Managing data derived from multiple sources in an X.500 Directory

Paul Barker

Department of Computer Science

University College London

Date: August 1st, 1990

ABSTRACT

X.500 Directories [1] will not often be used as the master source of data until such directories are much longer established, and the technology trusted. Until then there will remain the substantial problem of keeping an X.500 Directory up-to-date from a number of sources. Usually the volume of data will require that maintenance procedures are as automated as possible.

However, naive procedures will not suffice for a number of reasons: different sources will name the same object differently; different sources will name different objects the same; the data obtained from these sources may overlap to some extent, though one source may be regarded as more authoritative than another.

This paper draws primarily upon experience gained from piloting ECMA TR32 and X.500 directory services as part of the THORN project's Large Scale Pilot Exercise [2]. The tools developed for THORN, and the problems with them, are discussed.

It is recognised that a complete solution to these data merging and updating problems will be quite complex, and such a solution is not essayed here. Instead, this paper proposes some pragmatic solutions to the majority of the problems. It is anticipated that further understanding of the problems will result from experimentation with implementations of the tools described herein.

Managing data derived from multiple sources in an X.500 Directory

Paul Barker

Department of Computer Science

University College London

Date: August 1st, 1990

1. Introduction

X.500 Directory systems will inevitably be regarded by data administrators as prototype systems for a considerable time to come. Until those who manage data are convinced that X.500 offers them the required functionality through usable tools, X.500 systems will rely for their data on shadowing other data sources. It will be seen that trying to merge data from more than one source causes a number of problems.

A fundamental assumption which lies behind a distributed directory service such as X.500 is that de-centralised, locally managed data is better quality than centralised data. Locally managed data will often be updated more assiduously and more frequently than a centralised database. Thus, it is argued that there will be a general tendency to want to use more data sources rather than less. However, a number of problems are possible when trying to merge the data from disparate sources into a coherent whole. Often, there will be no common key between data sources to *join* two sets of information. The same object may be named differently in various data sources. Different objects may be given the same name in two sources. Some sources may be regarded as more reliably accurate for some attributes, in some namespaces.

It should be noted that the naming problems are particularly acute when trying to create entries for people. A good data source might have personal titles, forenames in full, surnames and even generational qualifiers; a poorer source may only provide a single initial and surname. Since this is such a major problem area, and also because X.500 directories are initially being used mainly for storing information about people, the particular focus of the paper is on dealing with data about people. However, notwithstanding some specifics about the naming of people, most of the arguments have general applicability.

Pilot directory services based on X.500 systems [3] [4] have now reached the point where several hundred organisations have loaded some data into the Directory. Many of these sites will have loaded data from a single source, using ad hoc methods for naming, and for loading entries into the Directory. Often, the difficulties of merging in even one additional source of data will have seemed too great without purpose built tools. The difficulty of this problem is directly related to the diversity of the data.

However, while it is difficult to solve all the problems, experience with piloting directory services suggests that many of the problems can be tackled given a fairly simple information framework and set of tools. This paper proposes some designs for such tools. It is suggested that by using such tools, experience with the panoply of data sets will indicate how the tools need to be enhanced.

2. Data from multiple sources

Unless an organisation has already centralised its information management, creating entries for people will usually involve drawing upon a number of separate sources, each used for some specific purpose. In fact, it is likely that additional sources of data will often be drawn on even where procedures exist for centralised data management.

The college where the author works poses a typical set of problems in this regard. To build up a reasonably complete entry for a person at UCL would require collating information from:

- The college's telephone system database;
- The computer centre's email database;
- The administration's personnel database;
- The library's database.

However, using these sources alone is far from satisfactory as the centralisation of the information, in some cases, leads to the information being of poor quality. For example, the telephone information is only kept accurate if departments keep the telephone system administrators advised of all changes. The computer centre's email addresses will generally be for mailboxes on the computer centre's hosts, whereas many users have mailboxes on departmental machines. In both these cases, it is likely that departments' administrative data will be more up-to-date than centrally managed data, and should be used if available in a readily processable form. Thus the number of data sources which it seems desirable to use might be as many as one per department: more than a hundred in the case of UCL!

