



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

<b>1</b>	<b>Overview .....</b>	<b>2</b>
1.1.1	Public .....	2
<b>2</b>	<b>The analysis .....</b>	<b>3</b>
2.1	Building a common and flexible context .....	3
2.1.1	Why do we need hierarchies? .....	3
2.1.2	Hierarchies and relational databases .....	5
2.1.3	LDAP theory and practice: the need for more flexible and dynamic directories ...	7
2.1.4	Conclusions: Keep LDAP simple and maximize cooperation/coordination between the directory and the applications/database environment .....	9
2.2	Leveraging the common context: Coordinating the work between previously isolated application/processes .....	9
2.2.1	Conclusions: A coordination services based on mainstream tools Connectors, message queues, XML transformation, and application server .....	10
<b>3</b>	<b>Illustration an example: Authentication and authorization with Netegrity .....</b>	<b>10</b>
<b>4</b>	<b>Appendix 1 : Extract from IPlanet 5.0 Documentation .....</b>	<b>15</b>



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

## 1 Overview

The success of LDAP as a general-purpose directory service comes from the simplicity of its protocol and information model. But this success also brought requirements for services that the current **LDAP implementations alone** cannot and **was not designed to support**.

Many examples can be found in the security and provisioning domains: **large-scale web-based customer authentication** with **diversified sources for identity**, complex **role-based authorization** with delegation, resources provisioning and user-driven profile management. Typically projects belonging to that category, require the presence of a common, rich, and **“extensible on demand”** context /hierarchies but also a **close coordination** between the directory information and the existing enterprise application infrastructure.

This note shows, through a general analysis and a specific example in the authentication/authorization domain, that in order to fulfill its promise **LDAP** needs to be complemented by two sets of **“middleware” services**.

1. A **brokering service** that allows
  - a. A LDAP directory to keep its simplicity, and generality (interoperability requirement) by maintaining a simple schema and set of objects (the “common identity context”)
  - b. And to cater to the specific and evolving needs of business processes by providing “dynamic” and flexible “directory views” on demand (“virtual directories”).
2. A **coordination/synchronization service** which through **connectors, messaging and transformations** provides a smooth integration of the directory with a diversified, distributed and (alas) complex IT environment.

### 1.1.1 Public

People interested only by the operational part can skip the analysis and jump directly to the example on authentication/authorization.

People interested in the fundamentals, such as system architects should read the analysis.



## 2 The analysis

### 2.1 Building a common and flexible context

#### 2.1.1 Why do we need hierarchies?

One of the key properties of an advanced directory services is the support of hierarchies of objects and a hierarchical namespace. To date the most successful naming/directory services, from DNS, X500 to LDAP through X500 are hierarchical. So what is so special about hierarchies?

A good answer is to look closely where they play a role in LDAP. It will be a surprise for no one if we say that the hierarchy is a key ingredient at any step of the design, deployment, security of a directory. A quick look at a subset of IPlanet 5.0 documentation is edifying (See Appendix 1). To keep things short, here is a direct quote:

The structure of your tree involves the following steps and considerations:

- ["Branching Your Directory"](#)
- ["Identifying Branch Points"](#)
- ["Replication Considerations"](#)
- ["Access Control Considerations"](#)

The first two items in the list have obvious and direct relationships with the directory tree so we will skip them.

But what is the role of the hierarchy in replication and as we can see also elsewhere in topology/ and directory database partitioning? What is the role of the hierarchies in access rights and security? The answers are obvious to any person familiar with the directory space (so obvious that we tend to take it for granted and forget about it, but nonetheless essential).

To keep things short here are the essential points:

1. A tree establishes an **explicit relationship** between objects that is **easy to navigate and discover**.
2. Navigating downward a hierarchy is a **unit of containment**.
3. Navigating upward a hierarchy is a **unit of aggregation**.
4. Orientation in a tree counts. Navigating from parent to child is semantically different from navigating from child to parent. (A manager is in charge of the members of his unit. The members report to this manager)

Hierarchy and

If we know explicitly the relationships between objects (1) and we know how they are contained (2), then we can easily segregate, delegate, distribute the load and the responsibility of each part



# White Paper

## Why LDAP needs additional services: an Authorization example

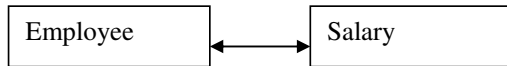
C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

of the tree to different servers. This is the secret behind the versatility of DNS (.com, .edu, etc...) and when it is correctly implemented (Referral and Knowledge references) the strength behind the **LDAP topology**. It is also an essential support for an effective replication strategy.

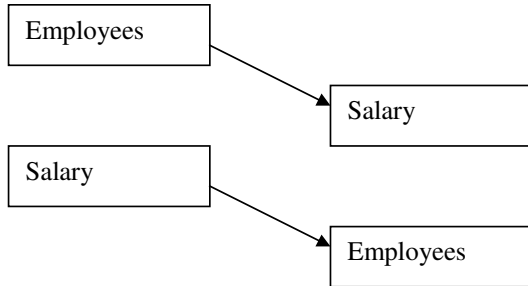
Security, authentication, authorization access rights is not about people, resources, rights, functions, responsibilities separately in isolation. In fact the biggest challenge is to imagine, to simulate all the contexts for security breaches.

A very simple example will make things clearer.

We have the following relationship between employees and salary. Each employee has a salary.



From this relationship we can “navigate” in two ways, we can we have the two following trees (OK we will call it a tree, even if for each parent we can have only one child...☺).



It is very easy to understand from a business point of view what access path/tree to information makes sense

A hierarchy (or a set of hierarchies) making explicit the relationship (1) between people and functions /roles/responsibilities and the access to resources is a key-enabler to a comprehensive security policy. Each of these trees shows a different point of view, a different security “context”. In fact each of these paths/ trees is the digital representation of what in real life we call a context (for a formal definition of what is a context see RLI White Paper on Digital Contexts). **By exposing all these potential relationships explicitly a directory enables a clear business decision in term of security. The security policies become more declarative rather than being hidden in some code module difficult to maintain.**

Finally the hierarchy by defining clearly the responsibilities is key for **security administration and delegation**.

