Slides for Chapter 13: Name Services

From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems: Concepts and Design

Names

- My 15” MacBook Pro
- The rightmost computer on my desk
- Paul’s aluminum laptop, but not the big or the small one.
- hedwig
- hedwig.pk.org
- 192.168.60.148
- 00:14:51:ec:f2:5b

Naming things

- User names
  - Login, email
- Machine names
  - rlogin, email, web
- Files
- Devices
- Variables in programs
- Network services

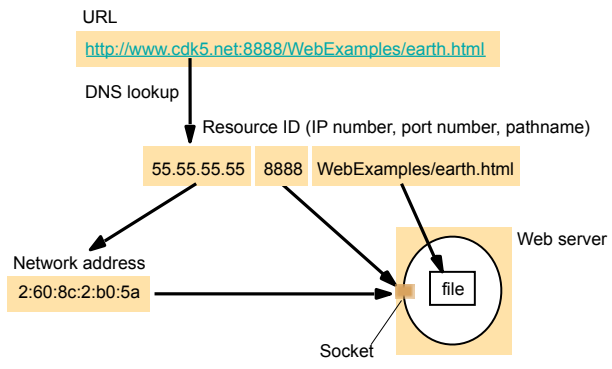
Names

- Need names for:
  - Services: e.g., time of day
  - Nodes: computer that can run services
  - Paths: route
  - Objects within service: e.g. files on a file server
- Naming convention can take any format
  - Ideally one that will suit application and user
  - E.g., human readable names for humans, binary identifiers for machines

What's a name?

- **Name**: refers to what you want
  - any resource can be named
- **Address**: identifies where it is, its access point
  - needed in order to operate on the resource
  - a resource may have several access points
    - phone numbers, file path, ip address, ip:port, …
    - can change during time (especially in distributed systems)
  - the same access point can be re-assigned to different resource
- **Route**: identifies how to get there, to the access point

Figure 13.1
Composed naming domains used to access a resource from a URL



Names, Identifiers, And Addresses

- Names and addresses can be re-assigned
- Identifiers instead are names with particular properties:
  - An identifier refers to at most one entity.
  - Each entity is referred to by at most one identifier.
  - An identifier always refers to the same entity
- examples: Ethernet address, memory address, Universally Unique IDentifier (UUID), Tax Code

"What's in a name? That which we call a rose
By any other name would smell as sweet"

Terms: Naming convention

Naming system determines syntax for a name

- Unstructured names: flat sequences of bits
- Unix file names:
  - Parse components from left to right separated by /
  - /home/paul/src/gps/gui.c
- Internet domain names:
  - Ordered right to left and delimited by . www.cs.rutgers.edu
- LDAP names
  - Attribute/value pairs ordered right to left, delimited by , cn=Paul Krzyzanowski, o=Rutgers, c=US

Terms: Context, Naming System, Name Space

- A **context** is a particular set of (name,object) bindings
- Each context has an associated naming convention
- A **name** is always interpreted relative to some context
  - E.g., directory /usr in the UNIX file system
- **Naming system**: Connected set of contexts of the same type (same naming convention) along with a common set of operations. For example:
  - System that implements DNS
  - System that implements LDAP
- **Name space**: Set of names in the naming system
  - For example, names of all files and directories in a UNIX file system

Binding

- **Binding**: associates a name with an address
  - address is often not easy to use and not flexible
  - “choose a lower-level-implementation for a higher-level semantic construct” RFC 1498: Inter-network Naming, addresses, routing
- The association can be
  - Static binding
    - Hard-coded
  - Early binding
    - Look up binding before use
    - Cache previously used binding
  - Late binding
    - Look up just before use

Naming Service

- A **Naming Service** allows you to look up names
  - Often returns an address as a response
  - sometimes corresponds to routing, i.e., finding where is the resource (e.g. DNS)
- Might be implemented as
  - Search through file
  - Client-server program
  - Database query (possibly distributed)
  - …

Directory Service

- Extension of naming service:
  - Associates names with objects
  - Allows objects to have attributes
  - Can perform search based on attributes, not only names
  - For example: LDAP
- Directory can be object store:
  - Look up printer object and send data stream to it

Name resolution

To send data to a service:

1. Find a node on which the service resides (service name resolution)
2. Find an address (or network attachment point) for that node (node name location)
3. Find a path from this location to the service (routing service)

E.g., access “paul’s service”:

- File lookup: “paul’s service”→cs.rutgers.edu:1234
- DNS lookup: cs.rutgers.edu →128.6.4.2
- ARP resolution: 128.6.4.2 →08:00:20:90:9c:23
- IP routing:route: remus →lcsr-gw →aramis

Name resolution

- In distributed systems, often name resolution is strictly related to routing
- e.g. resolving www.dimi.uniud.it requires to
  - resolve .it to find the (address of the) server NS(it)
  - send www.dimi.uniud to NS(it), which resolves .uniud finding the server NS(uniud.it)
  - send www.dimi to NS(uniud.it), which answers with 158.110.144.132
- the resolution path is close to the route path: each server gets closer to the access point

Types of naming services

- Simple naming
  - flat, unstructured names containing no information of how to resolve
  - various solutions, from simple broadcast to hierarchical domains
- Structured naming
  - names are structured according a specified grammar
  - structure explains how to solve the name
- Attributed naming
  - names are objects with attribute-value pairs
  - often used in directories

Simple naming: broadcast

- Broadcast and multicast
  - a message containing the name is broadcasted to all nodes
  - only the node(s) which contains the resource with that name will answer with the corresponding address
  - ARP: resolution IP address -> Ethernet address
- Effective only on LANs; broadcast does not scale to WAN
- Possible on multicast networks
  - e.g., for finding address of mobile nodes (i.e. real IP varying, but multicast group is the same)
  - or choosing a replica among many equivalent ones

Structured naming

- The approaches seen so far are classified as flat naming:
  - good for machines,
  - not very convenient for humans.
- The alternative is to use structured names composed by simple human-readable names.
- This is the usual scheme adopted in
  - file systems,
  - host naming on the Internet,
  - ...

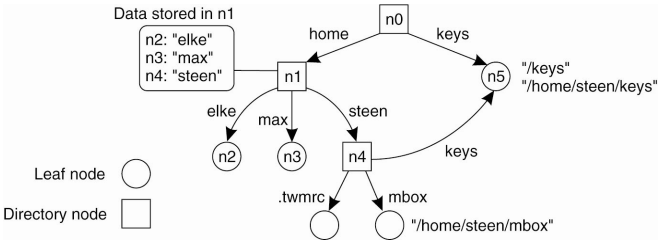
Structured naming (examples)

- Uniform Resource Identifiers (URIs):
  - goal: identify resources in a coherent way;
  - URIs are uniform: their syntax incorporates that of indefinitely many individual types of resource identifiers (URI schemes);
  - there are automatic procedures for managing the global namespace of schemes.
- Uniform Resource Locators (URLs):
  - they contain information useful to locate and access a resource;
  - issue: moving or deleting resources makes URLs dangling;
  - “cool URLs do not change”.
- Uniform Resource Names (URNs):
  - pure resource names.
  - e.g., mid:0E4FC272-5C02-11D9-B115-000A95B55BC8@hpl.hp.com, urn:ISBN:0-201-62433-8, urn:doi:10.555/music-pop-1234

Name Spaces (1)

- Names are commonly organized into a **name space**.
- The usual representation of a name space is a **labeled**, **directed** graph with 2 types of nodes:
  leaf node: represents a named entity (no outgoing edges);
  directory node: it has some outgoing (name-labeled) edges.
- Each **node** in a naming graph represents another **entity** (with an associated identifier) in a distributed system.
- A directory node stores a **directory table** where each entry *(edge label, node identifier)* represents an outgoing edge.

Name Spaces (2)



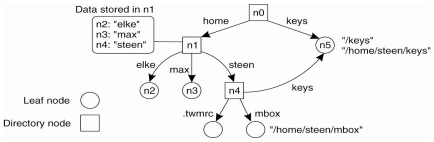
A general naming graph with a single **root** node.

Name Spaces (3)

- A path in a naming graph is represented by the sequence of labels corresponding to its edges, i.e.,
  N: <label-1, label-2, ..., label-n>
  where *N* refers to the first node in the path.
- If *N* is the root of the naming graph, then we speak of an **absolute path name**, otherwise of a **relative path name**.
- A **global name** is a name denoting the same entity, no matter where it is used.
- A **local name** is a name whose interpretation depends on where it is used.

