syntlogo

Design of a Directory Information Tree

giovanni.baruzzi@syntlogo.de

Why do we need a good tree design?

syntlogo

- Once deployed, a DIT is difficult to change.
- The Design has to be possibly simple to understand, just because we are going to talk about it with every customer (developers and application designers).
- Complex things are an impediment to further development

Requirements for the user's objects

syntlogo

Schema is beyond the our scope, but we can state some simple facts:

- A very good start for the user object is the class "inetOrgPerson"
- For containers use "organizationalUnit" or "organization"
- The groups should be "groupOfNames" or "groupOfUniqueNames"
- You may have to define one or two auxiliary classes.

Requirements for the applications

syntlogo

- **Applications with no integration to a Directory**
  - Very difficult to manage: use metadirectory techniques
- **Applications that can use a common user management and integrate authentication**
  - Very simple requirements if uid and password policy can be used
  - sometime you can talk to developers
- **Applications with full integration of user management, authentication and authorizazion, at least for roles**
  - This is usual for new application, when you can enforce your framework

Requirements for User's authentication

- Choose a simple password policy
- ensure that the passowrd and the password policy are encorfed IN THE DIRECTORY and not from someone else like a portal application

Requirements for the directory access

synlogo

- **Design you acces control before you need it:**
  - Classify your attributes in sensitivity classes
  - determine the access need from management applications and user's application
  - define the ACL's along your sensitivity classes, containers and access need
  - Define groups for the various access modes
  - Assign your users (service accounts) to the above groups

Requirements for the security

synlogo

- Avoid using the „root“ for access
- consider to split the root password between more people
- Define an administrator group with full rights
- Assign account to this group
- define an auditors group

Our simple Design

- **Think minimalist**
- **Think as you would define a file system and directories below that**
- **You can always define a new container**
- **move objects around may be later difficult**
  - you may have already defined groups (DN!)
  - you may already have users which grew up used to your structure

Design of a DIT: as flat as possible, as deep as needed

syntlogo

- Here your architectural skills and creativity are needed: there is no absolute rule
- use a hierarchical structure everytime it makes sense
- but avoid unnecessary containers

Design of a DIT: as flat as possible, as deep as needed

syntlogo

When you do need a different container?

- to keep together logically bound or homogeneous objects
- to simplify access control (ACI)
- to avoid name collisions (RDN)
- to keep the design clean & tidy
- comfortable browsing of the information
- to allow replication of different part of the DIT

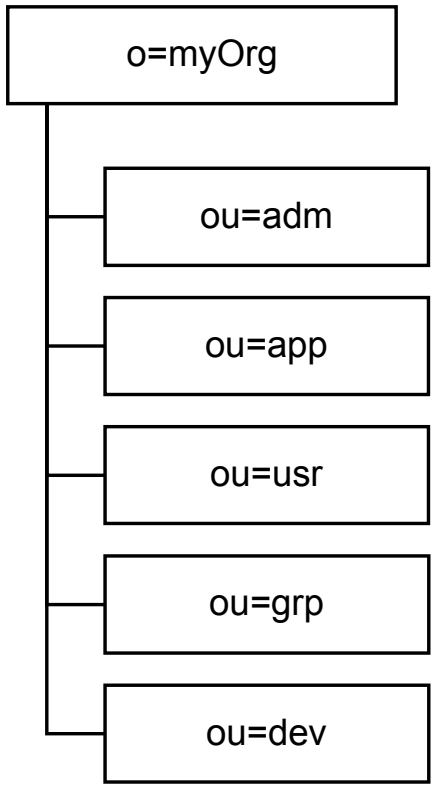
Design of a DIT: as flat as possible, as deep as needed

syntlogo

When you do *not* need a different container?

