Network Working Group
INTERNET-DRAFT

S.E. Kille
ISODE Consortium
July 1993
Expires: January 1994

MHS use of Directory to support MHS Routing

Status of this Memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of this or any other Internet Draft.

Abstract

This document specifies an approach for X.400 Message Handling Systems to perform application level routing using the OSI Directory [16, 1]. Use of the directory in this manner is fundamental to enabling large scale deployment of X.400.

This draft document will be submitted to the RFC editor as a protocol standard. Distribution of this memo is unlimited. Please send comments to the author or to the discussion group <mhs-ds@mercury.udev.cdc.com>.

Contents

List of Figures

List of Tables

1 Introduction

MHS Routing is the problem of controlling the path of a message as it traverses one or more MTAs to reach its destination recipients.

Routing starts with a recipient O/R Address, and parameters associated with the message to be routed. It is assumed that this is known a priori, or is derived at submission time as described in Section 27.

The key problem in routing is to map from an O/R Address onto an MTA (next hop). This should be an MTA which in some sense is "nearer" to the destination UA. This is done repeatedly until the message can be directly delivered to the recipient UA. There are a number of things which need to be considered to determine this. These are discussed in the subsequent sections. A description of the overall routing process is given in Section 29.

2 Work to be done

This section notes things which still need to be done to this document, and also to other documents in the series. This includes:

1. Formalising ASN.1: defining modules and variable imports. (all documents)
2. Provide pseudo-code to describe the algorithm more clearly
3. Write a general note on MHS Use of Directory, summarising what is already defined in X.402, and showing how this work relates to it.
4. More examples
5. Notes on performance
6. Acknowledgements
7. Add appendix on regex
8. List of attributes which affect routing
9. Chase through note to ensure Application Process / Application Entity
10. Various minor changes marked with *** or as editor's notes throughout the document
11. Some restructure to improve overall clarity of document. Sections 18-22 will be the major targets.

3 Goals

Application level routing for MHS is a complex procedure, with many requirements. The following goals for the solution are set:

- Straightforward to manage. Non-trivial configuration of routing for current message handling systems is a black art, often involving gathering and processing many tables, and editing complex configuration files. Many problems are solved in a very ad hoc manner. Managing routing for MHS is the most serious headache for most mail system managers.
- Economic, both in terms of network and computational resources.
- Robust. Errors and out of date information should cause minimal and local damage.
- Deal with link failures. There should be some ability to choose alternative routes. In general, the routing approach should be redundant.
- Load sharing. Information on routes should allow "equal" routes to be specified, and thus facilitate load sharing.
- Support format and protocol conversion
- Dynamic and automatic. There should be no need for manual propagation of tables or administrator intervention.
- Policy robust. It should not allow specification of policies which cause undesirable routing effects.
- Reasonably straightforward to implement.
- Deal with X.400, RFC 822, and their interaction.
- Extensible to other mail architectures
- Recognise existing RFC 822 routing, and coexist smoothly.
- Improve RFC 822 routing capabilities. This is particularly important for RFC 822 sites not in the SMTP Internet.
- Deal correctly with different X.400 protocols (P1, P3, P7), and with 1984 and 1988 versions.
- Support X.400 operation over multiple protocol stacks (TCP/IP, CONS, CLNS) and in different communities.
- Messages should be routed consistently. Alternate routing strategies, which might introduce unexpected delay, should be used with care (e.g. routing through a protocol converter due to unavailability of an X.400 MTA).

	High Priority	Normal Priority	Low Priority
50% max delay	0.5 hour	1 hour	12 hours
98% max delay	3 hours	12 hours	24 hours
max delay	6 hours	72 hours	96 hours

Table 1: Possible target end to end delays

- Delay between message submission and delivery should be minimised. Table 1 indicates the sort of target which might be aimed for.

  The figures are illustrative of the sort of target which might be set. They are better than achieved in most current Research Networks. They are larger that the CEN/Cenelec figures, which were probably written by someone who has never run an MHS Network. The long tail-offs on Normal and Low priority recognise the fact that some end systems will not get 24 hour operator coverage (whereas any MTAs providing ADMD service should). In the case of a high priority message, a non-delivery notification should be returned within the suggested time.

- Interact sensibly with ADMD services provided by PTTs or RPOAs.
- Be global in scope
- Routing strategy should deal with a scale of order of magnitude 10^6 – 10^8 MTAs.
- Routing strategy should deal with of order 10^6 – 10^8 Organisations.
- Information about alterations in topology should propagate rapidly to sites affected by the change.
- Removal, examination, or destruction of messages by third parties should be difficult. This is hard to quantify, but “difficult” should be comparable to the effort needed to break system security on a typical MTA system.
- As with current Research Networks, it should be recognised that prevention of forged mail will not always be possible. However, this should be as hard as can be afforded.
- Sufficient tracing and logging should be available to track down security violations and faults.
- Optimisation of routing messages with multiple recipients, in cases where this involves selection of preferred single recipient routes.

The following are not initial goals:

- Advanced optimisation of routing messages with multiple recipients, noting dependencies between the recipients to find routes which would not have been chosen for any of the single recipients.
- Dynamic load balancing. The approach does not give a means to determine load. However, information on alternate routes is provided, which is the static information needed for load balancing.

4 Approach

A broad problem statement, and a survey of earlier approaches to the problem is given in the COSINE Study on MHS Topology and Routing [8]. The interim (table-based) approach suggested in this study, whilst not being followed in detail, broadly reflects what the MHS community is doing. The evolving specification of the RARE table format is defined in [5]. This document specifies the envisaged longer term approach. Although this work is important and needed now, there is a coherent interim approach, and this work should not be rushed.

Some documents have made useful contributions to this work:

- My paper on MHS use of directory, which laid out the broad approach of mapping the O/R Address space on to the DIT [7].
- The current OSI Standardisation work on MHS use of Directory for routing. The concept of Routing Entity is a useful one [17].
- The work of the VERDI Project [3].
- Work by Kevin Jordan of CDC [6].
- The routing approach of ACSNet [4, 15] paper. This gives useful ideas on incremental routing, and replicating routing data.
- A lot of work on network routing is becoming increasingly relevant. As the MHS routing problem increases in size, and network routing increases in sophistication (e.g., policy based routing), the two areas have increasing amounts in common. For example, see [2].

5 Direct vs Indirect Connection

Two extreme approaches to routing connectivity are:

1. High connectivity between MTAs. An example of this is the way the Domain Name Server system is used on the DARPA/NSF Internet. Essentially, all MTAs are fully interconnected.

2. Low connectivity between MTAs. An example of this is the UUCP network.

In general an intermediate approach is desirable. Too sparse a connectivity is inefficient, and leads to undue delays. However, full connectivity is not desirable, for the reasons discussed below.

A number of general issues related to relaying are now considered. The reasons for avoiding relaying are clear. These include.

- Efficiency. If there is an open network, it should be used.
- Extra hops introduce delay, and increase the (very small) possibility of message loss. As a basic principle, hop count should be minimised.
- Busy relays or Well Known Entry points can introduce high delay and lead to single point of failure.
- If there is only one hop, it is straightforward for the user to monitor progress of messages submitted. If a message is delayed, the user can take appropriate action.
- Many users like the security of direct transmission. It is an argument often given very strongly for use of SMTP.

Despite these very powerful arguments, there are a number of reasons why some level of relaying is desirable:

- Charge optimisation. If there is an expensive network/link to be traversed, it may make sense to restrict its usage to a small number of MTAs. This would allow for optimisation with respect to the charging policy of this link.
- Copy optimisation. If a message is being sent to two remote MTAs which are close together, it is usually optimal to send the message to one of the MTAs (for both recipients), and let it pass a copy to the other MTA.
- To access an intermediate MTA for some value added service. In particular for:
  - Message Format Conversion
  - Distribution List expansion
- Dealing with different protocols. The store and forward approach allows for straightforward conversion. Relevant cases include:
  - Provision of X.400 over different OSI Stacks (e.g. Connectionless Network Service).

  - Use of a different version of X.400.
  - Interaction with non-X.400 mail services

- To compensate for inadequate directory services: If tables are maintained in an ad hoc manner, the manual effort to gain full connectivity is too high.
- To hide complexity of structure. If an organisation has many MTAs, it may still be advantageous to advertise a single entry point to the outside world. It will be more efficient to have an extra hop, than to (widely) distribute the information required to connect directly. This will also encourage stability, as organisations need to change internal structure much more frequently than their external entry points. For many organisations, establishing such firewalls is high priority.
- To handle authorisation, charging and security issues. In general, it is desirable to deal with user oriented authorisation at the application level. This is essential when MHS specific parameters must be taken into consideration. It may well be beneficial for organisations to have a single MTA providing access to the external world, which can apply a uniform access policy (e.g. as to which people are allowed access). This would be particularly true in a multi-vendor environment, where different systems would otherwise have to enforce the same policy — using different vendor-specific mechanisms.

In summary there are strong reasons for an intermediate approach. This will be achieved by providing mechanisms for both direct and indirect connectivity. The manager of a configuration will then be able to make appropriate choices for the environment.

Two models of managing large scale routing have evolved:

1. Use of a global directory/database. This is the approach proposed here.
2. Use of a routing table in each MTA, which is managed either by a management protocol or by directory. This is coupled with means to exchange routing information between MTAs. This approach is more analogous to how network level routing is commonly performed. It has good characteristics in terms of managing links and dealing with link related policy. However, it assumes limited connectivity and does not adapt well to a network environment with high connectivity available.