Name Spaces (4)

- Naming graphs are very close to the implementation of **file systems**.
- For instance, instead of writing a sequence of labels, a pathname in a file system is a single string in which labels are separated by a **special separator character** (e.g., "/" in UNIX-like systems), indicating also the root node:



n0: < home, steen, mbox> ——————→ /home/steen/mbox

Name Spaces (5)

- There are many different ways to organize a name space:
  - **single root** node vs. **multiple root** nodes,
  - **strictly hierarchical**, i.e., the naming graph is a **tree**,
  - the naming graph is a **directed acyclic graph**: each node can have multiple (absolute) pathnames,
  - the naming graph is a **general graph** (possibly with cycles).

Name Spaces: name resolution

- **Name resolution**: the process of **looking up a name**.
- Let us consider a pathname *N: <label$_1$, label$_2$, ..., label$_n$>*
  - resolution starts at *N* looking up *label$_1$* in the directory table;
  - this procedure returns the identifier of the node to which *label$_1$* refers;
  - resolution goes on by looking up *label$_2$* in the directory table of the identified node and so on;
  - if the pathname exists, the resolution stops at the node referred to by *label$_n$*, returning its contents.

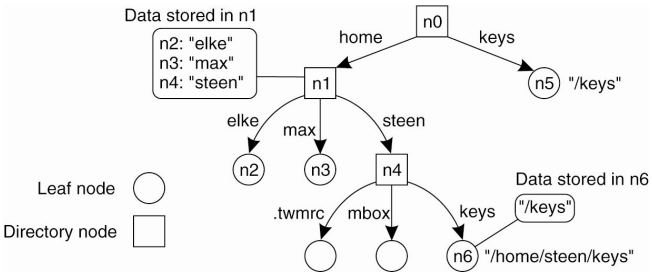
Name Spaces: closure mechanism

- Name resolution can take place only if we know how and where to start, e.g., in the previous example:
  - where: the initial node of a pathname;
  - how: accessing its directory table.
- **Closure mechanism**: knowing how and where to start name resolution.
- Closure mechanisms are not always trivial:
  - in a UNIX file system name resolution works based on the fact that the operating system knows the inode of the root directory.

Linking and Mounting (1)

- **Alias**: another name for the same entity.
- In naming graphs there are two different ways to implement an alias:
  - allowing multiple absolute pathnames to refer to the same node (in UNIX terminology, these are called **hard links**);
  - representing an entity by a leaf node N and storing an absolute pathname instead of the address or state of that entity (in UNIX terminology, these are called **symbolic links**).

Linking and Mounting (2)



The concept of a symbolic link explained in a naming graph.

Linking and Mounting (3)

- So far, name resolution is taking place completely within a **single** name space.
- However, name resolution can also **merge different** name spaces in a transparent way using the **mounting** mechanism.
- A mounted file system is represented by:
  - letting a directory node store the identifier of a directory node from a different name space (also called **foreign name space**);
  - the directory node storing such identifier is called a **mount point**.
- This approach is followed in many **distributed file systems**, where each name space can be implemented by a different server.

Linking and Mounting (4)

- Mounting a foreign name space NS, may require to communicate over a network, since NS may be running on a different machine.
- Thus, in general, the information required to mount a foreign name space in a distributed system is at least the following:
  - the **name** of an **access protocol**,
  - the **name** of the **server**,
  - the **name** of the **mounting point** in the foreign name space.

Linking and Mounting (5)

Notice that each of the previous names **must be resolved**:

•the name of the protocol must be resolved to an implementation of the access protocol in order to communicate with the server of the foreign space;

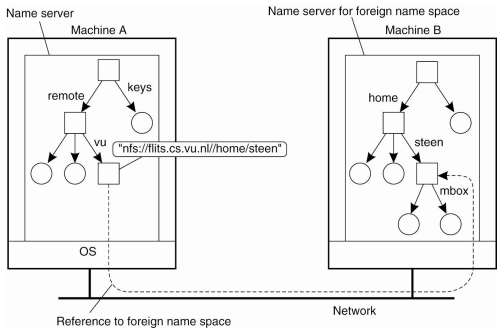
•the name of the server must be resolved to an address where the server can be reached;

•the name of the mounting point must be resolved to a node identifier in the foreign name space.

•Moreover, we need name spaces and implementations for the access protocol and the server name.

•A common solution is to represent the three names as a URL (e.g., with Sun's NFS: nfs://flits.cs.vu.nl//home/steen).

