Research Note RN/90/88
December 20th 1990

Research
Note

Management in QUIPU

S.E. Kille

Abstract

This document is a design plan for Management Extensions of the QUIPU Directory System [Kil88]. Management of the Directory is an increasing problem as pilot operation scales. This document describes a number of tools which will be developed. The aim here of this work is to manage the Directory and its configuration, but not the data in it. The latter is is an equally important, but rather separate issue.

The problems of directory management are not well understood, and so this specification has many research aspects. The design is expected to evolve in the light of experience. Some parts of this design have been implemented.

Department of Computer Science
University College London
Gower Street, London, WC1E 6BT

Contents

1 Introduction **2**
1.1 Goals 2

2 Storage of Management Information **2**
2.1 Access Mechanism 3
2.2 Information associated with a DSA 3

3 Monitoring **4**
3.1 DSA Dynamic Monitoring 4
3.2 DSA Control 5
3.3 DSA Console 5
3.4 Schema 5

4 Configuration Management **7**
4.1 Aliases 7
4.2 DSA Installation 7
4.3 DSA Configuration Verification 8
4.4 Consistency Checks 8
4.5 Directory Configuration Inspection 9
4.6 Configuration Changes 9
4.7 New aspects 13
4.8 Changing Names and Arboriculture 13

5 Authorisation **13**

6 Statistics and Accounting **15**
6.1 DIT Size 15

List of Figures

1 Management Schema 7
2 Authorisation Schema 15
3 DIT Statistical Information 16

1 Introduction

A significant planned evolution of QUIPU is the addition of the management facilities described in this document. Whilst most of the things specified here can be achieved in the current version of QUIPU (6.0 and subsequent beta releases), much needs to be done by direct editing of the EDB files. It is expected to move to a very high level of remote management. This document is concerned with management of the directory. Management of user data is considered elsewhere.

1.1 Goals

The aims of this work is nicely summarised in the initial contract. In more general terms, the following should be expected as a result of this work:

- Tools so that a system (DUAs and DSAs) can be installed, and then configured by use of the directory itself. There should be minimal need for configuring of internal files.
- Tools for verifying that a DSA is correctly installed as a part of the Directory.
- Tools for monitoring the operation of the Directory, so that continued correct operation of the system can be easily verified.
- Tools to change DSA configuration, particularly with respect to replication and knowledge.
- Mechanisms to authorise access to the Directory. This is intentionally a limited approach, but will provide some features needed urgently.

2 Storage of Management Information

A pre-requisite of some of the following work is to store additional information in the directory about the directory. The following information needs to be represented in the DIT, independent of DSA.

- Management information on the size of the DIT, and nature of information contained in it.

The following information needs to be stored on a per DSA basis, and should be remotely accessible using standard protocols.

- Static status attributes of the DSA, such as number of entries
- Dynamic status of the DSA, such as the number of open calls
- Statistical information
- Virtual attributes, used to control the DSA, some may be write only, and others read/write.
- Information on a per-EDB basis of EDBs in the DSA

2.1 Access Mechanism

The first choice is the access mechanism to read and modify information. Two basic choices available.

1. Management Protocols. This might be CMIS/CMIP or SNMP. The former is not sufficiently mature either in terms of specification or implementation to be a serious candidate in the time-frame. SNMP's simple information model is not so useful for some of the things proposed. There is a certain attraction in using a management protocol to perform management, although it is not clear that these protocols are ideal for application management.

2. Use of the Directory Protocols. This was what was envisaged in the original specification. It has the advantage of using existing infrastructure, and being easily integratable with the Directory. Reading of information can utilise the distributed operations of the directory, and no new protocol is needed.

The second approach will be taken for the reasons given.

2.2 Information associated with a DSA

For efficiency reasons, it is important that some of this information is stored in the DSA in question, and not replicated. It seems clean that the entry used is the DSA's own entry. Because of the way replication is achieved (to prevent deadlock), the DSA's own entry is currently stored in a different DSA. Certain information about the DSA (in particular the presentation address and supported application contexts) need to be widely replicated. Some of the new information is highly dynamic, and should not be replicated at all. The following mechanism is proposed.

1. The DSA's entry will be mastered in the DSA.

2. Selected information (in particular presentation address and supported application contexts) will be replicated in the parent EDB, and this replicated information will then be in line with the usual QUIPU model. This can be thought of as the parent entry having a subordinate reference to the DSA itself, with a maintained copy of information needed to access the DSA.