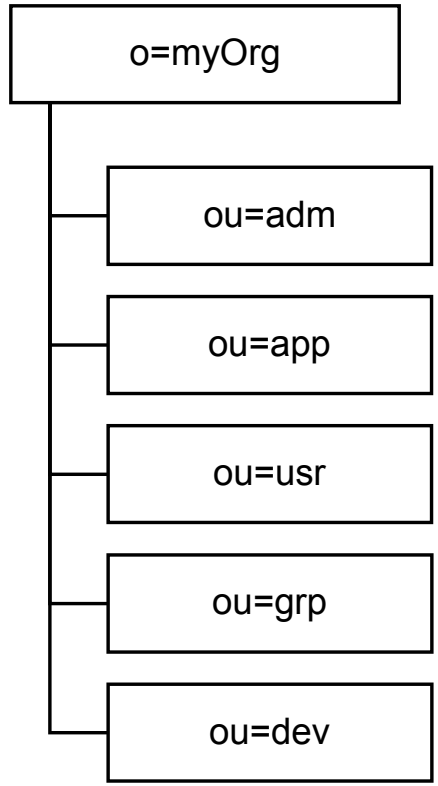
- If the number of objects is small (e.g. less than 10.000)
- if the functional scope is similar (e.g. people of the same organization, authorizations groups for applications, technical accounts)
- If the selection can be made through an attribute value (use filter, acl)

Our simple design



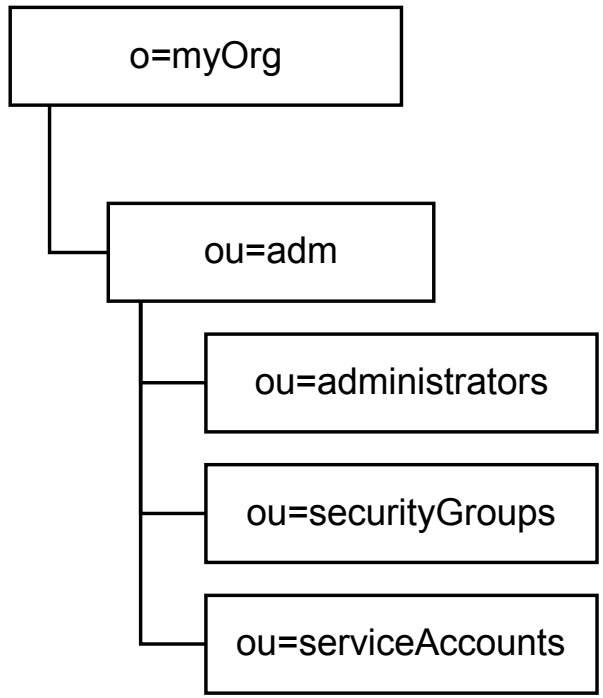
The Root Level

- Avoid long names for the root container. Remember: you have to carry them around everywhere
- My preferred choice is simply “o=<your organization’s name>”



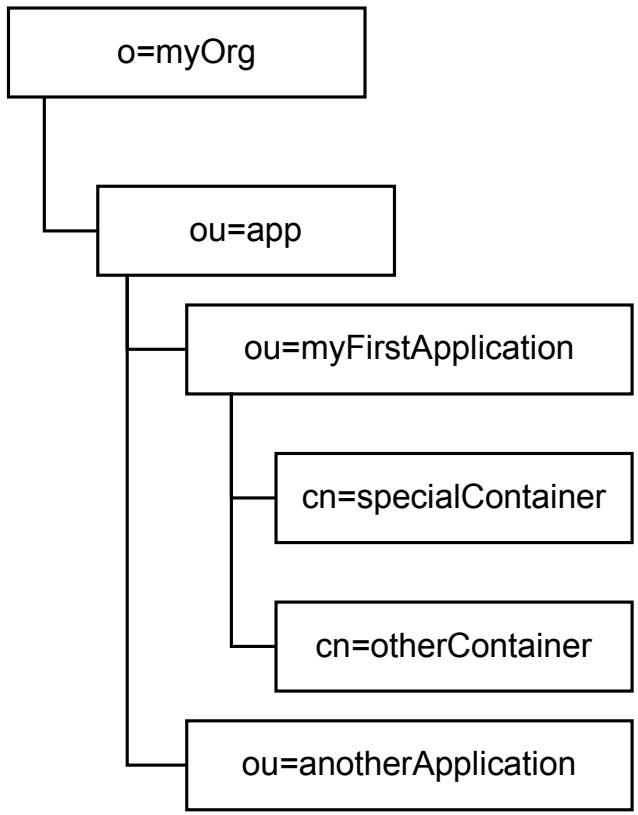
“ou=adm”: an administrative container

- hold the information directly involved in the administration of the directory itself
- The technical or service accounts should find place here
- Place here the few groups (groupsOfUniqueNames) needed to implement the access control