Linking and Mounting (6)



Name Spaces: implementation

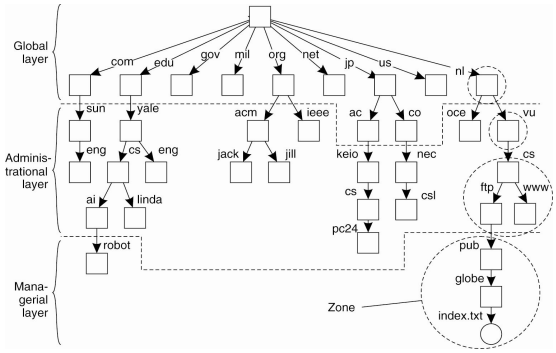
- A name space is the core of a naming service.
- Naming service: service allowing users and processes to add, remove and lookup names.
- A naming service is implemented by name servers.
- In a LAN a single name server is often a viable solution.
- In large-scale distributed systems it is necessary to spread the implementation of a name service across multiple name servers.

Name Space Distribution (1)

- In order to effectively implement a large name space (corresponding to a worldwide distributed system), it is convenient to partition it into the following logical layers:
  - **global layer**: root node and its children (stable nodes representing organizations or groups of organizations: directory tables are rarely changed);
  - **administrational layer**: nodes managed by a single organization (nodes are relatively stable);
  - **managerial layer**: nodes changing regularly (e.g., hosts in a LAN, shared files, user-defined directories and files etc.).

33

Name Space Distribution (2)



An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Name Space Distribution (3)

- Name servers in **each layer** have to meet **different requirements**.
- In the **global** layer:
  high **availability** is critical for name servers,
  caching by clients can be effectively used,
  thus, for the global layer, **availability** and **performance** can be obtained by **replicating servers** and **client-side caching** (updates do not have to come into effect quickly).
- In the **administrational** layer:
  availability is important for clients **within the same organization** as the name server,
  client-side caching is effective, but **updates** must be **processed quickly** (name servers must be high-performance machines).
- In the **managerial** layer:
  availability is **less demanding** (a single dedicated machine as a name server is sufficient),
  performance is **crucial**,
  client-side caching is less effective.

Name Space Distribution (4)

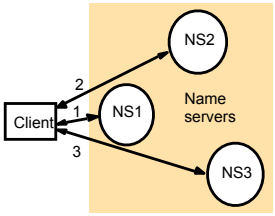
Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, an administrational layer, and a managerial layer.

Name resolution - navigation

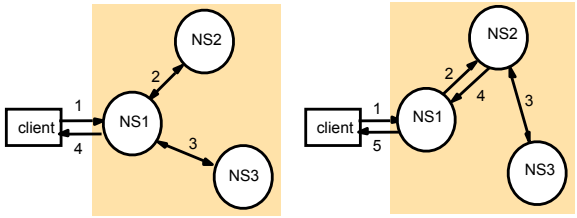
- Iterative navigation
- Multicast navigation
- Non-recursive server-controlled navigation
- Recursive server-controlled navigation

Figure 13.2
Iterative navigation



A client iteratively contacts name servers NS1–NS3 in order to resolve a name

Figure 13.3
Non-recursive and recursive server-controlled navigation

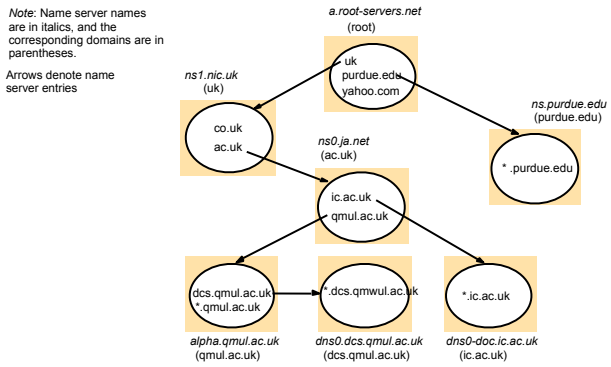


Non-recursive server-controlled

Recursive server-controlled

A name server NS1 communicates with other name servers on behalf of a client

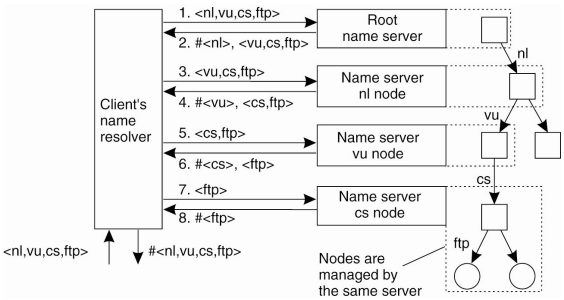
Figure 13.4
DNS name servers



Implementation of Name Resolution (1)