**Comment [m1]:** In fact, despite its deceptive simplicity, a hierarchy can be the “physical” support for two very broad and totally different categories of relationships: the inheritance relationship (the so-called Is-A relationship) and the association relationship (the Has or Is-Associated-To relationship). In the digital world, to establish relationships between objects, a same programming “device/artifact” the tree (the 1 to N relationship) is used to represent the two most generic verbs any language to be and to have ... (No surprise, that a lot of people see directory trees everywhere. At the same time such a huge versatility if not carefully analyzed, tends to drive to a lot of confusion and religious debate too ☺ )



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\1- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

#### 2.1.2 Hierarchies and relational databases

So we have seen the pervasive role played by the hierarchies in directories. In fact one question we can have is how could the database world ignore such an important question? (Well honestly, it does not ignore it at all. I just needed a junction to bring my two grains of salt to the <sigh> debate about RDBMS and directory).

Now (long, long time ago ☺) we learnt from RDBMS 101 that a relational model is the mother of any kind of hierarchical or even network data model. Aren't we missing anything here?

To make things even more confusing, some directory implementation are quite specific (IPlanet, NDS) in term of storage server, and other (Active Directory, Oracle OID, IBM Secureway) are based on standard RDBMS. So did our teachers lie to us or did we fall asleep that day?

No, no one lies to us. The fact is Oracle, DB2, SQL Server can serve an application with as many hierarchical views as it needs. One to N relationships, the so-called "Master-Details" relationship, can be handled quite efficiently through "join", the hallmark of RDBMS and is the work horse for all kind of applications. RDBMS are just not so effective with one very specific category of hierarchy the so-called "recursive relationships". Bill of materials, "Managers and ReportsTo" kind of relationships are examples of that type. Nested OU (Organization Unit) hierarchies could be another example if we were to map a directory into an RDBMS.

So the question is if RDBMS can generate all the hierarchies we need, knowing their maturity/stability etc... why would anyone even needs a separate directory engine or at least an additional directory layer?

The reason is a directory is at a different scope level and justifies an extra abstraction layer. The fact is the unlimited flexibility of an RDBMS in term of relationships management has also a high price in term of programming complexity. Navigating between objects in a relational world is syntactically cumbersome (joins) and requires an extra knowledge from a program (the so-called schema metadata, or knowledge of the relationships between objects). More specifically, dynamic / on the fly discovery of relationships between objects, an essential feature for a directory is not a given in a RDBMS. It needs to be built. A directory client can "discover" relationships between objects just by crawling the hierarchy, or at least the part of the directory he is allowed to explore. An SQL client has to learn first a schema, and then navigate the links he understands. To use a metaphor, a directory client is like a traveler who can have access to a set of predefined "tours" or itineraries exposed by the directory designers. An SQL client, is like a traveler who have access first to a map, some of them of poor quality, and then he needs to figure out a path, an itinerary out of that map. A map contains all potential itineraries, but you still have to decipher the map first and then trace your route.



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\1- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

An example will makes think clearer.

The following picture (fig 1) shows four entities/tables (Customers, Orders, Order\_Details, Product) and their relationships as they exist in a “database schema”. These relationships can be interpreted as follows: A customer places orders. An order consists of a header and lines of order details. Each order detail line references some quantity of product.

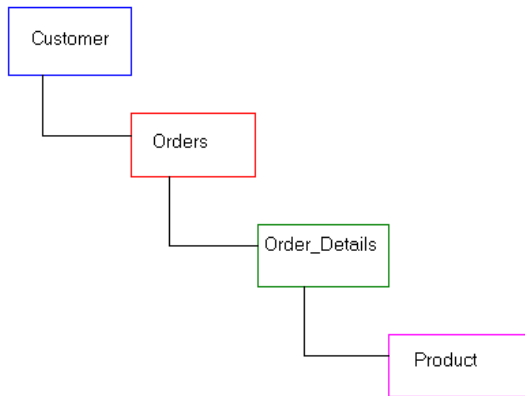


Figure 1

And let say that we want to know for the customer “Joe”, the order number 101, the order line details 6, what are the products ordered. We would have something like this in SQL

```
SELECT ... "Product columns list" FROM Customers, Orders, Order_Details, Products WHERE (Order_Details.ProductID = 6) AND ( Order_Details.OrderID = 101 ) AND ( Orders.CustomerID = Customers.CustomerID ) AND (Customers.CustomerName="Joe").
```

On top of that, a programmer to be able to generate this SQL command, would need access to a full description of the objects (data dictionary level) as shown in Fig 1 and detect the relationships existing between each of these objects. RDBMS can manage any kind of relationships for you, including hierarchies, and they excel at that, but at a relatively high programming cost when it comes to object discovery and navigation.

Let suppose now that an LDAP hierarchy matching this model was in place. We could access the same information from a program with a notation like

Product = \*, Orders details = 6,Orders = 101, Customer= X

A notation that I believe is a lot more “natural” and easy to grasp.



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

The point to remember here is not to add one more bad reasons to the sometime “religious” debate between database and directory. It is to see that directories and databases are playing different roles at different layers, serving different purposes.

In our example both system can drive you to the same results, but the directory approach is at a higher “abstraction” layer (“the view level” where SQL would be more at the model level). This layer allows allowing an easier object navigation and discovery. There is also a cost for this simplicity; you can only discover what the directory exposes. At the SQL layer you have more control and access to more paths, but you need more programming and more rights too. There is nothing like a free lunch ☺.

Understanding the role of each “layers” allows us to take advantage of the strength of each system while canceling out their respective weaknesses.

#### 2.1.3 LDAP theory and practice: the need for more flexible and dynamic directories

In reviewing the key values propositions for a directory we have seen that the support for hierarchical views are essential. Each of these views brings us a different aspect of a question and helps us to solve it. If we dive more into real life cases we can see that the problem is even more complex. What we need is trees that in fact “point” to each other. In real life cases we have a lot of “network” structure that we would like to navigate through specific hierarchical paths. This is the kind of situation where RDBMS shines, but LDAP alone fail.