However, a number of problems occur when trying to merge several sets of data into a coherent, single set of data. The following problems are common:

- Usually there is no common key to help to join the data together. Different sources allow the derivation of different relative distinguished names (RDNs) for the same person. For example, the UCL Computer Science departmental database might know of a "Steve Kille"; the college mail list might know of S. Kille; and the college phone list might refer to Mr. S.E. Kille.
- Two or more different sources might use the same name for different people. This will occur when sets of similar names are not disambiguated in the same way in corresponding sets of data.
- More than one source of data might provide information on, for example, email addresses. There has to be a mechanism for deciding which is the authoritative source for a particular attribute.
- Some attributes for an entry might be multi-valued, where separate values are derived from each source. Some, but not all, of these attribute values might occur in more than one source. Some attribute values might appear to be multi-valued when

in fact there is simply a representational difference between two or more sources.

3. Automation of procedures.

It is desirable that the Directory is updated from the primary sources as frequently as possible. It is thus very important that automated procedures require as little intervention as possible.

However, not all aspects of Directory maintenance have to, or even may, be automated. The naming of organisations will eventually be under the control of national naming authorities. Organisations' administrations will in turn usually wish to exercise control over the naming of their organisational units, rather than using algorithmically derived names. This will require mappings to be created between algorithmically derived names and the names that administrators decide they would like to appear in the Directory. For example, deriving a list of department names from the telephone directory at UCL might produce a name such as "Elec & Elec Eng": since longer name forms generally have more universal meaning than abbreviated forms, this name would probably be mapped onto the more suitable Directory name of "Electronics and Electrical Engineering". This naming task should not be too onerous, as the underlying organisational unit data should not change very frequently.

People are a different matter: there are a lot of us; we move around frequently; we can usually, but not always, be correctly and unambiguously identified if referred to by any of these names.

Experience with THORN showed that the automation of Directory maintenance could be considered as a set of three separate problems.

i Initial naming of entries;

ii Subsequent amendment of entries;

iii Deletion of expired entries.

The desirability of automating these tasks as much as possible is now briefly discussed.

3.1. Initial naming of entries

For many organisations, in particular academic institutions, the arrival of new people is far from uniformly distributed. At UCL, for example, the vast majority of people joining the college each year will do so in early October. Since this may be several thousand people, it is clearly desirable to be able to add these people into the Directory as automatically as possible. This surrenders the allocation of RDNs to an algorithm. Providing the source data is well structured, this should not be difficult. However, it is anticipated that this bulk loading of new data into the Directory will inevitably cause a substantial number of name clash and name matching problems initially, which will require manual intervention to resolve.

Almost all institutions will experience a steady change in personnel. While, in this instance, the volume of data does not render manual methods unworkable, consistency and accuracy are better served by automated methods.

While algorithmically deriving RDNs from source data is not of itself a difficult task, deriving the best RDN when there are a number of data sources is more complicated. It would be nice to avoid the following situation where possible:

> A directory entry is created and named from a source which contains only a single initial and surname. Some other sources offer better RDN choices having a personal title, a full list of forenames and a surname. It is thus deemed desirable to rename the entry.

Unfortunately the scenario just described is likely to be typical. Frequently the first sources of data used to load the Directory will be large, centralised sources, with less naming information than smaller de-centralised sources. As these smaller sources are drawn upon, it will be desirable to rename the entries loaded from the poorer quality sources. While it should be possible to automate much of this process under the supervision of an administrator, it might be preferable not to load entries initially with RDNs of the format "initial surname". Whether to allow the creation of entries with such short RDNs should thus be under the control of the directory administrator.

3.2. Subsequent amendment of entries

It is obviously desirable to keep the Directory as up-to-date as possible by frequent updates from the source data. In fact in the U.K., with the advent of the Data Protection Act, it is a legal requirement to keep the data accurate. Usually, the volume of source data will make it desirable to automate this task as much as possible.

In the previous section, it was shown that modifying entries correctly when they are maintained from a number of sources can be a non-trivial problem. For example, the same attribute in an entry might be sourced from two different places but with the same data represented slightly differently. One would hope to avoid the situation where the values in the directory were continually modified as the Directory was updated from different sources. To this end, it seems desirable to have some notion of preference between data sources.

3.3. Deletion of expired entries

Redundant entries must be removed from the Directory as quickly as possible. Almost nothing erodes confidence in the quality of data in the Directory as seeing a lot of data therein which is clearly out-of-date. In the U.K., the need to remove data which is no longer extant is stronger than this: it is *required* by the Data Protection Act.

It is not always clear though, using automated procedures, when to delete an entry. If an entry disappears from one source, it cannot always be inferred that the directory entry should be deleted from the Directory if that entry is drawn from many sources. One possibility is to use preference information which would indicate that deletion from a particular source should result in the deletion of the Directory entry. Another possibility is to carefully delete all the appropriate attributes as an entry disappears from each source, and only to delete the entry when it becomes *empty.* But this policy is fraught with difficulty if users are allowed to modify their own entries and to add additional attributes. The problem is not insuperable, but again requires an information framework which details which attributes are maintained by the user.

