished Name (RDN) of the entry.

For example, if a *EDB* file has an entry whose RDN is `o=Computer Science`, and if the entry has children, and if your Level-1 DSA stores this information locally, then the information can be found in the file *o=Computer Science/EDB* relative to the current directory. Note that:

- QUIPU is case sensitive when it looks for the subordinate *EDB* file — the case of the attribute type (e.g., `o`) is taken from the file *oidtable.at* in the ISODE `ETCDIR` directory, whilst the case of the attribute value is taken from the superior *EDB* file; and,

- blanks are *actually* present in the sub-directory name.[2]

[2]An interesting example of the maxim *can implies shall* — because the Berkeley UNIX file system *can* let us have long directory names with embedded spaces, we *shall* make use of that feature.

Thus, for a Level-1 DSA, the initial database hierarchy looks like:

> *EDB*
> *c=US/EDB*
> *c=US/o=O_i/EDB*
> *c=US/o=O_i/ou=U_j/EDB*

So, *dsaconfig* built these files for you.

2.7.4 Testing the stand-alone DSA

The first thing to do is to start the DSA interactively so as to see that it can successfully load its local EDB files. So your Level-1 DSA is started thusly:

```
# cd wildlife/
# $(SBINDIR)ros.quipu -t ./quiputailor &
```

If your DSA is configured properly, it will print out something like:

```
DSA wildlife has started on '0101'H/Internet=130.117.128.3+17003
```

2.7.5 DSA Failure

Instead, if the DSA prints out something like:

```
FATAL ERROR: <<reason>>
```

then something has gone wrong.

There are a few reasons why the DSA might fail. First verify that the ISODE is operating correctly (see Section 2.4 on page 54.) Next, verify that the DUA is operating correctly (see Section 2.5 on page 54.)

There are generally three causes of failure:

- there is an attribute-related schema problem;
- there is a structure-related schema problem; or,
- something else is running at the OSI presentation address of the DSA.

Since you are using files which have been automatically generated, the first two errors are unlikely. However, they might occur due to spelling errors or catastrophic editing on your part. In order to diagnose the problem, you will need to look at the tail of the file *dsap.log* in the *wildlife/* directory.

Attribute Errors

If you misspell the name of an attribute, then the diagnostic will look something like this:

```
line 14: unknown attribute type '<<bad-name>>'
File ...wildlife/c=US/o=0_i/EDB not loaded
```

Alternately, if you specify an illegal attribute for an object (a violation of the object's schema), then the diagnostic will look something like this:

```
attribute '<<illegal-name>>' not allowed in the specified objectclass
Schema error in entry ending line 17...
File ...wildlife/c=US/o=0_i/EDB not loaded
```

Conversely, if you leave out an attribute which is required by an object:

```
'Must' attribute missing '<<mandatory-name>>'
Schema error in entry ending line 16...
File ...wildlife/c=US/o=0_i/EDB not loaded
```

Structure Errors

If a subordinate object belongs to a class which is not permitted by the schema of its immediate parent (as determined by the parent's `treeStructure` attribute), then the diagnostic looks something like this:

```
Specified object class is not in the tree structure (schema) list
Schema error in entry ending line 16...
File ...wildlife/c=US/o=0_i/EDB not loaded
```

Note that this means you have to look in the file *wildlife/c=US/EDB* to find the `treeStructure` attribute of the parent entry and then compare that to the `objectClass` attribute of the entry in error.

Network Errors

If the local EDB files are consistent, but another process is already listening on the IP address and TCP port for the DSA, then your DSA will print:

```
TNetListen: [Congestion at TSAP] start_tcp_server failed:
        Address already in use
Address: '0101'H/Internet=130.117.128.3+17003

FATAL ERROR: Couldn't start the DSA!!
```

Perhaps another copy of this DSA is already running or the TCP port in the entry for the DSA (found in the file *wildlife/c=US/EDB/*) is wrong. It is also possible that an early invocation of the DSA was ungracefully terminated and left the TCP port in a `FIN_WAIT_x` state. Use the *netstat* program to find out. If so, reboot UNIX.

Alternately, you might see:

```
TNetListen: [Congestion at TSAP] start_tcp_server failed:
        Can't assign requested address
Address: '0101'H/Internet=130.117.128.3+17003

FATAL ERROR: Couldn't start the DSA!!
```

This indicates that you mis-identified the IP address of your Level-1 DSA in your *dsaconfig* configuration file.

2.7.6 Editing the DUA tailoring file

The file *dsaptailor* in the ISODE `ETCDIR` directory contains run-time tailoring information for the DUAs provided by the pilot software. You will need to be the super-user to edit this file.

You should now edit the *dsaptailor* file so that your DUAs will contact that DSA by default. This is done by looking for these two lines in the *dsaptailor* file:

```
# the level-1 DSA
#dsa_address "wildlife name"      '0101'H/Internet=aaa.bbb.ccc.ddd+port
```

and then removing the '`#`'-sign at the beginning of the second line.

Next, you should also have your DUAs start, by default, in your portion of the Directory tree. This is done by looking for these two lines in the *dsaptailor* file:

```
local_DIT    "c=US"
#local_DIT   "c=US@O_i"
```

and then adding a '#'-sign at the beginning of the first line, and removing the '#'-sign at the beginning of the second line.

Section 12.2 of *Volume Five* discusses the options available for run-time tailoring. Typically, you will have no need of further editing this file.

Now run the *dish*(1c) program again, telling it to connect to your Level-1 DSA.

```
% dish -c "wildlife name"
Welcome to Dish (DIrectory SHell)
Dish ->
```

indicates that the DUA connected to your Level-1 DSA. Otherwise consult Section 2.5.1 on page 55 and try to debug the problem.

Now look around the Directory tree using *dish*:

```
Dish -> list
```

A good test to run is to try and bind to your own entry, but to do so by dereferencing the alias for the Manager of your DMD:

```
Dish -> bind -user "@c=US@o=O_i@cn=Manager"
Enter password for "@c=US@o=O_i@cn=Manager":
Dish ->
```

Indicates that you are now bound to the directory as that DN. Instead, if you see:

```
Dish -> bind -user "@c=US@o=O_i@cn=Manager"
Enter password for "@c=US@o=O_i@cn=Manager":
Security Error - check name and password
```

then either you may have entered the DN or password wrong. Try again. If not, or if you encounter some other problem, contact a *Camayoc* for assistance.