3. Further slave copies will be maintained by the usual QUIPU EDB replication mechanisms.

4. Queries which reference the master copy (e.g., updates) or refer to an attribute stored only within the DSA need to be directed to the DSA itself.

5. Search and Compare queries will never be routed to the DSA, so this restricts the attributes to which they can be applied.

6. ModifyRDN will not be permitted on DSAs.

7. DSA Names will always be leaf entries.

8. If a DSA with a slave copy (including the one with the spot-shadowed copy) receives a query about a DSA with don't use copy requested, it should pass the query to the DSA itself. This may be done by chaining or referral, using the usual criteria such as reachability for selection.

9. Replication in the parent EDB will be achieved by spot shadowing. Here the DSA with the parent EDB will pull a copy of the DSAs entry. This means that when a DSA changes its presentation address, it should stay listening on the old address long enough for this copy to happen. The manager of the DSA should ensure that this is the case.

10. Each DSA will store, in EDB format, its information which should be preserved between DSA reboots. This will be done by using the (slave or cache) copy of the EDB file which the DSA holds. If the DSA replicates the EDB, it will ensure that the master information is not overwritten. Transient information will be in memory only.

The schema associated with this information is defined in Figures 1 and 2.

3 Monitoring

The issues considered under monitoring are those relating to dynamic management of the DSA, as opposed to static ones considered under configuration.

3.1 DSA Dynamic Monitoring

There is a need to determine the operational state of a given DSA. This will be done either by use of a QUIPU specific operation or by reading pseudo-attributes. It will include information on:

- Listens available
- Calls open, and information about the calls
- Queries in progress
- Recent queries (to allow for debugging and monitoring of distributed operations)
- Query rate

The details are defined in Figure 1.

3.2 DSA Control

There is a need to control the operation of a DSA. In the current version of QUIPU this is achieved by use of a special attribute stored in the DSAs own EDB ("DSAControl"). This (virtual) attribute is only writable by the DSA Manager, and is not readable. Values written into it control operation of the DSA. These are described fully in the QUIPU Manual. The operations include:

- Lock the DSA (to prevent updates).
- Alter the logging level, This is primarily for debugging.
- Refresh: reread DSA parameters, and adapt to any changes. This operation is used when changing the Directory Configuration.
- Stop DSA (cause process to exit)

This function will be transferred to the DSA's own entry. This will allow control to utilise the normal distributed operations, rather than forcing a direct connection to the DSA. The syntax of the attribute (different from the current one) is documented in Figure 1.

3.3 DSA Console

There will be a DSA Console, equivalent to the PP MTA Console, to be implemented using X–Windows. This should display the status of one or more DSAs, and also allow for DSA control when appropriately bound. It should allow automatic "following" of distributed operations through multiple DSAs. This will allow distributed operation to be both demonstrated and debugged more easily. This tool can be seen as the user interface onto the management functions described in the previous two sections.

3.4 Schema

The new attributes associated with all of this work are define in Figure 1.

–– A set of attributes to be added to QUIPU DSA Object Class

–– Attributes are read–only unless otherwise indicated

MasterEntries **ATTRIBUTE**
WITH SYNTAX integerSyntax
SINGLE VALUE

10

SlaveEntries **ATTRIBUTE**
WITH SYNTAX integerSyntax
SINGLE VALUE

CacheEntries **ATTRIBUTE**

```
        WITH SYNTAX integerSyntax
        SINGLE VALUE


OpenCall ATTRIBUTE                                                      20
        WITH SYNTAX Call

Call ::= SEQUENCE {
        protocol ENUMERATED {
                dap(0),
                dsp(1),
                quipudsp(2)},
        association-id INTEGER,
        AuthLevel,
        initiated-by-dsa BOOLEAN,                                       30
        DistingushedName OPTIONAL,
        net-address OCTET STRING,
        start-time UTCTime,
        finish-time UTCTime OPTIONAL, -- cache info for monitoring
        pending-operations SET OF Operation,
        invoked-operations SET OF Operation }


AuthLevel ::= ENUMERATED {
                none(0),                                                40
                indentified(1),
                simple(2),
                protected-simple(3),
                strong(4) }

Operation ::= SEQUENCE {
        operation-id INTEGER,
        base-object DistinguishedName,
        start-time UTCTime,
        finish-time UTCTime OPTIONAL, -- cache info for monitoring       50
        chained-ops SEQUENCE OF SEQUENCE {
                association-id INTEGER,
                operation-id INTEGER } }

UsageRate ATTRIBUTE
        WTIH SYNTAX integerSyntax
                -- operations per minute averaged over 5 minutes


                                                                        60


DSAControl ATTRIBUTE                    -- write-only
        WITH SYNTAX DSAControl


DSAControl ::= CHOICE {
        lockdsa [0] NULL,
        setLogLevel [1] PrintableString,
        refresh [2] OptionalDN,                                         70
        stopDSA [3] NULL,
        unlock [4] NULL,
```

```
        resyynch [5] OptionalDN,
        changeTailor [6] PrintableString,
        updateSlaveEDBs [7]
        }

OptionalDN ::= CHOICE {
        NULL,                                                    80
        DistinguishedName }
```