4. Experience with maintaining directory entries

In this section, the experience gained with the THORN system, and the lessons learned from it, are considered.

A simple bulk data loading and merging tool was developed, which attempted to tackle some, but not all of the problems. Each data set was converted into bulk load data. This data represented Directory entries as a set of attribute type-value pairs. The bulk data merging tool could create new entries, amend existing entries and delete old entries in a single invocation of the program. However, as before, each data maintenance task is considered separately.

4.1. Initial naming of entries

Each data set was processed such that, if possible, RDNs of the form "preferred_forename surname" were derived. This form was considered to give the most natural naming. If no entry with the chosen name was found in the Directory, a new entry was created. Since not all data sources contained forename information, or failed to indicate which one was preferred, alternative RDNs were sometimes derived, resulting in two or more entries for a person being created. This option was chosen in preference to choosing a shorter RDN, or failing to use data sets which didn't provide forenames. In the next section, a method which is believed to be a more satisfactory compromise is proposed.

4.2. Subsequent amendment of entries

If an entry with the same RDN was found in the Directory, it was read and compared, attribute by attribute, with the information in the bulk load data. If an attribute in the bulk load data was not present in the Directory, that attribute was added into the Directory. If the attribute was in the Directory but differed to the value in the bulk load file, the value in the Directory was overwritten with the value in the bulk load file.

This method has a number of deficiencies. It was not possible to delete attributes as the possibility existed that attributes in the Directory, which were not in the bulk load data, had been loaded from another data source, possibly added by the data subject him/herself.

The method only allowed for an attribute to be derived from a single data source. A different strategy could have been chosen here: add in a new value if that value was not one of the set of values in the Directory. But this causes another problem: if an attribute value changes, the new value is now included along with the old value in the Directory.

Since an attribute value replacement strategy was used, accurate data from reliable sources could be overwritten by poorer quality data. When the better data source was again used, the original values would be restored, and so on.

The only remedy to the above problems was to try to get all the values for an attribute in a particular namespace from a single source. Bulk load data for that attribute from other sources would have to filtered to exclude entries for namespaces where such duplication occurred. This was done on an ad hoc basis. A more satisfactory method is proposed in the next section.

4.3. Deletion of entries

A simple-minded attempt was made to solve this difficult problem. An option was provided to allow the deletion of Directory entries not found in the bulk load data, for a specified subtree. The method required that all the data for that subtree was presented to the bulk load program. This required the collation of all the various sources of bulk load data.

The method worked, and was used to a limited extent in practice. However it was hard to administer. To keep the volumes of data handled by the bulk loading program manageable required the excising of the appropriate data from all the separate bulk data files. Two possible methods for tackling this problem more satisfactorily are proposed in the next section.

5. A Data Maintenance algorithm

There are several major changes between the mechanisms proposed here, and those operated with the TR32 system. It is not claimed that the proposed methods solve all data maintenance problems. A review of some of the known deficiencies of what is proposed is given later. However, it is believed that the method outlined does offer a useful basis for further experimentation and evaluation. The main features of the proposed method are:

- Attention is given to the problem of creating more than one entry for the same person, with slightly different names.
- The name matching algorithms attempt to allow some looseness of matching, but refer ambiguous cases for administrator intervention.
- Information on matching names in data sources with names in the directory, where algorithmic matching is not possible or is wrong, is stored in name-mapping files.
- There is a mechanism, requiring administrator intervention, for improving the directory names of entries, if they are originally created with a less than ideal RDN. This mechanism makes use of the afore-mentioned name-mapping files.
- Information about which data sources hold the "best" source of information for an attribute or set of attributes for a particular domain is stored in the directory.
- Entry deletion can be achieved by relatively simple methods.

5.1. Load data format

The data management tools proposed in this section require that all data sources have their data converted to a common format. An example of a suggested syntax is presented in Appendix A. The suggested syntax is based heavily on the syntax used in the THORN system’s X.500 bulk load files and Quipu’s EDB files. Some syntax extensions are proposed which allow for data management rather than just data loading. Henceforth, files in this format will be referred to as DM (Data Management) files. The main features of DM file syntax are:

- The source of a set of data is included in the data as a special attribute.
- The subtree in the directory where the information should be loaded is included in the data (rather than mapping the DIT onto the file system tree, as with THORN’s bulk load files and Quipu’s EDBs).