2.7.7 Joining the Pilot DMD

Once your DUA is configured to use your Level-1 DSA and your Level-1 DSA appears to be functioning properly in stand-alone mode, it is time for it to join the Pilot DMD.

First, backup your EDB files:

```
# cp EDB EDB.stand-alone
# cp c=US/EDB c=US/EDB.stand-alone
```

Contact the c=US Manager

Next, contact the Manager of the `c=US` portion of the DMD. Use the *dish* program to get the necessary contact information, e.g.,

```
% dish -c "Alpaca"
Welcome to Dish (DIrectory SHell)
Dish -> show @c=US@cn=Manager -fred
```

If this fails, contact a *Camayoc* for assistance. Otherwise, e-mail the file *c=US/EDB* along with your phone number to the Manager and await a phone call or message in return. Leave your DSA running until you hear from the Manager.

The Manager will enter these into the MASTER `c=US` EDB and then try to connect to your DSA to test it out. The Manager will also give you a "time slot" when your *nightly.sh* script should run. (This is a feeble attempt to prevent multiple systems from mailing statistical information simultaneously each night.)

If everything looks fine, the Manager will direct you to proceed.

Connecting In and Checking Out

Restart your DSA and use *dish* to bind as the manager. Next, direct the DSA to update its slave EDBs:

```
% dish -user "@c=US@o=O_i@cn=Manager"
Welcome to Dish (DIrectory SHell)
Enter password for "@c=US@o=O_i@cn=Manager":
Dish -> dsacontrol -slave
Scheduled
Dish -> quit
```

The `Scheduled` message means that the EDB updates have been scheduled for execution. The DSA will actually perform these in the background, so it may be a few minutes (or longer, depending on DSA/network availability) before the EDBs are updated.

To verify that your Level-1 DSA is updating its ROOT and `c=US` EDB files, look for the files *EDB.bak* and *c=US/EDB.bak* in the *quipu/wildlife/* directory. If present, these indicate that your Level-1 DSA was able to successfully update the EDBs. If not, look at the *dsap.log* log file and contact a *Camayoc* for assistance.

Finally, check that the global Directory tree knows about your DSA (the `c=US` Manager verified this, but you should check anyway). Use the *dish* but connect to a Level-0 DSA:

```
% dish -c "Alpaca"
Welcome to Dish (DIrectory SHell)
Dish -> moveto "@c=US@o=O_i"
Dish -> list
   ...
Dish -> list "ou=U_j"
   ...
```

If this succeeds, the Level-0 DSAs know how to talk to your Level-1 DSA.

The other Level-1 DSAs won't know about your organization and Level-1 DSA for another day or so. (The Manager of the `c=US` may cause a few Level-1 DSAs to immediately reload the `c=US` in order to facilitate things.)

More System Administration

Once everything checks out, its time to restart the DSA in the background. First, edit the *quiputailor* file for your DSA by changing the line

```
update  off
```

to

```
update  on
```

This tells your DSA to automatically update its SLAVE EDB files on a regular basis.

Next, use *dish* to abort the DSA:

```
% dish -user "@c=US@o=O_i@cn=Manager"
Welcome to Dish (DIrectory SHell)
Enter password for "@c=US@o=O_i@cn=Manager":
Dish -> dsacontrol -abort
*** Problem with DSA ***
Dish -> quit
```

Now run the *startup.sh* script:

```
% $(ETCDIR)quipu/wildlife/startup.sh
```

Take a look at the log files it creates and once you're satisfied that it is operational, use *dish* one last time before considering things up and running.

Finally, it's time for the last bit of system administration:

1. Add an entry to the file */etc/rc.local*:

   ```
   if [ -d $(ETCDIR)quipu/wildlife ]; then
      $(ETCDIR)quipu/wildlife/startup.sh & \
                                  (echo -n ' wildlife') > /dev/console
   fi
   ```

   in the section where the network servers are started. If your *rc.local* file starts *tsapd*(8c), then place this entry after the one which starts *tsapd*.

   The next time your system reboots, check to make sure that your DSA started. On *some* systems running Sun's "Yellow Pages", it has been observed that YP prevents the *ros.quipu* from starting. No cause or cure is known for this. The problem just seems to "go away" after a while.

2. Based on the time that the `c=US` manager gave you, modify the *crontab* file according; e.g.,

   ```
   0 4 * * * $(ETCDIR)quipu/wildlife/nightly.sh
   ```

   If the directory database for the Level-1 DSA is owned by a user-ID other than `root` (e.g., `daemon`), then instead the line should look something like this:

   ```
   0 4 * * * su daemon -c "/bin/sh $(ETCDIR)quipu/wildlife/nightly.sh"
   ```

   or perhaps this

   ```
   0 4 * * * su daemon -c "sh $(ETCDIR)quipu/wildlife/nightly.sh"
   ```

   Be sure that the ownership of the *nightly.sh* file matches that of the userid that will be running under cron. Verify that the user's shell, paths, and so on, are all correct as well.

Congratulations! Your Level-1 DSA has now joined the pilot DMD. Further, the Manager of `c=US` has elevated you to the role of junior *Camayoc* by adding your electronic mail address to the `wpp-camayocs` mailing list. Each *Camayoc* is expected to share experiences with others so as to expand and advance understanding of running a white pages service using the OSI Directory.

2.8 Configuring the White Pages

The very last step you need do is to configure the white pages service which resides on top of the OSI Directory.

You should now determine if a substantive portion of your user community does not have easy access to a machine running the pilot project software. If this is the case, then you will probably want to make the white pages service available via the network (either via TELNET or by emulating WHOIS) or via electronic mail.

2.8.1 White Pages via TELNET

Choose a machine in your local environment which is running the pilot project software. This machine will offer the white pages service via TELNET.

On this machine, add a line to the */etc/passwd* file for a user called `fred`

```
fred::30:30:Anonyous White Pages User:/:$(SBINDIR)fredsh
```

where group-id `30` is the `guest`.

Fred-only hosts

After bringing up the software for both the Directory and the White Pages, you might wish to copy the executables to other machines, just to make *fred* available. To do this, you need the following files:

- In the ISODE `BINDIR` directory:

  dish
  editentry
  fred

- In the ISODE `SBINDIR` directory:

  fredrc
  dsaptailor
  *oidtable.**
  isobjects
  isologs
  isomacros
  isotailor

Note that the *isotailor* file need not be present, depending on the dynamic changes to your system configuration.

