Network Working Group
Requests for Comments 1276

S.E. Hardcastle-Kille
University College London

November 1991

Replication and Distributed Operations extensions to provide an Internet Directory using X.500

Status of this Memo

This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

Some requirements on extensions to X.500 are described in the RFC[HK91b], in order to build an Internet Directory using X.500(1988). This document specifies a set of solutions to the problems raised. These solutions are based on some work done for the QUIPU implementation, and demonstrated to be effective in a number of directory pilots. By documenting a de facto standard, rapid progress can be made towards a full-scale pilot. These procedures are an INTERIM approach. There are known deficiencies, both in terms of manageability and scalability. Transition to standard approaches are planned when appropriate standards are available. This RFCwill be obsoleted at this point.

Contents

List of Figures

1 Approach

There are a number of non-negotiable requirements which must be met before a directory can be deployed on the Internet [HK91b]. These problems are being tackled in the standards arena, but there is currently no stable solution. One approach would be to attempt to intercept the standard. Difficulties with this would be:

- Defining a coherent intercept would be awkward, and the effort would probably be better devoted to working on the standard. It is not even clear that such an intercept could be defined.
- The target is moving, and it is always tempting to track it, thus causing more delay.
- There would be a delay involved with this approach. It would be too late to be useful for a rapid start, and sufficiently close to the timing of the final standard that many would choose not to implement it.

Therefore, we choose to take a simple approach. This is a good deal simpler than the full X.500 approach, and is based on operational experience. The advantages of this approach are:

- It is proven in operation. This RFCis simply documenting what is being done already.
- There will be a minimum of delay in starting to use the approach.
- The approach is simpler, and so the cost of implementation is much less. It will therefore be much more attractive to add into an implementation, as it is less effort, and can be further ahead of the standard.

These procedures are an INTERIM approach. There are known deficiencies, both in terms of manageability and scalability. Transition to standard approaches are planned when appropriate standards are available. This RFCwill be obsoleted at this point.

2 Extensions to Distributed Operations

The distributed operations of X.500 assume that all DUAs and DSAs are fully interconnected with a global network service. For the Internet Pilot, this assumption is invalid. DSAs may be operated over TCP/IP, TP4/CLNS, or TP0/CONS.

The extension to distributed operations to support this situation is straightforward. We define the term *community* as an environment where direct (network) communication is possible. Communities may be separated because they operate different protocols, or because of lack of physical connectivity. Example communities are the DARPA/NSF Internet, and the Janet private X.25 network. A network entity in a community is addressed by its Network Address. If two network entities are in the same community, they can by definition communicate. A community is identified by a set of network address prefixes. For the approach to be useful, this set should be small (typically 1). For TCP/IP Networks, and X.25 Networks not providing CONS, the approach is described in [HK91a] allows for communities to be defined for the networks of operational interest.

This model can be used to determine whether a pair of application entities can communicate. For each entity, determine the presentation address (typically by directory lookup). Each network address in the presentation address will have a single associated community. The set of communities to which each application entity belongs can thus be determined. If the two application entities have a common community, then they can communicate directly.

Two extensions to the standard distributed operations are needed.

1. Consider a DSA (the *local DSA*) which is contacted by either a DUA or DSA (the *calling entity*) to resolve a query. The *local DSA* determines that the query must be progressed by another DSA (the *referred-to DSA*). The DSA will make a chain/referral choice. If chaining is prohibited by service control, a referral will be passed back. Otherwise, if the *local DSA* prefers to chain (e.g., for policy reasons) it will then chain. The remaining situation is that the *local DSA* prefers to give a referral. It shall only do so if it believes that the *calling entity* can directly connect to the *referred-to DSA*. If the *calling entity* is a DUA, it should be assumed to belong only to the community of the called network address. If the *calling entity* is a DSA, its communities should

be determined by lookup of the DSA's presentation address in the directory. The communities of the *referred-to DSA* can be determined from its presentation address, which will either be present in the reference or can be looked up in the directory. If the *calling entity* and the *referred-to DSA* do not have a common community, then chaining shall be used. Otherwise, a referral may be passed back to the *calling entity*.