- o Entries consist of a sequence of attribute type-value pairs, which are separated from each other by one or more blanks lines.
- o It is possible to specify that an entry or attribute value should be deleted from the Directory.

5.2. Name matching and entry creation

Non-ambiguous human name matching, given the the number of variant formats, is a non-trivial task. Some data sources will not provide more than an initial and a surname. Others might provide several initials and a surname. Further sources might provide a list of forenames and surname. It should be noted that as directory entries can potentially be created from any source, all sources must be able to supply all the "MUST CONTAIN" attributes for the object classes required for the entry. This should not be a problem as currently personal entries only mandate the commonName and surname attributes (the objectClass attribute must of course also be specified).

The key points of a naming matching algorithm are now discussed. The algorithm in pseudo-code is reproduced in Appendix B.

A few name definitions will help to clarify the following section.

- o An *algorithmic name* is a name which is derived algorithmically from a data source.
- o A *purported name* is a name which is chosen for a record in a source data set. This will often, but not always, be the *algorithmic name.*
- o A *directory name* is the RDN of an entry in the directory.

It should first be noted that name matching and the choice of names are separate issues.

5.2.1. Name matching

For name matching, it is clear that the more naming information that is provided, the better. In such cases, a reasonably high degree of confidence can be held in the decision as to whether a match is found in the directory. Equally, name matching on the basis of a purported name of a single initial and surname does not provide much confidence that the matching will always be correct. The matching algorithm proposed here takes account of this by allowing more automation of matching if longer name forms are provided, and more deferment to administrators if only short name forms are available. A mechanism will be described shortly which will minimise the need to defer to administrators.

The matching algorithm proposed is purpose-designed, matching on surname and first initial at the very least, and preferably further on forenames or subsequent initials, and personal title. Unfortunately, off-the-peg, fuzzy-matching algorithms are probably of no use in this instance. Soundex, for example, could be considered. However, soundex’s strength is in finding a number of loose matches when incomplete or incorrect names are specified, which is not the case here.

It should be noted that name matching can be attempted on more than just the directory name. Alternative common names can also be tested against purported names. However matching against the directory names has primacy as such names must be unique with respect to all sibling entries, while choice of alternative names is subject to no such restrictions.

5.2.2. Choosing RDNs for people

A consistent policy on choice of purported names greatly facilitates the name matching algorithms. The suggestions which follow depart slightly from the standard, as the author believes that the standard overloads the common name attribute, when used for naming people. The standard suggests that personal title information - e.g. Mr, Ms, etc. - and academic and professional qualifications should be included in the common name attribute. The view taken here is that personal title information and qualifications should be stored in separate attributes, and common name used solely for forenames or initials and surnames. Since personal title information is often available, a personalTitle attribute is thus proposed.

Others have noted that more structuring of personal names is desirable [5].

The suggestions for naming people are:

- The best option for the RDN will usually be a single forename and surname in the commonName attribute, and this can be regarded as the *preferred* name form. (However, it should be noted that Chinese names, for example, will often be better with three name parts. Such decisions will need to be taken by a directory administrator.)
- Personal title information should not be included in the commonName attribute, but should be stored separately in the personalTitle attribute.
- While longer name forms are not preferred for RDNs, such forms should be included in the directory as alternate forms of commonName. This enhances the approximate matching capabilities of the name matching algorithm.
- Multiple value RDNs, using personal title information, should be used to disambiguate entries with otherwise identical common names.

Alternative names should also be chosen if they add more information than that conveyed by the RDN. A number of variants are possible, and should be used if it is felt that they will help searching on the Directory. However, the name matching algorithm proposed does not rely on all the variant forms being present, as the comparisons are algorithmic rather than merely being of equivalence.

5.2.3. Minimising administrator intervention

The proposed method pivots around decisions which cannot be made without fear of error being resolved by administrator intervention. Too little intervention by an administrator will probably mean that the directory will contain duplicates and errors. On the other hand, if the requirement for intervention is too frequent, the procedures will be too time-consuming and unworkable. It is hoped that the following method strikes a satisfactory balance.

Every time a new source of data is tapped, it is anticipated that there will be a number of name matching problems which will have to be deferred to an administrator. The administrator must decide how algorithmic names should be mapped onto purported names. It is proposed that, for each source, these mappings are preserved in a file. Every time that the directory is updated from a source, the mapping file information is used, when it exists, to transform algorithmic names into purported names. Thus only new entries in the data source might require further administrator intervention.