Figure 1: Management Schema

4 Configuration Management

4.1 Aliases

Whilst aliases are, in some senses, user data, special tools are provided for dealing with them as a part of Directory Configuration.

A special interface is provided for managing aliases (built on DISH). We take a "schema consistent" view of aliases. A "correct" alias has a number of characteristics:

- The aliased entry contains only:
  - Object Class
  - AliasedObjectName
  - ACL
- AliasedObjectName points at an entry in the DIT
- Structure rules concerning the aliased object are preserved (i.e., the alias is "schema correct"

This has three functions, achieved as described.

Add alias Add an alias if the proposed parameters are correct

Delete alias Remove an alias

Check Alias Verify that an alias is correct. This can be applied recursively, to check parts of the DIT for correct aliases.

4.2 DSA Installation

This is only concerned with initial installation, and not the configuration issues discussed in Section 4.6. Therefore, only minimal issues beyond software installation need to be covered. The first stage of installation is the basic software, which occurs in all cases.

There are two types of installation.

1. Where the installer already has (local) access to the directory service, of which the new DSA is effectively a user. Here, the **Add DSA** procedures described below are appropriate.

2. Where this a a new DSA, which will be bootstrapped locally, and later added in to the global Directory after some local work. Here, a second stage of "default bootstrap" will be applied, to give the DSA a default standalone configuration. Data can then be added by user data tools, and the Add EDB procedures described here. This can then be added into the global DIT by the **Insert DSA** procedure.

4.3 DSA Configuration Verification

The basic level of management is of a single DSA. This section discussed procedures for dealing with a single DSA. Installation of a single DSA affects multiple DSAs, and is discussed in Section 4.6.

There will be tools to view the the configuration of a single DSA. These are provided by DISH scripts (QUIPU 6.0). Dish is described in the QUIPU Manual [KRRT90]. First there is an overall summary, which includes.

- Software version
- List of EDBs held
- Address

Then for each EDB, the following information may be obtained:

- Version number
- Master / Slave / Cache
- Number of Entries
- Volume of Data
- All RDNs
- All data

There will be a tool to list the tree (name only) for all the information held in a DSA.

All of this functionality will be accessible through the DSA Console.

4.4 Consistency Checks

The following consistency checks should be carried out by a DSA. Some are applied at DSA startup. Others are costly, and so should be applied at intervals using special tools. The following checks should be made:

1. Spurious (Unix) directories in the database

2. Extra attribute files

3. For all master non-leaf entries:

   - Check that the master and slave DSAs referenced (EDB Info) have copies of the implied EDB.
   - Check that referenced DSAs are accessible (directly or indirectly). This might be expensive, and so not be undertaken too often.

4. For the DSAs own EDB Info (master and slave), as determined from the DSAs own entry:

   - check that the EDB files are present (should happen at boot time)
   - Check that from and to DSAs and correctly listed (they exist, have the implied EDB, and the distance is acceptable)

   Note that the introduction of replication by subtree may well have an effect on this.

5. Check that all file attributes referenced in each EDB are present, and that their contents have a valid syntax

4.5 Directory Configuration Inspection

It will be important to have tools to enable viewing of the overall Directory Configuration. These will be provided by the use of DISH scripts. The basic function should operate at an EDB level, and for an EDB should be able to provide:

- Master DSA
- Registered Slave DSAs
- Unregistered Slave DSAs
- Propagation tree
- References to non-QUIPU DSAs
- Spot shadow information
- List of child EDBs (a basis for recursive operation)

4.6 Configuration Changes

General Approach This section describes how the directory is manipulated to give a coherent configuration. There are some general principles:

- As much as possible is done by use of standard directory operations.
- These operations are for DSA managers, and not naïve users. Access Control needs to be set up appropriately.

Many of these operations will be provided by use of DISH scripts, giving manual directions where external action must be taken (e.g., loading software).