- In the ISODE `SBINDIR` directory:

    fredsh
    in.whitepages

  Note that you need *in.whitepages* only if you plan on using this host to offer the white pages via WHOIS emulation.

2.8.2 White Pages via WHOIS

Choose a machine in your local environment which is running the pilot project software. This machine will offer the white pages service via a network port offering an emulation of the WHOIS service.

On this machine, edit the file */etc/services* so that it has an entry like this:

```
whitepages 17005/tcp
```

Next, edit the file */etc/servers* so that it has an entry like this:

```
whitepages tcp  $(SBINDIR)in.whitepages
```

Because most user interfaces to WHOIS, e.g., *whois*(1c), do not allow the user to specify a special port, you should probably also add this line as well:

```
whois   tcp     $(SBINDIR)in.whitepages
```

If you already have a line for `whois` in the *servers* file, then you are already running a WHOIS service, and you should **not** add a second `whois` line. This machine is not a good choice for running the white pages via WHOIS emulation.

Note that on newer systems derived from Berkeley UNIX, */etc/servers* is called */etc/inetd.conf*.

The whitepages Command

On those systems which are to access the white pages via the network and not locally (i.e., those systems which are not running the pilot project software), you should determine how a user invokes the WHOIS service via the network. For UNIX systems, you should provide a shell script like this:

```
: run this script through /bin/sh

exec /usr/ucb/whois -h wp.nyser.net "$*"
```

where the name of a host running the pilot project software is substituted for `whitepages`, e.g., `wp.nyser.net`. This host must have the files */etc/services* and */etc/servers* edited as described above.

2.8.3 White Pages via mail

In addition, you might want to make the whitepages available via electronic mail. To do this, choose a machine in your local environment which is running the pilot project software.

On this machine, edit the file */usr/lib/aliases* so that it has an entry like this:

```
whitepages: "|/usr/local/bin/fred -m"
```

After editing the *aliases* file, you will need to run the *newaliases* command as the super-user:

```
# newaliases
```

Also make sure that the *sendmail.cf* file lists *sh* and not *csh* as the `prog` mailer.

Once these changes are in place, Whenever someone sends to the address `whitepages` on this machine, the subject line or body of the message will be interrogated for WHOIS-like commands and the appropriate reply generated.

Note that on systems not running *sendmail*(8c), a similar procedure is followed. However, it's up to you to figure out how to define aliases that run commands on receipt of mail.

You should tell the local PostMaster about the white pages service. In this fashion, when the PostMaster get queries from other sites, they can the sender of the message (and the Postmaster at the sender's site) about the service.

2.8.4 An Undocumented Feature

In some environments, in which the IP-address of a host is reasonably trustworthy, it may be desirable to have an automatic means of mapping an IP-address into a Distinguished Name. *fred* provides such a mechanism under the following conditions:

1. Create a file called *fredmap* in the ISODE `ETCDIR` directory.

   The syntax of this file is a series of entries, one to a line. Each line contains:

   - a IP-address mask (in the familiar "dot" notation, e.g., `255.255.255.0`);
   - a IP-network address (in the familiar "dot" notation, e.g., `192.52.180`);
   - a Distinguished Name (e.g., `"c=US@o=DMD@cn=anon"`); and,
   - the password to use when binding to that DN (e.g, `"secret"`).

   Blanks and/or tab characters are used to separate items. However, double-quotes may be used to prevent separation for items containing embedded whitspace. The sharp character (`'#'`) at the beginning of a line indicates a commentary line.

   This file should be mode `0640`, owned by user `root` and some previously unused group, e.g., `wp`.

2. After installing *fred* with the

   ```
   # (cd others/quipu; ./make inst-pilot)
   ```

   command (this was done way back in Section 2.1.3 on page 52), make the group of the binary the same as the *fredmap* file, and change the mode to `02755`.

3. When a local user invokes *fred fred* looks-up the local IP-address, reads the *fredmap* file looking for the first entry satisfying:

   $$\text{(local IP-address)} \wedge \text{(IP-address mask)} \equiv \text{(IP-network address)}$$

   Upon finding such an entry, *fred* will bind to the Directory using the DN and password for that entry.

4. When *fred* is accessed via the network (e.g., with the `in.whitepages` mechanism described in Section 2.8.2 on page 70), *fred* will automatically read the *fredmap* file, looking for the first entry satisfying:

   $$\text{(remote IP-address)} \wedge \text{(IP-address mask)} \equiv \text{(IP-network address)}$$

   Upon finding such an entry, *fred* will bind to the Directory using the DN and password for that entry.

Of course, the ordering of entries in the *fredmap* file is important! It is suggested that the file begin with host-specific entries (those with an IP-address mask of `255.255.255.255`), and then entries follow in decreasing order by the number of bits on in the IP-address mask.

Although use of a "default" entry, one which will match any IP-address, is strongly discouraged, it is possible to define such an entry. This should be the very last entry in the *fredmap* file. The format of this entry is simply:

```
0.0.0.0 0.0.0.0    "some DN"    "some password"
```

It should be noted that this scheme provides a convenient mechanism for allowing "local" users to automatically be authorized as specific entries in the local DMD. By doing so, this eases the administrative burden of assigning passwords to users in an environment which contains read-only but, nonetheless, sensistive information. The information is protected via access control, e.g.,

```
acl= prefix # c=US@o=DMD # read # attributes # <attribute-list>
acl= self # write # attributes # <attribute-list>
acl= others # none # attributes # <attribute-list>
```

Note that for a writeable environment, whilst the access control scheme is still used, this simplification of locally anonymous DNs usually can not be made.

Note that under the current QUIPU access scheme, the access control list above will prevent any user, regardless of the DN they are bound as (including local users), from searching on the values of the attributes named in `<attribute-list>`. As such, a less restrictive set of rules might be:

```
acl= prefix # c=US@o=DMD # read # attributes # <attribute-list>
acl= self # write # attributes # <attribute-list>
acl= others # compare # attributes # <attribute-list>
```

which will enable searching for all users. However, it will also allow any user to compare specific values against the attributes named in `<attribute-list>`.