5.2.4. Renaming entries

It is preferable if entries can be initially created with preferred name forms. Such entries are easier to match with than entries with directory names of format "initial surname". However, creating entries with a directory name in this format will not always be possible, and quite often short form algorithmic names will be used as the purported name, and corresponding directory names created. If a data source with more naming information subsequently becomes available, it will generally be desirable to rename the entries which have short form directory names. While such renaming needs to be under the control of an administrator, a suggested mapping (or choice of mappings) of existing short name to preferred name will be possible. Any renaming of entries requires changes to be propagated to all the mapping files, so that algorithmic names are mapped onto the new purported names.

5.2.5. Some examples to clarify the intention of the algorithm

The idea behind the algorithm is best explained by the following examples which reference the sample DM files in Appendix A.

The algorithm should allow for the following testing cases:

- o The creation of an entry with RDN "cn=Pat Jones", when one has already been created with RDN "cn=Peter Jones".
- o The creation of an entry with RDN "cn=Pat Jones", with an alternative common name of "cn=Pat Sarah Jones", if an entry with an RDN of "cn=P A Jones" exists.
- o The creation of an entry with RDN "cn=P Barker" so long as there no other entries where any common name value is of the form "cn=P*Barker".
- o The recognition that a purported name of "cn=P Barker" corresponds to the directory entry named "cn=Paul Barker", if there is only a single match on names of the form "P*Barker". This matching decision should be logged for ratification by the administrator, and updating of the name mapping file.
- o The creation of an entry with multi-valued RDN "cn=Pat Jones % personalTitle=Ms", if there was already an entry with an RDN of "cn=Pat Jones" but with a personalTitle attribute other than "Ms".
- o The refusal to create an entry with RDN "Paul Barker" if one already exists with RDN "P Barker", and has no disambiguating alternative names. This would be flagged for intervention, with the suggestion that "Paul Barker" was probably a preferred name form for "P Barker". The manager would presumably concur with this and sanction a modRDN operation. All mapping files would now have to be updated with the information that *"P Barker -> Paul Barker"*.
- o If a purported name of "P Jones" is given and matches several entries, the administrator will be given a list of the possible matches.

5.3. Maintaining the attributes in existing entries

We have seen already that there are a number of problems in trying to build and maintain a coherent set of attribute information for an entry, when the information is derived from multiple sources. These might be summarised as: attribute representational differences; how to favour better quality source data; deletion of attribute values. The ideas described in this section are meant to be a reasonably lightweight solution to these problems. They do not provide the ultimate solution, but they should prove workable.

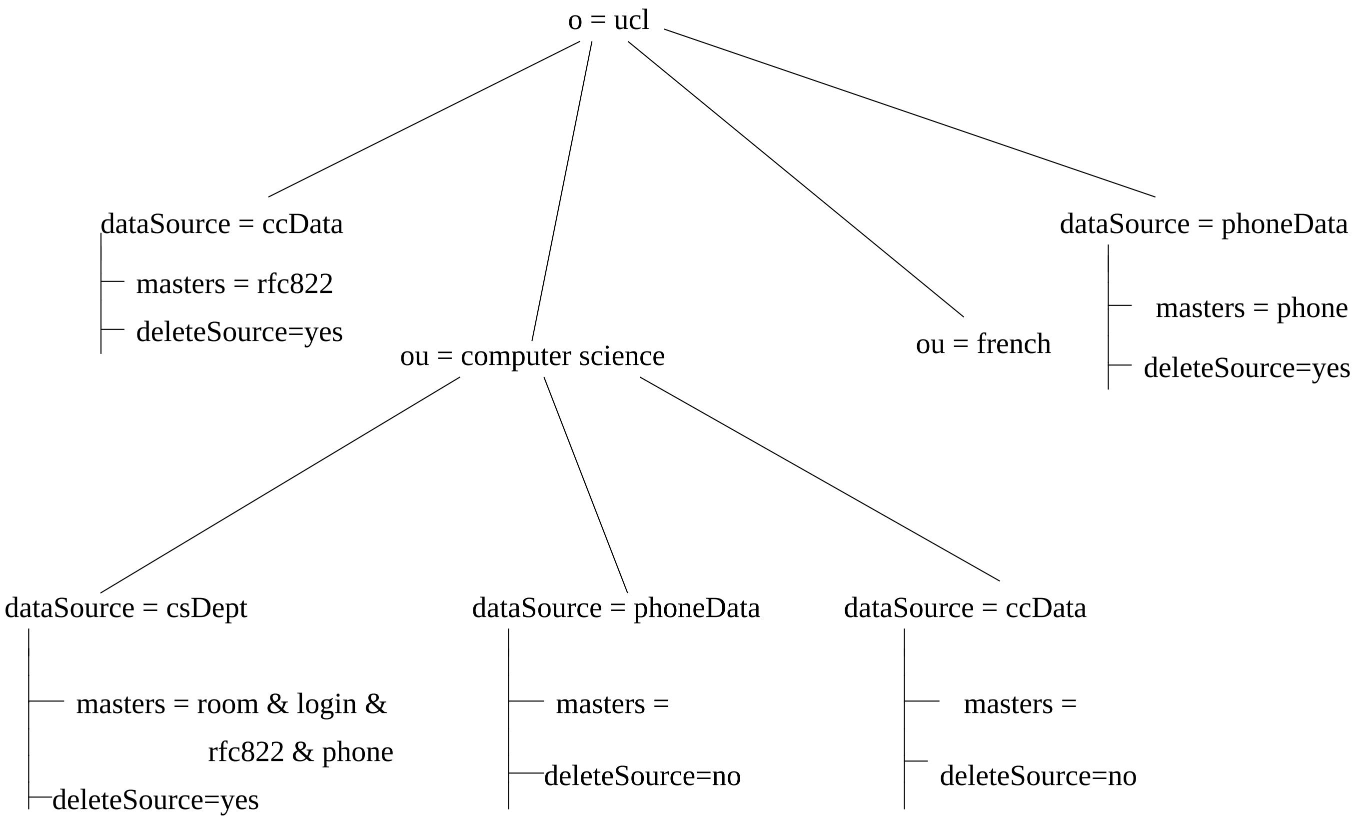
Before discussing the proposed ideas in any detail, one problem is dismissed out of hand. Representational differences have to be solved by means external to the directory. The tools which are used to process the various sources of data into the common load format must be consistent in their representation of, for example, room and telephone numbers. Any inconsistencies must be removed before the data is presented to any directory data management tools.

