NSF Middleware Initiative

internet2-mace-dir-groups-best-practices-200210.html

Tom Barton

University of Memphis

October 2002

Comments to: nmi-support@nsf-middleware.org

Development of this document was supported with funding from the University of Memphis, Internet2, and the NSF Middleware Initiative (Cooperative Agreement No. ANI-0123937).

# Practices in Directory Groups

## Abstract

Experiments and early experiences with the facilitation of authorization in applications and the facilitation of group messaging with the use of directory services in institutions of higher education were surveyed. Several concepts, good practices, open issues, and a few principles extracted from this are presented.

This document is a product of the Internet2 Middleware Initiative, and the Middleware Architecture Committee for Education (MACE), published under the auspices of the National Science Foundation Middleware Initiative (NMI). Internet2 is a member of the Enterprise and Desktop and Integration Technologies (EDIT) Consortium, participating in the NMI.

This is one of a growing set of documents created by MACE, detailing various issues surrounding the planning, deployment, configuration, maintenance, and security of enterprise directories.

For additional information and related topics and resources see the following sites:

| | |
|---|---|
| Internet2 Middleware Initiative: | http://middleware.internet2.edu/ |
| MACE: | http://middleware.internet2.edu/MACE/ |
| EDIT: | http://www.nmi-edit.org/ |
| NMI: | http://www.nsf-middleware.org/ |

# Table of Contents

1.Introduction

2. Concepts and definitions

2.1. Types of aggregates

2.2. Group representations

2.3. Group update circumstances

2.4. Access use cases

2.4.1. Is X in groupA?

2.4.2. List all groups of which X is a member.

2.4.3. List all members of groupA.

2.4.4. Boolean combinations of groups.

2.4.5. Group scoping.

3. Managing directory groups

3.1. Apology: lack of standardization of directory ACLs

3.2. Types of groups

3.3. Group names & namespace overloading

3.4. Maintenance & indexing of membership attributes

3.5. Management of delegated groups

3.6. Personal groups

3.7. Maintaining referential integrity

3.8. Group math

3.9. Privacy and visibility of groups

3.10. Forward references

3.11. Aging groups

4.  Using directory groups

4.1. Directory access controls

4.1.1. Application access to the directory

4.1.2. Helpdesk View of the the directory

4.2. White pages

4.3. Course management, portal, and application server systems

4.4. Group scheduling

4.5. Group messaging

4.6. Dialup & wireless authorization

4.7. Unix group maps

4.8. Referencing groups in applications

4.9. Don’t slurp!

5. Acknowledgements

6. References

7. Appendices

7.1. *memberOf* Algorithm

8. Contact Information

# 1. Introduction

Many higher education institutions have extended and adapted their core IT infrastructures to provide enterprise directory services. Following practices exemplified by the Internet2 Early Harvest Best Practices in Identification and Authentication [1] and the LDAP Recipe [2], their experiences have demonstrated the significant value afforded by use of this type of architecture (cf. Internet2 Early Adopters Generic Middleware Business Case [3]). It is expected that a similar value will be obtained by extending this architecture to supply authorization and attribute services, so that the data for access-control policies need not be independently provisioned within each application and service platform.

This document focuses on the representation of and access to information needed to implement an infrastructure service to support automatic facilitation of access control processes and group messaging. The details of particular models for role- or relationship-based access-control systems are not directly considered here. Instead, this document explores the institutional-level groups deployment as a basis for enterprise access-control and group messaging architectures.

There are several ways of representing group information, no one of which is innately better than another. Which representations are implemented in an enterprise directory depends on

- how the group information will be maintained,
- how it will be most commonly accessed, and
- how potential interactions arise between the type of representation, the nature of the group (such as size and privacy requirements), and capabilities of the particular directory service agent (DSA) being used.

The document is organized into three main sections.

- Section 2 introduces several concepts and terms to enhance readability in the rest of the paper.
- Section 3 addresses several aspects of managing directory groups.
- Section 4 briefly offer examples applications that make use of directory groups, intended to stimulate thought on what may be possible rather than to be a comprehensive catalog.

Terms being defined are underlined. Special technical terms, such as names of attributes and objectclasses, are rendered in *italics* to improve readability.

# 2. Concepts and definitions

The notion of a "group" is so basic and common to so many different contexts that it can be difficult to communicate effectively about this subject, unless the terms and scope are defined for a particular context. This section will introduce and define terms and identify several related algorithms and concepts that together frame the technical matters under consideration in this document.

It is convenient to organize these topics into four categories. The first one covers terms relating to types of aggregates such as *group, role,* and *affiliation*. The second category contains terms used to describe the representation of the membership of a group, including *static, dynamic, forward reference,* and *spatial*. The third category identifies terms used to classify groups according to how they are maintained. These are *automated, manual, enterprise, delegated, joinable,* and *personal*. The fourth category lists the types of queries that applications may use when accessing information concerning groups. Taken as a whole the four categories address the basic concept of a group, how they are staged in an enterprise directory, how such things are maintained, and how they may be accessed. These are all essential to understanding the use and management of group information.

## 2.1. Types of aggregates

Here the term group is used to refer to any means of representing a collection of objects in an X.500 directory. It is important to distinguish between a group and a group object, which is an object in an X.500 directory whose schema is designed to express information about a group. Use of a group object is one way to represent a group. Others will be discussed in section 2.2 (Group Representation).

The chief characteristic of a group is its membership, i.e., the set of objects that belong to the group. The member objects may be of any type. Most often we are concerned with groups whose members are either *person* objects or other group objects.

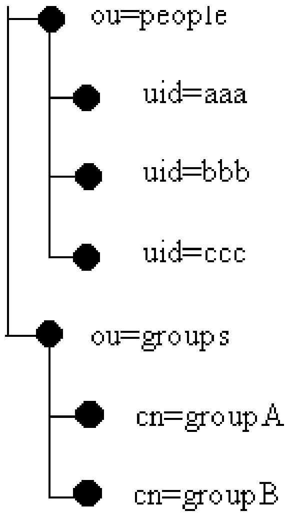
The term role refers to a way of identifying a set of entitlements. Entitlements are meant to translate into actions mediated by other applications. For the purposes of this document they can remain an abstract concept. *Person* and other types of objects are associated with one or more roles through a variety of means. The collection of these roles, the objects that are assigned them, and the entitlements they are associated with form a database that may be used by a role-based access-control system. A discussion of the design of role-based access-control systems is beyond the scope of this document. The term is defined here to distinguish it from a group. Likewise, the term relation, which figures in relationship based access-control systems, should be distinguished from the usage of "group" in this document.

An affiliation is a special case of a relation between a person and an organization. The organization involved may be implicit - that whose directory service holds affiliation information about the person. The types of such relationships are sometimes only locally meaningful, although the *eduPersonAffiliation* controlled vocabulary defines several affiliation values in use across much of higher education, including *student, faculty, staff, alum,* and *member* (cf. [4]). Affiliations will often identify courses in which a person is a student or an instructor, programs to which they belong, departments in which they are employed, status of alumni involvement, or perhaps just their expression of interest in applying for admission or visiting the campus.