Of course, this simplification is severely limited in that it works only for IP-addresses, and further assumes that such addresses are trustworthy. The trustworthiness depends entire upon the network environment in which *fred* runs. For example, since IP-addresses can be readily forged, one might rely on routers on the edge of the local environment to simply drop IP datagrams with local source IP-addresses that originate from the outside world. In this case, entries in the *fredmap* file containing local IP-addresses can be thought to be "reasonably" secure against outside attack. Of course, nothing prevents a local host from forging a local IP-address. If the trust policy is simply to distinguish between local and non-local users, then the *fredmap* mechanism is adequite.

Given all of these concerns, this "undocumented feature" is a useful, albeit non-OSI, local mechanism.

2.9 Other Services

There are some additional programs that you might wish to install for your user community.

2.9.1 Faces

When *fred* and *dish* display the entry for someone, they check to see if there is a photograph associated with the user. This is stored in facsimile format in the `photo` attribute for the entry. If a photograph is present, then the *dsaptailor* file is consulted for directives indicating how the picture should be displayed. The most common entry is:

```
photo    xterm Xphoto
```

which says that if the user's terminal type is `xterm` then the program *Xphoto* should be run.

Although there are many different programs that can be used to display a photograph, only the *XPhoto* is supported by the sponsors of the pilot.

XPhoto

To generate the *XPhoto* program, run the following command:

```
% (cd others/quipu/photo; ./make all XPhoto)
```

To install the program, become the super-user and do:

```
# (cd others/quipu/photo; ./make inst-all)
```

Next, add this line at the end of the *dsaptailor* file:

```
photo    xterm Xphoto
```

xwho and xface

There are two other programs which consult the Directory for people's faces. These are *xwho*, who for X windows; and, *xface*, face agent for X windows.

To generate these programs, first generate *Xphoto* as described above. next:

```
% (cd others/image; ./make all)
```

To install the program, become the super-user and do:

```
# (cd others/image; ./make inst-all)
```

The *xwho* program is free-standing, but requires that the local system run the *rwhod*(8c) daemon.

The *xface* program runs, for each user, in the background. When a mail reading program displays a message, it sends a wake-up call to *xface* which must then try to find the picture associated with the sender of the message.

At present, only one mail reading system has been modified to communicate with *xface*, the MH system. The file *others/image/mhlsbr.c* is a replacement for the *uip/mhlsbr.c* in the MH distribution. Once MH has been modified, the manual entry for *xface*(1c) describes how the user tells MH to display faces.

Finally, the file *others/image/READ-ME* describes how images may be collected and placed in the Directory.

2.9.2 Mail Composition

In addition to using the Directory when messages are being displayed, the White Pages may also be consulted when a message is being composed.

At present, only one mail composition system has been modified to use the White Pages, the MH system.

The file *others/quipu/uips/fred/MH-patches* contains a patch set to MH to use this facility along with instructions for apply the patch set.

Once installed, users can specify a name by bracketing a White Pages query between "`<<`" and "`>>`" using the `whois` syntax of the *fred* command, e.g.,

```
To: << rose -org psi >>
```

At the `What now?` prompt, the user can say `whom` to have the names expanded into addresses. Alternately, the `send` option can be used as well. For each query appearing between "`<<`" and "`>>`", *fred* will be asked to perform a White Pages resolution. All matches are printed and the user is asked to select one. If one is not selected, the user remains with *fred*, to make more queries, until eventually one is selected (or the user exits *fred* to abort the expansion process).

Note that expansion can occur only if `whom` or `send` is invoked interactively. If the `push` option is used instead, then the expansion will fail because *fred* will be unable to query the user to select/confirm the right entry to use for the substitution.

2.10 Tell The Users!

OK, everything is installed and running, so it's time for you to tell your users about the system! Print out copies of *The Handbook* and the accompanying *White Pages Quick Reference Sheet*, and start distributing them. Also, encourage your users to subscribe to the

```
wpp-users@nisc.nyser.net
```

discussion group. This is done by sending a message to the

```
wpp-users@nisc.nyser.net
```

mailbox and asking to be added.

Chapter 3

Maintenance

Now that your Level-1 DSA has successfully joined the pilot project DMD, you must maintain your portion of the Directory tree.

3.1 A file you should know about

The file *quiputailor* file in the *quipu/wildlife/* directory contains runtime configuration for your Level-1 DSA. It was automatically created earlier.

Section 13.3 of *Volume Five* discusses the options available for run-time tailoring. Typically, you will not need to edit this file.

3.2 Nightly Maintenance

One of the last tasks performed when you Level-1 DSA joined the pilot project DMD was to direct your system to run a shell script, *nightly.sh*. This script performs two tasks:

- it mails your logs to the pilot project sponsors (this is only a temporary measure to aid our understanding of how the software is behaving); and,
- it cycles your logs.

You might wish to modify this script so that you are also informed of the activities of your Level-1 DSA. Actually, the logs record only crude information. During the course of the pilot project, the pilot software might be upgraded to provide more meaningful information. This will be examined as experience is gained.

3.2.1 Logs

The pilot software does a lot of logging. There are two logs which are generated, the first, *dsap.log*, contains information on general DSA activity, whilst the second, *stats.log*, contains statistical information on the DSA.

3.2.2 Limiting the size of Logs

If your Level-1 DSA is particularly busy, it may generate large logs. As such, you might find it desirable to limit the maximum size that a log may grow to. Since two logs are generated, one for DSA activity and the other for statistics, you will need to make two edits.

Look at the *quiputailor* file in the *quipu/wildlife/* directory. There should be four lines similar to this:

```
# minimal logging
dsaplog level=exceptions dflags=tty file=dsap.log

# full statistics
stats level=all dflags=tty file=stats.log
```

Add the string `size=100` at the end of the two lines, e.g.,

```
# minimal logging
dsaplog level=exceptions dflags=tty file=dsap.log size=100

# full statistics
stats level=all dflags=tty file=stats.log size=100
```

This will limit the size of each log to 100 Kbytes. If you wish other limits, change the value `100` accordingly.

3.2.3 Reading Logs

This is currently a black art. When the sponsors of the pilot project have mastered this, *The Guide* will be updated accordingly. In the meantime, if you figure something out, share it with the `wpp-camayocs` list. Have fun.

3.3 Adding Entries

Now comes the fun part: entering data into the Directory. In general, there are two kinds of activities: small, incremental changes are best made using *dish*(1c). However, for the wholesale entry of massive amounts of data, the easiest way is to run your favorite text editor and create EDB files manually. (In a future release of the pilot project software, more management tools will be available to automate this process somewhat.)