5.3.1. Preferring better quality data sources

It is a fundamental assumption behind this paper that, in pursuance of better quality data in the X.500 directory, there will be a tendency for administrators to want to draw on an increasing number of data sources. Frequently these will offer high quality information for some attributes in particular subtrees, and thus will be preferred over other sources. We therefore need a mechanism for describing these preferences. The natural place to store this information is in the directory itself.

The information framework is very simple. For each directory subtree, one entry per data source will contain the attributes which are *mastered* by that source. The amount of preference information can probably be kept quite small by the notion of a subordinate being able to inherit preference information from a superior if none is explicit.

The following diagram should clarify these points by example.



A possible source mastering and preference tree

This example indicates the following:

- o The "csDept" data source masters room, login, rfc822 mailbox and phone number information for the subtree "o=ucl@ou=computer science".
- o The "ccData" (computer centre) source masters rfc822 mailbox information for the subtree "o=ucl", but not for the subtree "o=ucl@ou=computer science".
- o The "phoneData" (telephone system database) source masters phone number information for the subtree "o=ucl", but not for the subtree "o=ucl@ou=computer science".
- o The subtree "o=ucl@ou=french" has no explicit source preference information, but inherits it from "o=ucl".

5.3.2. Aligning the attributes in the directory with the values in the source data

This problem is trivial if only a single source is allowed to master each attribute for a set of entries, but much more complex if multiple masters are allowed. It is not even clear at present how useful this facility is. After some consideration, it has been decided to support multiple master sources for an attribute, in a particular subtree. The information framework proposed makes it easy to represent muliple mastering, although the semantics are less straightforward. These issues are discussed shortly.

The effect of this may be summarised as follows:

- When an attribute is loaded from a master source, any attribute value(s) in the source data, which are not in the directory, are added into the directory.
- Whan an attribute is loaded from a non-master source, attribute values are optionally added, but only if there is no such attribute in the directory.
- If an attribute is not explicitly mastered by any source, the directory is updated by the first source which presents that attribute to the Directory's data management tools. While this seems somewhat arbitrary behaviour, lack of explicit mastering information implies that such arbitrariness is what is required.
- Attribute values can be deleted from the directory by means of special DM file syntax. This is discussed in the next sub-section.

5.3.3. Deletion of attribute values

The ability to discard redundant attributes and values is ultimately as important as the ability to create and amend them. Multiple sourcing of an attribute makes this process very difficult if one is restricted to comparing the values in a source data file and the values in the Directory. Values in the directory which are not in the source data might have been loaded from other data sources. One solution to this problem is to detect changes in the source data and to present these changes as part of the DM data. These differences will include both attributes to be added and to be removed.