## 2.2. Group representations

A static group is one represented by a group object that contains a multi-valued attribute listing the member objects. That attribute is called its membership attribute. The *groupOfNames* and *groupOfUniqueNames* objectclasses defined in X.521 ([5] and [6]) specify types of static group objects with membership attributes *member* and *uniqueMember*, respectively.

**Figure 1**



To illustrate a static group, consider the example directory information tree (DIT) in Figure 1. The object *cn=groupA , ou=groups, dc=foo, dc=edu* could be the static group defined by the following LDIF (cf. [7]):

```
dn: cn=groupA,ou=groups,dc=foo,dc=edu
cn: groupA
member: uid=bbb,ou=people,dc=foo,dc=edu
member: uid=ccc,ou=people,dc=foo,dc=edu
```

A group is dynamic if its membership is expressed by evaluating an ldap url (cf. [8]). A corresponding group object may, but does not have to, exist. Typically, the values of one or more attributes maintained in *person* objects are used to define a group dynamically. In this case, the scope descriptors in the defining ldap url specify a search of the DIT subtree containing *person* objects for everyone. An application will then execute the defining ldap url to determine the group members at that moment. If a corresponding group object exists, it should have a dynamic-membership attribute whose value is the ldap url that defines member objects. For example, iPlanet's *groupOfURLs* objectclass uses the *memberURL* attribute to store the defining ldap url. A dynamic group without a corresponding group object might be thought of as existing only in an application that executes the defining ldap url, but still is considered as a representation of a group within the directory. For example, Apache's mod_ldap module uses *require group* and *require filter* directives to enable information in an ldap directory to be used for access control decisions. The first form checks membership in static groups, while the second form checks membership in dynamic groups.

Let’s refer again to Figure 1 to help illustrate a dynamic group. Suppose that the *person* objects under *ou=people* are partially defined by

```
dn: uid=aaa,ou=people,dc=foo,dc=edu
ou: student
-
dn: uid=bbb,ou=people,dc=foo,dc=edu
ou: staff
-
dn: uid=ccc,ou=people,dc=foo,dc=edu
ou: student
ou: staff
```

Then the following ldap url dynamically defines the group of people who are *staff*:

```
ldap://ldap.foo.edu:389/ou=people,dc=foo,dc=edu?dn?2?(ou=staff)
```

Forward referencing is a practice in which an object's group memberships are listed as the values of an attribute of the object. This attribute (and each of its values) is called a forward reference. For example, an attribute named *isMemberOf* might be used to list the groups of which the object is a member. Typically, a forward reference identifies actual group objects to which the member object belongs. The group objects themselves may be static or dynamic. It should be noted that a forward reference could also be used merely to tag members of groups without there being a corresponding group object. Such tags might also be used to associate an object with other types of objects such as roles. Below in section 3.10: (Forward references), we will consider how to implement forward referencing in greater detail.

An example of forward referencing is given by the following partial definitions of the *person* objects in Fig. 1:

```
dn: uid=aaa,ou=people,dc=foo,dc=edu
isMemberOf: groupB
-
dn: uid=bbb,ou=people,dc=foo,dc=edu
isMemberOf: groupA
-
dn: uid=ccc,ou=people,dc=foo,dc=edu
isMemberOf: groupA
isMemberOf: groupB
```

Those *person* objects with a *groupA* value of their *isMemberOf* attributes in fact belong to the static group named *groupA* in the example of a static group above.

A group is considered spatial if its membership is inferred from the location in the DIT. We will also use the term spatial more generally to refer to characteristics stemming from an object’s location within the DIT.

Referring once again to Figure 1, an example of a spatial group is the set of all objects immediately subordinate to the *ou=people organizationalUnit* object.

**Synthesis**

The basic distinction between static and dynamic groups is how an object’s membership in the group is determined. For static groups, an equality search of the group’s membership attribute determines the object’s membership. For dynamic groups, an ldap query scoped to the object itself determines membership. Forward references are a particular implementation of dynamic groups. Spatial groups are the original means of representing groups and still the best choice for certain purposes. They too are special cases of dynamic groups, since all objects in a given node of the DIT can be selected by an appropriate ldap url.

## 2.3. Group update circumstances

Much of the information represented in a directory is derived from external authoritative sources. The method of updating group membership information may constrain how various groups are represented in the DIT. It may also influence the nature of ACLs and other items controlling update access to them. For example, most colleges probably prefer that their payroll and registrar's offices do not maintain each other's core business systems. In planning how groups are to be maintained, consideration must be given to who or what may modify which information for which groups. The choice of a type of group representation, the spatial location of a group object, and the nature of an ACL mediating access to a group all affect this aspect of directory design.

Such design decisions may be better informed if one recognizes the circumstances under which the update process occurs. These circumstances include deciding whether the process is automatic or manual; and if enterprise directory administrators manage it or delegate it to others with fewer directory-update privileges. It is probably not wise to have both automated and manual processes routinely maintain information in the same location (such as an *isMemberOf* attribute). Also, certain attributes of *person*, group, or other types of objects should not be modifiable by (potential) delegates. The adjectives automated, manual, and delegated, then, reflect the way a given group is updated.

A personal group is one that is created and maintained by end users. Personal groups chiefly are a convenience provided to end users, permitting them to make autonomous use of certain enterprise infrastructure services for purposes not necessarily related to the institution's mission.

A group is joinable if end users are permitted to add or remove themselves from the group's membership list.

Finally, institutional refers to groups that aren't personal.

## 2.4. Access use cases

In this section, the basic group-information access-use cases will be presented.The way a group's information is most often accessed can strongly influence how it should be represented in the DIT, the nature of ACLs controlling read access to that information, and the performance tuning of the DSAs servicing the ldap access requests. Even the number of directory replicas maintained to serve applications could be affected. The implications of access will be addressed in subsequent sections of this document.

### 2.4.1. Is X in groupA?

This basic membership question is treated differently for dynamic and static groups. For static groups, a (*membershipAttributeName*=DN) query is scoped to the group object *groupA*. For dynamic groups, the search filter defining *groupA* is scoped to the object X. In both cases, the membership query response is positive if the ldap query returns a nonempty selection. In the dynamic case, this by itself is sufficient. In the static case, this is further complicated by the possibility that *groupA* may contain other groups as members (subgroups). This can be resolved by substituting the query 2.4.2 List all groups of which X is a member instead.

Examples of these follow. For these assume X has

```
dn:uid=aaa,ou=people,dc=foo,dc=edu
```

the static group *groupA* is a *groupOfUniqueNames* with

```
dn:cn=groupA,ou=groups,dc=foo,dc=edu
```

and the dynamic group *groupB* is given by the ldap url

```
ldap://ldap.foo.edu:389/ou=people,dc=foo,dc=edu?dn?2?(ou=staff)
```

Then the following two ldap urls respectively determine whether X is a member of the static group *groupA* or a member of the dynamic group *groupB*:

```
ldap://ldap.foo.edu:389/cn=groupA,ou=groups,dc=foo,dc=edu?dn?0?
(uniqueMember=uid=aaa,ou=people,dc=foo,dc=edu)
```

```
ldap://ldap.some.edu:389/uid=aaa,ou=people,dc=foo,dc=edu?dn?0?(ou=staff)
```

### 2.4.2. List all groups of which X is a member.

An efficient algorithm, outlined in Appendix 7.1: *memberOf* Algorithm, can be used to find all static groups to which X belongs. The algorithm functions in a way that avoids recursively searching for memberships through subgroups of static groups. Use of forward referencing (cf. section 3.10: Forward references) also trivializes the DSA's processing of this query in exchange for the effort needed to maintain forward references.