6 X.400 and RFC 822

This document defines mechanisms for X.400 routing. It is important that this can be integrated with RFC 822 base routing, as many MTAs will work in both communities. This routing document is written with this problem in mind, and support for RFC 822 routing using the same basic infrastructure is defined in a companion document [13]. In addition support for X.400/RFC 822 gatewaying is needed, to support interaction. Directory based mechanisms for this are defined in [12]. The advantages of the approach defined by this set of specifications are:

- Uniform management for sites which wish to support both protocols.
- Simpler management for gateways.
- Improved routing services for RFC 822 only sites.

For sites which are only X.400 or only RFC 822, the mechanisms associated with gatewaying or with the other form of addressing are not needed.

7 Objects

It is useful to start with a manager's perspective. Here is the set of object classes used in this specification. It is important that all information entered relates to something which is being managed. If this is achieved, configuration decisions are much more likely to be correct. In the examples, distinguished names are written using the String Syntax for Distinguished Names [10]. The list of objects used in this specification is:

User An entry representing a single human user. This will typically be named in an organisational context. For example:

```
CN=Steve Kille, OU=Computer Science,
O=University College London, C=GB
```

This entry would have associated information, such as telephone number, postal address, and mailbox.

MTA A Message Transfer Agent. In general, the binding between machines and MTAs will be complex. Often a small number of MTAs will be used to support many machines, by use of local approaches such as NFS. MTAs may support multiple protocols, and will identify addressing information for each protocol.

To achieve support for multiple protocols, an MTA is modelled as an Application Process, which is named in the directory. Each MTA will have one or more associated Application Entities. Each Application Entity is named as a child of the Application Process, using a common name which conveniently identifies the Application Entity relative to the Application Process. Each Application Entity supports a single protocol, although different Application Entities may support the same protocol. Where an MTA only supports one protocol or where the addressing information for all of the protocols supported have different attributes to represent addressing information (e.g., P1(88) and SMTP) the Application Entity(ies) may be represented by the single Application Process entry.

User Agent (Mailbox) This defines the User Agent (UA) to which mail may be delivered. This will define the account with which the UA is associated, and may also point to the user(s) associated with the UA. It will identify which MTAs[1] are able to access the UA.

Role Some organisational function. For example:

```
CN=System Manager, OU=Computer Science,
O=University College London, C=GB
```

The associated entry would indicate the occupant of the role.

Distribution Lists There would be an entry representing the distribution list, with information about the list, the manger, and members of the list.

8 Communities

There are two basic types of agreement in which an MTA may participate in order to facilitate routing:

Open Agreements An agreement between a collection of MTAs to behave in a cooperative fashion to route traffic. This may be viewed as a general bilateral agreement.

Bilateral Agreements An agreement between a pair of MTAs to route certain types of traffic. This MTA pair agreement usually reflects a higher level agreement. It is an important special case of the first, as bilateral information must be held for the link at both ends. In some cases, this information must be private.

It is important to ensure that there are sufficient agreements in place for all messages to be routed. This will usually be done by having agreements which correspond to the addressing hierarchy. For X.400, this is the model where a PRMD connects to an ADMD, and the ADMD provides the inter PRMD connectivity, by the ability to route to all other ADMDs. Other

[1] In the formal X.400 model, there will be a single MTA delivering to a UA. In many practical configurations, multiple MTAs can deliver to a single UA. This will increase robustness, and is desirable.

agreements may be added to this hierarchy, in order to improve the efficiency of routing. In general, there may be valid addresses, which cannot be routed to, either for connectivity or policy reasons.

We model these two types of agreements as communities. A community is a scope in which an MTA advertises its services and learns about other services. Each MTA will:

1. Register its services in one or more communities.
2. Look up services in one or more communities.

In most cases an MTA will deal with a very small number of communities — very often one only. There are a number of different types of community.

The open community This is a public/global scope. It reflects routing information which is made available to any MTA which wishes to use it.

The local community This is the scope of a single MTA. It reflects routing information private to the MTA. It will contain an MTA's view of the set of bilateral agreements in which it participates, and routing information private and local to the MTA.

Hierarchical communities A hierarchical community is a subtree of the O/R Address tree. For example, it might be a management domain, an organisation, or an organisational unit. This sort of community will allow for firewalls to be established. A community can have complex internal structure, and register a small subset of that in the open community.

Closed communities A closed community is a set of MTAs which agrees to route amongst themselves. Examples of this might be ADMDs within a country, or a set of PRMDs representing the same organisation in multiple countries.

Formally, a community indicates the scope over which a service is advertised. In practice, it will tend to reflect the scope of services offered. It does not make sense to offer a public service, and only advertise it locally. Public advertising of a private service makes more sense, and this is shown below. In general, having a community offer services corresponding to the scope in which they are advertised will lead to routing efficiency. Examples of how communities can be used to implement a range of routing policies are given in Section 10.2.

9 Routing Trees

Communities are a useful abstract definition of the routing approach taken by this specification. Each community is represented in the directory as a *routing tree*. There will be many routing trees instantiated in the directory. Typically, an MTA will only be registered in and make use of a small number of routing trees. In most cases, it will register in and use the same set of routing trees.

9.1 Routing Tree Definition

Each community has a model of the O/R address space. Within a community, there is a general model of what to do with a given O/R Address. This is structured hierarchically, according to the O/R address hierarchy. A community can register different possible actions, depending on the depth of match. This might include identifying the MTA associated with a UA which is matched fully, and providing a default route for an O/R address where there is no match in the community — and all intermediate forms.

The name structure of a routing tree follows the O/R address hierarchy, which is specified in a separate document [11]. Where there is any routing action associated with a node in a routing tree, the node is of object class `routingInformation`, as defined in Section 11.

9.2 The Open Community Routing Tree

The routing tree of the open community starts at the root of the DIT. This routing tree also serves the special function of instantiating the global O/R Address space in the Directory. Thus, if a UA wishes to publish information to the world, this hierarchy allows it to do so.

The O/R Address hierarchy is a registered tree, which may be instantiated in the directory. Names at all points in the tree are valid, and there is no requirement that the *namespace* is instantiated by the owner of the name. For example, a PRMD may make an entry in the DIT, even if the ADMD above it does not. In this case, there will be a "skeletal" entry for the ADMD, which is used to hang the PRMD entry in place. The skeletal entry contains the minimum number of entries which are needed for it to exist in the DIT (Object Class and Attribute information needed for the relative distinguished name). This entry may be placed there solely to support the subordinate entry, as its existence is inferred by the subordinate entry. Only the owner of the entry may place information into it. This might be thought of as directory knowledge information. An analogous situation in current operational practice is to make DIT entries for Countries and US States.

9.3 Routing Tree Location

All routing trees follow the same O/R address hierarchy. Routing trees other than the open community routing tree are rooted at arbitrary parts of the DIT. These routing trees are instantiated using the subtree mechanism

routingTreeRoot **OBJECT−CLASS**
SUBCLASS OF routingInformation, subtree
::= oc−routing−tree−root

Figure 1: Location of Routing Trees

defined in the companion document "Representing Tables and Subtrees in the Directory" [11]. A routing tree is identified by the point at which it is rooted. An MTA will use a list of routing trees, as determined by the mechanism described in Section 10. Routing trees may be located in either the organisational or O/R address structured part of the DIT. All routing trees, other than the open community routing tree, are rooted by an entry of object class `RoutingTreeRoot`, as defined in Figure 1.

9.4 Example Routing Trees

Consider routing trees with entries for O/R Address:

```
PRMD=UK.AC; ADMD=Gold 400; C=GB;
```

In the open community routing tree, this would have a distinguished name of:

```
PRMD=UK.AC, ADMD=Gold 400, C=GB
```

Consider a routing tree which is private to:

```
O=University College London, C=GB
```

They might choose to label a routing tree root "`UCL Routing Tree`", which would lead to a routing tree root of:

```
CN=UCL Routing Tree, O=University College London, C=GB
```

The O/R address in question would be stored in this routing tree as:

```
PRMD=UK.AC, ADMD=Gold 400,
C=GB, CN=UCL Routing Tree,
O=University College London, C=GB
```

9.5 Use of Routing Trees to look up Information

Lookup of an O/R address in a routing tree is done as follows:

```
routingTreeList ATTRIBUTE
        WITH ATTRIBUTE−SYNTAX RoutingTreeList
        SINGLE VALUE
        ::= at−routing−tree−list

RoutingTreeList ::= SEQUENCE OF RoutingTreeName


RoutingTreeName ::= DistinguishedName,
```

Figure 2: Routing Tree Use Definition

1. Map the O/R address onto the O/R address hierarchy described in [11] in order to generate a Distinguished Name.
2. Append this to the Distinguished Name of the routing tree, and then look up the whole name.
3. Handling of errors will depend on the application of the lookup, and is discussed later.

Note that it is valid to look up a null O/R Address, as the routing tree root may contain default routing information for the routing tree. This is held in the root entry of the routing tree, which is a subclass of `routingInformation`. The open community routing tree does not have a default.

Routing trees may have aliases into other routing trees. This will typically be done to optimise lookups from the first routing tree which a given MTA uses. Lookup needs to take account of this.

10 Routing Tree Selection

The list of routing trees which a given MTA uses will be represented in the directory. This uses the attribute defined in Figure 2.

This attribute defines the routing trees used by an MTA, and the order in which they are used. Holding these in the directory eases configuration management. It also enables an MTA to calculate the routing choice of any other MTA which follows this specification, provided that none of its routing trees have access restrictions. This should facilitate debugging routing problems.

10.1 Routing Tree Order

The order in which routing trees are used will be critical to the operation of this algorithm. A common approach will be:

1. Access one or more shared private routing trees to access private routing information.
2. Utilise the open routing tree.
3. Fall back to a default route from one of the private routing trees.