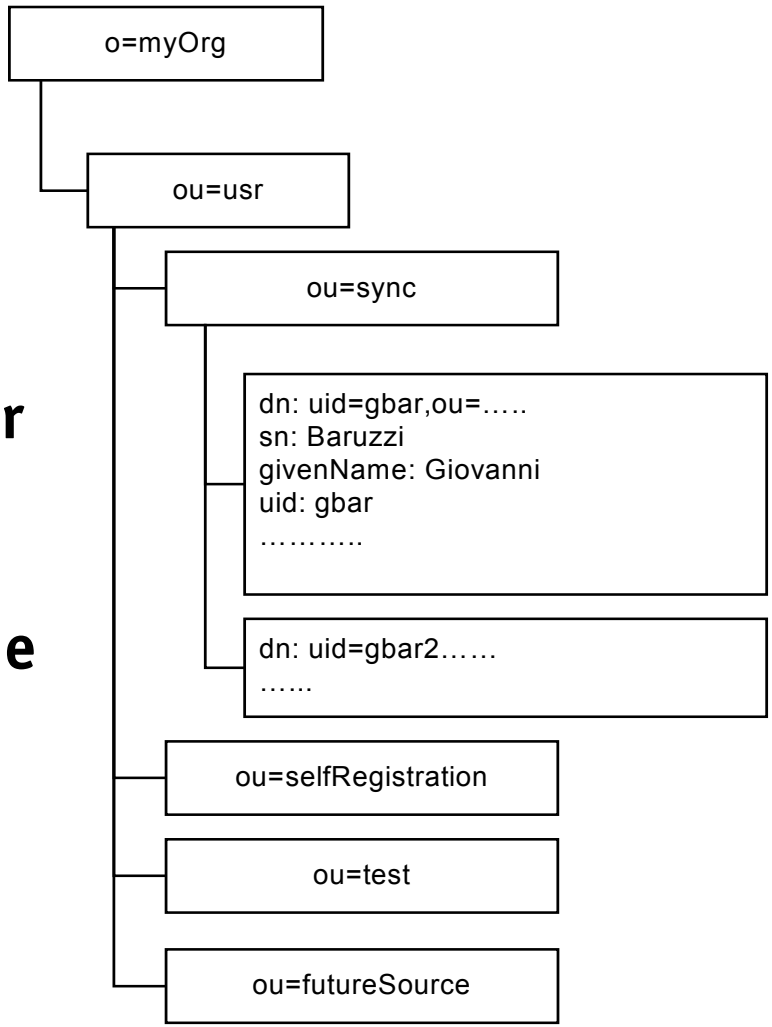
“ou=app”: an extra container for applications

- A dedicated container (“ou=app”) for applications’ private data
- Below this container you may define a dedicated container for every application.
- The single application would receive full access to its container.



The simple design: “o=usr”: the People container

- central to an LDAP Design
- the search base to locate the users
- No additional information is needed from the application than the name of this container and a filter expression
- conceal containers below this against applications (except the managing applications)
- define containers below it as needed (security, replications, logical needs)