### 2.4.3. List all members of groupA.

As with the question *is X in groupA?*, the constructed ldap query to list all members of groupA must be phrased differently depending on whether A is static or dynamic. Statement 2.4.3 is also ambiguous when considering that groups may have subgroups. Does it ask to list all values of *groupA*'s membership attribute, or to list all objects belonging either to *groupA* or to any group having *groupA* as a supergroup? Its meaning must be clarified in each application context.

Using the details of the example under *Is X in groupA* above, the membership of the static group *groupA* is given by

```
ldap://ldap.foo.edu:389/cn=groupA,ou=groups,dc=foo,dc=edu?uniqueMember?
0?(objectclass=*)
```

while the membership of the dynamic group *groupB* is given by

```
ldap://ldap.foo.edu:389/ou=people,dc=foo,dc=edu?dn?2?(ou=staff)
```

### 2.4.4. Boolean combinations of groups.

It is likely that not all groups of significant value to applications and users will be provisioned within the directory. Unions, intersections, and relative complements (all members of A not belonging to B) of existing groups may need to be addressed to satisfy some applications. See section 3.8: (Group math) for a discussion of handling this in a standard way.

### 2.4.5. Group scoping.

Queries of the form "which objects in groupA satisfy a given search filter" are analogous to the queries formulated using ldap urls, except that whereas an ldap url scopes a search spatially using search base and search scope parameters, these queries scope the search to a group. That's group scoping.

Conceptually, this use case is like executing an ldap url whose scope specification is replaced by the membership of a group. Since it is not possible to scope an ldap search to the membership of a static group, using an ldap url to implement a group scoped query can only be considered for dynamic groups. In this case a group scoped query can be formulated with an ldap url whose scope is a subtree of the one in the ldap url defining the dynamic group, and whose search filter matches the search filter in the ldap url defining the dynamic group ANDed with the desired search filter.

The group scoping problem for static groups can be overcome by reliance on forward references, since that method effectively tags all objects with their group memberships with no spatial dependence whatsoever. For example, assuming that the *isMemberOf* attribute is used for forward referencing and that *groupA* is a valid forward reference value, the following ldap url might be used to determine which members of *groupA* are static-group objects:

```
ldap://ldap.foo.edu:389/dc=foo,dc=edu?dn?2?(&(isMemberOf=groupA)(|
(objectclass=groupOfUniqueNames)(objectclass=groupOfNames)))
```

# 3. Managing directory groups

Automation of the provisioning of authorized and/or customized access to web content and applications, and group messaging, provide motivation to implement large numbers of directory groups. Collected here are several principles and practices pertinent to the design of any groups implementation.

This document does not attempt to contemplate the various technical and non-technical implementation problems associated with gathering data from source systems and using that data to populate and maintain static or dynamic group information in a directory. Neither does it posit any particular directory architecture. A forthcoming "Group Implementers Guide" being discussed in the MACE-Dir working group should address the down and dirty issues confronting the project manager of a groups implementation project. Also see "Metadirectory Practices for the Enterprise Directory in Higher Education" [11] for a discussion of directory provisioning.

## 3.1. Apology: lack of standardization of directory ACLs

The use of access controls internal to the directory is critical to the design. Unfortunately, at present there are neither standard ways of referring to these nor standard methods of implementation in the leading directory products. As a result, non-trivial directory designs cannot be vendor independent. Switching enterprise DSA technologies is likely to cause disruption to many directory-enabled enterprise IT services.

Assertions in this document about ACL capabilities are based principally on experience with the ACL implemented in iPlanet's DSA and secondarily on ACL capabilities implemented in openLDAP. The writer assumes that the ACL capabilities of all other leading DSA products are at least as powerful and expressive as these.

## 3.2. Types of groups

Authorization is the determination of whether a given authenticated entity is is permitted to perform some requested action. Customization is the adaptation of the user interface depending on the affiliation(s) of the authenticated user and is typically used by course management systems and portals. The information required by both drives many requirements for directory groups (note that personalization, which is the adaptation of the user interface depending on preferences selected by the authenticated user, does not rely on group information). That information may be obtained from traditional core business systems such as human resources, financial records, student information, alumni information, physical facilities inventory, scheduling, databases of computer account information, and a variety of *ad hoc* sources.

Three categories of groups determined from information in core business systems are:

- *Enterprise groups,* whichincludes faculty, staff, employees, various categories of employees as defined by the payroll system, students, alumni, and various categories of computer-account holders.

- *Departmental groups,* which includes faculty, staff, or graduate assistants differentiated by department, school, college, or other organizational division; and department heads (and above) differentiated by organizational division.

- *Academic groups,* which includes students in each academic program, college, or level; and students (and instructors) in each course section (per term) and in each course (for courses with multiple sections).

Some groups deriving from *ad hoc* institutional sources are:

- *Application-specific groups,* which accommodate the authorization or customization needs of an enterprise application or service and are maintained in the enterprise directory rather than using application-specific mechanisms. These provide a standard means of delegating the management of enterprise applications and services through the maintenance of their application-specific groups. Examples include groups used to control access to stored database queries, access to web or application-server mediated processes or content, or access to network-access services, the access-control information for which is not stored in core business systems.

- *Activity-specific groups,* which are similar to departmental groups for activities not modeled in core business systems. Examples include student activity groups and those facilitating work-team activities.

Typically, update of personal, application, and activity specific groups is delegated. A chief distinction between personal and the other two types is that of institutional value. Application and activity specific groups may need to continue to exist even after the particular humans who maintain them leave the institution, as opposed to personal groups (cf. section 3.6) which, under good practice, cease to exist when the individual leaves the institution.

How these groups should be represented and how their membership information flows to applications is determined by considering several factors:

- How and by whom each group will be maintained.

- The types of application access use cases each group will most typically need to satisfy.

- ACLs within the directory.

- DSA constraints such as processor limits or group size limits.

The remaining topics in this section should help to illuminate these considerations.

## 3.3. Group names & namespace overloading

Standards-based person and group objectclasses, and potentially others, require the *cn* attribute. All such objects contribute to *cn* indices maintained by the DSA and, unless specifically excluded by ACLs or scoping of ldap urls, are potential selectees of an ldap search referencing the *cn* attribute. Such searches are a very common type of search conducted against many enterprise directories. A mature groups implementation may produce a population of group objects in the directory even larger than the set of *person* objects. This means that ordinary white pages searches will tend to have lots of group and *person* objects returned together. That makes it harder for humans to find what they're looking for, and can result in pressure to relax DSA search limits. For this reason, there is a need to be concerned about the values assigned to *cn* across all objects which possess that attribute.

Each institution will likely have its own version of a naming plan for groups, but a trend is emerging in which group names are structured with a prefix indicative of its origin or scope followed by a more arbitrary name presumably meaningful to group members, maintainers, or users. The group prefix selection algorithm tends to follow a classification of groups along the lines illustrated in section 3.2: (Types of groups).