Initially, the open routing tree will be very sparse, and there will be little routing information in ADMD level nodes. Access to many services will only be via ADMD services, which in turn will only be accessible via private links. For most MTAs, the fallback routing will be important, in order to gain access to an MTA which has the right private connections configured.

In general, for a site, UAs will be registered in one routing tree only, in order to avoid duplication. They may be placed into other routing trees by use of aliases, in order to gain performance. For some sites, Users and UAs with a 1:1 mapping will be mapped onto single entries by use of aliases.

10.2 Example use of Routing Trees

Some examples of how this structure might be used are now given. Many other combinations are possible to suit organisational requirements.

10.2.1 Fully Open Organisation

The simplest usage is to place all routing information in the open community routing tree. An organisation will simply establish O/R addresses for all of its UAs in the open community tree, each registering its supporting MTA. This will give access to all systems accessible from this open community.

10.2.2 Open Organisation with Fallback

In practice, some MTAs and MDs will not be directly reachable from the open community (e.g., ADMDs with a strong model of bilateral agreements). These services will only be available to users/communities with appropriate agreements in place. Therefore it will be useful to have a second (local) routing tree, containing only the name of the fallback MTA at its root.

Thus, open routing will be tried first, and if this fails the message will be routed to a single selected MTA.

10.2.3 Minimal-routing MTA

The simplest approach to routing for an MTA is to deliver messages to associated users, and send everything else to another MTA (possibly with backup).

An organisation using MTAs with this approach will register its users as for the fully open organisation. A single routing tree will be established, with the name of the organisation being aliased into the open community routing tree. Thus the MTA will correctly identify local users, but use a fallback mechanism for all other addresses.

10.2.4 Organisation with Firewall

An organisation can establish an organisation community to build a firewall, with the overall organisation being registered in the open community. This is an important structure, which cannot be achieved easily with current technology (e.g., DNS with MX records).

- Some MTAs are registered in the open community routing tree to give access into the organisation. This will include the O/R tree down to the organisational level. Full O/R Address verification will not take place externally.
- All users are registered in a private (organisational) routing tree.
- All MTAs in the organisation are registered in the organisation's private routing tree, and access information in the organisation's community. This gives full internal connectivity.
- Some MTAs in the organisation access the open community routing tree. These MTAs take traffic from the organisation to the outside world. These will often be the same MTAs that are externally advertised.

10.2.5 Well Known Entry Points

Well known entry points will be used to provide access to countries and MDs which are oriented to private links. A private routing tree will be established, which indicates these links. This tree would be shared by the well known entry points.

10.2.6 ADMD using the Open Community for Advertising

An ADMD uses the open community for advertising. It advertises its existence and also restrictive policy. This will be useful for:

- Address validation
- Advertising the mechanism for a bilateral link to be established

10.2.7 ADMD/PRMD gateway

An MTA provides a gateway from a PRMD to an ADMD. It is important to note that many X.400 MDs will not use the directory. This is quite legitimate. This technique can be used to register access into such communities from those that use the directory.

- The MTA registers the ADMD in its local community (private link)
- The MTA registers itself in the PRMD's community to give access to the ADMD.

11 Routing Information

Routing trees are defined in the previous section, and are used as a framework to hold routing information. Each node, other than a skeletal one, in a routing tree has information associated with it, which is defined by the object class `routingInformation` in Figure 3.

```
routingInformation OBJECT-CLASS
  SUBCLASS OF top
  MAY CONTAIN {
          subtreeInformation,
          routingFilter,
          routingFailureAction,
          mTAInfo,
          accessMD,
          nonDeliveryInfo,
          badAddressSearchPoint,
          badAddressSearchAttributes}
  ::= oc-routing-information
                  -- No naming attributes as this is not a
                  -- structural object class


subtreeInformation ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX SubtreeInfo
  SINGLE VALUE
  ::= at-subtree-information

SubtreeInfo ::= ENUMERATED {
  all-children-present(0),
  not-all-children-present(1) }


routingFilter ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX RoutingFilter
  ::= at-routing-filter


RoutingFilter ::= SEQUENCE{
```

```
               attribute-type OBJECT-IDENTIFIER,
               weight RouteWeight,
               dda-key String OPTIONAL,
               regex-match IA5String OPTIONAL,
               node DistinguishedName }

String ::= CHOICE {PrintableString, TeletexString}

routingFailureAction ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX RoutingFailureAction
  SINGLE VALUE
  ::= at-routing-failure-action

RoutingFailureAction ::= ENUMERATED {
                next-level(0),
                next-tree-only(1),
                next-tree-first(2),
                stop(3)  }


mTAInfo ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX MTAInfo
  ::= at-mta-info

MTAInfo ::= SEQUENCE {
                name DistinguishedName,
                weight [1] RouteWeight DEFAULT preferred-access,
                mta-attributes [2] SET OF Attribute OPTIONAL,
                ae-info [4] SEQUENCE OF SEQUENCE {
                        aEQualifier PrintableString,
                        ae-weight RouteWeight DEFAULT preferred-access,
                        ae-attributes SET OF Attribute OPTIONAL} OPTIONAL
}

RouteWeight ::= INTEGER  {endpoint(0),
                        preferred-access(5),
                        backup(10)} (0..20)
```

Figure 3: Routing Information at a Node

For example, information might be associated with the node:

`PRMD=CDC, ADMD=ATTmail, C=US`

If this node was in the open community routing tree, then the information represents information published by the owner of the PRMD relating to public access to that PRMD. If this node was present in another routing tree, it would represent information published by the owner of the routing tree about access information to the referenced PRMD. The attributes associated with a `routingInformation` node provide the following information:

- That the node corresponds a valid O/R address. This is implicit in the existence of the entry.

- If the node is a UA. This will be true if the node is of object class `routedUA`. This is described further in Section 12. If it is not of this object class, it is an intermediate node in the O/R Address hierarchy.
- Whether or not the node is authoritative for the level below is specified by the `subtreeInformation` attribute. If it is authoritative, indicated by the value `all-children-present`, this will give the basis for (permanently) rejecting invalid O/R Addresses. The attribute is encoded as enumerated, as it may be later possible to add partial authority (e.g., for certain attribute types). If this attribute is missing, the node is assumed to be non-authoritative (`not-all-children-present`). For example, consider the node:

  ```
  MHS-O=X-Tel, PRMD=UK.AC, ADMD=Gold 400, C=GB
  ```

  An organisation which has a bilateral agreement with this organisation has this entry in its routing tree, with no children entries. This is marked as non-authoritative. There is a second routing tree maintained by X-Tel, which contains all of the children of this node, and is marked as authoritative. When considering an O/R Address

  ```
  MHS-G=Random + MHS-S=Unknown, MHS-O=Fast Fruit ltd.,
  PRMD=MHS plc, ADMD=Gold 400, C=GB
  ```

  only the second, authoritative routing tree can be used to determine that this address is invalid.
- A list of MTAs and associated information defined by the mtaInfo attribute. This information is discussed further in Sections 16 and 19. This information is the key information associated with the node. When a node is matched in a lookup, it indicates the validity of the route, and a set of MTAs to connect to. Selection of MTAs is discussed in Sections 19 and Section 11.1.
- An action to be taken if none of the MTAs can be used directly (or if there are no MTAs present) is defined by the `routingFailureAction` attribute. When an routing tree is being used, it is usually in the context of a sequence of routing trees. If a matched node cannot be used directly, the routing algorithm will have the choice to move up a level in the current routing tree, or to move on to the next routing tree with an option to move back to the first tree later. This option to move back is to allow for the common case where a tree is used to specify two things:

  1. Routing information private to the MTA (e.g., local UAs or routing info for bilateral links).

2. Default routing information for the case where other routing has failed.

The actions allow for a tree to be followed, for the private information, then for other trees to be used, and finally to fall back to the default situation. For very complex configurations it might be necessary to split this into two trees. The options are:

1. Move up a level in the current routing tree. This is the action implied if the attribute is omitted. This will usually be the best action in the open community routing tree. It is the default action if the attribute is not present.
2. Move to the next tree. This will be useful optimisation for a routing tree where it is known that there is no useful additional routing information higher in the routing tree.
3. Move to the next tree, and then default back to the next level in this tree if no route is found. This will be useful for an MTA to operate in the sequence:
   (a) Check for optimised private routes
   (b) Try other available information
   (c) Fall back to a local default route

- The `accessMD` attribute is discussed in Section 11.3. This attribute is used to indicate MDs which provide indirect access to the part of the tree that is being routed to.

- The `badAddressSearchPoint` and `badAddressSearchAttributes` are discussed in Section 16. This attribute is for when an address has been rejected, and allows information on alternative addresses to be found.

- A set of routing filters, defined by the `routingFilter` attribute. This attribute provides for routing on information in the unmatched part of the O/R Address. This is described in Section 11.2.

11.1 MTA Choice

This section considers how the choice between alternate MTAs is made. First, it is useful to consider the conditions why an MTA is entered into a node of the routing tree are:

- The manager for the node of the tree must place it there. This is a formality, but critical in terms of overall authority.
- The MTA manager should agree to it being placed there.

- The MTA will in general (for some class of message) be prepared to route to any valid O/R address in the subtree implied by the address. The only exception to this is where the MTA will route to a subset of the tree which cannot easily be expressed by making entries at the level below. An example might be an MTA prepared to route to all of the subtree, with certain explicit exceptions.