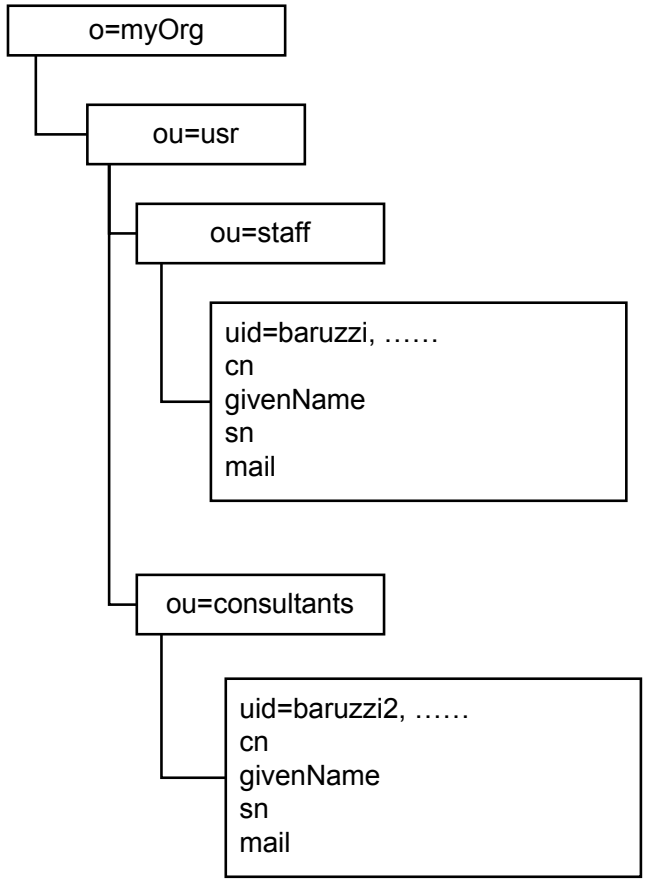


Difficult decisions

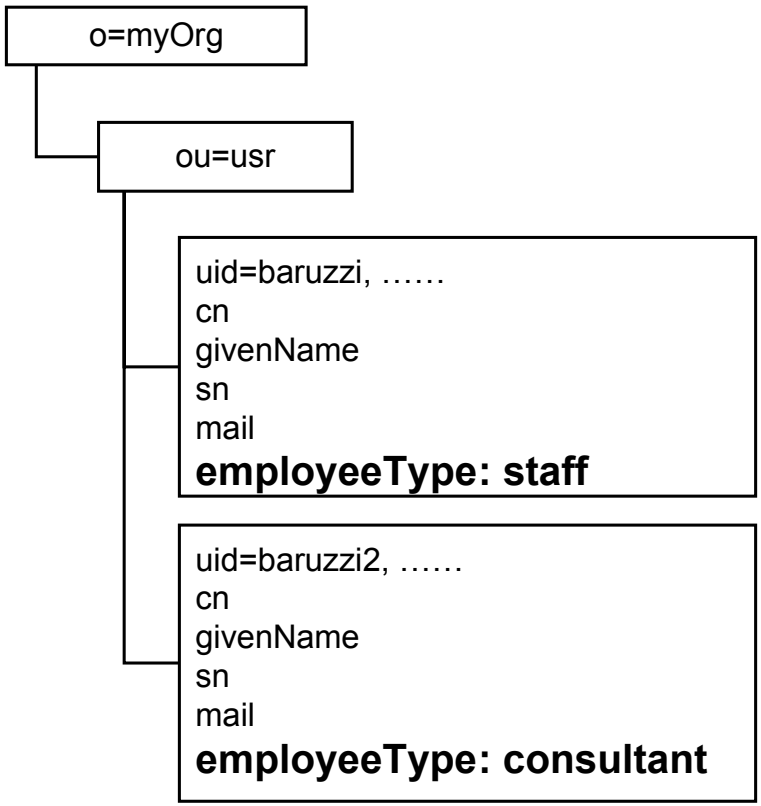
syntlogo

... arise when we work with very similar objects differentiated only by an attribute and only time to time we need to select between them.

Difficult decisions: different containers



Difficult decisions: additional attribute



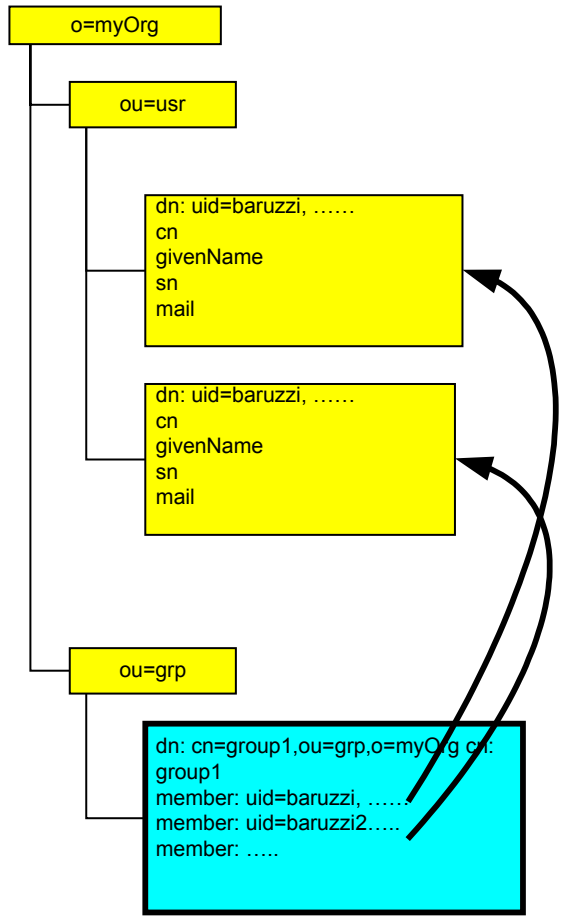
“ou=grp”: Container for Roles and Groups

