tree will usually represent objects such as countries (e.g., "GB") and organisations (e.g., "University College London"), entries at or near the leaves of the tree will represent people (e.g., "Colin Robbins"), equipment or application processes.

- Read and search operations will dominate over modification operations.
- Temporary inconsistencies in the data are acceptable. This greatly facilitates the replication of data in the directory by obviating concerns about record locking and atomic operations.

3.2 Information Representation

There are various structures needed to represent the data required in the directory database. These are described briefly below.

3.2.1 Object Identifiers

An important part of the OSI world is the object identifier (OID).

An OID is a hierarchy of numbers used uniquely to describe various objects within the OSI world: for example, "1.0.8571.1.1" describes the FTAM protocol; "0.9.2342.19200300.99.1" defines a QUIPU attribute type. The strings of numbers look "horrible" and can be difficult to remember. As such, QUIPU provides a mapping from OIDs to "strings", so that easy-to-remember strings can be used in place of the numeric OIDs, for example "iso ftam" or "quipuAttributeType".

The mappings are defined in a set of oidtables which are discussed in Section 14.11.

3.2.2 Attributes

The directory holds information about various entities, such as a person or an organisation. The information is held in an information object, typically referred to as an *entry*. An entry consists of a set of one of more attributes (see Figure 3.2).

An attribute is represented by an attribute type and a set of attribute values. An attribute with a single value is represented as follows:

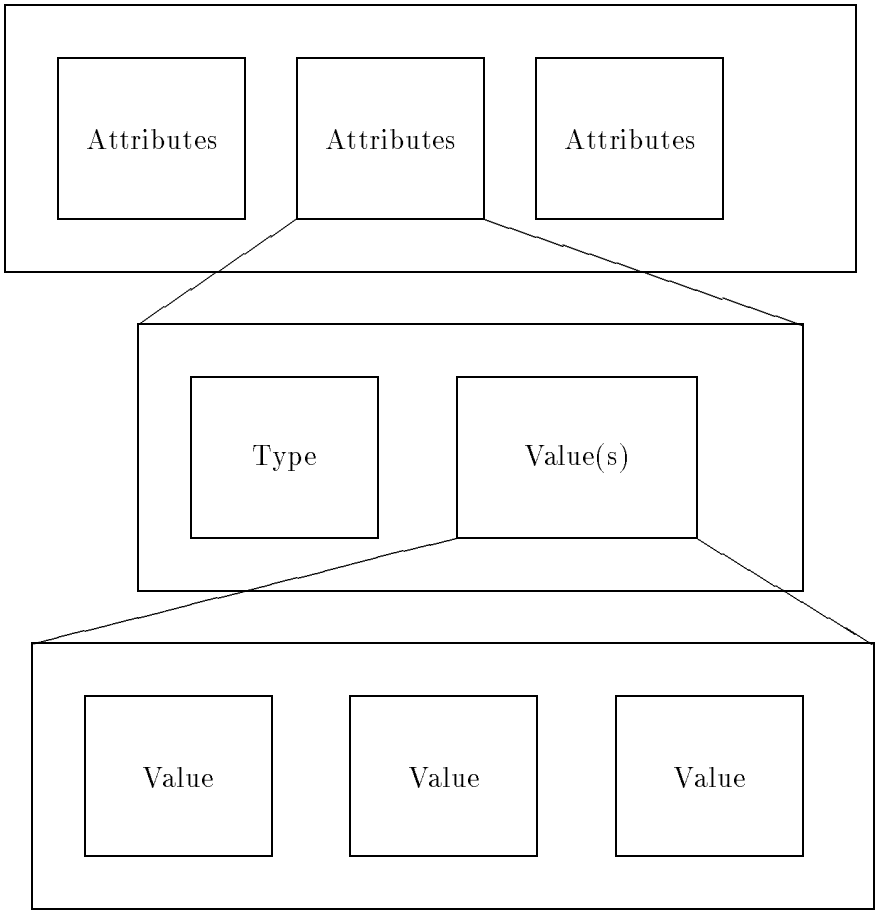

Figure 3.2: Structure of an Entry

<attribute type> "=" <attribute value>

The attribute type can either be a string (which must be in the attribute oidtable — see Section 14.11) or the OID in a dotted decimal format.

Each attribute value is represented by a string, the format of the attribute value depends upon the syntax defined in the oidtables, but in general will be a string.

So

```
roomNumber = G24
2.5.4.20   = 453-5674
commonName = Colin Robbins & Colin John Robbins
photo      = {ASN} 0308207b4001488001fd...
```

is an example of an entry, with 4 attributes. The "commonName" attribute is multivalued, with the two values separated by the "&" symbol. Notice how the "photo" attribute does not have a specific string format, so a hexadecimal "ASN.1" representation is used. "2.5.4.20" is an object identifier represented in numeric form.