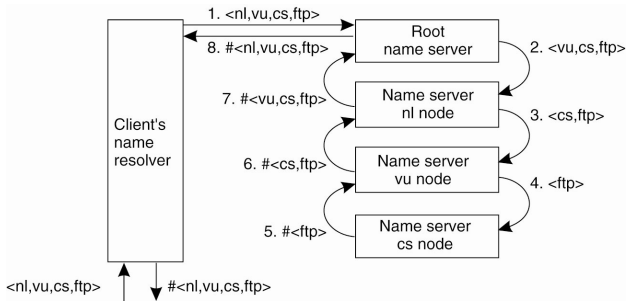
- A distributed name space across multiple name servers affects the implementation of name resolution.
- Let us suppose we have to resolve the following path name: *root: <nl, vu, cs, ftp, pub, globe, index.html>*, corresponding to the URL *ftp://ftp.cs.vu.nl/pub/globe/index.html*
- For the resolution we can implement two approaches:
  an **iterative** name resolution,
  a **recursive** name resolution.

Implementation of Name Resolution (2)



Iterative name resolution

Implementation of Name Resolution (3)



Recursive name resolution

Implementation of Name Resolution (4)

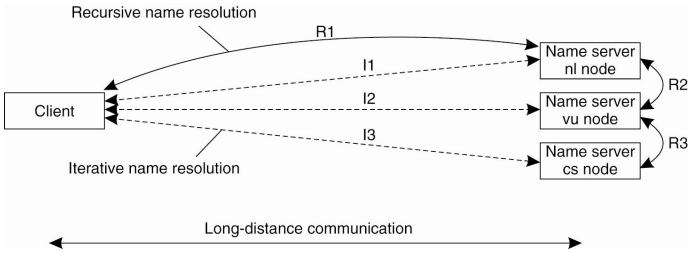
Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	<ftp>	#<ftp>	—	—	#<ftp>
vu	<cs,ftp>	#<cs>	<ftp>	#<ftp>	#<cs> #<cs, ftp>
nl	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
root	<nl,vu,cs,ftp>	#<nl>	<vu,cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>	#<nl> #<nl,vu> #<nl,vu,cs> #<nl,vu,cs,ftp>

Recursive name resolution of *<nl, vu, cs, ftp>*. Name servers cache intermediate results for subsequent lookups.

Implementation of Name Resolution (5)

- Recursive name resolution puts a higher performance demand on each name server.
- As a consequence name servers in the global layer implement only iterative name resolution.
- Recursive name resolution has two advantages:
  - caching is more effective (in the iterative approach caching works only client-side),
  - communication costs may be reduced.

Implementation of Name Resolution (6)



The comparison between recursive and iterative name resolution with respect to communication costs.

Example: The Domain Name System

- The Internet Domain Name System (DNS) is one of the largest distributed naming services.
- DNS is mainly used to lookup IP addresses of hosts in the Internet.
- The DNS name space is a rooted tree, where labels are alphanumeric strings composed by a maximum of 63 characters.
- Pathnames are restricted to 255 characters (lists of labels separated by a dot “.”).
- A subtree is called a domain and the path to its root node is called a domain name.
- Each node contains a collection of resource records, representing several entities.

DNS Implementation

- In general DNS name space can be divided in:
  - global layer,
  - administrational layer.
- The managerial layer is formed by local file systems.
- Each zone is implemented by a name server (replicated for availability).
- Updates are handled modifying the DNS database of the primary server.
- Secondary name servers request the primary server to transfer its contents (zone transfer).
- A DNS database is formed by a small collection of files.
- Thus, a node identifier corresponds to an index in a file.

Figure 13.5
DNS resource records

Record type	*Meaning*	*Main contents*
A	A computer address	IP number
NS	An authoritative name server	Domain name for server
CNAME	The canonical name for an alias	Domain name for alias
SOA	Marks the start of data for a zone	Parameters governing the zone
WKS	A well-known service description	List of service names and protocols
PTR	Domain name pointer (reverse lookups)	Domain name
HINFO	Host information	Machine architecture and operating system
MX	Mail exchange	List of <*preference*, *host* > pairs
TXT	Text string	Arbitrary text