Note that if you edit the EDB files directly, you **must** tell your DSA to re-read these files after you are done editing. This is accomplished by either killing and restarting the DSA or using the '`-refresh`' option to the *dish* command `dsacontrol`.

To aid the process, a number of templates for the objects you might add are found in the directory *quipu/templates/*:

alias	`alias` object
dsa	Level-2 DSA
person	`pilotPerson` object
role	`organizationalRole` object
unit	`organizationalUnit` object

Each of these files contains editing instructions.

In the *quipu/* source directory, there is a program called *testedb*, which can be used to check EDB files for correctness:

```
% cd quipu/
% ./make testedb
% ./testedb < EDB
```

The *testedb* program will find the vast majority of errors in an EDB file. It can not however, find errors due to schema violation (i.e., not conforming to the `treeStructure` attribute of the EDB's parent). If your DSA fails to boot properly, the log will indicate:

```
DSA Halted
```

In this case, invoke the DSA interactively to determine the cause of the problem, e.g.,

```
% $(SBINDIR)ros.quipu -t ./quiputailor
Schema error in entry ending line 16...
```

```
*** Attribute error ***
<<DN of entry in error>>
Attribute type objectClass - Constrain violation
File ...wildlife/c=US/o=O_i/EDB not loaded
FATAL ERROR: DSA Halted
```

3.3.1 Using Dish

If you use the first approach, then your Level-1 DSA will automatically update the database directory. Thus, all you need be able to do is run one of the user interfaces.

First, identify yourself to *dish* as the manager of the DSA holding the entries you want to modify:

```
% dish -c "wildlife name" -user "c=US@o=O_i@cn=Manager"
Enter password for "c=US@o=O_i@cn=Manager": secret
Dish ->
```

You can now use the `add` and `modify` commands as appropriate. For the `add` command, it is suggested you start with one of the supplied templates, e.g.,

```
Dish -> add ou=Corporate -template $(ETCDIR)quipu/templates/unit
```

will create a new organizational unit under the current node.

The only tricky part is when objects of class `organizationalUnit` or `dsa` are added.

3.4 Adding organizationalUnits

When an organizational unit is added, you must also modify the entries for the DSAs holding MASTER or SLAVE copies of the subordinates of the organizational unit.

NOTE: At the present time, the pilot sponsors strongly recommend against adding Level-2 DSAs.
A Level-2 DSA should be added only when a Level-1 DSA is too large to run on an available system. In this case, a Level-2 DSA can be used to reduce the memory requirements on the system running the Level-1 DSA.

If the organizational unit is to be mastered by your Level-1 DSA, then the procedure is straight-forward: First, create a directory in your *wildlife/c=US/o=O_i/* directory with the name of the organizational unit, e.g.,

```
wildlife/c=US/o=NYSERNet Inc./ou=Corporate
```

Second, create an *EDB* file in this UNIX directory containing information on the entries in that organizational unit. Third, create an entry for that organizational unit in the Directory, e.g., by running *dish*, moving to your organization's entry, typing:

```
Dish -> add ou=Corporate -template $(ETCDIR)quipu/templates/unit
```

and then following the editing instructions in the file.

3.5 Adding a Level-2 DSA

There are three aspects to adding a Level-2 DSA: first, the entry for your organization and Level-1 DSA must be modified, and an entry for your Level-2 DSA must be created; second, the Level-2 DSA must be configured; and, third, parts of the Directory tree mastered by the Level-1 DSA may be moved over to be mastered by the Level-2 DSA.

First, you must choose a name for your new Level-2 DSA. Since there will probably be more Level-2 DSAs then endangered species of South American Wildlife, you do not have to use a wildlife name for a Level-2 DSA. Choose something associated with your organization or state.

For the purposes of the pilot project, the name of each Level-2 DSA taks the form:

```
c=US@o=O_i@cn=wildlife name
```

As usual, you will have to pick a "sanitized" name that will be used for the UNIX directory which will contain the database for your Level-2 DSA.

To remain consistent with the discussion on configuring a Level-1 DSA, we'll call the name of the DSA `wildlife name` and the directory will be called *wildlife/*.

3.5.1 Modifying the Level-1 DSA

To the entry for your Level-1 DSA, you will need to add these lines:

```
eDBinfo= # # c=US@o=O_i@cn=wildlife name
eDBinfo= c=US # # c=US@o=O_i@cn=wildlife name
eDBinfo= c=US@o=O_i # # c=US@o=O_i@cn=wildlife name
```

This says that your Level-1 DSA provide copies of the ROOT, `c=US` and your organization's EDBs to your Level-2 DSA.

This addition is done using the `modify` command to *dish*:

```
Dish -> modify "@c=US@cn=wildlife name"
```

Note that even though your Level-2 DSA will contain a slave copy of the EDB for your organization, you do not add a `slaveDSA` attribute to your organization's entry to reflect this. In order to contact your Level-2 DSA, it is necessary to find its `presentationAddress` attribute by asking the directory. Since the entry for your Level-2 DSA is kept beneath your organization's entry, anyone asking for information about your Level-2 DSA would already have information on your organization!

Finally, you need to add an entry for your Level-2 DSA to the EDB for your organization:

```
Dish -> add "cn=wildlife name" -template $(ETCDIR)quipu/templates/dsa
```

The template file contains these editing instructions:

1. Change each occurrence of `O_i` to your organization's name; e.g.,

   ```
   NYSERNet Inc.
   ```

2. Change each occurrence of `wildlife name` to the common name of your DSA; e.g.,

   ```
   beeblebrox
   ```

3. For each organizational unit, `U_j`, this DSA will master, add a line:

   ```
   eDBinfo = c=US@o=O_i@ou=U_j # # c=US@cn=level-1 DSA
   ```

   where `c=US@cn=level-1 DSA` is the name of your Level-1 DSA.

4. Change the value of the `presentationAddress` attribute to contain the IP address of the host running the Level-2 DSA, and select an unused TCP port at this IP address (port 17010 is suggested for Level-2 DSAs); e.g.,

   ```
   '0101'H/Internet=130.117.118.3+17010
   ```

   If other Level-2 DSAs are to be run on this host, it is suggested that ascending port numbers, starting at 17011, be assigned. However, running multiple Level-2 DSAs on a single host is not recommended.