2. Consider that a DSA (or DUA), termed here the *local entity* is following a referral (to a *referred-to DSA*). In some cases, the *local entity* and *referred-to DSA* will not be able to communicate directly (i.e., not have a common community). There are two approaches to solve this:

   (a) Pass the query to a DSA it would use to resolve a query for the entry one level higher in the DIT. This will work, provided that this DSA follows this specification. This default mechanism will work without additional configuration.

   (b) Use a "relay DSA" to access the community. A relay DSA is one which can chain the query on to the remote community. The relay DSA must belong to both the remote community and to at least one community to which the *local entity* belongs. The choice of relay DSA for a given community will be manually configured by a DSA manager to enable access to a community to which there is not direct connectivity. Typically this will be used where the default DSA is a poor choice (e.g., because relaying is not authorised through this DSA).

   A DSA conforming to this specification shall follow these procedures. A DUA may also follow these procedures, and this will give improvements in some circumstances (i.e., the ability to resolve certain queries without use of chaining). However, this specification does not place requirements on DUAs.

3 Alternative DSAs

There is a need to give information on slave copies of data. This can be done using the standard protocol, but modifying the semantics. This relies on the fact that there may only be a *single* subordinate reference or cross reference.

If there is a need to include references to master and slave data (EDB copies) in a referral, then this should be done in a referral by specifying a subordinate reference with multiple values. This cannot be a standard subordinate reference, which would only have a single value. Therefore, this usage does not conflict with standard references. The first reference is the master copy, and subsequent references are slave copies.

4 Data Model

The X.500 data model takes the unit of mastering data as the entry. A DSA may hold an arbitrary collection of entries. We restrict this model so that for the replication protocol defined in this specification the base unit of replication (shadowing) is the complete set of immediate subordinate entries of a given entry, termed an Entry Data Block (EDB). An EDB is named by its parent entry. It contains the relative distinguished names of all of the children of the entry, and each of the child entries. For each entry, this comprises all attributes of the entry, the relative distinguished name, and knowledge information associated with the entry. If a DSA holds (non-cached) information on an entry, it will hold information on all of its siblings. One DSA will hold a *master EDB*. This will contain two types of entry:

1. Entries for which this DSA is the master.
2. Slave copies of entries which are mastered in another DSA, indicated by a subordinate reference. This copy must be maintained automatically by the DSA holding the master EDB.

Thus the master EDB contains a mixture of master entries, and entries which are mastered elsewhere and shadowed by the DSA holding the master EDB on an entry by entry basis. Other DSAs may hold slave copies of this EDB (*slave EDBs*), which are replicated in their entirity directly or indirectly from the master EDB. This approach has the following advantages.

- Name resolution is simplified, and performance improved.
- Single level searching and listing have good performance, and are straightforward to implement. In a more general case of applying the standard, without sophisticated replication, these operations might require to access very many DSAs and be prohibitively expensive.

5 DSA Naming

All DSAs must be named in the DIT, and the master definition of the presentation address stored in this entry. X.500 (including some of the extension work) implies that the presentation address information is extensively replicated (manually). The management overhead implied by this is not acceptable.

Care must be taken to prevent deadlock in determining a DSAs address. This is solved by:

1. Use of a well known DSA with "root knowledge"
2. Naming DSAs in a manner which prevents deadlocks. Currently this is done by giving DSAs names high in the DIT.

The Internet Pilot will need to define detailed policies for naming DSAs, in conjunction with the replication policy. This will be defined in a future RFC.

6 Knowledge Representation

Knowledge information is represented in the DIT. It seems unreasonable to manage this by any other means. Knowledge information is represented in an entry by use of *knowledge attributes.* These attributes are considered separately from all the other attributes in the entry which are termed "user attributes". Each entry in a master EDB will be in one of four categories.