LDAP hierarchies are static and were not designed for that purpose. References in LDAP to other objects has to be used with moderation, since explicit navigation by chasing referral is an operational that is delegated to the client.

Since the hierarchies are hard-coded in LDAP, managing changes in a hierarchy is quite cumbersome. The result is a kind of paradox or split personality. In theory LDAP can be as hierarchical as needed; in practice it is better to keep the tree in LDAP as flat as possible. Here is again a quote from IPlanet documentation on directory that I believe is quite revealing:

#### Creating Your Directory Tree Structure

You need to decide whether you use a flat or hierarchical tree structure. As a general rule, strive to make your directory tree as flat as possible. However, a certain amount of hierarchy can be important later when you partition data across multiple databases, prepare replication, and set access controls.



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

Another quote:

#### **Branching Your Directory**

Design your hierarchy to avoid problematic name changes. The flatter a namespace is, the less likely the names are to change. The likelihood of a name changing is roughly proportional to the number of components in the name that can potentially change. The more hierarchical the directory tree, the more components in the names, and the more likely the names are to change.

In reality we believe these “difficulties” come from the way LDAP was designed initially and a changing usage pattern. One of the greatest contributions of LDAP has been a greater simplification of the X500 model and the support of TCP/IP. Now the initial design for the Slapd engine was geared more toward the support of a relatively simple directory tree, where the essential function was simple look up for very specific class of entries (initially e-mail address look-ups). Relationships and hence explicit hierarchy of objects at arbitrary level of depth was not in the design, contrarily to X500. That was perhaps a place where simplification was an oversight.

Unfortunately as LDAP is seeing its role expanding to support more complex directory enabled application such identity security/access management, the requirement for more flexible and dynamic views is growing too. One-way to solve partially the problem is to introduce more attributes into the main class of entries like people. The categorization can then be done by these criteria through a search on these attributes. It is equivalent to build an implicit dynamic hierarchy based on the value of these additional attributes. The problem to this approach is the entry becomes a kitchen sink, and as the system evolves the very well-known syndrome of update anomaly will appear. The RDBMS has learnt long time ago the normalization rule (but also sometime the normalization excesses), but it is very possible that the lesson will have to be rediscovered by the LDAP world ☺.

We believe that a lot more pragmatic solution is to use each tools for what it was designed for. LDAP storage should be kept simple and flat. A directory should contain only the essential information, since the cost of synchronization and replication will always be high.

If the directory content should be kept simple, limited to the essential, it should be essentially a repository for identities. Instead of storing the data itself, it is a lot better to store the “pointers”, the keys to the data. By leveraging brokering service such as virtual directories services, these static and simple directories trees can be “dynamically” extended at will. Reconfiguring a virtual directory is a matter of modifying an RDBMS views. With a clever caching, we can obtain the best results from both world: the explicit navigation and simplicity from LDAP trees; the industrial strength, extensive search criteria, triggers, transactions from the RDBMS world.



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

2.1.4 Conclusions: Keep LDAP simple and maximize cooperation/coordination between the directory and the applications/database environment.

## **2.2 Leveraging the common context: Coordinating the work between previously isolated application/processes** (*Work in progress*)

A solid infrastructure needs to be robust and comprehensible. At the same time this infrastructure needs to fully support the complex and evolving demand of the market. There is no point in opposing directories and databases. Each service covers a different spectrum of needs. In fact the challenge is to leverage the strength of each services, make them cooperate and cancel out their respective weaknesses.

A central directory, a reference source of identity is a requirement for security, provisioning etc...But the traditional unsolvable dilemma is to keep a balance between what need to be centralized and what need to be distributed. A high level of information centralization in a directory has a very high cost in term of consistency, replication, synchronization, maintenance etc...and as such unrealistic. On the other side, a too simplistic structure cannot fulfill the business requirements.

This chicken and egg dilemma that has driven many directories to the quick sands of over-design or no-go just ignore a fundamental fact:

**The value, the ROI of a directory does not come just from building it. The value resides in the coordination of the previously “unmanaged/uncoordinated” distributed services that is now made possible by the directory.**

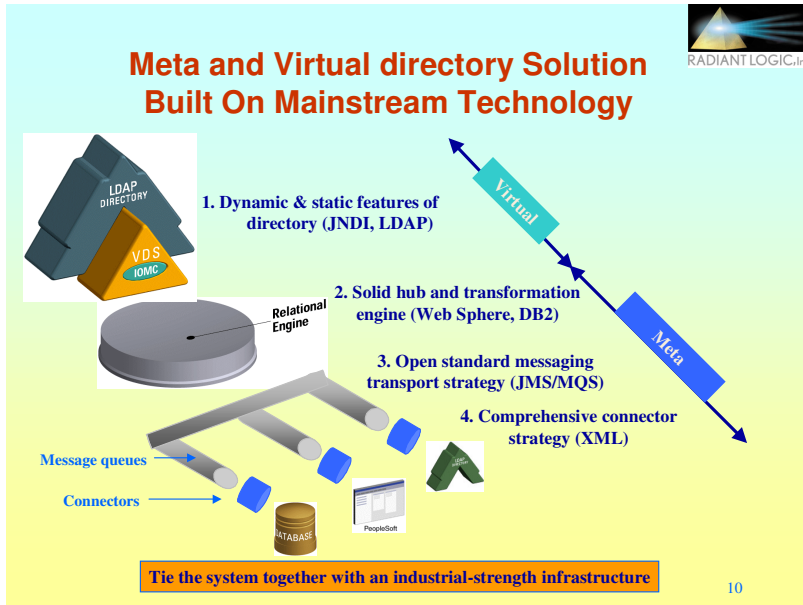
It is not required to have everything inside a directory, what is needed is to find the minimal information needed in a directory plus a service that can use these “keys/pointers” to pull dynamically the information from the right place ( a database or an application). Unified identity management for authentication and services provisioning is a typical example of these kind of benefits.