Information on each MTA is stored in an `mTAInfo` attribute. This attribute contains:

name The Distinguished Name of the MTA (Application Process)

weight A weighting factor (Route Weight) which gives a basis to choose between different MTAs.

mta-attributes Attributes from the MTAs entry. Information on the MTA will generally be stored in the MTA's entry. The MTA is represented here as a structure, which enables some of this entry information to be represented in the routing node. This can lead to considerable performance optimisation. For example if ten MTAs were represented at a node, another MTA making a routing decision might need to make ten reads in order to obtain the information needed. If any attributes are present here, all of the attributes needed to make a routing decision will be included, and also all attributes at the Application Entity level

Where an MTA supports a single protocol only, or the protocols it supports have address information that can be represented in non-conflicting attributes, then the MTA may be represented as an application process only. In this case, the `ae-info` structure which gives information on associated application entities may be omitted, as the MTA is represented by a single application entity which has the same name as the application process. In other cases, the names of all application entities shall be included. A weight is associated with each application entity to allow the MTA to indicate a preference between its application entities.

ae-qualifier A printable string (e.g. "x400-88"), used to derive the relative distinguished name of the application entity.

ae-weight A weighting factor (Route Weight) which gives a basis to choose between different Application Entities (not between different MTAs).

ae-attributes Attributes from the AEs entry.

Route weighting is a mechanism to distinguish between different route choices. A routing weight may be associated with the MTA in the context of a routing tree entry. This is because routing weight will always be context dependent. This will allow machines which have other functions to be used as backup MTAs. The Route Weight is an integer in range 0–20. The lower the value, the better the choice of MTA. Where the weight is equal, and no other factors apply, the choice between the MTAs should be random to facilitate load balancing. If the MTA itself is in the list, it should only route to an MTA of lower weight. The exact values will be chosen by the manager of the relevant part of the routing tree. For guidance, three fixed points are given:

- 0. For an MTA which can deliver directly to the entire subtree implied by the position in the routing tree.
- 5. For an MTA which is preferred for this point in the subtree.
- 10. For a backup MTA.

When an organisation registers in multiple routing trees, the route weight used is dependent on the context of the subtree. In general it is not possible to compare weights between subtrees. In some cases, use of route weighting can be used to divert traffic away from expensive links.

Attributes that are available in the MTA entry and will be needed for making a routing choice are:

*** list attributes which must be present here if they are in the MTA ***

A full list of MTA attributes, with summaries of their descriptions are given in Section 17.

*** delete following list when referenced section is filled in ***

- Authentication policy of the MTA (e.g., some MTAs may require strong authentication). This is discussed in Section 21.2.
- Information on the stack offered by the MTA. This is discussed in Section 19.1.
- Policy for the traffic an MTA will route. This is discussed in Section 22.

11.2 Routing Filters

This attribute provides for routing on information in the unmatched part of the O/R Address, including:

- Routing on the basis of an O/R Address component type
- Routing on the basis of a substring match of an O/R address component. This might be used to route X121 addressed faxes to an appropriate MTA.

When present, the procedures of analysing the routing filters should be followed before other actions. The routing filter overrides `MTAinfo` and `accessMD` attributes. The components of the `routingFilter` attribute are:

attribute-type This gives the attribute type to be matched, and is selected from the attribute types which have not been matched to identify the routing entry. The filter applies to this attribute type. If there is no regulear expression present (as defined below), the filter is true if the attribute is present. The value is the object identifier of the X.500 attribute type (e.g., `at-prmd-name`).

weight This gives the weight of the filter, which is encoded as a Route Weight. If multiple filters match, the weight of each matched filter is used to select between them. If the weight is the same, then a random choice should be made.

dda-key If the attribute is domain defined, then this parameter may be used to identify the key.

regex-match This string is used to give a regular expression match on the attribute value. The syntax for regular expressions is defined in Appendix E.

node This defines where to get routing information for the filter. It will be an entry with object class `routingInformation`, which can be used to determine the MTA or MTA choice.

An example of use of routing filters is now given, showing how to route on X121 address to a fax gateway in Germany. Consider the routing point.

```
PRMD=UK.AC, ADMD=Gold 400, C=GB
```

The entry associated would have two routing filters:

1. One with type `x121` and no regular expression, to route a default fax gateway.
2. One with type `x121` and a regular expression `^9262` to route all German faxes to a fax gateway located in Germany with which there is a bilateral agreement. This would have a lower weight, so that it would be selected over the default fax gateway.

11.3 Indirect Connectivity

In some cases a part of the O/R Address space will be accessed indirectly. For example, an ADMD without access from the open community might have an agreement with another MD to provide this access. This is achieved

```
accessMD ATTRIBUTE
        WITH ATTRIBUTE−SYNTAX distinguishedNameSyntax
        ::= at−access−md
```

Figure 4: Indirect Access

by use of the `accessMD` attribute defined in Figure 4. If this attribute is found, the routing algorithm should read the entry pointed to by this distinguished name and route according to the information retrieved, in order to route to this access MD.

The attribute is called an MD, as this is descriptive of its normal use. It might point to a more closely defined part of the O/R Address space.

It is possible for both access MD and MTAs to be specified. This might be done if the MTAs only support access over a restricted set of transport stacks. In this case, the access MD should only be routed to if it is not possible to route to any of the MTAs.

This structure can also be used as an optimisation, where a set of MTAs provides access to several parts of the O/R Address space. Rather than repeat the MTA information, a single access MD is used as a means of grouping the MTAs. The value of the Distinguished Name of the access MD will probably not be meaningful in this case.

12 Local Addresses (UAs)

Local addresses (UAs) are a special case for routing: the endpoint. The definition of the `routedUA` object class is given in Figure 5. This identifies a User Agent in a routing tree. This is needed for several reasons:

1. To allow UAs to be defined without having an entry in another part of the DIT.
2. To identify which (leaf and non-leaf) nodes in a routing tree are User Agents. In a pure X.400 environment, a UA (as distinct from a connecting part of the O/R address space) is simply identified by object class. Thus an organisation entry can itself be a UA. A UA need not be a leaf, and can thus have children in the tree.
3. To allow UA parameters as defined in X.402 (e.g., the `mhs-deliverable-eits`) to be determined efficiently from the routing tree, without having to go to the user's entry.
4. To identify the MTA which supports a UA. The MTAs which support a UA directly are noted in the `SupportingMTA` attribute, which may be multi-valued. X.400 models that only one MTA is associated with

```
routedUA OBJECT-CLASS
  SUBCLASS OF routingInformation
  MAY CONTAIN {
                        -- from X.402
        mhs-deliverable-content-length,
        mhs-deliverable-content-types,
        mhs-deliverable-eits,
        mhs-message-store,
        mhs-preferred-delivery-methods,
        -- ** Note to add supported P2 extensions
        -- ** SEK cannot find standard reference
                        -- defined here
        mandatoryRedirect,
        supportingMTA,
        filteredRedirect,
        userName,
        nonDeliveryInfo}
  ::= oc-routed-ua

supportingMTA ATTRIBUTE
  SUBTYPE OF mtaInfo
  ::= at-supporting-mta

userName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
  ::= at-user-name
```

Figure 5: UA Attributes

a UA. In practice, it is possible and useful for several MTAs to be able to deliver to a single UA. This attribute is a subtype of `MTAinfo`, as it is possible for an MTA to be registered to route to a UA, without it actually being able to deliver messages to it.

This attribute must be present, unless the address is being non-delivered or redirected.

5. The attribute `nonDeliveryInfo` mandates non-delivery to this address, as described in Section 25.

6. The attributes `mandatoryRedirect` and `filteredRedirect` control redirects, as described in Section 23.

7. The attribute `userName` points to the distinguished Name of the user, as defined by the mhs-user in X.402. **REF** The pointer from the user to the O/R Address is achieved by the `mhs-or-addresses` attribute. This makes the UA/User linkage symmetrical.

When routing to a UA, an MTA will read the `supportingMTA` attribute. If it finds its own name present, it will know that the UA is local, and invoke appropriate procedures for local delivery (e.g., co-resident or P3 access information). The cost of holding these attributes for each UA at a site will often be reduced by use of shared attributes (** REF X.500(92)).

The linkage between the UA and User entries was noted above. It is also possible to use a single entry for both User and UA, as there is no conflict between these. In this case, the entries should be in one part of the DIT, with aliases from the other.

Many sites will need to support local RFC 822 mailboxes (UAs). As the RFC 822 mailboxes are untyped, it is necessary for such a site to represent every mailbox as an `routedUA` object.

To support UAs of other object classes, there will then need to be aliases to the correct entry. For example, suppose that there is a UA:

```
MHS-CN=Postmaster, MHS-OU=CS, MHS-O=UCL,
PRMD=UK.AC, ADMD=Gold 400, C=GB
```

There will have to be an alias:

```
CN="/CN=Postmaster/", MHS-OU=CS, MHS-O=UCL,
PRMD=UK.AC, ADMD=Gold 400, C=GB
```

which points to the real entry. For user convenience, it will also be desirable to have an alias:

```
CN=Postmaster, MHS-OU=CS, MHS-O=UCL,
PRMD=UK.AC, ADMD=Gold 400, C=GB
```

If there is a mailbox for:

```
MHS-OU=Sales, MHS-OU=CS, MHS-O=UCL,
PRMD=UK.AC, ADMD=Gold 400, C=GB
```

There will need to be an alias:

```
CN="/OU=Sales/", MHS-OU=CS, MHS-O=UCL,
PRMD=UK.AC, ADMD=Gold 400, C=GB
```

In general, aliases can be used extensively at this level to provide alternate values for names. For routing purpose, UAs (Mailboxes) will only be located by the read operation, not by searching. This is for efficiency of overall routing.

To support many UNIX mail utilities, it is important for UNIX sites to alias the UNIX login ids as alternate values, and to have tools which maintain this binding automatically.