1. The entry is a leaf entry mastered in this EDB, and so only contains user attributes
2. The level below has an associated EDB (i.e., the DIT continues downwards to use the data model of this specification). All attributes of this entry will be mastered in this entry. The entry will contain an attribute with the name of the DSA which holds the master of the associated EDB. Optionally, it will contain an attribute holding the names of DSAs which hold slave EDBs. The entry may not hold a subordinate reference attribute. The DIT is followed by use of the master and slave attributes.

3. The entry is mastered in a DSA which does not follow this specification. The entry in the EDB will contain a master attribute, which holds a subordinate reference (or cross reference) to the DSA which holds the master entry. The user attributes of the entry will be mastered in the DSA pointed to by the reference. The DSA holding the master EDB, which actually acts as an intermediate shadow for this entry, will read these attributes from the DSA indicated by the reference, so that it will have a full copy of the entry, using a standared DSP Read operation. This technique is called "spot shadowing". Any access control on the entry being spot shadowed must be configured so that all attributes can be copied by the DSA holding the master EDB. DSAs taking slave copies of the EDB will not do spot shadowing. However, the knowledge attributes will be copied, and may be used by this DSA (e.g., for modify operations).

4. The entries at the level below are held in DSAs which do not follow this specification, and all of these are indicated by a set of NSSRs (Non Specific Subordinate Reference). The NSSRs are stored as an attribute of the entry. The user attributes are either mastered in the EDB.

   It is important to note that NSSRs are stored at the level above subordinate references. At a given point in the DIT, if there are subordinate references, these are stored in shadow entries below that point, and named by the RDN. If there are NSSRs, they are stored in the entry itself, as there is no RDN associated with an NSSR. This approach is cleanest where there are either NSSRs or subordinate references, but not both. For example, consider an Organisation HP, whose many OUs are stored in a set of DSAs indicated by by NSSRs. Here, the NSSR attributes will be used to identify these DSAs.

   This model of replication is not tightly integrated with NSSRs. Where there is a mixture of NSSRs and Subordinate references at a given point in the DIT, this is handled by giving a single subordinate reference to a DSA which follows standard X.500 distributed operations and can cleanly handle this mixture. In practice, this is equivalent to not allowing a mixture of subordinate references and NSSRs.

The information framework needed to support this is defined in Figure 1.

```
InternetDSNonLeafObject ::= OBJECT-CLASS
          SUBCLASS OF top
          MUST CONTAIN {masterDSA}
          MAY CONTAIN {slaveDSA}

ExternalDSObject ::= OBJECT-CLASS
          SUBCLASS OF top
          MAY CONTAIN {SubordinateReference, CrossReference,
                    NonSpecificSubordinateReference}
                              -- will contain exactly one of these references

MasterDSA ::= ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
   SINGLE VALUE

SlaveDSA ::= ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax

SubordinateReference ::= ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX AccessPoint
   SINGLE VALUE

CrossReference ::= ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX AccessPoint
   SINGLE VALUE

NonSpecificSubordinateReference ::= ATTRIBUTE
   WITH ATTRIBUTE-SYNTAX AccessPoint

AccessPoint ::= SET {
          ae-title [0] Name,
          address  [2] PresentationAddress OPTIONAL }

                    -- Same definition as X.500 AccessPoint,
                    -- but presentation address is optional
```

Figure 1: Knowledge Attributes

Two object classes are defined to support this approach:

InternetDSNonLeafObject This is for where the level below follows the model defined here, and there is an Entry Data Block (EDB) containing the sibling entries. The Entry itself contains master data. The associated attributes are:

MasterDSA The name of the DSA where the master EDB is held.

SlaveDSA The names of DSAs which hold slave copies of the EDB for public access.