3.2.3 Names

Within an entry, some attributes (generally one) have special importance, and are called distinguished attributes. The collection of these attributes form the relative distinguished name (RDN).

An RDN is usually represented by a single valued attribute, thus

```
commonName = Alan Turland
organization = University College London
```

are valid RDNs.

An RDN made up of multiple distinguished attributes uses the `%` symbol to separate the values, for example

```
userid = quipu % commonName = Colin Robbins
```

A distinguished name (DN) is the sequence of RDNs that uniquely define a node in the directory information tree (DIT). These are represented as follows:

RDN ["@" RDN ...]

So

```
countryName = GB @ organization = University College London
```

is an example of a valid DN.

In some cases an RDN or DN is specified relative to a node other than the root in the DIT. To be unambiguous a DN may be rooted using a leading "@", for example

```
@ countryName = GB @ organization = University College London
```

3.3 Directory User Agent

The directory user agent (DUA) is the entity you as a user will connect to when you interact with the directory service. It will help you formulate your queries, wrap them up in the required protocol, pass them to the directory, and then show you the results obtained.

The X.500 standard defines only the protocol the DUA should use when talking to the directory, and as such DUAs come in many varieties, but the basic concepts are the same.

A DUA talks to the directory using the directory access protocol (DAP).

The following chapters of this part of the manual describe several such DUA interfaces available in the ISODE. It is left to system managers to decide which form of interface is appropriate at your site, and which to install. There now follows a brief description of each interface:

DISH: The **DI**rectory **SH**ell (Chapter 4).
This interface give a user full access to the DAP, and as such may be complex for novice users.

SID: **S**teve's **I**nterface to **D**ISH (Chapter 5).
A set of scripts that make DISH usable for novice users.

DE: **D**irecory **E**nquires (Chapter 6).
A DUA for line mode terminals

FRED: FRont **E**nd to **D**ISH (Chapter 7).

FRED provides a white pages user interface, which hides most of the complexity of the OSI directory.

SD: Screen **D**irectory (Chapter 8).
A curses-based DUA.

POD: POpup **D**irectory (Chapter 9).
A DUA for the X Window System[1].

3.4 Directory System Agent

The directory system agent (DSA) is the entity that generates the results to requests from a DUA. It may handle the request itself, ask another DSA (using the directory system protocol), or advise the DUA to contact another DSA (see Figure 3.3). This part of the directory is discussed more in Part III of this manual. Here we shall concentrate on the DUA.

[1]The X Window System is a trademark of the Massachusettes Institute of Technology

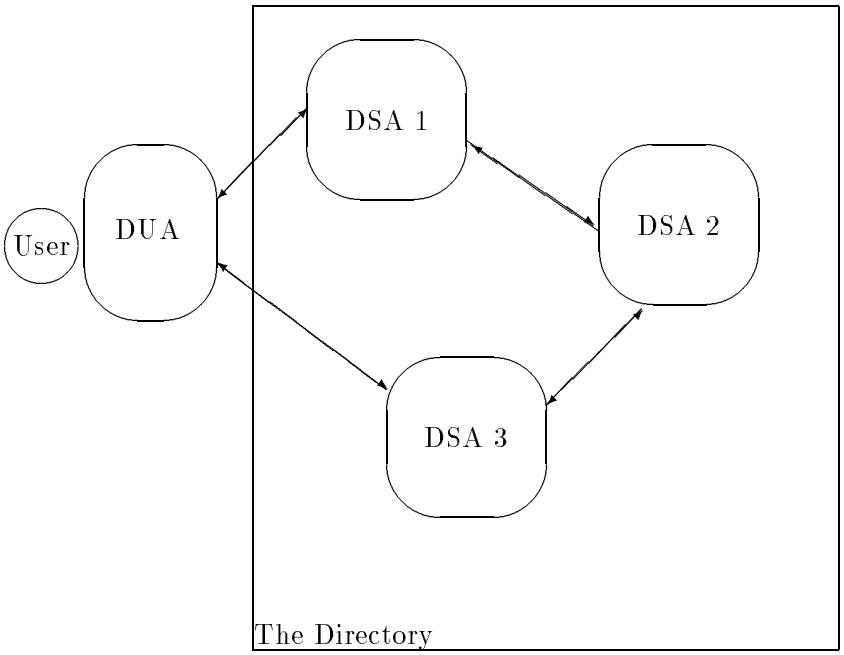

Figure 3.3: DUA/DSA Interaction

Chapter 4

DISH

This chapter describes DISH, a directory shell interface to the directory. It provides an interface to the directory in an similar manner to the way that MH provides an interface to a message handling systems. As with MH, DISH can either be invoked as a single process with the full repertoire of commands built in, or it can be invoked by individual shell commands. This latter style allows DISH to be used with others tools to provide very flexible access to the directory.

DISH provides a very powerful interface onto the directory and gives a user access to the full directory access protocol(DAP). The price of such comprehensiveness is complexity. DISH has a large number of flags, and both the syntax and volume of typing required can seem forbidding to the novice user. One compromise solution is to use DISH to build interfaces which are easier to use and more intuitive.