Operations will often affect multiple DSAs. The tools will be designed so that if later operations fail, earlier ones will be unrolled. There is clearly some potential for getting the overall system into an incoherent state. At this point the tool will complain loudly, and expect intelligent intervention.

Three cases of management will be dealt with

- Single DSA Manager
- Multiple managers, but represented by the same individual
- Two managers. Here one manager will run the script, and make the changes it is possible to make. A script to make the other changes will be generated, and this can be sent out of band (e.g., by mail) to the other manager.

Add DSA Adding a new DSA proceeds in the following manner:

1. Install software at OSI location. No tailoring should be needed, other than that in the software configuration. Installation will include:
   - All of the relevant DSA binaries
   - A default DSA tailor file, which will include pointers to "well known" DSAs.
   - A base directory for the database, owned by an appropriate uid.
2. Add DSA entry to the global directory (using a DUA pointed at an existing DSA).
3. Boot new DSA.
4. Add local DSA information (over DAP).
5. Refresh DSA (see Section 3.2).

For testing purposes, step 2 may be replaced by adding an EDB which gives the DSA a dummy name stored locally. This will enable a DSA to be tested in isolation from the rest of the directory.

Insert DSA This procedure is to add a DSA which has been booted standalone, and contains a single master subtree.

1. Needs name of DSA and Subtree
2. Register DSA Information in the global DIT
3. Add entry corresponding to subtree root in global DIT with master DSA
4. Change EDB continuing subtree root entry in DSA to be slave or cache

Remove DSA

1. Needs name of DSA
2. Ensure no EDBs held by DSA
3. Turn off DSA
4. Remove DSA entry
5. Remove software

Modify DSA info Most DSA information can be changed as an aspect of local configuration. The name of a DSA cannot be changed externally. It is necessary to remove one DSA and then recreate a new one (or vice versa). In practice, the tools for DSA creation and EDB manipulation will make this straightforward.

Moving the OSI location of a DSA is straightforward, as the naming rules will ensure that this propagates without deadlock. Local tables will need to be adjusted for local DUAs. This will cause some level of disruption, and so OSI location changes should be avoided as far as possible.

Note that the presentation address of a DSA is stored in two entries, and that both must be modified.

Add new EDB AddEntry will only work if the appropriate EDB is in place (although subtree propagation might enable a sensible defaulting mechanism to be achieved). AddEntry should not create EDBs, as this affects multiple DSAs, and a failure could leave the directory in an inconsistent state. This also gives management control over the addition of new layers into the DIT. The following steps are followed by the EDB creation tool.

1. Need to specify Name of EDB and Master DSA
2. Check the EDB of parent exists
3. Add EDB info to DSA entry of the master DSA
4. Add master pointer in parent Entry
5. Refresh master DSA, which will cause EDB file to be created

Add EDB copy Usually, an EDB will be given a number of slave copies.

1. Needs name of EDB, slave DSA, update mechanism, and whether EDB will be visible
2. Add EDB info to slave DSA
3. Modify EDB Info of DSA sending update (often, but not necessarily the master)

4. Refresh slave DSA and DSA sending update
5. Optionally add slave pointer to entry. Every EDB will be given a small number of "visible" copies. EDBs high in the tree will often have many private copies.

Remove EDB copy An EDB copy is removed as follows:

1. Needs name of EDB and DSA name
2. Ensure no copies being sent from this DSA
3. Modify EDB Info of sending DSA
4. Modify pointer in Entry (if needed)
5. Remove EDB Info from DSA
6. Refresh DSA

Remove EDB After removing all copies, a master EDB is removed as follows:

1. Ensure no copies being sent from the DSA
2. Ensure that EDB has no remaining entries (might optionally remove all such entries)
3. Remove pointer in entry
4. Remove EDB from DSA
5. Get DSA to refresh (or wait for reboot)

Change EDB Master It is sometimes necessary to change a master or EDB propagation path, and their should be tools to facilitate this.

The propagation path can be viewed as a tree. The basic tool acts like the unix mv command. It works as follows:

1. Needs the EDB/DSA pair being moved, and name of new feed DSA
2. If incremental update is used, remove item from old feed DSA
3. Optionally remove DSA from getEDB list of old feed DSA
4. Change source in DSA
5. If incremental update is used, add item to new feed DSA
6. Optionally add DSA to getEDB list of new feed DSA
7. Optionally modify the associated entry list of slaves

8. Extra twiddles if moving a parent to a child

Changing the master is a special case of this. In addition:

- The EDB entry must be updated to reflect change of master
- The new master must not

4.7 New aspects

Additional management will be implied by the ongoing work on distributed operation.