For example, enterprise groups might uniformly start with the prefix "All" or "University", such as "All Faculty" or "University Enrollment Management Task Force". The prefix of a departmental group stems from the name, possibly hierarchical, of the organizational element, such as "A&S Chemistry Graduate Assistants".

Similar prefixes are suggested by other categories in your groups taxonomy. Personal groups might be named starting with "Personal".

The group naming convention at The University of Memphis is reproduced here for consideration:

```
AreaLabel [ SubAreaLabel ] DescriptivePhrase
```

"AreaLabel" is the name of the high level area most closely affiliated with the group's membership or purpose, chosen from a list of labels for the Divisions, Colleges, MajorProgramCodes, Courses, and a set of special labels ("All", "University", "Personal").

The term "All" is applied to groups that span any finer division of the university.

The term "University" is used as a prefix for groups representing university-wide committees, roles, or other activities whose name is fixed.

Divisions are IS, B&F, Provost, Athletics, M&A, StudentAffairs, President.

Colleges are ACU, AS, CFA, EDU, ENG, FBE, GRA, LAW, NUR, and UC.

The "Course" is the 4-digit course number or the 7-digit course+section number.

"MajorProgramCode" is one of about 100 4-character program codes for the various academic majors defined in SIS.

"Personal" would signify a group created by an end-user, should we ever enable that. This value is the one exception to the rule that the AreaLabel reflects the group's membership. In this case it reflects the group's pedigree, needed I think to keep the name space for "administrative groups" separate from the namespace for end-user created groups.

"DescriptivePhrase" is essentially what the group might be named independently of the AreaLabel, the "real common name". Or it might, in conjunction with the AreaLabel, be a completion of the "real common name".

A general principal is to prefix the name with a label identifying the group's "area" or scope. Sometimes it is meaningful for there to be one or more additional scopings. For example, a group established for a given department in a given college. That's what "SubAreaLabel" is for - a more specific scoping added to the prefix of the "real group name".

Some applications, unrelated to *cn* indexing, may require that names of groups they reference be subject to additional constraints. For example, names of group objects of the *posixGroup* objectclass, used to represent Unix groups in ldap directories, must comply with group naming requirements for unix operating systems.

## 3.4. Maintenance & indexing of membership attributes

Membership attributes for static groups such as *member* and *uniqueMember* should be equality indexed to speed searches.

As membership for a static group changes, unless the change is a substantial fraction of the overall membership, the values for these attributes should be incrementally maintained rather than replaced, i.e., deprecated values should deleted and new values added. For example, the following ldif fragment removes one member and adds two new ones to the membership list of a group:

```
changetype: modify
delete: uniquemember
uniquemember: uid=uid1,ou=people,dc=foo,dc=edu
-
add: uniquemember
uniquemember: uid=uid2,ou=people,dc=foo,dc=edu
uniquemember: uid=uid3,ou=people,dc=foo,dc=edu
```

## 3.5. Management of delegated groups

Some organizations find that certain aspects of enterprise IT provisioning are best managed outside of the central IT organization. For example, the most expert knowledge and experience with institutional financial information is likely to be found outside of IT. Such experts may be drawn on to write and load certain stored queries into an enterprise decision support system. The maintenance of who is permitted such access may be delegated to officials outside of IT who are familiar with persons possessing that expertise.

Such circumstances may be supported by a design in which directory chosen delegates administer groups associated with the operation of an enterprise facility. A group management ACL (cf. [2]) in the directory together with a directory enabled web application for managing attributes in group objects provide sufficient infrastructure to support this.

The group management ACL grants users listed in the *owner* attribute of a group object management privileges over the group. Among the types of information to consider granting management privilege to are membership, ownership (to enable subdelegation), descriptive information, and certain mail related information if the group is mail enabled. The *cn* attribute should not be modifiable by owners of delegated groups so that group naming conventions can be upheld.

## 3.6. Personal groups

Personal groups enable end-users to facilitate authorization in their own web sites, particularly if those reside on centrally provided web-hosting servers. They also provide a means for taking advantage of any directory enabled group messaging facilities that may be available.

Such groups require special consideration with regard to spatial location, naming, and aging. They need to go away when their creator does, and most *cn* substring searches should not select these. In particular, there is a need to ensure that no personal group will ever be selected by an institutional process, e.g., for making an access control decision in an institutional application or for sending a message to the members of an institutional group.

Two different approaches to personal groups surfaced among the responses to the group practices survey, although none could be said to be both a mature and fully satisfactory implementation. The types of implementation considerations raised include:

- DIT location. Locate personal group objects under their creator's *person* object as in

`cn=uid=aUid:MyGroup,ou=groups,uid=aUid,ou=people,dc=foo,dc=edu`

or locate all personal group objects in a branch of the DIT designated for this purpose, as in

`cn=uid=aUid:MyGroup,ou=personalGroups,dc=foo,dc=edu`.

The former enforces a requirement that subordinate personal group objects to be removed before a person's *person* object can be deleted. The latter relies on a periodic external process to remove groups whose *owner* no longer exists in the directory.

- Enforced naming convention. If the *cn* attribute will be used on personal groups, in particular if personal groups will be static and use or extend the *groupOfNames* or *groupOfUniqueNames* objectclasses, then there is a namespace issue to contend with (cf. section 3.3: Group names & namespace overloading). Using a name stem such as "uid=aUid:" ensures that institutional groups and personal groups can always be distinguished by their names. To enforce a naming convention, end users must not be enabled to directly create their own personal groups. Instead, a privileged process must create and name the group and then delegate management of it to the end user as described in section 3.5: (Management of delegated groups).

## 3.7. Maintaining referential integrity

As objects that may be members of static groups leave the directory, the groups of which they are members need to have their membership attributes updated accordingly. This is an example of referential integrity (RI): each DN value of a membership attribute should refer to an object that actually exists. Note that referential integrity is an issue only for static groups.

There is no standard that embraces a means of maintaining referential integrity in an ldap directory. Novell's, iPlanet's, Microsoft's, and possibly other, directory products each include a proprietary means of maintaining referential integrity.  If either the vendor changes the nature of their referential integrity facility or if a directory implementation includes custom objectclasses then relying on the vendor for referential integrity may not work either.

To remain in control of this issue, an external referential integrity process will need to be constructed (aka "RI dialysis machine"). An ldif dump of the directory is periodically processed to produce a set of changes needed to maintain referential integrity, in accord with the advice in section 3.4: (Maintenance & indexing of membership attributes).

Further requirements for an RI process will be mentioned below in sections 3.9: (Privacy and visibility of groups) and 3.10: (Forward references).

## 3.8. Group math

A successful initial deployment of groups may bring further needs to light. Among these is the need to construct new groups from institutional groups already being maintained. In some cases extending the set of automated groups will best accommodate this, but some needs are better met by delegating the authority to create new combinations. For example, a department or individual may wish to grant access to a resource to all members of an institutional group except those on a short list they maintain. Depending on the application involved, it may be easy, difficult, impossible, or otherwise undesirable to accomplish this in the application, if it's not done in the directory.