ExternalDSObject This is for where the *entry* and levels below are mastered according to X.500. There are attributes corresponding to the standard knowledge references, which are used to resolve queries. The presentation address is optional in these attributes. If not present, it should be looked up in the DSAs own entry. For NonSpecificSubordinateReference, the master of the entry will be in the master EDB, For SubordinateReference or CrossReference[1] the DSA which masters the EDB will "spot shadow" the entry, by reading it at intervals. This will ensure that the master EDB contains a copy of each entry. Single level searching can then be done efficiently where it is not required to access the master copy of the data. DSAs holding slave copies of the EDB do not perform spot shadowing, but do receive copies of the references.

7 Replication Protocol

```
GetEntryDataBlock ABSTRACT-OPERATION
        ARGUMENT GetEntryDataBlockArgument
        RESULT GetEntryDataBlockResult
        ERRORS {nameError,ServiceError,SecurityError,EDBVersionError}

EDBVersionError ABSTRACT-ERROR
        PARAMETER versionHeld EDBVersion


GetEntryDataBlockArgument ::= SET {
        entry [0] DistinguishedName,
        CHOICE {
                sendIfMoreRecentThan [1] EDBVersion,
```

[1]These references are really the same. The function and value are the same. The name depends on where the reference is stored. It may be preferable to have only one attribute.

```
                    getVersionNumber [2] NULL,
                    getEDB [3] NULL,            -- force retrieval
                    continuation [4] SEQUENCE {
                            EDBVersion,
                            nextEntryPosition INTEGER }
                    },
          maxEntries [5] INTEGER OPTIONAL
                            -- if omitted return whole EDB in
                            -- one operation
}

GetEntryDataBlockResult ::= SEQUENCE {
                    versionHeld [0] EDBVersion,
                    [1] SEQUENCE OF RelativeEntry OPTIONAL,
                            -- if omitted, only version is returned
                    nextEntryPostion INTEGER OPTIONAL
                            -- if omitted there are no more entries
          }



RelativeEntry ::= SEQUENCE {
          RelativeDistinguishedName,
          SET OF Attribute
          }

EDBVersion ::= UTCTime
```

Figure 2: Replication Protocol

A ROS operation to support replication is defined in Figure 2. This pulls an entire copy of the EDB. In normal use, the initiator specifies the EDB Version held. If the responder has a more recent version, then all of the entries in the EDB are returned. There are options to rerequest only the version of EDB held, or to return the full EDB irrespective of the version held by the initiator.

For large EDBs, transfer of an entire EDB in a single operation would lead to very large ROS PDUs. This gives a definite scaling limitation. To overcome this, the protocol allows an EDB to be retrived in chunks of a size (in number of entries) specified by the initiator. The responder specifies a number which indicates the next entry to be transferred. The same operation can be used to retrieve the next chunk of the EDB, with EDBVersion and the same integer as parameters.

This approach is simple to implement. It is less efficient than an incremental technique. When scaling dictates that an incremental technique must be used, it is expected that a suitable standard will be available.

An implementation issue that must be noted is how to deal with updates whilst a multi-operation transfer is in progress. There are two possible approaches:

1. Refuse/block updates until the EDB is transferred. This may cause problems where the rate of update and transfer is high, as this may make update very difficult (for the manager).

2. Create a new version of the EDB, whilst retaining the old EDB to complete the bulk transfer. A suitable retentions strategy would be to hold an EDB version as long as the association on which it is being pulled it remains active.

3. Allow the update and fail subsequent transfer requests for the EDB. This may cause both transfer failure and excessive waste of bandwidth due to retries if the rate of update and transfer is high.

If option 1. or 3. is chosen, for a widely replicated EDB where the update rate is greater than a few changes per day, it is recommended to configure the master EDB in a DSA which only replicates to one other DSA. This second DSA can then control its update rate, and safely perform a large fanout of replications (option 3). The first DSA will have reasonable availability for modifications (option 1).