References Facilities will be provided to establish, update, and remove references to non-QUIPU DSAs.

Spot Shadows Facilities will be provided to establish, update, and remove spot shadows.

4.8 Changing Names and Arboriculture

It would useful to be able to change non-leaf names, and to move around portions of the DIT. This is rather easy to do locally, because of the nature of the QUIPU EDB format. However, much care must be taken to correctly preserve external references. Often, there will be the need to use aliases as a part of the transition.

Moving is not supported directly by the directory. This needs to be done by creation and deletion, with appropriate additional EDB management (pointers). Tools should be provided to:

- Delete EDBs (recursively)
- Copy EDBs
- Move EDBs (combining the previous two)

These sound easy, but much care must be taken with the propagation aspects. It is not currently planned to implement this functionality.

5 Authorisation

We will tackle authorisation of access. The following will be considered:

1. Who can connect
    - DSA and DUA rights
    - Specification by entry/subtree/filter
2. Which networks the authorised entity is allowed access to

3. Level of access (authentication policy variations)

The functionality is defined in Figure 2. Briefly, access is considered as a set of grants and denials. This relates to access to DSAs, and not to data. If no grants are present, open access is assumed as the default. If any grants are present, closed access is assumed as the default. A denial will always have priority over a grant. Grants and denials are pairwise as follows:

- Whether the entity is a DUA or DSA
- The level of authentication
- An identification of the entity, which is one of:
  - An explicit DUA or DSA
  - A list, identified by a group entry in the directory
  - A subtree, identified by the base object and a single Attribute Value Assertion as a filter (e.g., UserClass ≡ Student). It might be necessary to allow for a more complex filter.
  - A network, identified by the NSAP prefix. This is used to control access to charging networks.
  - The local entity (typically to control local access)
  - Everything (typically to grant open access to some resource)

```
-- These are attributes needed for authorisation
-- They should be added to QUIPUDSA object class


-- If no "grants" are present then open access is the default

GrantAccess ATTRIBUTE
        WITH SYNTAX AccessRight
                                                        10

DenyAccess ATTRIBUTE
        WITH SYNTAX AccessRight



AccessRight ::= SEQUENCE {
        from SET OF USER
        to SET OF USER }
                                                        20

User ::= SEQUENCE{
        dua BOOLEAN, -- otherwise a DSA
        auth-level AuthLevel DEFAULT simple,
        CHOICE {
        explicit [0] DistinguishedName,
        group [1] DistinguishedName,
        subtree [2] SEQUENCE {
                base DistinguishedName
```

```
            filter AVA OPTIONAL },
            -- simple filtering, could be made more general
network [3] OCTET STRING,
            -- prefix
local [4] NULL,
everything [5] NULL
}}
```

Figure 2: Authorisation Schema

6 Statistics and Accounting

Statistics logging will be provided, which produces logs which can be machine processed for:

1. Directory Usage. This will indicate who has been using the directory and for what
2. Performance monitoring. This will note performance of operations, and availability of DSAs, on the basis of normal operation (as distinct from explicit probing).
3. DSA availability statistics will be generated
4. Tables of data sizes in the DIT

6.1 DIT Size

Attributes are defined to indicate the size of the DIT Figure 3. This is stored in the entry at the root of the subtree. This information is updated by the DSA which contains the EDB in question, which will always have a slave or cache copy of the information. The DSA with the master will always allow the other DSA to make these changes — the access control is implicit. This should be done at reasonable intervals to prevent ripple effects causing undue load on DSAs mastering information high in the tree.

```
-- Attributes for measuring the size of the DIT
-- Should be added to QUIPU non-leaf object


SingleLevelCount ATTRIBUTE
        WITH SYNTAX integerSyntax


SubtreeCount ATTRIBUTE
        WITH SYNTAX integerSyntax


TypedSubtreeCount ATTRIBUTE
        WITH SYNTAX SET {
                objectClass OBJECT IDENTIFIER,
                count INTEGER }
```

```
-- represent each object once (primary object class)
```

Figure 3: DIT Statistical Information

References

[Kil88] S.E. Kille. The QUIPU directory service. In *IFIP WG 6.5 Conference on Message Handling Systems and Distributed Applications*, pages 173–186. North Holland Publishing, October 1988.

[KRRT90] S.E. Kille, C.J. Robbins, M. Roe, and A. Turland. The ISO development environment: User's manual (version 6.0), January 1990. Volume 5: QUIPU.