The way to solve the dilemma is to put in place a set of services that bridge the two worlds: a context and coordination server. Provide the sophisticated services required by your business by coordinating the work of the two worlds, so that directory and applications/databases work hand in hand.

Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

A directory by itself is of limited value, unless it can act as an enabler to coordinate the work of previously isolated applications. To really unlock the promise of high ROI a directory deployment should go hand on hand with its environment. See architecture diagram



2.2.1 Conclusions: A coordination services based on mainstream tools Connectors, message queues, XML transformation, and application server

### 3 Illustration an example: Authentication and authorization with Netegrity

**Example** - For authentication purposes, a flat list of people is sufficient. Therefore, when most people create their DIT it will be relatively flat because it is easier to maintain, and meets authentication needs. However, when it comes to authorization, and being able to authenticate people based on the context of their role within a group, an entirely different hierarchy is needed.

The use case scenario from a Netegrity perspective:

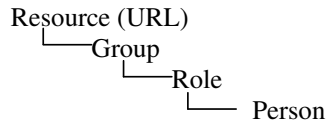


## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

- User logs in
- Netegrity goes to LDAP to authenticate him/her – a flat list of people is sufficient (and in fact necessary) here
- User attempts to access a resource (today it might be a URL but down the road it might be a web service)
- Resource is protected by Group and Role (i.e. Netegrity now needs to search by Group and Role to determine if user is a member)
- The ideal DIT structure for this search is illustrated below:



This challenge has been encountered at many LDAP Netegrity implementation. One alternative to addressing the issue is to use the concept of "dynamic groups" (Thanx to Dan Beckett from DewPoint for this suggestion, that is also the approach recommended by Iplanet)

Consider the following:

LDAP context:

```
o=foo.com
ou=people
cn=userY
  role=abc
  role=xyz
```

```
o=foo.com
ou=groups
ou=groupX
cn=roleZ (objectclass=groupofuniquenames; objectclass=groupofurls)
```

```
memberurl: ldap://sub??(role=xyz)
```

Using this approach, the groups and roles are defined once. From the standpoint of performance, if roles do not change at a very high frequency, the penalty of storing role information directly in the user will be minimal.

There are search by attribute benefits with this approach, not the least of which is that ACIs can now be enforced based on membership of group, derived from a users attributes. Furthermore, as



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

soon as the user's role attributes are modified, the appropriate groups/roles are immediately updated by virtue of their dynamic query. So there is a gain here in database performance and administration by not having to administer the user AND the group every time a role changes.

The above approach also has some drawbacks:

- The concept of dynamic groups is NOT an LDAP v3 standard, it is only found in iPlanet. Therefore, the code used is not portable to other LDAP-enabled directories such as Active Directory.
- If you want to search for people based on criteria other than Group/Role, dynamic groups won't be sufficient. The more criteria you want to base the search on, the more specific hierarchies you will need or you will have to introduce new attributes in InetOrgPerson.
- Storing all kind of attributes in one entry for grouping purpose can be difficult to maintain because they can change frequently. Good design dictates you only keep the attributes that are directly related to the object. (aka of "normalization" rule)
- Group/Role information is often stored in an RDBMS, and copying this data into the directory requires synchronization, and maintaining two copies of the data.
- Searching for other criteria requires a join...and directories aren't designed well for joining!
- Netegrity protects the resource based on roles and groups...not people. This is why it is important to have a tree that starts with resource → groups → roles → people.

With the RadiantOne solution, you are given the flexibility to easily extend the namespace, and create multiple virtual hierarchies that you can build and change on the fly. A typical example would be a company who stores authentication information in a directory, and authorization information in a database. For authentication, a branch could look like Figure A.

C:\Documents and Settings\stims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

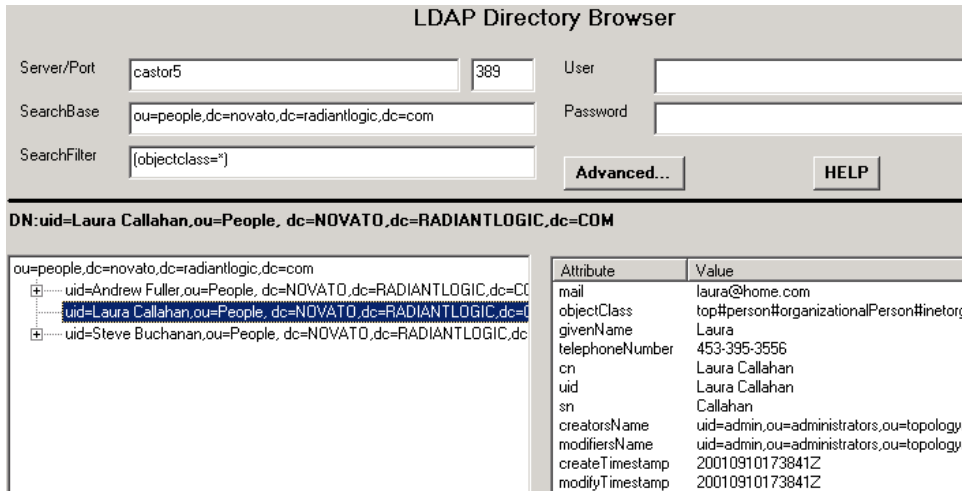


Figure A

To meet authorization requirements, the tree will have to be built starting from groups, and then navigate to the roles within the group, and finally to the users with the role. Groups and role information are stored in a relational database, and the user information is inside the directory. With RadiantOne, the steps to build the needed tree would be as follows:

1. Map needed schemas with Schema Manager tool
2. Build needed tree hierarchy (See Figure B)

### From Database

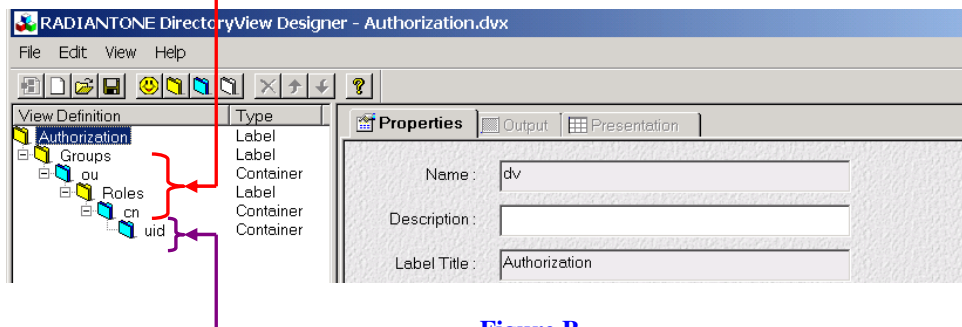


Figure B

### From Directory





## 4 Appendix 1 : Extract from IPlanet 5.0 Documentation

### Designing Your Directory Tree

---

This section guides you through the major decisions you make during the directory tree design process. The directory tree design process involves the following steps:

- Choosing a suffix to contain your data
- Determining the hierarchical relationship among data entries
- Naming the entries in your directory tree hierarchy

The following sections describe the directory tree design process in more detail.

### Creating Your Directory Tree Structure

You need to decide whether you use a flat or hierarchical tree structure. As a general rule, strive to make your directory tree as flat as possible. However, a certain amount of hierarchy can be important later when you partition data across multiple databases, prepare replication, and set access controls.

The structure of your tree involves the following steps and considerations:

- ["Branching Your Directory"](#)
- ["Identifying Branch Points"](#)
- ["Replication Considerations"](#)
- ["Access Control Considerations"](#)

### Branching Your Directory

Design your hierarchy to avoid problematic name changes. The flatter a namespace is, the less likely the names are to change. The likelihood of a name changing is roughly proportional to the number of components in the name that can potentially change. The more hierarchical the directory tree, the more components in the names, and the more likely the names are to change.

Following are some guidelines for designing your directory tree hierarchy:

- Branch your tree to represent only the largest organizational subdivisions in your enterprise.



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

Any such branch points should be limited to divisions (Corporate Information Services, Customer Support, Sales and Professional Services, and so forth). Make sure that divisions you use to branch your directory tree are stable; do not perform this kind of branching if your enterprise reorganizes frequently.

- Use functional or generic names rather than actual organizational names for your branch points.

Names change and you do not want to have to change your directory tree every time your enterprise renames its divisions. Instead, use generic names that represent the function of the organization (for example, use `Engineering` instead of `Widget Research and Development`).

- If you have multiple organizations that perform similar functions, try creating a single branch point for that function instead of branching based along divisional lines.

For example, even if you have multiple marketing organizations, each of which is responsible for a specific product line, create a single `Marketing` subtree. All marketing entries then belong to that tree.

Following are specific guidelines for the enterprise and hosting environment.

#### Branching in an Enterprise Environment

Name changes can be avoided if you base your directory tree structure on information that is not likely to change. For example, base the structure on types of objects in the tree rather than organizations. Some of the objects you might use to define your structure are:

- `ou=people`
- `ou=groups`
- `ou=contracts`
- `ou=employees`
- `ou=services`

A directory tree organized using these objects might appear as follows:



#### Branching in a Hosting Environment

For a hosting environment, create a tree that contains two entries of the object class `organization(o)` and one entry of the `organizationalUnit(ou)` object class beneath the root suffix. For example, the ISP I-Zed branches their directory as follows:



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

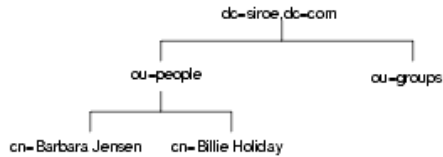


### Identifying Branch Points

As you decide how to branch your directory tree, you will need to decide what attributes you will use to identify the branch points. Remember that a DN is a unique string composed of attribute-data pairs. For example, the DN of an entry for Barbara Jensen, an employee of Siroe Corporation, appears as follows:

`cn=Barbara Jensen,ou=people,dc=siroe,dc=com`

Each attribute-data pair represents a branch point in your directory tree. For example, the directory tree for the enterprise Siroe Corporation appears as follows:



The directory tree for I-Zed, an internet host, appears as follows:



Beneath the root suffix entry, `o=i-zed,c=US`, the tree is split into three branches. The ISP branch contains customer data and internal information for I-Zed. The internet branch is the domain tree. The groups branch contains information about the administrative groups.

There are some points to consider when choosing attributes for your branch points:

- Be consistent.



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

Some [LDAP client](#) applications may be confused if the [distinguished name](#) (DN) format is inconsistent across your directory tree. That is, if `l` is subordinate to `o` in one part of your directory tree, then make sure `l` is subordinate to `o` in all other parts of your directory.

- Try to use only the traditional attributes (shown in [Table 4-1](#)).

Using traditional attributes increases the likelihood of retaining compatibility with third-party LDAP client applications. Using the traditional attributes also means that they will be known to the default directory schema, which makes it easier to build entries for the branch DN.

Table 4-1 Traditional DN Branch Point Attributes  <b>Attribute Name</b>	Definition
c	A country name.
o	An organization name. This attribute is typically used to represent a large divisional branching such as a corporate division, academic discipline (the humanities, the sciences), subsidiary, or other major branching within the enterprise. You should also use this attribute to represent a domain name as discussed in " <a href="#">Suffix Naming Conventions</a> ".
ou	An organizational unit. This attribute is typically used to represent a smaller divisional branching of your enterprise than an organization. Organizational units are generally subordinate to the preceding organization.
st	A state or province name.
l	A locality, such as a city, country, office, or facility name.
dc	A domain component as discussed in " <a href="#">Suffix Naming Conventions</a> ".

---

**Note** A common mistake is to assume that you search your directory based



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

on the attributes used in the distinguished name. However, the distinguished name is only a unique identifier for the directory entry and cannot be searched against.