12.1 Searching for Local Users

The approach defined in this specification performs all routing by use of reads. This is done for performance reasons, as it is a reasonable expectation that all DSA implementations will support a high performance read operation. For local routing only, an MTA in cooperation with the provider of the local routing tree may choose to use a search operation to perform routing. The major benefit of this is that there will not be a need to store aliases for alternate names, and so the directory storage requirement and alias management will be reduced. The difficulty with this approach is that it is hard to define search criteria that would be effective in all situations and well supported by all DUAs. There are also issues about determining the validity of a route on the basis of partial matches.

13 Direct Lookup

Where an O/R address is registered in the open community and has one or more “open” MTAs which support it, this will be optimised by storing MTA information in the O/R address entry. In general, the Directory will support this by use of attribute inheritance or an implementation will optimise the storage or repeate informaion, and so there will not be a large storage overhead implied. This is a function of the basic routing approach.

As a further optimisation of this case, the User’s distinguished name entry may contain the MTAInfo attribute. This can be looked up from the distinguished name, and thus routing on submission can be achieved by use of a single read.

14 Alternate Routes

14.1 Finding Alternate Routes

The routing algorithm selects a single MTA to be routed to. It could be extended to find alternate routes to a single MTA with possibly different weights. How far this is done should be a local configuration choice. Provision of backup routing is desirable, and leads to robust service, but excessive use of alternate routing is not usually beneficial. It will often force messages onto convoluted paths, when there was only a short outage on the preferred path.

It is important to note that this strategy will lead to picking the first acceptable route. It is important to configure the routing trees, so that the first route identified will also be the best route.

14.2 Sharing routing information

So far, only single addresses have been considered. Improving routing choice for multiple addresses is analogous to dealing with multiple routes. This section defines an optional improvement. When multiple addresses are present, and alternate routes are available, the preferred routes should be chosen so as to maximise the number of recipients sent with each message.

Specification of routing trees can facilitate this optimisation. Suppose there is a set of addresses (e.g., in an organisation) which have different MTAs, but have access to an MTA which will do local switching. If each address is registered with the optimal MTA as preferred, but has the "hub" MTA registered with a higher route weight, then optimisation will occur when a message is sent to multiple addresses in the group.

15 Looking up Information in the Directory

The description so far has been abstract about lookup of information. This section considers how information is looked up in the Directory. Consider that an O/R Address is presented for lookup, and there is a sequence of routing trees. At any point in the lookup sequence, there is one of a set of actions that can take place:

Handle MTA Info Information from the node should be examined. If suitable information is found, one of the following is done:

- Use the information and finish the routing process
- Continue the routing process in order to find more options to choose from

Unroutable Address Potentially valid address, which cannot be routed

Bad Address

Temporary Reject Try again later (** what action is implied?)

Permanent Reject Administrative error on the directory which should be fixed, but which prevents routing.

***** Add info on what to do and which diagnostic codes and general clarification

Each routing tree should be processed in turn. To start a new routing tree, the full O/R address should be looked up in the routing tree. The next routing tree should be started when:

- An entry is reached with routing action `next-tree-only` or `next-tree-first`.

- A lookup is done in the routing tree at the top level (i.e., country), and no routing action is returned. In this case, it should behave as if the action was `next-tree-first`. (*** SEK editorial note here that I did not understand).

Unless the action is `next-tree-only`, or the root of the routing tree has been processed, the routing tree which is about to be left should be pushed onto a stack of routing trees for future processing. The position in the tree should be retained.

Errors from the lookup (directory read) should be handled as follows:

AttributeError This leads to a permanent reject.

NameError The `matched` parameter is used to determine the number of components of the name that have matched (possibly zero). The read is then repeated with this name. This is the normal case, and allows the "best" entry in the routing tree to be located with two reads.

Referral The referral should be followed.

SecurityErrror Strip one component of the O/R address and continue.

ServiceError This leads to a temporary reject (see above).

**** Clarify Popping

16 Naming MTAs

MTAs need to be named in the DIT, but the name does not have routing significance, it is simply a unique key. Attributes associated with naming MTAs are given in Figure 6. This figure also gives a list of attributes, which may be present in the MTA entry. The use of most of these is explained in subsequent sections. The `mTAName` and `globalDomainID` attributes are needed to define the information that an MTA places in trace information. As noted previously, and MTA is represented as an Application Process, with one or more Application Entities.

In X.400, MTAs are named by MD and a single string. This style of naming is supported, with MTAs named in the O/R Address tree relative to the root of the DIT (or possibly in a different routing tree). The MTAName attribute is used to name MTAs in this case. For X.400(88) it is assumed that the Distinguished Name may be passed as an AE Title.

MTAs may be named with any other DN, which can be in the O/R Address or Organisational DIT hierarchy. There are several reasons why MTAs might be named differently.

```
mTAName ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
        caseIgnoreIA5StringSyntax (SIZE(1..ub-mta-name-length)
  SINGLE VALUE
  ::= at-mta-name
                          -- used for naming when
                          -- MTA is named in O/R Address Hierarchy

globalDomainID ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX
        GlobalDomainIdentifier
  SINGLE VALUE
  ::= at-global-domain-id
                          -- both attributes present when MTA
                          -- is named outside O/R Address Hierarchy
                          -- to enable trace to be written

mTAApplicationProcess OBJECT CLASS
  SUBCLASS OF application-process
  MAY CONTAIN {
        mTAWillRoute,
        globalDomainID,          -- Default
        routingTreeList,
        accessMD,
        localAccessUnit,
        accessUnitsUsed
  }
  ::= oc-mta-application-process

mTA OBJECT CLASS  -- Application Entity
  SUBCLASS OF mhs-message-transfer-agent
  MAY CONTAIN {
        mTAName,
        globalDomainID,          -- per AE variant
        responderAuthenticationRequirements,
        initiatorAuthenticationRequirements,
        responderPullingAuthenticationRequirements,
        initiatorPullingAuthenticationRequirements,
        initiatorP1Mode,
        responderP1Mode,
        polledMTAs,
        transportCommunity,
        respondingRTSCredentials,
        initiatingRTSCredentials,
        callingPresentationAddress,
        callingSelectorValidity,
        bilateralTable
        }
  ::= oc-mta
```

Figure 6: MTA Definitions

- The flat naming space is inadequate to support large MDs. MTA name assignment using the directory would be awkward.
- An MD does not wish to register its MTAs in this way (essentially, it prefers to give them private names in the directory).
- An organisation has a policy for naming application processes, which does not fit this approach.

In this case, the MTA entry must contain the correct information to be inserted in trace. The MTAName and GlobalDomainID attributes are used to do this. They are single value. For an MTA which inserts different trace in different circumstances, a more complex approach would be needed.

An MD may choose to name its MTAs outside of the O/R address hierarchy, and then link some or all of them with aliases. A pointer from this space may help in resolving information based on MTA Trace.

16.1 Naming 1984 MTAs

Some simplifications are necessary for 1984 MTAs, and only one naming approach may be used. In X.400, MTAs are named by MD and a single string. This style of naming is supported, with MTAs named in the O/R Address tree relative to the root of the DIT (or possibly in a different routing tree). The MTAName attribute is used to name MTAs in this case.

17 Attributes Associated with the MTA

This section lists the attributes which may be associated with an MTA as defined in Figure 6, summarises their use, and gives pointers to the relevant section.

*** tbs

18 Bilateral Agreements

*** Editorial note. Work thru this to sort out AP/AE tangle

*** this section needs to be clarified in general

In many cases, X.400 routing has to be on the basis of bilateral agreement. We can now consider how these agreements are represented in the Directory. A bilateral agreement is represented by one entry associated with each MTA participating in the bilateral agreement. For one end of the bilateral agreement, the agreement information will be keyed by the name of the MTA at the other end. Each party to the agreement will set up the entry which represents its half of the agreed policy. The fact that these correspond

```
mTABilateralTableEntry OBJECT−CLASS
  SUBCLASS OF mTA, distinguishedNameTableEntry
  ::= oc−mta−bilateral−table−entry
```

Figure 7: MTA Bilateral Table Entry

is controlled by the external agreement. In many cases, only one half of the agreement will be in the directory. The other half might be in an ADMD MTA configuration file.

MTA bilateral information is stored in a table, as defined in [11]. An MTA has one such table, which controls agreements in both directions. The definition of entries in this table are defined in Figure 7. This table will usually be access controlled so that only a single MTA or selected MTAs which appear externally as one MTA can access it.

Each entry in the table is of the object class `DistinguishedNameTableEntry`, which is used to name the entry by the distinguished name of the MTA. In some cases discussed in Section 21.1, there will also be aliases of type `textTableEntry`. The MTA attributes needed as a part of the bilateral agreement (typically MTA Name/Password pairs), as described in Section 21.3, will always be present. Other MTA attributes (e.g., presentation address) may be present for one of two reasons:

1. As a performance optimisation
2. Because the MTA does not have a global entry

Every MTA with bilateral agreements will define a bilateral MTA table. When a connection from a remote MTA is received, its Distinguished Name is used to generate the name of the table entry. For 1984, the MTA Name exchanged at the RTS level is used as a key into the table. The location of the bilateral table used by the MTA is defined by the `bilateralTable` attribute in the MTA entry, which is defined in Figure 18.

For outbound connections, an attribute in the MTA's entry (either read directly, or from `mTAInfo` from the routing tree) will indicate that a bilateral connection should be used. The entry containing the bilateral information for the MTA can be derived as for an incoming connection.

**** Tie in the use of RTS parameters, and add control of various types of trace stripping and orginator munging

```
transportCommunity ATTRIBUTE
  WITH ATTRIBUTE SYNTAX objectIdentifierSyntax
  ::= at−transport−community
```

Figure 8: Transport Community Definition

19 MTA Selection

19.1 Dealing with protocol mismatches

MTAs may operate over different stacks. This means that some MTAs cannot talk directly to each other. Even where the protocols are the same, there may be reasons why a direct connection is not possible. An environment where there is full connectivity over a single stack is known as a transport community [9]. The set of transport communities supported by an MTA is specified by use of the TransportCommunity attribute defined in Figure 8. This is represented as a separate attribute for the convenience of making routing decisions.