5. Change the value of the `description` attribute for your Level-2 DSA accordingly. The first value should be the wildlife description. Note that you should fully explain the meaning of the Level-2 DSA's common name. Another description value should be added for each organizational unit mastered by this DSA, e.g.,

   ```
   Master DSA for U_j under O_i
   ```

3.5.2 Configuring a Level-2 DSA

Configuring a Level-2 DSA is currently a pain as *dsaconfig* is not currently used for this task. Before following the steps below, drop a note to the `wpp-camayocs` list and ask if a new version of *dsaconfig* is available!

Start by copying the database directory for your Level-2 DSA:

```
# cd quipu/
# cp -r level-1-dsa wildlife
# chmod 700 wildlife
# find wildlife -exec chown daemon {} \;
# find wildlife -exec chgrp daemon {} \;
# su daemon
# cd wildlife/
```

Editing the DSA tailoring file

Now edit the *quiputailor* file in the *quipu/wildlife/* directory. There are three things to do:

1. Change the `mydsaname` variable to reflect the Distinguished Name of the DSA. For example:

```
    mydsaname        "c=US@o=O_icn=wildlife name"
```

becomes

```
    mydsaname        "c=US@o=NYSERNet Inc.@cn=beeblebrox"
```

2. Change the `logdir` variable to reflect the UNIX directory where QUIPU log files are to reside. For example:

   ```
       logdir  $(ETCDIR)quipu/wildlife/
   ```

   becomes

   ```
       logdir  $(ETCDIR)quipu/beeblebrox/
   ```

   (Note the trailing slash.)

3. Change the `treedir` variable to reflect the UNIX directory where the DSA's database resides. For example:

   ```
       logdir  $(ETCDIR)quipu/wildlife
   ```

   becomes

   ```
       logdir  $(ETCDIR)quipu/beeblebrox
   ```

   (Note the lack of a trailing slash.)

Section 13.3 of *Volume Five* discusses the options available for run-time tailoring. You will have no need of editing this file.

Editing the DSA startup file

Now edit the *startup.sh* file in the *quipu/wildlife/* directory. There are two things to do:

1. Change the `W` variable to reflect the wildlife name of the DSA. For example:

   ```
       W=wildlife
   ```

   becomes

   ```
       W="Beeblebrox"
   ```

2. Change the D variable to reflect the UNIX directory where the DSA's database resides. For example:

   ```
   D=$(ETCDIR)quipu/wildlife
   ```

   becomes

   ```
   D=$(ETCDIR)quipu/beeblebrox
   ```

Building an Initial Database

The directory database you created with the *cp* command earlier has done virtually all the work for you. Now all you need do is edit each EDB file to initially mark each as a SLAVE copy. A simple way of doing this is:

```
# find . -name EDB -a -exec vi {} \;
```

which will run *vi* on each EDB file. If the first line of this file says `MASTER`, change it to `SLAVE`. Otherwise the first line should say `SLAVE` (if the first line of the EDB file says `CACHE`, then contact a *Camayoc* for assistance.)

Testing the Level-2 DSA

At this point, your Level-2 DSA should be configured and you should start and test it:

```
# $(SBINDIR)ros.quipu -t ./quiputailor &
```

If your DSA is configured properly, it will print out something like:

```
-- '0101'H/Internet=130.117.128.3+17010 --
DSA Started
```

If your Level-2 DSA does not boot for some reason, consult Section 2.7.5 on page 62.

You should now try connecting to the Level-2 DSA.

```
% dish -c "wildlife name"
Welcome to Dish (DIrectory SHell)
Dish ->
```

indicates that the DUA connected to your Level-1 DSA. Otherwise consult Section 2.5.1 on page 55 and try to debug the problem.

Editing the DUA tailoring file

Once your Level-2 DSA is operational, you should edit the file *dsaptailor* in the ISODE `ETCDIR` directory so that your DUAs will know about this DSA. This is done by adding this line

```
dsa_address "wildlife name"      '0101'H/Internet=aaa.bbb.ccc.ddd+port
```

after the `dsa_address` line for your Level-1 DSA and then making these edits:

1. Substitute the common name of your DSA for `wildlife name`; e.g.,

   ```
   Beeblebrox
   ```

2. Change the IP address and TCP port number to correspond to the OSI presentation address you defined earlier in the entry for the DSA; e.g.,

   ```
   '0101'H/Internet=130.117.118.3+17010
   ```

Now run the *dish*(1c) program again, telling it to connect to your Level-1 DSA.

```
% dish -c "wildlife name"
Welcome to Dish (DIrectory SHell)
Dish ->
```

indicates that the DUA connected to your Level-1 DSA. Otherwise consult Section 2.5.1 on page 55 and try to debug the problem.

Now look around the Directory tree using *dish*. Descend to `c=US@o=O_i` to check on your own entries. A good test to run is to try and bind to your own entry, but to do so by dereferencing the alias for the Manager of your DMD:

```
Dish -> bind -user "c=US@o=O_i@cn=Manager"
Enter password for "c=US@o=O_i@cn=Manager":
Dish ->
```

Indicates that you are now bound to the directory as that DN. Instead, if you see:

```
Dish -> bind -user "c=US@o=O_i@cn=Manager"
Enter password for "c=US@o=O_i@cn=Manager":
Security Error - check name and password
```

then either you may have entered the DN or password wrong. Try again. If not, or if you encounter some other problem, contact a *Camayoc* for assistance.

3.5.3 Moving portions of the Directory Tree

For each organizational unit, `ou=U_j`, the Level-2 DSA will master, you now need to do two things:

- tell your Level-1 DSA that it no longer masters the EDB; and,
- tell your Level-2 DSA that it now masters the EDB.

The first step is done as follows. Begin by using *dish* to bind to the Level-1 DSA:

```
% dish -c "Level-1 DSA" -user "c=US@o=O_i@cn=Manager"
```

Next:

1. Modify the entry for the organizational unit which is held by your Level-1 DSA. Change the `masterDSA` attribute of the entry corresponding to that unit from:

   ```
   masterDSA= c=US@cn=Level-1 DSA
   ```

   to:

   ```
   masterDSA= c=US@o=O_i@cn=wildlife name
   ```

   You will also need to add a `slaveDSA` attribute to this entry:

   ```
   slaveDSA= c=US@cn=Level-1 DSA
   ```

   These additions are done using *dish*:

   ```
   Dish -> modify "@c=US@o=O_i@ou=U_j"
   ```

2. Modify the entry for your Level-1 DSA. Change the line that says:

   ```
   eDBinfo= c=US@o=O_i@ou=U_j # #
   ```

   to:

   ```
   eDBinfo= c=US@o=O_i@ou=U_j # c=US@o=O_i@cn=wildlife name #
   ```

   This says that your Level-1 DSA will receive copies of the EDB for each organizational unit held by your Level-2 DSA.