The basic set theoretic operations of union, intersection, and complement used to describe combinations of groups don't have good counterparts in either standard X.500 directory objects or in ldap. One might represent a union as a new group whose members are the group objects being unioned, but there is no general representation of the complement of a static group or of an intersection of static groups.

The problem of representing combinations of groups becomes more tractable for groups represented by forward references. In this case the boolean operations in the search filter of an ldap url can be applied to values of forward reference attributes to determine if a given object belongs to any specified combination of groups, or to list the membership of a combination of groups, depending on how the ldap url is scoped. For example, if an attribute named *isMemberOf* is used for forward referencing, then the ldap search filter

```
(&(isMemberOf=groupA)(!(isMemberOf=groupB)))
```

selects all objects in the search scope belonging to groupA but not to groupB. If the search is scoped to a specific object, the filter selects the object if and only if it belongs to the specified combination of groups. If the search is scoped to the entire DIT, the complete membership of that combination of groups is selected, provided that DSA search limits are not exceeded.

Forward referencing permits the representation of group combinations. This reduces the problem into two sub problems: maintaining a forward reference attribute, and inserting appropriate ldap queries somewhere, if not in the application. Let’s look at how to attack these sub problems.

Like any attribute, a given forward reference attribute should only be maintained by a single source to avoid the possibility of one maintenance procedure undermining the work of another. One way to deal with this is to rely on an external, periodically run process to automatically maintain a forward reference attribute. This process is closely related to maintaining referential integrity of group membership attributes and might be conveniently accomplished by the same program. This is discussed in section 3.10: (Forward references). It is also worth noting that iPlanet Directory Server v5.x includes a "virtual" forward reference attribute (named *role*) on all objects whose value is calculated by the DSA dynamically when it is read. This can be used instead only if you are willing to rely on proprietary mechanisms and if privacy requirements do not preclude it.

The second sub problem generally is outside the scope of this document. However, a directory enabled web application can be created that uses a forward reference attribute to enable end-users (or any other group deemed appropriate) to create their own directory resident groups out of combinations of previously existing ones. Probably the most challenging task in designing this application is determining how to represent the set of groups available to users that they may wish to combine.

## 3.9. Privacy and visibility of groups

Different organizations will have different requirements with regard to how widely disseminated knowledge of the existence of a group or the membership of a group is permitted to be. Some leave all groups and their memberships visible while some others locate group objects spatially according to a visibility policy implemented using ACLs on containing *ou*’s. Below are descriptions of two, more sophisticated, approaches to this contributed by group practices survey respondents.

Virginia Polytechnic Institute (VPI) designed their group implementation with visibility as a requirement. VPI group members have a viewability attribute. Chad La Joie of VPI summarizes its use:

```
A user will have the option of setting their group visibility, defined as the
visible attribute in the member object, with one of three values. A value of
"public" means the user may be seen in any public listing of the group's
membership. A value of "group" means the person will only be visible to other
people in the group. A value of "private" means the person may never be displayed
in a group membership listing. It should be noted that a value of "private" does
not disallow services requiring this information to see group listings; it just
```

means they cannot display the list. This is to protect people who are in groups that may be targeted for certain types of crude or bigotous behavior.

Brendan Bellina of Notre Dame describes another approach to provisioning privacy for static groups which depends on a custom auxiliary objectclass being applied to a standard group object:

A group may be private or public. Viewing access to the membership list of public groups may be open (unrestricted) or closed (restricted). The group type (private, public-open, public-closed) is recorded in the ndGroupType attribute. The membership lists of private groups are always viewable only to the group owner(s). For all group types the owner of the entry can see all group attributes including group common name (cn), description, group membership, and applications that group members are authorized to by virtue of group membership (listed in the ndAuthEligible attribute). Through the group administration utility a group owner can update its membership list, and name (authorized applications must be updated by the directory administrator to prevent admins from granting access to directory administration utilities) (utility yet to be written).

A private group's existence is hidden from anonymous access. A member of a private group can view the common name (cn), description, and owner of groups for which he/she is a member. In order to view the names of the groups that a person is a member of the person would need to bind and search all groups for the existence of ndGroupMember and return cn.

The ndGroupMember for a group is only visible to members of the group. This prevents a group member from using access to the uniquemember attribute to determine the names of other members of the group. The problem solved by the ndGroupMember attribute is described in the email snippet below by Jeremy McCarty of the ND OIT Infrastructure Services group:

> Since we'll have public groups with hidden (restricted) membership and public groups with public (unrestricted) membership and hidden (private) groups, you cannot just search for all groups and assume those returned are ones you can see because you are a member. You can do this for hidden groups only. You can see your own dn in public groups with public membership, but for public groups with hidden membership you'd need to be able to search uniquemember. The problem with allowing searching against uniquemember for hidden membership is that you can search for another dn (not just your own) and get back groups for that dn that the two share in common (are both members of). I can't easily think of a way to limit this yet other than to have a certain attribute (ndGroupMember) that would only be visible to members for searching (not anonymously). Then you could search on this attribute as well or use its appearance in a group listing to determine your membership.

The name, description, and owner of public groups, whether open or closed, are visible to anonymous searching. This will allow non-members to search by name or description for groups of possible interest and contact the group owner to register. The ndGroupMember attribute is visible only to group members, allowing a person to request a list of public groups to which they belong.

If a group has an open membership list then its uniquemember attribute can be viewed by anonymous search. If the membership list is closed, then the uniquemember attribute is accessible only to the group owner(s).

Note that neither of the preceding two methods encompasses use of forward referencing, nor is it clear how they might be extended to accommodate it. But see the next section for more information.

## 3.10. Forward references

Certain issues raised elsewhere in this document can be resolved by use of forward references: chasing down membership in subgroups (section 2.4.3: List all members of groupA.), scoping a search to a group's membership (section 2.4.5: Group scoping.), representing set theoretic combinations of groups (section 3.8: Group math), and DN independent referencing of groups in applications (section 4.8: Referencing groups in applications). This approach does not seem to have drawn much attention among directory designers in higher education, and it is hoped that they will consider using this technique. It is also hoped that DSA designers will give thought to implementing better ways to support its use as well as addressing its main weakness – a difficulty in expressing privacy and visibility policy for groups represented as forward references.

If it were possible to completely avoid use of static group objects and only use forward reference attributes to designate group membership, then hypothetically the only aspect of forward reference management that would remain is determining the ACLs that should pertain to forward reference attributes. It is unlikely that many directory deployments can avoid use of static groups, because many off the shelf directory enabled applications require them. Management of forward references is partly a referential integrity type problem. As static group memberships change, corresponding changes to member objects' forward reference attributes must also occur, and vice-versa.

Processes that provision directory groups should be designed or extended to maintain forward referential integrity. Unless such processes are the only way group membership can be impacted, which is unlikely to either be the case or remain that way for long, maintaining forward referential integrity must be a primary task of some piece of directory related technology.

There are two alternatives for maintaining forward referential integrity: capture relevant changes to the directory and complement them as needed for referential integrity, or periodically run an external process that examines an ldif dump of the directory to make the updates needed to maintain this integrity. The first might be done in real time using methods proprietary to the DSA (e.g., iPlanet v5.x's virtual *role* attribute; using a plug-in API), by embedding the DSA within a suitable metadirectory architecture, or periodically by parsing DSA change logs appropriately. The second approach can be done in a manner independent of DSA particulars.