This protocol will be used by DSAs to obtain copies of EDBs high in the tree (typically root and national EDBs). DSAs which need these copies should establish bilateral agreements to access them[2].

This protocol should only transfer user attributes. In particular, implementation specific attributes such as those needed to support private access control should not be transferred. There may be bilateral agreements on access control policy of the information (e.g., size limits on listing), which are implemented by (different) system specific techniques.

[2] QUIPU defines some attributes to register such agreements, but these are probably not appropriate for this specification.

8 New Application Context

A DSA which follows these procedures will support a new ApplicationContext "Internet DSP" defined in Appendix A. This will be stored in the DSAs entry, so that support of the extensions defined here can easily be determined.

9 Policy on Replication Procedures

To be effective, a directory configuration must be laid out. These protocols will need to be used in the framework of a pilot, and service providers making available data for replication.

There is a requirement to manage the replication process. This can be done by a combination of local configuration (to register shadowing agreements) and directory operations to set pointers to master and slave copies of the data.

10 Use of the Directory by Applications

Care must be taken by users of the directory when replication is available. This is not a change from current use of X.500, but is noted here as it is important. Normal read requests should allow use of copy information. If the user of the directory believes that information may be out of date (e.g., because an association could not be established), then the request should be repeated and use of copy data prohibited by service controls.

11 Migration and Scaling

The major scaling limit of this approach is the non-incremental update. This will put a limit on the maximum DIT fanout which can be supported. Given an average entry size of around a thousand bytes, and a maximum reasonable transfer size is tens of megabytes, then the fanout limit of this approach is of order 10 000. Note that smaller organisations will tend to be registered geographically (e.g., in the US, by State), so that the limit of the number of Organisations is somewhat larger. It should be noted that although the replication technique described here is general, it is only intended for high levels of the DIT. These figures assume this.

These techniques do not preclude use of other techniques for replication. It would be quite reasonable to replicate data using this approach, and that which will be defined in X.500(92).

References

[HK91a] S.E. Hardcastle-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.

[HK91b] S.E. Hardcastle-Kille. Replication requirement to provide an internet directory using X.500. Request for Comments RFC 1275, Department of Computer Science, University College London, November 1991.

12 Security Considerations

Security considerations are not discussed in this memo.

13 Author's Address

Steve Hardcastle-Kille
Department of Computer Science
University College London
Gower Street
WC1E 6BT
England