   This change is done using the `modify` command to *dish*:

```
Dish -> modify "@c=US@cn=wildlife name"
```

3. Lock the Level-1 DSA's copy of the EDB file:

   ```
   Dish -> dsacontrol -lock "c=US@o=O_i@ou=U_j"
   ```

4. Edit the EDB file kept in the **Level-1** DSA's directory database by changing the first line from `MASTER` to `SLAVE`. (If the first line of this file does not say `MASTER`, then you are editing the **wrong** directory database.)

5. Unlock the Level-1 DSA's copy of the EDB file:

   ```
   Dish -> dsacontrol -refresh "c=US@o=O_i@ou=U_j"
   Dish -> dsacontrol -unlock "c=US@o=O_i@ou=U_j"
   ```

The second step is done as follows. Begin by using *dish* to bind to the Level-2 DSA:

```
% dish -c "Level-2 DSA" -user "c=US@o=O_i@cn=Manager"
```

Next:

1. Modify the entry for the for organizational unit which is to be mastered by your Level-2 DSA. Change the `masterDSA` attribute of the entry corresponding to that unit from:

   ```
   masterDSA= c=US@cn=Level-1 DSA
   ```

   to:

   ```
   masterDSA= c=US@o=O_i@cn=wildlife name
   ```

   You will also need to add a `slaveDSA` attribute to this entry:

   ```
   slaveDSA= c=US@cn=Level-1 DSA
   ```

   These additions are done using *dish*:

   ```
   Dish -> modify "@c=US@o=O_i@ou=U_j"
   ```

2. Modify the entry for your Level-2 DSA. Add this line:

   ```
   eDBinfo= c=US@o=O_i@ou=U_j # # c=US@cn=Level-1 DSA
   ```

This says that your Level-1 DSA will receive copies of the EDB for each organizational unit held by your Level-2 DSA.

This change is done using the `modify` command to *dish*:

```
Dish -> modify "@c=US@o=O_i@cn=wildlife name"
```

3. Lock the Level-2 DSA's copy of the EDB file:

```
Dish -> dsacontrol -lock "@c=US@o=O_i@ou=U_j"
```

4. Edit the EDB file kept in the **Level-2** DSA's directory database by changing the first line from `SLAVE` to `MASTER`. (If the first line of this file does not say `SLAVE`, then you are editing the **wrong** directory database.)

5. Unlock the Level-2 DSA's copy of the EDB file:

```
Dish -> dsacontrol -unlock "@c=US@o=O_i@ou=U_j"
```

You should now reboot your Level-2 DSA and then try to connect to it using *dish*. Once this is successfully, you should reboot your Level-1 DSA and also use *dish* to connect to it.

When you restart the Level-2 DSA, it will try to update its ROOT, `c=US`, and `o=O_i` EDB files from your Level-1 DSA. The first two should be the same as what your Level-2 is running, so no update will take place. The third will be different however, so you should see a file *c=US/o=O_i/EDB.bak* created.

When you restart the Level-1 DSA, in addition to trying to reload its ROOT and `c=US` EDB files from the Level-0 DSAs, it will try to reload the EDB file for each organizational unit mastered by the Level-2 DSA. Since these will be the same, initially no update will take place.

Editing the DUA tailoring file

You now edit the *dsaptailor* file one more time. Move the `dsa_address` line for your Level-2 DSA above the line for your Level-1 DSA. This will tell your DUAs to contact the Level-2 DSA by default, rather than the Level-1 DSA.

System Administration

Once everything checks out, its time to restart the DSA in the background. Use *dish* to abort the DSA and then run the *startup.sh* script:

```
% $(ETCDIR)quipu/wildlife/startup.sh
```

Take a look at the log files it creates and once you're satisfied that it is operational, use *dish* one last time before considering things up and running.

Finally, it's time for the last bit of system administration:

1. Add an entry to the file */etc/rc.local*:

   ```
   if [ -d $(ETCDIR)quipu/wildlife ]; then
      $(ETCDIR)quipu/wildlife/startup.sh & \
                                      (echo -n ' wildlife') > /dev/console
   fi
   ```

   in the section where the network servers are started. If your *rc.local* file starts *tsapd*(8c), then place this entry after the one which starts *tsapd.*

2. Edit the file *quipu/wildlife/nightly.sh*, by looking for these three lines

   ```
   W="dsa name from dsaptailor, e.g., Beeblebrox"
   D="wildlife directory, e.g., $(ETCDIR)quipu/beeblebrox"
   ```

   and editing them appropriately.

3. Based on the time that the `c=US` manager gave you for your Level-1 DSA, add one hour and modify the *crontab* file according; e.g.,

   ```
   0 5 * * * $(ETCDIR)quipu/wildlife/nightly.sh
   ```

   If the directory database for the Level-2 DSA is owned by a user-ID other than `root` (e.g., `daemon`), then instead the line should look something like this:

   ```
   0 5 * * * su daemon < $(ETCDIR)quipu/wildlife/nightly.sh
   ```

Congratulations! Your Level-2 DSA has now joined the pilot DMD.

3.6 Miscellaneous Topics

Here is information on a wide range of topics, arranged in no particular order.

3.6.1 Moving a Level-1 DSA

For various reasons you might need to move your DSA from one host to another. In OSI terminology, you need to change the presentation address of the DSA. The steps to do this are:

1. Modify the `presentationAddress` attribute of your DSA using *dish* to include the second host. Also edit your /dsaptailor file to include this new address. In both cases, you simply add the string:

   ```
   |Internet=aaa.bbb.ccc.ddd+portno
   ```

   to the address. So, if the old address was

   ```
   '0101'H/Internet=192.33.4.20+17003
   ```

   the new address might be

   ```
   '0101'H/Internet=192.33.4.20+17003|Internet=130.117.128.2+17003
   ```

2. Edit your DSA's entry in the *c=US/EDB* file and remove the line

   ```
   eDBinfo= c=US # cn=Alpaca #
   ```

   This will prevent your DSA from seeing this change to it's presentation address, which is necessary since your DSA can't listen on the new address at the moment.