To run DISH in interactive mode invoke *dish*, then issue any of the commands described in Section 4.1. This mode of operation is especially useful for novice users. The process of connecting to the directory — binding — takes place automatically when the program is invoked, and an unbind is issued when you quit.

The arguments to DISH are the same as for bind (see Section 4.1.13), with the addition of a "-pipe" flag which is used to start DISH in "shell" mode (see Section 4.7) (Note: "dish -pipe &" has the same effect as the shell command "bind").

4.1 Commands

Each of the DISH commands described in this section has a large number of valid flags. The full names of the flags are given below; however, the shortest unique name is sufficient to select the flag. Similarly, when DISH is used interactively the shortest unique name is taken for the command name (e.g., "l" for "list"). There is an exception in that "sh" is interpreted as "showentry", "shown" must be typed for "showname".

The additional flag `-help` can be specified with every command to get limited runtime help.

4.1.1 Moveto

With nearly every command it is possible to supply the distinguished name of the object you want to reference. In the syntax used to describe the DISH commands this is represented by

```
<object>.
```

An initial "`@`" in an object description specifies that the name is relative to the root of the directory information tree (DIT), otherwise the name is taken as being relative to the current position in the DIT. The special name component ".." is used to mean "one position up" from the current position. Some examples of valid names and their interpretation relative to how a distinguished name is built up are shown below by the following sequence of names:

`@c=GB@o=University College London@ou=Computer Science:`
: The specified object is
"c= GB @ o= University College London @ ou= Computer Science".

`cn=Colin Robbins:` describes the object cn=Colin Robbins relative to the current position ("c= GB @ o= University College London @ ou= Computer Science @ cn= Colin Robbins"). Spaces in a name will be seen as the start of a separate argument by the shell, so the name must be quoted.

`..@cn=Steve Titcombe:` describes the entry cn=Steve Titcombe at the same level in the DIT as the current position ("c= GB @ o= University College London @ ou= Computer Science @ cn= Steve Titcombe").

Objects can also be expressed in the form of sequence numbers and nicknames. See Sections 4.2 and 4.4.1.

When you specify an object, the current position in the DIT is not changed. To change the current position you should use the command

```
moveto [-[no]pwd] [-[no]check] <position>
```

The `-pwd` flag tells moveto to print the current position in the DIT. The `-nocheck` flag tells moveto NOT to check the entry exists, normally, moveto will invoke `read` to check that the named entry exists.

The current position can also be changed with the `-move` flag to showentry and list commands (see Sections 4.1.3 and 4.1.2).

4.1.2 Showentry

```
showentry [<object>] [-[no]cache] [-[no]name]
          [-[no]move]
          [<any of the read arguments>]
          [<any of the FRED arguments>]
```

Showentry will display some or all of the attributes of the specified entry. A combination of the read argument flags and your *.quipurc* file described later in the chapter are used to define which attributes to show.

The `-noname` option tells showentry to show the distinguished name of the current entry as well as the attributes.

`-cache` is used to tell showentry to use cached information only — do not issue a read operation. This is used to see what the DUA has cached, and then get information when the local DSA is unavailable.

`-nocache` is used to enforce a directory read. This is used to update the cache directly.

`-move` is used to tell DISH to change the current position in the DIT to the object specified as well as showing the requested attributes.

See Section 4.1.4 for a discussion of the flags for FRED.

read flags

These flags are used by *showentry*, *showname* and *search*.

```
[-[no]types <attribute-type> *] [-[no]all]
[-[no]value] [-[no]show]
[-[no]key] [-edb]
[-proc <syntax> <process>]
```

The `all` flag request that all attributes are read (default).

The `value` flag reads the attribute value, which is the default, the inverse is to read the attribute types only.

The `novalue` flag says print the attribute types only.

The `show` flag is used to make DISH show the requested attributes (this is the default for read, but not for search).

The `key` flag determines whether the key (attribute type) will be shown along with the attribute value.

The `edb` flag requests that the data is displayed in "EDB" format, that is, a format which can be used in QUIPU data files. This is not such a nice format to present users, but is easier to manage in shell scripts.

The `type` flag requests that only the supplied attributes are read from the DSA; the inverse of this `-notype` does not prevent the attributes being read (as there is no easy mechanism to perform this within X.500), but it does stop them being shown.

The `proc` flag is used to instruct the display routines to use an external process to display the attribute. For example if you have a process that displays presentation elements representing OIDs in a particular way, then you can use

```
showentry -proc OID /usr/bin/oid_display_process.
```

All OIDs will then be displayed using "/usr/bin/oid_display_process". The process should read the presentation element from the standard input, and place any textual results on the standard output.

4.1.3 List

```
list [<object>] [-nocache] [-noshow] [-[no]move]
```