All supergroups of a forward referenced group should also be referenced in a forward reference attribute to facilitate authorization and messaging applications. An object may have forward reference values for supergroups of a group of which it is explicitly a member without necessarily being listed in the membership attribute of any of the supergroups. So an algorithm that maintains forward referential integrity must take care to add the object to a group's membership attribute only when appropriate. One way to accomplish this, is to run the memberOf algorithm (Appendix 7.1) for the object and compare the results with the values of its forward reference attributes. Then add the object to the membership attribute of only those groups present in the forward reference list but absent from the *memberOf* result.

The value syntax for a forward reference attribute must also be chosen carefully. Values might be either abstract labels or DNs. The latter is possible only if group objects are always instantiated. A DN value syntax is fragile with regard to the location of group objects within the directory. An abstract label syntax avoids this problem and removes the necessity of instantiating a group object, thereby enabling forward referencing of dynamic groups as well. When creating forward references to group objects it also becomes necessary to give target group objects an attribute whose value is this label, so that forward referenced group objects can be found with an equality search of group objects having that attribute. It is sufficient to use the *cn* attribute of standard group objects as the value of forward reference attributes. However, it may be desirable to extend the schema of group objects to include an attribute specifically for the purpose of providing a target for forward reference values.

A forward referencing design is complicated by privacy requirements. It is not clear that current ACL implementations are sufficiently expressive to provide access to a forward reference attribute in strict accord with a particular directory privacy policy. To address this one might use ACLs to limit access to forward reference attributes only to "blessed" applications, perhaps using a "read only service DNs" approach as described in section 4.1: Directory access controls.

## 3.11. Aging groups

Groups not being automatically maintained by directory administrators, i.e., delegated and personal groups, may eventually become unmanaged and unused and should be removed to reduce namespace overuse and to avoid potential accidental misuse. Two models for doing this have emerged. A "keep alive" model is one in which group objects have attributes containing an expiration date and contact information. An external process runs periodically to notify contacts of groups past their expiration date and gives them an interval in which to renew the group. Groups past expiration date plus a grace interval are removed. This approach depends upon two things: groups being created by a tool that always sets the expiration date and gathers contact information; and a mechanism to do renewals.

Another approach is an "inactivity timer" model. Each group has a last modify timestamp and contact information associated with it. A periodic external process identifies all group objects whose last modify time is older than a set period of time, the inactivity timeout. One or more notifications are given to group contacts during a grace interval, at the end of which groups are deleted. Removal of a group is avoided by someone "touching" it, causing an update to its last modified timestamp. Some DSAs automatically supply last modified timestamps for all objects. For others, a custom process, either external to the directory or an extension to the DSA, will need to maintain a last modified attribute. In the case of an external process this might be done by referring to appropriate DSA log files.

The contact information may be held in an attribute designed for this purpose. Alternatively, the DN valued *owner* attribute of standard static groups may also serve as a pointer to contact information residing in the object(s) it references.

The length of grace periods, default lifetimes for the keepalive model, and inactivity periods for the inactivity timeout model, are purely local preference. A good practice is simply to employ some means consistently for grooming of stale groups.

# 4. Using directory groups

This section contains brief descriptions of some particular and general types of applications that use directory groups. It is hoped that these will stimulate thought concerning some of the good, bad, and debatable practices chronicled here.

## 4.1. Directory access controls

Perhaps the most prevalent and earliest use of groups is to control access to objects and attributes within the directory itself. The LDAP Recipe [2] included a section addressing this, which might also account for its prevalence. Two best practices of this sort have surfaced since the LDAP Recipe was published:

### 4.1.1. Application access to the directory

Access to some attributes of *person* and other types of objects must be controlled in order to satisfy needs for privacy or confidentiality. ACLs must deny anonymous viewing of these attributes. If any are needed by a directory enabled application in order to provide service to people (and in most organizations this will be the case), it is necessary for that application to bind to the directory in a non-anonymous way. The alternative is to deny service to people requiring protection of their private directory data.

Directory enabled applications should each be assigned an object to bind to the directory as. This is called a *service DN*. Access to attributes and objects that are required by some directory enabled applications and are not anonymously viewable, should be made accessible to members of groups defined for this purpose. ACLs referencing these groups specify the additional access privileges to be granted to member service DNs. Each service DN should be made a member of all such groups needed by its application in order to provide service.

Some schools lump all such needs into one group, a "read only service DNs" group. This group provides read only access to the entire directory to member service DNs. The other extreme is to define one group for each cluster of attributes and/or objects for which some application needs special access, and to make each service DN a member of the smallest set of such groups needed for it to function. Local circumstances will determine which model is chosen. The best practice is simply to use the service DNs approach to controlling application access to the directory.

There are further operational benefits that stem from implementing service DNs, but discussion of these is beyond the scope of this document.

### 4.1.2. Helpdesk view of the directory

Helpdesk and other personnel offering end-user technical support need access to information to help them identify causes for common service issues. A group should be defined whose members are permitted read-only access to attributes needed to fulfill the support function. The use of subgroups in this circumstance is very likely. The subgroups will themselves contain DNs of different sets of people who have this common need (e.g., central IT helpdesk staff, College helpdesk staff, etc.). This group and its subgroups will likely be delegated.

Note that it may be necessary to provide FERPA training to individuals with a helpdesk view of the directory. Also, integration between a helpdesk operation and the enterprise directory is a fertile area – many possibilities exist beyond merely providing helpdesk operators with access to directory resident information to facilitate their jobs.

## 4.2. White pages

White pages is usually the first deployed directory-enabled application, and is used to display contact and affiliation information about people. Departmental affiliation data should be added to *person* objects to facilitate white pages applications. Using the *ou* attribute has the benefit of supporting some off the shelf clients that look there for what they deem "departmental" affiliation information. It should be either substring or approximately (aka soundex) indexed to facilitate searching since ordinary people might not know exact forms of official department names. General affiliation status, such as "faculty", "staff", "student", "member", "alum", etc, may also be advantageously located in the *ou* attribute and used to constrain white pages searches. With respect to students, programs of study and other academic affiliation information should *not* be added to the values of any publicly viewable attribute to assist in maintaining a FERPA compliant directory.

Two models for how to determine departmental affiliation values for each person are: (1) mine it out of financial and payroll data, and (2) have independent business processes track it. Since financial and payroll data exists for other purposes, the latter is probably best, but it may be that a campus must start with or settle for the former. It can happen that, if affiliation data mined from financial systems are insufficiently accurate in expressing actual organizational affiliation, pressure builds until new business processes are implemented to gather more accurate data. An in-depth discussion of these models is beyond the scope of this document.

How best to represent departmental affiliation information for applications other than white pages is an interesting question for which insufficient experience could be found on which to base recommendations for this document.

## 4.3. Course management, portal, and application server systems

Some application servers can be configured to reference an enterprise directory service for groups (e.g., J2EE). Others may have their own groups store in which some groups are integrated with the enterprise directory by one-way synchronization processes.

Affiliation information is most critical to customizing a portal. Like application servers, some integrate with an enterprise directory service and some provide their own customization mechanisms. In the latter case, implementers will be required to develop custom processes to integrate the portal into the enterprise infrastructure.