A mechanism to perform this is fairly straightforward. First, perform a difference operation on the most recent version of the data and the previous version which was loaded into the Directory. New values can be represented as before as an *attribute type, attribute value* pair. Values which should be deleted should follow the same format but the line should commence with the *not* operator "!". See Appendix A for an example of the syntax.

While this method enables the deletion of attribute values, it has the side-effect of optimising the addition of entries to the directory, as only new attributes are presented to the directory by the management tool.

A further, and at first sight undesirable, consequence of this method of deleting attribute values is possible. Consider the following situation:

> An attribute is loaded into the Directory from source "A", which explicitly masters that attribute. Let us say that this attribute has value "abc". Source "B" which contains values for that attribute, but does not master the attribute, is subsequently processed and differences between the previous and current versions of the source "B" data are produced. These differences indicate that the attribute value "abc" should be deleted. As there is nothing in the proposed information framework to indicate that the attribute value "abc" was loaded from source "A", the value will be deleted from the Directory.

While this initially appears to be wrong, it is argued that this is acceptable behaviour. Any change in source data can usually be taken as being an indicator of data quality. To totally ignore such a change would be to waste information about the data. One of two things might now ensue:

1. Doing differences on source "A" will also result in a delete operation for the attribute value "abc". This will fail harmlessly as the value has already been

deleted.

2. The value will not have been deleted from source "A", and thus will be reloaded into the Directory when the data management tools next present source A’s data to the Directory. It should be noted that presenting diff data only will result in this anomaly not being corrected.

If this approach is regarded as unacceptable, an alternative method would be to refer such potential deletions to an administrator for sanction. Experience will decide which of these two methods is the more preferred.

5.3.4. Defects with the proposed method

The proposed method is not free of defects. First, allowing multiple sources for an attribute places reliance on DM file differences to effect deletion of attribute values. If some differences are lost or not applied, the directory and the source data can become unsynchronised. To some extent this problem can be circumvented by periodically rebuilding the X.500 database from the various sources.

Second, it is not clear how much attention needs to be given to the problem of multiple sources for a single attribute. If it proves not to be a practical problem, the proposed method can be simplified to only allow a single source. If, on the other hand, it proves to be a serious problem, then more detailed information will be required about which sources have provided which attribute values.

Third, the semantics of attribute deletion are somewhat convoluted. This stems from trying to be as flexible as possible, while only having a lightweight information framework.

It is not clear at the moment how serious the problems just mentioned are in practice. The approach needs to be reviewed following some experimentation.

5.4. Deleting entries

Some complex entry deletion strategies can be envisaged. It might be decided that an entry should be deleted from the directory when all of a named set of attributes had been removed from the directory by attribute alignment mechanisms. While such an approach might be feasible, it will often be slow to achieve the desired effect as some data sources will only be updated infrequently.

Thus two simpler methods are suggested. First, the disappearance of an entry from a source can be detected by using the difference mechanisms described above. The decision to actually delete the entry should only be made if that data source is categorised as a *delete* source. Otherwise, the disappearance of the entry should be optionally flagged to the system administrator.

Another technique can be used if difference data is not used. In this case, all extant entries will have to be read as part of the data aligning process, and could be timestamped. An entry could now be deleted if it had not been timestamped for a specified period.

References

[1] X.500, *The Directory - overview of concepts, models and services,* CCITT /ISO IS 9594

[2] S.E. Kille, *The THORN Large Scale Pilot Exercise,* pp 143-145 in Computer Networks and ISDN Systems 16 (1988), published by North Holland.

[3] J. Craigie, *UK Academic Community Directory Service Pilot Project*, pp. 305-310 in Computer Networks and ISDN Systems 17 (1989), published by North Holland.

[4] M.T. Rose, *NYSERNet White Pages Pilot Project: Status Report*, in NYSERNet report no. 89--12--31--1, Dec 1989

[5] S.E. Kille, *Personal communications with the author*

APPENDIX A

1. Example data from telephone directory

DM file for UCL CS Dept created with information extracted from UCL’s telephone directory

```
# this is a comment line
# The next line identifies the data source entry
source= c=gb@o=university college london@dataSource=uclPhoneData

# the next line indicates where in the DIT the data should be loaded
rootedAt= c=gb@o=university college london@ou=computer science

cn= P Barker
telephoneNumber= +44 71-380-7366
surname = Barker

cn= Peter Jones
# add values +44 71-380-1234 and +44 71-380-2345 and remove +44 71-380-3456
telephoneNumber= +44 71-380-1234
telephoneNumber= +44 71-380-2345
!telephoneNumber= +44 71-380-3456
surname= jones

cn= Pat Jones
telephoneNumber= +44 71-380-4567
surname= jones

# delete this entry (only if deletion criteria are met)
# as found in this file
!cn= Fred Barker
!telephoneNumber= 1212
!surname= Barker
```

2. Example data from departmental database

DM file for UCL CS Dept created with information extracted from the departmental database.

source= c=gb@o=university college london@dataSource=csDeptData

rootedAt= c=gb@o=university college london@ou=computer science

cn= Paul Barker
telephoneNumber= +44 71-380-7366
roomNumber= G21
surname= Barker

etc.