Instead, search for entries based on the attribute-data pairs stored on the entry itself. Thus, if the distinguished name of an entry is `cn=Babs Jensen,ou=People,dc=siroe,dc=com`, then a search for `dc=siroe` will not match that entry unless you have explicitly put `dc:sun` as an attribute in that entry.

---

### Replication Considerations

During directory tree design, consider which entries you are replicating. A natural way to describe a set of entries to be replicated is to specify the [distinguished name](#) (DN) at the top of a subtree and replicate all entries below it. This subtree also corresponds to a database, a directory partition containing a portion of the directory data.

For example, in an enterprise environment you can organize your directory tree so that it corresponds to the network names in your enterprise. Network names tend to not change, so the directory tree structure will be stable. Further, using network names to create the top level branches of your directory tree is useful when you use replication to tie together different directory servers.

For example, Siroe Corporation has three primary networks known as `flightdeck.siroe.com`, `tickets.siroe.com`, and `hanger.siroe.com`. They initially branch their directory tree as follows:



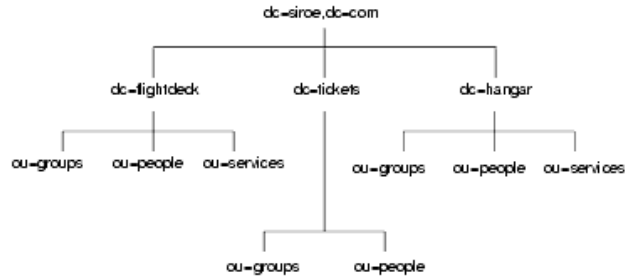
After creating the initial structure of the tree, they create additional branches as follows:



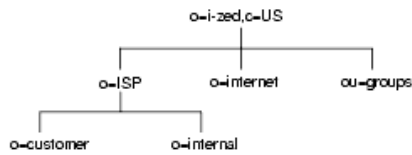
# White Paper

## Why LDAP needs additional services: an Authorization example

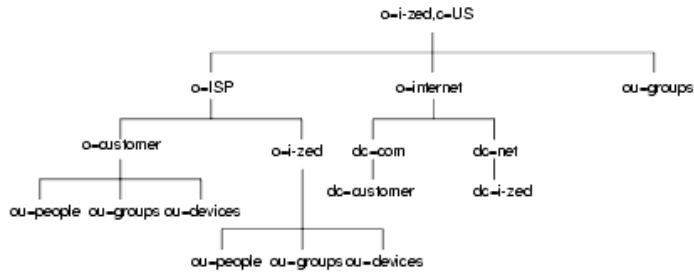
C:\Documents and Settings\stims\Desktop\1- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc



As another example, the ISP i-zed.com branches their directory as follows:



After creating the initial structure of their directory tree, they create additional branches as follows:



Both the enterprise and the hosting organization design their data hierarchies based on information that is not likely to change often.

### Access Control Considerations

Introducing hierarchy into your directory tree can be used to enable certain types of



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

access control. As with replication, it is easier to group together similar entries and then administer them from a single branch.

You can also enable the distribution of administration through a hierarchical directory tree. For example, if you want give an administrator from the marketing department access to the marketing entries and an administrator from the sales department access to the sales entries, you can do so through your directory tree design.

You can set access controls based on the directory content rather than the directory tree. The [ACI](#) filtered [target](#) mechanism lets you define a single access control rule stating that a directory entry has access to all entries containing a particular attribute value. For example, you could set an ACI filter that gives the sales administrator access to all the entries containing the attribute `ou=Sales`.

However, ACI filters can be difficult to manage. You must decide which method of access control is best suited to your directory: organizational branching in your directory tree hierarchy, ACI filters, or a combination of the two.

## Naming Entries

After designing the hierarchy of your directory tree, you need to decide which attributes to use when naming the entries within the structure. Generally, names are created by choosing one or more of the attribute values to form a [relative distinguished name](#) (RDN). The RDN is the left-most [DN](#) attribute value. The attributes you use depend on the type of entry you are naming.

Your entry names should adhere to the following rules:

- The attribute you select for naming should be unlikely to change.
- The name must be unique across your directory.

A unique name ensures that a DN can refer to at most one entry in your directory.

When creating entries, define the RDN within the entry. By defining at least the RDN within the entry, you can locate the entry more easily. This is because searches are not performed against the actual DN but rather the attribute values stored in the entry itself.

Attribute names have a meaning, so try to use the attribute name that matches the type of entry it represents. For example, do not use `l` (`locality`) to represent an organization, or `c` (`country`) to represent an organizational unit.

The following sections provide tips on naming entries:

- [Naming Person Entries](#)
- [Naming Group Entries](#)
- [Naming Organization Entries](#)
- [Naming Other Kinds of Entries](#)



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

### Naming Person Entries

The person entry's name, the DN, must be unique. Traditionally, distinguished names use the `commonName`, or `cn`, attribute to name their person entries. That is, an entry for a person named Babs Jensen might have the distinguished name of:

```
cn=Babs Jensen,dc=siroe,dc=com
```

While allowing you to instantly recognize the person associated with the entry, it might not be unique in an include people with identical names. This quickly leads to a problem known as DN [name collisions](#), multiple entries with the same distinguished name.

You can avoid common name collisions by adding a unique identifier to the common name. For example:

```
cn=Babs Jensen+employeeNumber=23,dc=siroe,dc=com
```

However, this can lead to awkward common names for large directories and can be difficult to maintain.

A better method is to identify your person entries with some attribute other than `cn`. Consider using one of the following attributes:

- `uid`  
Use the `uid` (`userID`) attribute to specify some unique value of the person. Possibilities include a user login ID or an employee number. A subscriber in a hosting environment should be identified by the `uid` attribute.
- `mail`  
Use the `mail` attribute to contain the value for the person's email address. This option can lead to awkward DNs that include duplicate attribute values (for example: `mail=bjensen@siroe.com, dc=siroe,dc=com`), so you should use this option only if you cannot find some unique value that you can use with the `uid` attribute. For example, you would use the `mail` attribute instead of the `uid` attribute if your enterprise does not assign employee numbers or user IDs for temporary or contract employees.
- `employeeNumber`  
For employees of the `inetOrgPerson` object class, consider using an employer assigned attribute value such as `employeeNumber`.