A community is identified by an object identifier, and so the mechanism supports both well known and private communities. A list of object identifiers corresponding to well known communities is given in Appendix B.

19.2 Supported Protocols

It is important to know the protocol capabilities of an MTA. This is done by the application context. There are standard definitions for the following 1988 protocols.

- P3 (with and without RTS, both user and MTS initiated)
- P7 (with and without RTS).
- P1 (various modes). Strictly, this is the only one that matters for routing.

In order to support P1(1984) and P1(1988) in X.410 mode, application contexts which define these protocols are given in Appendix C. This context is for use in the directory only, and would never be exchanged over the network.

For routing purposes, a message store which is not co-resident with an MTA is represented as if it had a co-resident MTA and configured with a single link to its supporting MTA.

In cases where the UA is involved in exchanges, the UA will be of object class mhs-user-agent, and this will allow for appropriate communication information to be registered.

```
restrictedSubtree OBJECT−CLASS
        SUBCLASS OF top
        MAY CONTAIN {
                subtreeDeliverableContentLength,
                subtreeDeliverableContentTypes,
                subtreeDeliverableEITs }
        ::= oc−restricted−subtree

subtreeDeliverableContentLength ATTRIBUTE
        SUBTYPE OF mhs−deliverable−content−length
        ::= at−subtree−deliverable−content−length

subtreeDeliverableContentTypes ATTRIBUTE
        SUBTYPE OF mhs−deliverable−content−types
        ::= at−subtree−deliverable−content−types

subtreeDeliverableEITs ATTRIBUTE
        SUBTYPE OF mhs−deliverable−eits
        ::= at−subtree−deliverable−eits
```

Figure 9: Subtree Capability Restriction

19.3 MTA Capability Restrictions

In addition to policy restrictions, described in Section 22, an MTA may have capability restrictions. The maximum size of MPDU is defined by the standard attribute mhs-deliverable-content-length.

It may be useful to define other capability restrictions, for example to enable routing of messages around MTAs with specific deficiencies.

It has been suggested using MTA capabilities as an optimised means of expressing capabilities of all users associated with the MTA. This is felt to be undesirable.

**** Add attribute which defines list of supported P1 extensions

19.4 Subtree Capability Restrictions

In many cases, users of a subtree will share the same capabilities. It is possible to specify this by use of attributes, as defined in Figure 9. This will allow for restrictions to be determined in cases where there is no entry for the user or O/R Address. This will be a useful optimisation in cases where the UA capability information is not available from the directory, either for policy reasons or because it is not there. This information may also be present in the domain tree (RFC 822).

This shall be implemented as a collective attribute, so that it is available to all entries in the subtree below the entry. This can also be used for default

```
initiatorP1Mode ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX P1Mode
  SINGLE VALUE
  ::= at-initiator-p1-mode

responderP1Mode ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX P1Mode
  SINGLE VALUE
  ::= at-responder-p1-mode

P1Mode ::= ENUMERATED {
  push-only(0),
  pull-only(1),
  twa(2) }

polledMTAs ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX PolledMTAs
  := at-polled-mtas

PolledMTAs ::= SEQUENCE {
        mta DistinguishedName,
        poll-frequency INTEGER OPTIONAL --frequency in minutes
        }

*** Need to add control mechanism (list) for MTA being polled
*** to identify which MTAs can poll it
```

Figure 10: Pulling Messages

setting in the subtree.

20 MTA Pulling Messages

Pulling messages between MTAs, typically by use of two way alternate, is for bilateral agreement. It is not the common case. There are two circumstances in which it can arise.

1. Making use of a connection that was opened to push messages.
2. Explicitly polling in order to pull messages

Attributes to support this are defined in Figure 10. These attributes indicate the capabilities of an MTA to pull messages, and allows a list of polled MTAs to be specified. If omitted, the normal case of push-only is specified.

21 Security and Policy

21.1 Finding the Name of the Calling MTA

A key issue for authentication is for the called MTA to find the name of the calling MTA. This is needed for it to be able to look up information on a bilateral agreement.

Where X.400(88) is used, the name is available as a distinguished name from the AE-Title provided in the A-Associate. For X.400(84), it will not be possible to derive a global name from the bind. The MTA Name exchanged in the RTS Bind will provide a key into the private bilateral agreement table. Thus for X.400(1984) it will only be possible to have bilateral inbound links or no authentication of the calling MTA.

Editor's Note CDC use a search here, as a mechanism to use a single table and an 88/84 independent access. This should be considered for general adoption. It appears to make the data model cleaner, possibly at the expense of some performance.

21.2 Authentication

The levels of authentication required by an MTA will have an impact on routing. For example, if an MTA requires strong authentication, not all MTAs will be able to route to it. The attributes which define the authentication requirements are defined in Figure 11.

The attributes specify authentication levels for the following cases:

Responder These are the checks that the responder will make on the initiator's credentials.

Initiator These are the checks that the initiator will make on the responders credentials. Very often, no checks are needed — establishing the connection is sufficient.

Responder Pulling These are responder checks when messages are pulled. These will often be stronger than for pushing.

Initiator Pulling For completeness.

If an attribute is omitted, no checks are required. If multiple checks are required, then each of the relevant bits should be set. If there are alternative acceptable checks, then multiple values of the attribute are used.

The values of the authentication requirements mean:

mta-name-present That an RTS level MTA parameter should be present for logging purposes.

```
responderAuthenticationRequirements ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX AuthenticationRequirements
  SINGLE VALUE
  ::= at-responder-authentication-requirements

initiatorAuthenticationRequirements ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX AuthenticationRequirements
  SINGLE VALUE
  ::= at-initiator-authentication-requirements

repsonderPullingAuthenticationRequirements ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX AuthenticationRequirements
  SINGLE VALUE
  ::= at-responder-pulling-authentication-requirements

initiatorPullingAuthenticationRequirements ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX AuthenticationRequirements
  SINGLE VALUE
  ::= at-initiator-pulling-authentication-requirements

AuthenticationRequirements ::= BITSTRING {
   mta-name-present(0),
   aet-present(1),
   aet-valid(2),
   network-address(3),
   simple-authentication(4),
   strong-authentication(5),
   bilateral-agreement-needed(6)}
```

Figure 11: Authentication Requirements

aet-present That a distinguished name application entity title should be provided at the ACSE level

aet-valid As for aet-present, and that the AET be registered in the directory. This may be looked up as a part of the validation process. If mta-name-present is set, the RTS value of mta and password must correspond to those registered in the directory.

network-address This can only be used for the responder. The AET must be looked up in the directory, and the `callingPresentationAddress` attribute matched against the calling address. This must match exactly at the network level. The validity of selectors will be matched according to the `callingSelectorValidity` attribute.

simple-authentication All MTA and password parameters needed for simple authentication must be used. This will usually be in conjunction with a bilateral agreement.

strong-authentication Use of strong authentication.

bilateral-agreement-needed This means that this MTA will only accept connections in conjunction with a bilateral agreement. This link cannot be used unless such an agreement exists.

These attributes may also be used to specify UA/MTA authentication policy. They may be resident in the UA entry in environments where this information cannot be modified by the user. Otherwise, it will be present in an MTA table (represented in the directory).

An MTA could choose to have different authentication levels related to different policies (Section 22). This is seen as too complex, and so they are kept independent. The equivalent function can always be achieved by using multiple Application Entities with the application process.

21.3 Authentication Information

This section specifies connection information needed by P1. This is essentially RTS parameterisation needed for authentication. This is defined in Figure 12. Confidential bilateral information is implied by these attributes, and this will be held in the bilateral information agreement. This should have appropriate access control applied. Note that in some cases, MTA information will be split across a private and public entry.

The parameters are:

Initiating Credentials The credentials to be used when the local MTA initiates the association. It gives the credentials to insert into the request, and those expected in the response.

```
respondingRTSCredentials ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX RTSCredentials
        SINGLE VALUE
        ::= at-responding-rts-credentials


initiatingRTSCredentials ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX RTSCredentials
        SINGLE VALUE
        ::= at-initiating-rts-credentials


RTSCredentials ::= SEQUENCE {
        request [0] MTAandPassword OPTIONAL,
        response [1] MTAandPassword OPTIONAL }


MTAandPassword ::= SEQUENCE {
        MTAName,
        Password }              -- MTAName and Password
                                -- from X.411


callingPresentationAddress ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX PresentationAddress
        MULTI VALUE
        ::= at-calling-presentation-address

callingSelectorValidity ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX CallingSelectorValidity
        SINGLE VALUE
        ::= at-calling-selector-validity

CallingSelectorValidity ::= ENUMERATED {
        all-selectors-fixed(0),
        tsel-may-vary(1),
        all-selectors-may-vary(2) }
```

Figure 12: MTA Authentication Parameters

```
mTAWillRoute ATTRIBUTE
  WITH ATTRIBUTE-SYNTAX MTAWillRoute
  ::= at-mta-will-route

MTAWillRoute ::= SEQUENCE {
        from [0]            SET OF ORAddressPrefix OPTIONAL,
        to [1]              SET OF ORAddressPrefix OPTIONAL,
        from-excludes [2]   SET OF ORAddressPrefix OPTIONAL,
        to-excludes [3]     SET OF ORAddressPrefix OPTIONAL }

ORAddressPrefix ::= DistinguishedName
```

Figure 13: Simple MTA Policy Specification

Responding Credentials The credentials to be used when the remote MTA initiates the association. It gives the credential expected in the request, and those to be inserted into the response.