3. Now wait a couple of days for this new information to propagate. This is important to avoid a transient service outage.

4. Stop the DSA and move its hierarchy over to the new host.

5. Edit the *c=US/EDB* file and change your DSA's `presentationAddress` attribute to have only the new address, e.g.,

   ```
   presentationAddress= '0101'H/Internet=130.117.128.2+17003
   ```

6. Start the DSA on the new host and verify that it is working okay.

7. Modify the `presentationAddress` attribute to remove the first host address for your DSA using *dish* The presentation address will now match the value in the *c=US/EDB* file you have locally. Also edit your *dsaptailor* file to have only the new address.

8. Edit your DSA's entry in the *c=US/EDB* file and add the line

   ```
   eDBinfo= c=US # cn=Alpaca #
   ```

   This will resume the automatic downloading of information for your DSA.

9. Don't forget to edit */etc/rc.local* and */usr/lib/crontab* on both systems.

3.6.2 Running a SLAVE Level-1 DSA

to be supplied...

3.6.3 A Final Word on DSAs and Knowledge Information

It is important to appreciate that a DSA may hold knowledge (have local *EDB* files) even though the DIT does not indicate this (no corresponding `slaveDSA` attribute for that portion of the tree). This is a feature.

For example, if you wish to speed access to certain parts of the tree for your users, then to the entry of the DSA which MASTERs that information, you add this attribute:

```
eDBinfo = interesting_EDB # # slave_DSA_name
```

to the entry of the DSA which is to have a copy, you add this attribute:

```
eDBinfo = interesting_EDB # master_DSA_name #
```

And you do **not** add any `slaveDSA` attribute to the entry corresponding that EDB.

This configuration has the effect that any user contacting the slave DSA, will find that information local, but, since this DSA is not listed in the `slaveDSA` attribute, then other DSAs won't bother it asking for information.

3.6.4 Naming People

As noted earlier, entries in the Directory are uniquely named by their Relative Distinguished Name (RDN). In the pilot software, the RDN is represented as the first line of each entry in the *EDB* file for its immediate parent. Thus, within an *EDB* file, all RDNs must be unique.

The simplest way of doing this is to use:

```
cn=FirstName LastName
```

as the RDN. However, in organizations with large numbers of people, this may not be sufficient to be unique. So, there are four alternative strategies:

1. Use

   ```
   cn=FirstName LastName
   ```

   whenever possible. However, whenever ambiguity occurs, use either

   ```
   cn=FirstName MiddleInitial LastName
   ```

   or

   ```
   cn=FirstName MiddleName LastName
   ```

   Note that, the shorter forms should also be included to aid in searching. Hence, the first three lines of an entry might look like:

   ```
   cn=FirstName MiddleName LastName
   cn= FirstName MiddleInitial LastName
   cn= FirstName LastName
   ```

   The first value is used for the RDN, and the other two for searching.

2. Always use

   ```
   cn=FirstName MiddleName LastName
   ```

   as the RDN for all entries, and include the shorter forms whenever possible.

3. Always generate a uniquely constructed string

   ```
   cn=FML1
   ```

and include whatever real naming information is available for searching purposes:

```
cn= FirstName MiddleName LastName
cn= FirstName MiddleInitial LastName
cn= FirstName LastName
```

The only caveat with this approach is that each time the EDB file is generated, it is best not to change the RDNs for entries which previously existed.

4. Use a multi-valued RDN (*deus ex machina* formed by a `commonName` attribute and some other distinguishing attribute:

   ```
   cn=FirstName MiddleName LastName%userid=Lastname
   ```

   The '`%`'-sign is used to concatenate attributes when forming an RDN. Hence, if the first line of an entry is

   ```
   cn=Marshall Rose%userid=mrose
   ```

   then the entry's RDN really has two parts: the most significant part is a `commonName` attribute, and the next significant part is a `userid` attribute. Other good choices besides `userid` are things like `localityName`.

   Of course, the '`%`'-notation can be used only on the first line of an entry, as it is used only for RDNs.

In all cases, regardless of the actual RDN chosen, it is strongly recommended to include as many alternate forms as possible, in order to aid searching.

3.6.5 Installing the Software on other hosts

You may wish to install the interfaces to the Directory, on other systems, whilst running a DSA on a single host. If the hardware/software configuration of the new hosts are the same as the initial host, then you can simply install the compiled binaries. Usually this is done by mounting the source hierarchy over the network, logging in to the new host, and using:

```
# ./make inst-all inst-quipu
# (cd others/quipu; ./make inst-pilot)
```

Then, you must copy over two files from the ISODE `ETCDIR` directory on the original host: *dsaptailor* and *fredrc*.

If, however, you wish to change the software configuration, then after moutning the source hierarchy, you must clean the existing binaries in the source hierarchy, then select the new configuration files and go through the generation and installation process:

```
% ./make distribution
% ./make once-only all all-quipu
% (cd others/quipu; ./make pilot)
# ./make inst-all inst-quipu
# (cd others/quipu; ./make inst-pilot)
```

Next, you must then copy over the *dsaptailor* and *fredrc* files from the ISODE `ETCDIR` directory on the original host.

Bibliography

[CCITT88] The Directory — Overview of Concepts, Models, and Service. International Telegraph and Telephone Consultative Committee, December, 1988. Recommendation X.500.

[ISO88] Information Processing Systems — Open Systems Interconnection — The Directory — Overview of Concepts, Models, and Service. International Organization for Standardization and International Electrotechnical Committee, December, 1988. International Standard 9594-1.

[IUCN82] The IUCN Mammal Red Data Book. International Union for Conservation of Nature and Natural Resources, 1982. Unwin Brothers Limited, The Gresham Press, Woking, Surrey, UK. ISBN 2–88032–600–1.

[SKill89a] Stephen E. Kille. *The Design of QUIPU.* Research Note RN/89/19, Department of Computer Science, University College London, March, 1989.

[SKill89b] Stephen E. Kille, Colin J. Robbins, and Alan Turland. *The ISO Development Environment: User's Manual, Volume 5: QUIPU.* April, 1989.

Index