- Grouping of objects is an essential function for directories, used mainly to grant rights
- groupOfNames and groupOfUniqueNames are broadly used in applications
- we put groups
  - in the ou=adm container
  - below the private application container (ou=appName,ou=app...)
  - below the ou=grp container
- define subcontainers like “sync”, “provisioning” or “manual”

Use of “groupOfNames”

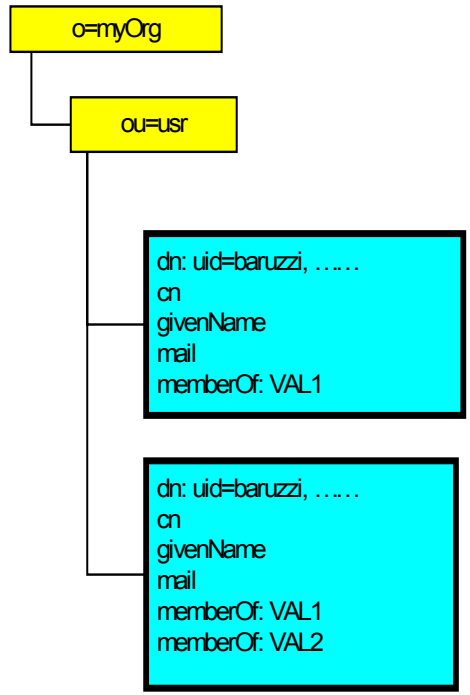
synlogo

- Secure (simple ACL)
- does not scale well beyond 50.000 members
- broadly used