Remote Presentation Address Valid presentation addresses, which the remote MTA may connect from.

If an MTA/Password pair is omitted. The MTA should default to the local MTA Name, and the password to a zero length OCTET STRING.

Note: It may be useful to add more information here relating to parameters required for strong authentication.

22 Policy and Authorisation

22.1 Simple MTA Policy

The routing trees will generally be configured in order to identify MTAs which will route to the destination. A simple means is identified to specify an MTA's policy. This is defined in Figure 13.

The multi-valued attribute gives a set of policies which the MTA will route. O/R Addresses are represented by a prefix, which identifies a subtree. A distinguished name encoding of O/R Address is used. There are three components:

from This gives a set of O/R addresses which are granted permission by this attribute value. If omitted, "all" is implied.

to This gives the set of acceptable destinations. If omitted, "all" is implied.

from-excludes This defines (by prefix) subtrees of the O/R address tree which are explicitly excluded from the "from" definition. If omitted, there are no exclusions.

to-excludes This defines (by prefix) subtrees of the O/R address tree which are explicitly excluded from the "to" definition. If omitted, there are no exclusions.

This simple policy should suffice for most cases. In particular, it gives sufficient information for most real situations where a policy choice is forced.

This simple prefixing approach does not deal explicitly with alias dereferencing. The prefixes refer to O/R addresses where aliases have been dereferenced. To match against these prefixes, O/R addresses being matched need to be "normalised" by being looked up in the directory to resolve alias values. If the lookup fails, it should be assumed that the provided address is already normalised. This means that policy may be misinterpreted for parts of the DIT not referenced in the directory.

The originator refers to the MTS originator, and the recipient to the MTS recipient, following any list expansion or redirect.

22.2 Complex MTA Policy

MTAs will generally have a much more complex policy mechanism, such as that provided by PP [14]. Representing this as a part of the routing decision does not seem worthwhile at present. Some of the issues which need to be tackled are:

- Use of charging and non-charging nets
- Policy dependent on message size
- Different policy for delivery reports.
- Policy dependent on attributes of the originator or recipient(e.g., mail from students)
- Content type and encoded information types
- The path which the message has traversed to reach the MTA
- MTA bilateral agreements
- Pulling messages
- Costs. This sort of policy information may also be for information only.

Policies relating to submission do not need to be public. They can be private to the MTA.

```
mandatoryRedirect ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
        SINGLE VALUE
        ::= at-mandatory-redirect

filteredRedirect ATTRIBUTE
        WITH ATTRIBUTE-SYNTAX FilteredRedirect
        SINGLE VALUE
        ::= at-filtered-redirect

FilteredRedirect ::= SEQUENCE {
        redirect-to DistinguishedName,
        SEQUENCE {
                min-size [1] INTEGER,
                max-size [2] INTEGER,
                content [3] ContentType,
                eit [4] ExternalEncodedInformationType }
        }
```

Figure 14: Redirect Definition

23 Redirects

There is a need to specify redirects in the Directory. This will be done at the O/R Address level (i.e., in a routing tree). This will be useful for alternate names where an equivalent name (synonym) defined by an alias is not natural. An example where this might be appropriate is to redirect mail to a new O/R address where a user had changed organisation. The definitions are given in Figure 14.

Mandatory redirects are specified by the `mandatoryRedirect` attribute. A filtered redirect is provided, to allow small messages, large messages, or messages containing specific EITs or content to be redirected. Message size is measured in kBytes. Where multiple elements are contained in the `FilteredRedirect` structure, they are treated as having a "logical or" relationship.

When a delivery report is sent to an address which would be redirected, X.400 would ignore the redirect. This means that every O/R address would need to have a valid means of delivery. This would seem to be awkward to manage. Therefore, the redirect should be followed, and the delivery report delivered to the redirected address.

These reidrects are handled directly by the MTA. Redirects can also be initiated by the UA, for example in the context of a P7 interaction.

```
nonDeliveryInfo ATTRIBUTE
        WITH ATTRIBUTE SYNTAX NonDeliveryReason
        ::= at−non−delivery−info

NonDeliveryReason ::= SEQUENCE {
        reason INTEGER (0..ub−reason−codes),
        diagnostic INTEGER (0..ub−diagnostic−codes),
        supplementaryInfo PrintableString }
```

Figure 15: Non Delivery Information

24 Underspecified O/R Addresses

X.400 requires that some underspecified O/R Addresses are handled in a given way. Where an underspecified O/R Address should be treated as if it were another O/R Address, an alias should be used. If the O/R Address should be rejected as ambiguous, and entry should be created in the DIT, and forced non-delivery specified for this reason.

25 Non Delivery

It is possible for a manager to define an address to non-deliver with specified reason and diagnostic codes. This might be used for a range of management purposes. The attribute to do this is defined in Figure 15.

26 Bad Addresses

If there is a bad address, it is desirable to do a directory search to find alternatives. This is a helpful user service and should be provided. This function is invoked after address checking has failed, and where this is no user supplied alternate recipient. This function would be an MTA-chosen alternative to administratively assigned alternate recipient.

Attributes to support handling of bad addresses are defined in Figure 16. The attributes are:

badAddressSearchPoint This gives the point (or list of points) from which to search.

badAddressSearchAttributes This gives the set of attribute types to search on. The default is common name.

Searches are always single level, and always use approximate match. If a small number of matches are made, this is returned to the originator by use

```
badAddressSearchPoint ATTRIBUTE
        WITH ATTRIBUTE−SYNTAX distinguishedNameSyntax
        ::= at−bad−address−search−point

badAddressSearchAttributes ATTRIBUTE
        WITH ATTRIBUTE−SYNTAX AttributeType
        ::= at−bad−address−search−attributes

alternativeAddressInformation EXTENSION
        AlternativeAddressInformation
        ::= id−alternative−address−information

AlternativeAddressInformation ::= SET OF SEQUENCE {
        distinguished−name DistinguishedName OPTIONAL,
        or−address ORAddress OPTIONAL,
        other−useful−info SET OF Attribute }
```

Figure 16: Bad Address Pointers

of the per recipient AlternativeAddresssInformation in the delivery report (DR). This should be marked non-critical, so that it will not cause the DR to be discarded. This attribute allows the Distinguished Name and O/R Address of possible alternate recipients to be returned with the delivery report. There is also the possibility to attach extra information in the form of directory attributes. Typically this might be used to return attributes of the entry which were matched in the search. A summary of the information should also be returned using the delivery report supplementary information filed (e.g., "your message could not be delivered to smith, try J. Smith or P. Smith"), so that the information is available to user agents not supporting this extension. Note the length restriction of this field is 256 (`ub-supplementary-info-length`).

If the directory search fails, or there are no matches returned, a delivery report should be returned as if this extra check had not been made.

Note: It might be useful to allow control of search type, and also single level vs subtree in future versions.

27 Submission

A message may be submitted with Distinguished Name only. If the MTA to which the message is submitted supports this service, this section describes how the mapping is done.

```
localAccessUnit ATTRIBUTE
        WITH ATTRIBUTE−SYNTAX AccessUnitType
        ::= at−local−access−unit

AccessUnitType ::= ENUMERATED {
        fax (1),
        physical−delivery (2),
        teletex (3),
        telex (4) }

accessUnitsUsed ATTRIBUTE
        WITH SYNTAX
        SelectedAccessUnit
        ::= at−access−units−used

SelectedAccessUnit ::= SEQUENCE {
        type AccessUnitType,
        providing−MTA DistinguishedName,
        filter SET OF ORAddress OPTIONAL }

*** update to provide a more general filtering mechanism
```

Figure 17: Access Unit Attributes

27.1 Normal Derivation

The Distinguished Name is looked up to find the attribute `mhs-or-addresses`. If the attribute is single value, it is straightforward. If there are multiple values, one O/R address should be selected at random.

27.2 Roles and Groups

Some support for roles is given. If there is no O/R address, and the entry is of object class role, then the `roleOccupant` attribute should be dereferenced, and the message submitted to each of the role occupants. Similarly, if the entry is of object class group, where the `groupMember` attribute is used.

28 Access Units

Attributes needed for support of Access Units are defined in Figure 17.

The attributes defined are:

localAccessUnit This defines the list of access units supported by the MTA.

accessUnitsUsed This defines which access units are used by the MTA, giving the type and MTA. An O/R Address filter is provided to control which access unit is used for a given recipient. For a filter to match an address, all attributes specificed in the filter must match the given address. This is specified as an O/R Address, so that routing to access units can be filtered on the basis of attributes not mapped onto the directory (e.g., postal attributes). Where a remote MTA is used, it may be necessary to use source routing.

Note: This mechanism might be used to replace the routefilter mechanism of the MTS routing.

29 The Overall Routing Algorithm

Having provided all the pieces, a summary of how routing works can be given. The very top level of the routing algorithm is:

1. Route the message according to the protocol by which it arrives (RFC 822 or X.400). In the case of RFC 822, this should follow [13].
2. If this fails, and gatewaying is supported, map the address and attempt to route according to the other protocol using [12].

29.1 X.400 Routing

The core of the X.400 routing is described in Section 11. A sequence of routing trees are followed. As nodes of the routing tree are matched, a set of MTAs will be passed upwards for evaluation. If all of these are rejected, the trees are followed further[2]. A set of MTAs is evaluated on the following criteria:

- If an MTA is the local MTA, deliver locally.
- Supported protocols. The MTA must support a protocol that the current MTA supports[3], as described in Section 19.2.
- The protocols must share a common transport community, as described in Section 19.1.
- There must be no capability restrictions in the MTA which prevents transfer of the current message, as described in Section 19.3.
- There must be no policy restrictions in the MTA which prevents transfer of the current message, as described in Section 22.