Whatever you decide to use for an attribute-data pair for person entry RDNs, you should make sure that they are unique, permanent values. Person entry RDNs should also be readable. For example, `uid=bjensen, dc=siroe,dc=com` is preferable to `uid=b12r56A, dc=siroe,dc=com` because recognizable DNs simplify some directory tasks, such as changing directory entries based on their distinguished names. Also, some directory client applications assume that the `uid` and `cn` attributes use human-readable names.



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\stims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

#### Considerations for Person Entries in a Hosted Environment

If a person is a subscriber to a service, the entry should be of object class `inetUser` and the entry should contain the `uid` attribute. The attribute must be unique within a customer subtree.

If a person is part of the hosting organization, represent them as an `inetOrgPerson` with the `nsManagedPerson` object class.

#### Placing Person Entries in the DIT

Here are some guidelines for placing people entries in your directory tree:

- People in an enterprise should be located in the directory tree below the organization's entry.
- Subscribers to a hosting organization need to be below the `ou=people` branch for the hosted organization.

#### Naming Group Entries

There are four main ways to represent a group:

- A static group  
The entry for this type of group uses the `groupOfNames` or `groupOfUniqueNames` object class, which contains values naming the members of the group. Static groups are suitable for groups with few members, such as the group of directory administrators. Static groups are not suitable for groups with thousands of members.  
Static group entries must contain a `uniqueMember` attribute value, because `uniqueMember` is a mandatory attribute of the `groupOfUniqueNames` object. This object class requires the `cn` attribute, which can be used to form the DN of the group entry.
- A member-based group  
This type of group uses a `memberOf` attribute in the entry of each group member.
- A dynamic group  
This type of group uses an entry representing the group with a search filter and subtree. Entries matching the filter are members of the group.
- A Directory Server [role](#)  
Roles are a new feature of Directory Server that unify the static and dynamic group concept. Refer to "[About Roles](#)" for more information.

In a deployment containing hosted organizations, we recommend using the `groupOfUniqueNames` object class to contain the values naming the members of groups used in directory administration. In a hosted organization, we also recommend



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

that group entries used for directory administration are located under the `ou=Groups` branch.

#### Naming Organization Entries

The organization entry name, like other entry names, must be unique. Using the legal name of the organization along with other attribute values helps ensure the name is unique. For example, you might name an organization entry as follows:

```
o=Company22+st=Washington, o=ISP, c=US
```

You can also use trademarks, however they are not guaranteed to be unique.

In a hosting environment, you need to include the following attributes in the organization's entry:

- `o` (`organizationName`)
- `objectClass` with values of `top`, `organization`, and `nsManagedDomain`

#### Naming Other Kinds of Entries

Your directory will contain entries that represent many things, such as localities, states, countries, devices, servers, network information, and other kinds of data.

For these types of entries, use the `commonName` (`cn`) attribute in the RDN if possible. Therefore, if you are naming a group entry, name it as follows:

```
cn=administrators, dc=siroe, dc=com
```

However, sometimes you need to name an entry whose object class does not support the `commonName` attribute. Instead, use an attribute that is supported by the entry's object class.

There does not have to be any correspondence between the attributes used for the entry's DN and the attributes actually used in the entry. However, a correspondence between the DN attributes and attributes used by the entry simplifies administration of your directory tree.

#### Grouping Directory Entries

---

Once you have created entries, you can group them for ease of administration. The Directory Server supports several methods for grouping entries and sharing attributes between entries:



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

- Using roles
- Using class of service

The following sections describe each of these mechanisms in more detail.

### About Roles

Roles are a new entry grouping mechanism. Your directory tree organizes information hierarchically. This hierarchy is a grouping mechanism, though it is not suited for short-lived, changing organizations. Roles provide another grouping mechanism for more temporary organizational structures.

Roles unify static and dynamic groups. You use static groups to create a group entry that contains a list of members. Dynamic groups allow you to filter entries that contain a particular attribute and include them in a single group.

Each entry assigned to a [role](#) contains the `nsRole` attribute, a computed attribute that specifies all of the roles an entry belongs to. A client application can check role membership by searching the `nsRole` attribute, which is computed by the directory and therefore always up-to-date.

Roles are designed to be more efficient and easier to use for applications. For example, applications can locate the roles of an entry, rather than select a group and browse the members list.

You can use roles to do the following:

- Enumerate the members of the role.  
Having an enumerated list of role members can be useful for resolving queries for group members quickly.
- Determine whether a given entry possesses a particular role.  
Knowing the roles possessed by an entry can help you determine whether the entry possesses the target role.
- Enumerate all the roles possessed by a given entry.
- Assign a particular role to a given entry.
- Remove a particular role from a given entry.

Each role has *members*, entries that possess the role. You can specify members either explicitly (meaning each entry contains an attribute associating it with a role) or dynamically (by creating a filter that assigns entries to roles depending upon an attribute contained by the entry). How you specify role membership depends upon the type of role you are using. There are three types of roles:

- Managed roles.  
A [managed role](#) allows you to create an explicit enumerated list of members. Managed roles are added to entries using the `nsRoleDN` attribute.



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

- Filtered roles.  
A [filtered role](#) allows you to assign entries to the role depending upon the attribute contained by each entry. You do this by specifying an LDAP filter. Entries that match the filter are said to possess the role.
- Nested roles.  
A [nested role](#) allows you to create roles that contain other roles. You specify the roles nested within it using the `nsRoleDN` attribute.

### Deciding Between Roles and Groups

Both methods of grouping entries have advantages and disadvantages. Roles reduce client-side complexity at the cost of increased server complexity. With roles, the client application can check role membership by searching the `nsRole` attribute. From the client application point of view, the method for checking membership is uniform and is performed on the server side.