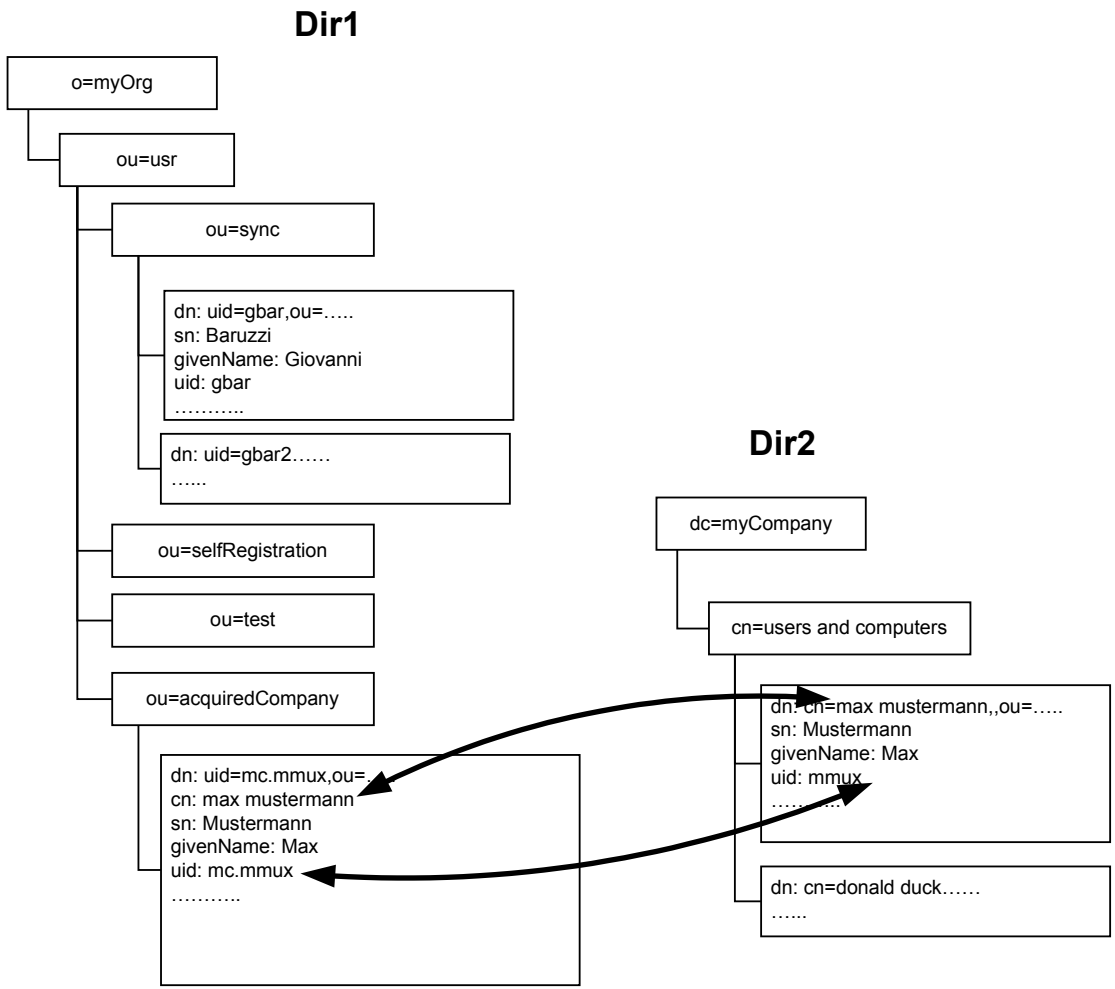


Use of „memberOf“

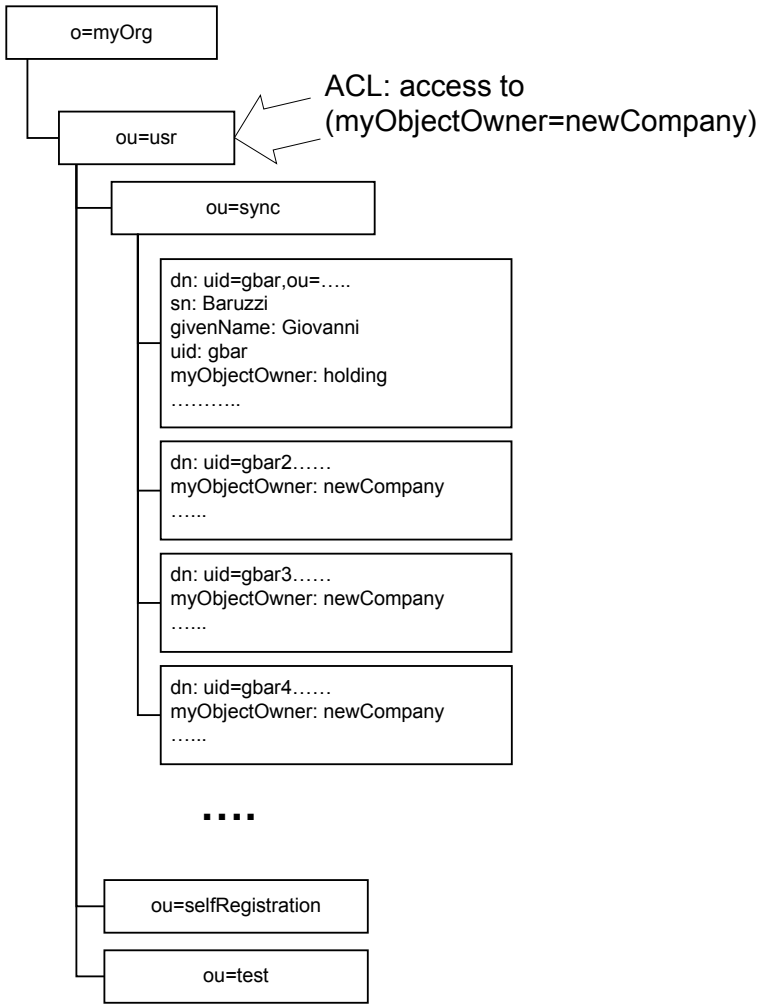
- Scales very well
- One access gets all group memberships
- If an administrator has the right to manage memberOf, she can grant membership to every group defined in this way



Growth of the design with the time: merge two directories



Growth of the design with the time: a directory split



Conclusion

for LDAP, complex structure are often an impediment to further development.
Our simple DIT model, as we have seen, can even survive a merger....

Thank you!