[2] It might be argued that the trees should be followed to find alternate routes in the case that only one MTA is acceptable. This is not proposed.

[3] Note that this could be an RFC 822 protocol, as well as an X.400 protocol.

- The authentication requirements of the MTA must be met by the local MTA, as described in Section 21.2.
- If the authentication (Section 21.2) indicates that a bilateral agreement is present, the MTA must be listed in the local set of bilateral agreements, as described in Section 18.
- In cases where the recipient UA's capabilities can be determined, there should either be no mismatch, or there must be an ability to use local or remote reformatting capabilities, as described in **** ref conversion I-D

29.2 Pseudo Code

This section describes the routing algorithm in pseudo-code.

***** tbs

29.3 Examples

to be supplied

30 Performance

*** notes on the overall performance of the scheme, and notes on implementation techniques to go faster

31 Acknowledgements

This ack section covers the whole document series.

*** tbs

This work was part funded by the COSINE Paradise project.

References

[1] The Directory — overview of concepts, models and services, December 1988. CCITT X.500 Series Recommendations.

[2] J.N. Chiappa. A new IP routing and addressing architecture, 1991. Internet Draft.

[3] A. Consael, M. Tschicholz, O. Wenzel, K. Bonacker, and M. Busch. DFN-Directory nutzung durch MHS, April 1990. GMD Report.

[4] P. Dick-Lauder, R.J. Kummerfeld, and K.R. Elz. ACSNET - the australian alternative to UUCP. In *EUUG Conference, Paris*, pages 60–69, April 1985.

[5] U. Eppenberger. Routing coordination for an X.400 MHS service within a multi protocol environment, October 1991. Version 1.0, RARE WG1 Document.

[6] K.E. Jordan. Using X.500 directory services in support of X.400 routing and address mapping, November 1991. 3rd draft, IETF WG.

[7] S.E. Kille. MHS use of directory service for routing. In *IFIP 6.5 Conference on Message Handling, Munich*, pages 157–164. North Holland Publishing, April 1987.

[8] S.E. Kille. Topology and routing for MHS. COSINE Specification Phase 7.7, RARE, 1988.

[9] S.E. Kille. Encoding network addresses to support operation over non-OSI lower layers. Request for Comments RFC 1277, Department of Computer Science, University College London, November 1991.

[10] S.E. Kille. A string representation of distinguished name. Request for Comments in preparation, Department of Computer Science, University College London, January 1992.

[11] S.E. Kille. Representing the O/R Address hierarchy in the directory information tree, July 1993. Internet Draft.

[12] S.E. Kille. Use of the directory to support mapping between X.400 and RFC 822 addresses, July 1993. Internet Draft.

[13] S.E. Kille. Use of the directory to support routing for RFC 822 and related protocols, July 1993. Internet Draft.

[14] S.E. Kille and J.P. Onions. The PP manual, December 1991. Version 6.0.

[15] P. Lauder, R.J. Kummerfeld, and A. Fekete. Hierarchical network routing. In *Tricomm 91*, 1991.

[16] CCITT recommendations X.400 / ISO 10021, April 1988. CCITT SG 5/VII / ISO/IEC JTC1, Message Handling: System and Service Overview.

[17] Zen and the ART of navigating through the dark and murky regions of the message transfer system: Working document on MTS routing, September 1991. ISO SC 18 SWG Messaging.

32 Security Considerations

Security considerations are not discussed in this INTERNET–DRAFT .

33 Author's Address

Steve Kille
ISODE Consortium
PO Box 505
London
SW11 1DX
England

Phone: +44-71-223-4062

EMail: S.Kille@ISODE.COM

DN: CN=Steve Kille,
O=ISODE Consortium, C=GB

UFN: S. Kille, ISODE Consortium, GB

A Object Identifier Assignment

```
mhs−ds OBJECT−IDENTIFIER ::= {iso(1) org(3) dod(6) internet(1) private(4)
        enterprises(1) isode−consortium (453) mhs−ds (7)}

routing OBJECT IDENTIFIER ::= {mhs−ds 3}

oc OBJECT IDENTIFIER ::= {routing 1}
at OBJECT IDENTIFIER ::= {routing 2}
id OBJECT IDENTIFIER ::= {routing 3}


oc−mta OBJECT IDENTIFIER ::= {oc 1}
oc−mta−bilateral−table−entry OBJECT IDENTIFIER ::= {oc 2}
oc−routing−information OBJECT IDENTIFIER ::= {oc 3}
oc−restricted−subtree OBJECT IDENTIFIER ::= {oc 4}
oc−routed−ua OBJECT IDENTIFIER ::= {oc 5}
oc−routing−tree−root OBJECT IDENTIFIER ::= {oc 6}
oc−mta−application−process OBJECT IDENTIFIER ::= {oc 7}

at−access−md OBJECT IDENTIFIER ::= {at 1}
at−access−units−used OBJECT IDENTIFIER ::= {at 2}
at−subtree−information OBJECT IDENTIFIER ::= {at 3}
at−bad−address−search−attributes OBJECT IDENTIFIER ::= {at 4}
at−bad−address−search−point OBJECT IDENTIFIER ::= {at 5}

at−calling−selector−validity OBJECT IDENTIFIER ::= {at 7}

at−filtered−redirect OBJECT IDENTIFIER ::= {at 38}
at−global−domain−id OBJECT IDENTIFIER ::= {at 10}
at−initiating−rts−credentials OBJECT IDENTIFIER ::= {at 11}
at−initiator−authentication−requirements OBJECT IDENTIFIER ::= {at 12}
at−initiator−p1−mode OBJECT IDENTIFIER ::= {at 13}
at−initiator−pulling−authentication−requirements OBJECT IDENTIFIER ::= {at 14}
at−local−access−unit OBJECT IDENTIFIER ::= {at 15}
at−mandatory−redirect OBJECT IDENTIFIER ::= {at 16}
at−mta−info OBJECT IDENTIFIER ::= {at 40}
at−mta−name OBJECT IDENTIFIER ::= {at 19}

at−mta−will−route OBJECT IDENTIFIER ::= {at 21}
at−calling−presentation−address OBJECT IDENTIFIER ::= {at 22}
at−responder−authentication−requirements OBJECT IDENTIFIER ::= {at 23}
at−responder−p1−mode OBJECT IDENTIFIER ::= {at 24}
at−responder−pulling−authentication−requirements OBJECT IDENTIFIER ::= {at 25}
at−responding−rts−credentials OBJECT IDENTIFIER ::= {at 26}
at−routing−failure−action OBJECT IDENTIFIER ::= {at 27}
at−routing−filter OBJECT IDENTIFIER ::= {at 28}
at−routing−tree−list OBJECT IDENTIFIER ::= {at 29}
at−subtree−deliverable−content−length OBJECT IDENTIFIER ::= {at 30}
at−subtree−deliverable−content−types OBJECT IDENTIFIER ::= {at 31}
at−subtree−deliverable−eits OBJECT IDENTIFIER ::= {at 32}
at−supporting−mta OBJECT IDENTIFIER ::= {at 33}
at−transport−community OBJECT IDENTIFIER ::= {at 34}
at−user−name OBJECT IDENTIFIER ::= {at 35}
at−non−delivery−info OBJECT IDENTIFIER ::= {at 36}
```

at−polled−mtas **OBJECT IDENTIFIER** ::= {at 37}
at−bilateral−table **OBJECT IDENTIFIER** {at 41}

id−alternative−address−information **OBJECT IDENTIFIER** ::= {id 1}

60

Figure 18: Object Identifier Assignment

B Community Identifier Assignments

```
ts-communities OBJECT-IDENTIFIER ::= {iso(1) org(3) dod(6) internet(1) private(4)
        enterprises(1) isode-consortium (453) ts-communities (4)}


tc-cons OBJECT IDENTIFIER ::= {ts-communities 1}          -- OSI CONS
tc-clns OBJECT IDENTIFIER ::= {ts-communities 2}          -- OSI CLNS
tc-internet OBJECT IDENTIFIER ::= {ts-communities 3}      -- Internet + RFC 1006
tc-int-x25 OBJECT IDENTIFIER ::= {ts-communities 4}       -- International X.25
                                                          -- Without CONS
tc-ixi OBJECT IDENTIFIER ::= {ts-communities 5}           -- IXI (Europe) [10]
tc-janet OBJECT IDENTIFIER ::= {ts-communities 6}         -- Janet (UK)
```

Figure 19: Transport Community Object Identifier Assignments

C Protocol Identifier Assignments

```
mail−protocol OBJECT−IDENTIFIER ::= {iso(1) org(3) dod(6) internet(1) private(4)
        enterprises(1) isode−consortium (453) mail−protocol (5)}

ac−p1−1984 OBJECT IDENTIFIER ::= {mail−protocol 1}        −− p1(1984)
ac−p1−1988−x410 OBJECT IDENTIFIER ::= {mail−protocol 1}  −− p1(1988) in X.410 mode
ac−smtp  OBJECT IDENTIFIER ::= {mail−protocol 2}          −− SMTP
ac−uucp OBJECT IDENTIFIER ::= {mail−protocol 3}           −− UUCP Mail
ac−jnt−mail OBJECT IDENTIFIER ::= {mail−protocol 4}       −− JNT Mail (UK)
```

Figure 20: Protocol Object Identifier Assignments

D ASN.1 Summary

To be supplied

E Regular Expression Syntax

*** tbs. Will be taken from ed(1) man page

*** Modified to be case insensitive and to handle leading and multiple spaces according to X.400 matching rules

F X.402 Definitions

Extract Attribute and Object Class definitions from X.402.

This appendix will be moved to the “overview” document when it is added

*** tbs