3. Example of a data source entry

Data source record for the UCL CS Dept data

dataSource= csDeptData
masters= room
masters= login
masters= rfc822
masters= telephoneNumber

APPENDIX B

The name matching algorithm

```
/* matchName returns one of the following codes

   EXACT          The purported name (algorithmically derived or, if
                  applicable, from a mapping file) exactly matched the
                  RDN of an entry.
   ONEMATCH       Using all the naming information provided, and comparing
                  it with all that in the directory, one match was found
   NOMATCH        No matches found, exact or approximate
   SEVERALMATCHES Several matches found, either exact matches on alternative
                  names or approximate matches.
   NEEDMULTRDN    There is already an entry with an RDN which matches
                  the purported name, but this entry has a different
                  personal title attribute
   MAPPINGFAILED  The algorithmically derived name has a mapping entry, but
                  this does not correspond to any entry in the directory
   MODRDN         The purported name appears to match an entry a single
                  entry in the Directory, but is of a more preferred form
                  than the existing RDN (e.g. forename and surname rather
                  than initial and surname).  Suggest modification of RDN

   and in the case of EXACT, ONEMATCH and NEEDMULTRDN, also returns the
   purportedName matched.
*/

matchName(dataSource, personalTitle, forenames, surname, purportedName):matchResult
{
   /* forenames may only be initials */
   algoName = makeAlgoName(forenames, surname)
   purportedName = mapName(dataSource, algoName)

   /* purportedName will be in the form "cn=Fred Bloggs"
      thus search will be for exact matches only */
   listRDNs = search(purportedName)
   if ((noMatches = countRDNs(listRDNs)) == 1)
   {
      x = head(listRDNs)
      if (purportName is RDN)
         if (x.personalTitle == ""  || personalTitle == "" ||
                                 x.personalTitle == personalTitle)
            return EXACT
         else
         {
            /* concoct multi-valued RDN using commonName and
               PersonalTitle */
```

```
            purportedName = purportedName "%personalTitle" personalTitle
            return NEEDMULTRDN
        }
}

if (algoName != purportedName)
{
    managerLog(dataSource, "mapping failed from algorithmic name:",
                    algoName, " to ", purportedName)
    return MAPPINGFAILED /* error in mapping file */
}

if (noMatches == 1) /* 1 exact match, but not on RDN */
{
    x = head(listRDNs)
    if (x.personalTitle == ""  || personalTitle == "" ||
                            x.personalTitle == personalTitle)
    {
        purportName = x.cn
        managerLog(dataSource, "Update mapping file(?):, algoName,
                                        purportedName)
        return ONEMATCH
    }
    else
        return NOMATCH
}

/* no exact matches found on preferred name form - try and match on
   name of form "cn=Initial*Surname" */
init = firstInitial(forenames)
listRDNs = search("cn=" firstInitial "*" surname)

if ((noMatches = countRDNs(listRDNs)) == 0)
    return NOMATCH
else
{
    /* from the list of entries which matched the filter cn=Init*Surname,
       select those where initials and forenames potentially match
       e.g. "A B Smith" could match "Arnold B Smith", but
            "A B Smith couldn't match "Andy John Smith" */
    listRDNs = initForenameCheck(listRDNs, forenames, surname,
                                        personalTitle)
    noMatches = countRDNs(listRDNs)
    if (noMatches == 1)
    {
        x = head(listRDNs)
        if (x.cn == firstInitial " " surname)
```

```
        {
          managerLog(dataSource, "ModRDN:", firstInitial " " surname,
                                                purportedName)
          return MODRDN
        }
        else
        {
          purportedName = x.cn
          managerLog(dataSource, "Update mapping file(?):, algoName,
                                              purportedName)
          return ONEMATCH
        }
      }
      else
        if (noMatches > 1)
        {
          for x in listRDNs do
            managerLog(dataSource, "Ambiguous:" x.cn, purportedName)
          return SEVERALMATCHES
        }
        else
          return NOMATCH
}
```