Dynamic groups, from an application point of view, offer no support from the server to provide a list of group members. Instead, the application retrieves the group definitions and then runs the filter. For static groups, the application must make sure the user is part of a particular `UniqueMember` attribute value. The method for determining group membership is not uniform.

You can use managed roles to do everything you would normally do with static groups. You can filter group members using filtered roles as you used to do with dynamic groups.

While roles are easier to use, more flexible, and reduce client complexity, they do so at the cost of increased server complexity. Determining role membership is more resource intensive because the server does the work for the client application.

### About Class of Service

A [class of service](#) (CoS) allows you to share attributes between entries in a way that is invisible to applications. With CoS, some attribute values may not be stored with the entry itself. Instead, they are generated by class of service logic as the entry is sent to the client application.

For example, your directory contains thousands of entries that all share the common attribute `facsimileTelephoneNumber`. Traditionally, to change the fax number, you would need to update each entry individually, a large job for administrators that runs the risk of not updating all entries. With CoS, you can generate the attribute dynamically. The `facsimileTelephoneNumber` attribute is stored in one place, and each entry points to that place to give a value to their fax number attribute. For the application, these attributes appear just like all other attributes, despite that they are not actually stored on the entries themselves.

Each CoS is comprised of the following entries in your directory:



## White Paper

### Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

- CoS Definition Entry.  
The [CoS definition entry](#) identifies the type of CoS you are using. It is stored as an LDAP subentry below the branch it affects.
- Template Entry.  
The [template entry](#) contains a list of the shared attribute values. Changes to the template entry attribute values are automatically applied to all the entries sharing the attribute.

The CoS definition entry and the template entry interact to provide attribute values to their *target entries*, the entries within their scope. The value they provide depends upon the following:

- The entry's DN (different portions of the directory tree might contain different CoS).
- A service class attribute value stored with the entry.  
The absence of a service class attribute can imply a specific default CoS.
- The attribute value stored in the CoS template entry.  
Each CoS template entry supplies the attribute value for a particular CoS.
- The [object class](#) of the entry.  
CoS attribute values are generated only when an entry contains an object class allowing the attribute when schema checking is turned on, otherwise all attribute values are generated.
- The attribute stored in some particular entry in the directory tree.

You can use different types of CoS depending upon how you want the value of your dynamic attributes to be generated. There are three types of CoS:

- Pointer CoS.  
A [pointer CoS](#) identifies the template entry using the template DN only. There may be only one template DN for each pointer CoS. A pointer CoS applies to all entries within the scope of the template entry.
- Indirect CoS.  
An [indirect CoS](#) identifies the template entry using the value of one of the target entry's attributes. The target entry's attribute must contain the [DN](#) of an existing entry.
- Classic CoS.  
A [classic CoS](#) identifies the [template entry](#) by both its [DN](#) and the value of one of the target entry's attributes. Classic Cos can have multiple template entries, including a default CoS template to be applied to those entries that do not belong to any other CoS template.

Roles and the Classic CoS can be used together to provide [role-based attributes](#). These attributes appear on an entry because it possesses a particular role with an associated class of service template. For example, you could use a role-based attribute to set the server look through limit on a role-by-role basis.



# White Paper

## Why LDAP needs additional services: an Authorization example

C:\Documents and Settings\tims\Desktop\I - web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

### Directory Tree Design Examples

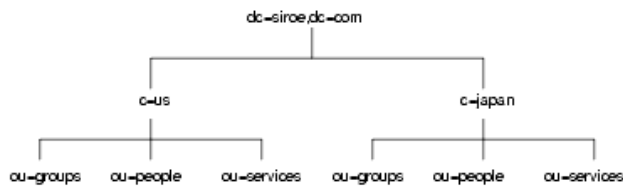
---

The following sections provide examples of directory trees designed to support a flat hierarchy as well as several examples of more complicated hierarchies.

#### Directory Tree for an International Enterprise

To support an international enterprise, root your directory tree in your Internet domain name and then branch your tree for each country where your enterprise has operations immediately below that root point. In "[Suffix Naming Conventions](#)", you are advised to avoid rooting your directory tree in a country designator. This is especially true if your enterprise is international in scope.

Because LDAP places no restrictions on the order of the attributes in your DNs, you can use the `c` (`countryName`) attribute to represent each country branch as follows:



However, some administrators feel that this is stylistically awkward, so instead you could use the `l` (`locality`) attribute to represent different countries:



#### Directory Tree for an ISP

Internet service providers (ISPs) may support multiple enterprises with their directories. If



# White Paper

## Why LDAP needs additional services: an Authorization example

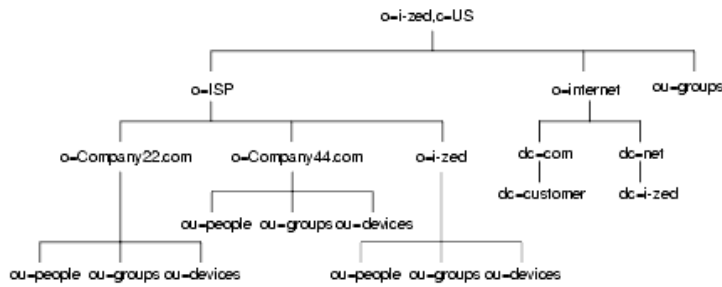
C:\Documents and Settings\tims\Desktop\I- web site\Resources\white papers\Authorizations challenge with the current LDAP structure.doc

you are an ISP, consider each of your customers as a unique enterprise and design their directory trees accordingly. For security reasons, each account should be provided a unique directory tree with a unique suffix and an independent security policy.

You can assign each customer a separate database, and store these databases on separate servers. Placing each directory tree in its own database allows you to back up and restore data for each directory tree without affecting your other customers.

In addition, partitioning helps reduce performance problems caused by disk contention, and reduces the number of accounts potentially affected by a disk outage.

For example, the directory tree for I-Zed, an ISP, appears as follows:



uid	cn	Other attributes	role
	userY		abc

ou	
groupX	