Course management systems are similar, with an additional requirement for representing specific roles such as instructor, developer, and student for each section. As of this writing, directory integrated course management systems implement proprietary group-like objectclasses for courses. The IMS standard (cf. [9]) describes how course related information should be transmitted between systems, but no standard yet exists for the representation of courses in an ldap directory.

## 4.4. Group scheduling

Several popular enterprise-scale ldap enabled calendar products are available that can use directory resident groups for group scheduling (e.g., invite the members of a group to a meeting). One caveat: watch out for where it looks for (and stores) groups. Several undesirable effects can occur, all examples of multitasking of data (generally a bad practice). End-users may be distracted by groups extraneous to scheduling purposes that may also appear there. Or there may be institutional groups there with varying privacy or visibility requirements, and the calendar service agent might not be able to honor those policies completely.

If possible, you may want to reconfigure the location used by the calendar for its groups so there is a part of the DIT dedicated solely for scheduling-related groups. If this is not possible, and if non-calendar groups can't be relocated elsewhere, consider running the calendar against an application-specific directory if your institutional metadirectory capabilities can support it.

## 4.5. Group messaging

Several of the institutions responding to the group practices survey provide a directory integrated group messaging facility using either iPlanet Messaging Server or "maildap", a program included with the openLDAP distribution. Both of these products require that groups to which messages can be sent contain attributes from an auxiliary objectclass that describes various details of how such email is to be processed. Both also have the capability to use credentials learned via SMTP AUTH to determine if a submitter is authorized to send a message to a given group. Both support use of iPlanet's *mailGroup* objectclass for mail enabling a group, although maildap can be configured to use alternatives as well.

Both static and dynamic group objects can be extended with *mailGroup* attributes, which make it very easy to add group messaging to the set of directory enabled applications. For example, if *person* objects are already maintained with attributes containing enterprise, departmental, or academic affiliation information, then a dynamic group object can be quickly created that permits messaging to a group dynamically defined by such attributes without having to first create metadirectory processes that maintain the membership attribute of corresponding static group objects. Of course, DSA search limits on general purpose replicas will likely require that a special purpose DSA replica be set up without search limits to support this use. This replica will need to be protected by access controls to inhibit trolling of the directory.

On the other hand, traditional Mail List Manager (MLM) products such as listserv, sympa, listproc, and majordomo (to name a few) implement mature designs for distributing mail to groups of recipients. They have sophisticated mechanisms for allied purposes such as bounce detection & other subtleties of SMTP handling of large distribution lists, archives, subscription, moderation, web-based administration, etc. Although sympa now incorporates some ldap directory integration, to date most MLM products lack integration with enterprise directories in several potentially valuable ways: subscription lists, authentication and access control in the administrative interface, and access control in list submission.

It is highly desirable to have something approximating a marriage of the above types of facilities. This marriage would create the ability to use mature MLM technology to send a message to a group maintained in the directory so that the group need not be maintained in two places separately or so that a custom integration process need not be created to synchronize it to the MLM database. There are some obstacles to directory enabling this function. MLM subscription lists are typically joinable (and leavable) by end users, whereas many directory groups may not be. This leads to what may become a commonplace group math problem: how to configure an MLM list so that it distributes to the membership of an institutional group (the basis for the list's existence), except that members of one manual group should be omitted and members of another manual group should be included. For example, an office may wish to send messages to all faculty except for a select few who wish to opt out, and to also include a set of non-faculty in the distribution.

Integration of the MLM administrative functions with the directory should produce the type of benefit typical of directory enabling an application. Changes to the directory would automatically provision users’ access to MLM administrative functions enabling authorized end users to autonomously create and manage mail lists. Likewise, incorporating a directory enabled access control procedure at the point of message submission supports the model of one-stop provisioning in the directory. In this case SMTP AUTH, an identity cert embedded in the message, or other means must be found to pass the submitter’s identity through the SMTP message submission point to the listserve software for an access control decision.

Happy marriages between directories and MLM products might be likelier to occur if objectclasses associated with MLM style applications become standardized. But no significant work in this area appears to have been undertaken since University of Michigan did the work underlying maildap and the IETF LASER BOF produced their final draft [10], which influenced how current directory enabled messaging products use the directory.

## 4.6. Dialup & wireless authorization

Modem pools and, increasingly, wireless access servers or wireless gateways, can refer both authentication and authorization to external services by means of RADIUS or other similar protocols. This enables an organization to integrate network access services (NAS) into the enterprise infrastructure by deploying directory enabled NAS authentication and authorization servers. Modem pools traditionally have different availability and session characteristics depending on affiliation information such as faculty, staff, student, and sometimes departmental affiliation. Similarly, wireless "zones", i.e., sets of access servers or gateways, may be provided to only serve specified groups, for example, the members of a given department or members of a manual group created for the purpose of granting or limiting access to a given wireless zone. Should the need arise, a NAS related attribute can also be added to *person* objects which can be set by authorized persons to administratively deny access to one or more NAS services.

The University of Memphis has implemented a solution of this type that relies on a custom objectclass used to define a NAS access control policy. The objectclass and its use in a RADIUS mediated authentication and authorization exchange are detailed:

```
objectclass uofmnas
   superior top
   requires
   cn,
   nasaccesstype,
   nasname,
   nasprofile
   allows
   nasadmitfilter
```

```
cn: name of this UofMNAS object.
```

```
nasAccessType: one of  "open", "closed", "userFiltered", "groupFiltered". An open
NAS service admits all authenticated users. One that’s closed admits nobody. User
and group filtered types rely on LDAP search filters to express access control
policy – either in terms of attributes in authenticated user’s object or in terms
of group memberships.
```

```
nasName: list of strings identifying this NAS service, typically names of NASes
providing this NAS service. Used by RADIUS server to find the UofMNAS object that
applies to a given session initiation request.
```

```
nasProfile: name of RADIUS server-based profile to apply to authorized sessions.
```

nasAdmitFilter: arbitrary LDAP search filter to be inserted into LDAP URL by the RADIUS server to determine if authenticated user is permitted access to this NAS service. If nasAccessType is "userFiltered", LDAP URL is scoped to the DN of the authenticated user. If it’s "groupFiltered", search scope is entire LDAP directory, but restricted to groupOfUniqueNames objects.

Attribute added to uMemphisPerson objectclass:

radiusDeny: "all" or list of nasNames for which to deny access.

Each NAS server sends its name in each RADIUS request packet in the "NameOfNAS" field. It is bound to a UofMNAS object expressing its access control policy by having that name listed among the values of the nasName attribute. Dialup NASes are further identified by using the "theNumberCalled" field from the RADIUS request packet. It is appended to NameOfNAS, after a hyphen. So, a virtual modem pool is identified as "NameOfNAS-theNumberCalled", i.e., that string appears in the nasName attribute of the UofMNAS object expressing the applicable access control policy.

The LDAP config script for the SteelBelted RADIUS server entails a variant sequence of six LDAP query steps, summarized as follows.

*Step 1 – Basic authentication*. Search for object with uid=UserName, but also require object to either be "active status" or a "guest" at UoM, and ensure that the object’s radiusDeny attribute doesn’t prohibit access to the NAS service being accessed.