Phone: +44-71-380-7294

EMail: S.Kille@CS.UCL.AC.UK

A ASN.1 Summary and Object Identifier Allocation

There are a few object identifiers needed. These are defined here.

```
InternetDSP  TAGS ::=
BEGIN

IMPORTS
   APPLICATION−SERVICE−ELEMENT, PORT, APPLICATION−CONTEXT,
   aCSE, ABSTRACT OPERATION
          FROM Remote−Operations−Notation−extension {joint−iso−ccitt
          remote−operations(4) notation−extension(2)}


  id−as−mrse, id−as−mase, id−as−ms
          FROM MTSAccessProtocol {joint−iso−ccitt mhs−motis(6)
          protocols(0) modules(0) object−identifiers(0)}

  chainedReadASE, chainedSearchASE, chainedModifyASE
          FROM DirectorySystemProtocol {joint−iso−ccitt ds(5)
                    modules(1) dsp(12)}

  DistinguishedName, RelativeDistinguishedName, Attribute
          FROM InformationFramework {joint−iso−ccitt ds(5)
                    modules(1) InformationFramework(1)}


  ATTRIBUTE, OBJECT−CLASS
          FROM InformationFramework {joint−iso−ccitt ds(5)
          modules(1) informationFramework(1)};



internet−dsp OBJECT IDENTIFIER ::= {ccitt data(9) pss(2342)
          ucl(19200300) internet−dsp(107)}

−− General

at OBJECT IDENTIFIER ::= {internet−dsp at(1)}
oc OBJECT IDENTIFIER ::= {internet−dsp oc(2)}


−− Object Classes needed for association

id−ac−idsp  OBJECT IDENTIFIER ::= {internet−dsp ac−idsp(3))}
```

```
id-as-idsp  OBJECT IDENTIFIER ::= {internet-dsp as-idsp(4))}
id-ase-replication  OBJECT IDENTIFIER ::= {internet-dsp ase-replication(5))}


-- Attribute Types

master-dsa MasterDSA ::= {at 1}
slave-dsa SlaveDSA ::= {at 2}
subordinate-reference SubordinateReference ::= {at 3}
cross-reference CrossReference ::= {at 4}
nssr NonSpecificSubordinateReference ::= {at 5}

-- Object Classes

internet-ds-non-leaf-object InternetDSNonLeafObject ::= {oc 1}
external-ds-object ExternalDSObject ::= {oc 2}


-- Operation and Error bindings

getEntryDataBlock GetEntryDataBlock ::= 10

eDBVersionError EDBVersionError ::= 10


-- Protocol Definitions

replicationASE APPLICATION-SERVICE-ELEMENT
   OPERATIONS {getEntryDataBlock}
   ::= id-ase-replication

internet-dsp APPLICATION-CONTEXT
   APPLICATION SERVICE ELEMENTS {aCSE}
   BIND MSBind
   UNBIND MSUnbind
   REMOTE OPERATIONS {rOSE}
   OPERATIONS OF { chainedReadADSm chainedSearchASE,
          chainedModifyASE, replicationASE }
   ABSTRACT SYNTAXES {
          id-as-acse,
          id-as-idsp }
   ::= id-ac-idsp
```

```
InternetDSNonLeafObject ::= OBJECT-CLASS
        SUBCLASS OF top
        MUST CONTAIN {masterDSA}
        MAY CONTAIN {slaveDSA}

ExternalDSObject ::= OBJECT-CLASS
        SUBCLASS OF top
        MAY CONTAIN {SubordinateReference, CrossReference,
                NonSpecificSubordinateReference}
                        -- will contain exactly one of these references

MasterDSA ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
    SINGLE VALUE

SlaveDSA ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax

SubordinateReference ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX AccessPoint
    SINGLE VALUE

CrossReference ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX AccessPoint
    SINGLE VALUE

NonSpecificSubordinateReference ::= ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX AccessPoint

AccessPoint ::= SET {
        ae-title [0] Name,
        address  [2] PresentationAddress OPTIONAL }

                -- Same definition as X.500 AccessPoint,
                -- but presentation address is optional

GetEntryDataBlock ABSTRACT-OPERATION
        ARGUMENT GetEntryDataBlockArgument
        RESULT GetEntryDataBlockResult
        ERRORS {nameError,ServiceError,SecurityError,EDBVersionError}

EDBVersionError ABSTRACT-ERROR
        PARAMETER versionHeld EDBVersion
```

```
GetEntryDataBlockArgument ::= SET {
        entry [0] DistinguishedName,
        CHOICE {
                sendIfMoreRecentThan [1] EDBVersion,
                getVersionNumber [2] NULL,
                getEDB [3] NULL,            -- force retrieval
                continuation [4] SEQUENCE {
                        EDBVersion,
                        nextEntryPosition INTEGER }
                },
        maxEntries [5] INTEGER OPTIONAL
                        -- if omitted return whole EDB in
                        -- one operation
}

GetEntryDataBlockResult ::= SEQUENCE {
                versionHeld [0] EDBVersion,
                [1] SEQUENCE OF RelativeEntry OPTIONAL,
                        -- if omitted, only version is returned
                nextEntryPostion INTEGER OPTIONAL
                        -- if omitted there are no more entries
        }


RelativeEntry ::= SEQUENCE {
        RelativeDistinguishedName,
        SET OF Attribute
        }

EDBVersion ::= UTCTime
END
```

Figure 3: Summary of the ASN.1