*Step 2 – Handle open NASes*. If nasAccessType is open, finish authentication process.

*Step 3 – Setup for userFiltered access type*. Obtain nasAdmitFilter for NAS object if nasAccessType is userFiltered.

*Step 4 - Apply user related access filter.* Apply nasAdmitFilter to user object to see if user meets access control policy. Reject if not, finish authentication process if so.

*Step 5 – Setup for groupFiltered access type.* Obtain nasAdmitFilter for NAS object if nasAccessType is groupFiltered.

*Step 6 – Apply group related access filter.* Apply nasAdmitFilter, augmented with an "AND objectclass=groupOfUniqueNames" clause, to entire LDAP directory to see if user meets access control policy. Reject if not, finish authentication process if so.

Examples of use include

1. All UoM People modem pool. Open to all active UoM people and official guests.

```
nasName: gelion-6783600
nas Name: sirion-6783600
nasAccessType: open
```

2. Faculty/Staff modem pool. Open only to faculty and staff, which status is automatically maintained in people objects’ "ou" attribute.

```
nasName: gelion-6784600
nasName: sirion-6784600
nasAccessType: userFiltered
nasAdmitFilter: ou=Faculty/Staff
```

```
3. Typical residence hall wireless access, for which the default policy is that
only residents may access.

nasName: WS109-A1
nasName: WSLBY-A1
nasName: WS127-A1
nasName: WS207-A1
nasName: WS221-A1
nasName: WS231-A1
nasName: WS309-A1
nasName: WS323-A1
nasName: WS337-A1
nasAccessType: groupFiltered
nasAdmitFilter: (|(cn=West Hall Residents)(cn=Network Services))
```

## 4.7. Unix group maps

Although no first hand experiences were reported in the survey responses collected for this document, as of the time of this writing, certain unixes using the Naming Services Switch, including Solaris and Linux, can resolve unix group names and memberships (and other standard "maps" such as *passwd*, *shadow*, *services*, *network*, and *netgroups* ) using an ldap directory. Groups in the *group* map are instances of the *posixGroup* objectclass, which is a static group. Users belonging to such groups also have their *person* objects extended to include the *posixAccount* and *shadowAccount* objectclasses.

## 4.8. Referencing groups in applications

Use of DNs outside of the directory itself is a fragile practice that breaks whenever objects may need to be moved around within the DIT. It is better to use a layer of indirection and rely on a spatially invariant reference to a group. With standard group objectclasses, that most often will take the form of a *cn*= search for objects with appropriate objectclass. For example,

```
(&(cn=groupA)(|(objectclass=groupOfUniqueNames)
(objectclass=groupOfNames)))
```

will select *groupA* wherever it may be located.

On the other hand, some common applications (like Apache, for example) use an ACL syntax that requires use of DNs to specify a static group. A forward reference attribute can be used instead to circumvent use of DNs in such cases. For example, rather than

```
require "cn=groupA,ou=groups,dc=foo,dc=edu"
```

one can use

```
require filter "(isMemberOf=groupA)"
```

## 4.9. Don’t slurp!

Slurping is an application practice of retrieving all members of each group being referenced. A few applications may require this, but most should take advantage of other application access use cases (cf. section 2.4) better suited to whatever the application’s function is. Slurping can cause a performance hit on the DSA and also cause directory designers to worry about where directory information may be cached so long that it becomes stale or is exposed to inappropriate release.

# 5. Acknowledgements

The experience (and patience!) of many people is represented in this document. Special thanks are due to the participants in the Internet2 Middleware Initiative's MACE-Dir-groups working group, to the members of the NMI Integration Testbed, to Internet2 staffers Renee Frost, Ellen Vaughan, and Lisa Hogeboom for millipedes of legwork, and to Ann West and Jeanette Fielden for bravely attempting to ungarble the original form of this document. All errors, misrepresentations, and opaque manners of expression are solely the author's.

This work was supported in part by the NSF Middleware Initiative - NSF 02-028.

# 6. References

[1] Early Harvest Technical Workshop, "Identifiers, Authentication, and Directories: Best Practices for Higher Education", 9 May 2000.

http://middleware.internet2.edu/docs/internet2-mi-best-practices-00.html

[2] Michael Gettes, "A Recipe for Configuring and Operating LDAP Directories",

http://middleware.internet2.edu/dir/docs/ldap-recipe.htm

[3] Early Adopters Generic Middleware Business Case, http://middleware.internet2.edu/earlyadopters/draft-internet2-ea-mw-business-case-00.pdf

[4] "eduPerson 1.0 Specification", http://www.educause.edu/netatedu/groups/pki/eduperson/spec.pdf

[5] "The Directory: Selected Object Classes", ITU-T Recommendation X.521, 4th edition, 1999

[6] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", RFC 2256, December 1997.

[7] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000.

[8] Howes, T., Smith, M., "The LDAP URL Format", RFC 2255, December 1997.

[9] "IMS Enterprise Specification", http://www.imsglobal.org/enterprise/index.cfm.

[10] Lachman, Hans and Shapiro , Gregory Neil , "LDAP Schema for Intranet Mail Routing", draft-lachman-laser-ldap-mail-routing-02, no formal archival home but still floating about the Internet.

[11] Jones, Richard et al., "Metadirectory Practices for Enterprise Directories in Higher Education", http://middleware.internet2.edu/dir/metadirectories

# 7. Appendices

## 7.1. *memberOf* Algorithm

Assume for the moment that only static groups are employed and that DN's of both people and groups may appear in group membership attributes. There is a fairly straightforward algorithm for determining the set of all static groups of which a given *givenDN* is a member.

A1: Initialize the set *memberships* and the set *newGroups* to the set of all static groups in which *givenDN* is a member with an ldap search selecting groups in which *givenDN* appears in any membership attribute.

A2: Find all groups to which any group in *newGroups* belongs with an ldap search selecting all groups in which any newGroup DN appears in any membership attribute. Replace *newGroups* with those elements of the selected set of groups that do not appear in *memberships*, then add all elements of *newGroups* to *memberships*.

A3: Repeat A2 until *newGroups* is empty.

The resulting *memberships* is all groups to which *givenDN* belongs directly, together with all supergroups of those groups to any number of levels.

The maximum number of iterations in the above algorithm is less or equal to the depth of group nesting plus the length of the longest cycle of nested groups in the search scope. The algorithm is pretty efficient if group membership attributes are equality indexed. It suffers from a possible interaction with the DSA's search limit, i.e., if you've lots of groups you'd want to ensure that these searches are not constrained by a search limit.

The algorithm avoids the problem of trying to determine if *givenDN* belongs to a given group by referring first of all to the membership list of the given group, and trying to chase that down through possible subgroups. The algorithm does provide more information than might be needed (i.e., you know whether or not *givenDN* belongs to the given group, but you can also answer any other group membership question for *givenDN*).

The algorithm doesn't particularly care which finite set of static group objectclasses are used - they simply appear as ANDed *objectClass*= clauses in the ldap search filter.

This algorithm is similar to what is used by the iPlanet v4.X web server in evaluating an ACL containing a group reference.

# 8. Contact Information

Tom Barton

University of Memphis

Email: tbarton@memphis.edu