Figure 13.6
DNS zone data records

domain name	*time to live*	*class*	*type*	*value*
dcs.qmul.ac.uk	*1D*	*IN*	*NS*	*dns0*
dcs.qmul.ac.uk	*1D*	*IN*	*NS*	*dns1*
dcs.qmul.ac.uk	*1D*	*IN*	*NS*	*cancer.ucs.ed.ac.uk*
dcs.qmul.ac.uk	*1D*	*IN*	*MX*	*1 mail1.qmul.ac.uk*
dcs.qmul.ac.uk	*1D*	*IN*	*MX*	*2 mail2.qmul.ac.uk*

domain name	*time to live*	*class*	*type*	*value*
www	*1D*	*IN*	*CNAME*	apricot
apricot	*1D*	*IN*	*A*	138.37.88.248
dcs	*1D*	*IN*	*NS*	*dns0.dcs*
dns0.dcs	*1D*	*IN*	*A*	*138.37.88.249*
dcs	*1D*	*IN*	*NS*	*dns1.dcs*
dns1.dcs	*1D*	*IN*	A	*138.37.94.248*

Berkeley Internet Name Domain (BIND)

- implemented in the `named` daemon, in routers, local servers, clients, etc.
- Three categories of name servers:
  - primary server: authoritative servers with information kept in local tables
  - secondary server: download data directly from primary servers, usually 1-2 times per day; useful for failure-tolerance
  - cache-only: no local tables, but only addresses of enough servers to resolve any name. Resolved address are cached

Discussion on DNS

- Quite efficient (short average response time)
- used also for storing other data beside names
- naming data may become inconsistent (time for update of servers can be hours, days)
- no detection of staleness or inconsistency
- still sufficient for most applications (kind of “lowest common denominator” of naming for Internet services)
- Structure of name space quite rigid (cannot be changed, not even locally)

Attribute-based Naming (1)

- In general, flat and structured names provide a unique and location-independent way of referring to entities (in a human-friendly way for structured names).
- However, the increasing amount of information requires effective searching techniques for entities.
- In order to do so, a user can provide merely a **description** of what he is looking for.
- A popular way in distributed systems to describe entities is to provide a collection of *(attribute, value)* pairs (**attribute-based naming**).
- Attribute-based naming systems are also called **directory services**.

Attribute-based Naming (2)

- Entities have a set of associated attributes that can be used for searching.
- In some cases the choice of the right set of attributes is simple:
  e-mail system: sender, recipient, subject, body etc.
- However, in most cases it is a non-trivial task which must be carried out by hand.

Attribute-based Naming (3)

- Storing resource descriptions allows one to query that storage in a way common to attribute-based naming systems.
- **Resource descriptions** are usually stored in a **central location**.
- Resources can be distributed.
- Distributing the descriptions can lead to performance problems.
- Indeed, **lookup** in attribute-based naming systems requires an **exhaustive search** through all the descriptors.
- Formal model: OSI X.500 specification.

Figure 13.10
X.500 service architecture

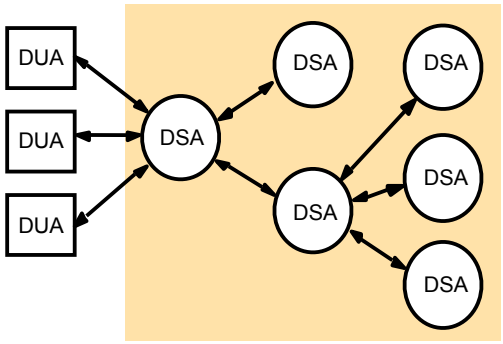


Figure 13.11
Part of the X.500 Directory Information Tree

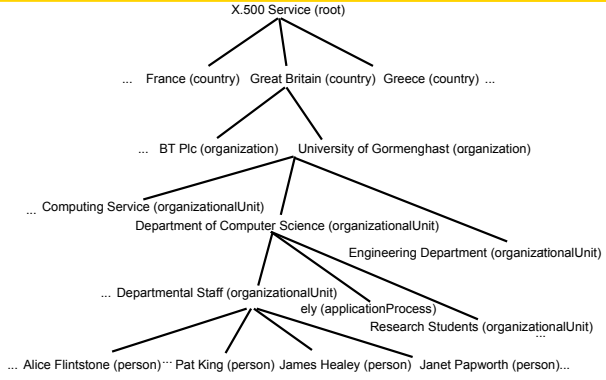


Figure 13.12
An X.500 Directory Information Base (DIB) Entry

info
Alice Flintstone, Departmental Staff, Department of Computer Science, University of Gormenghast, GB

commonName
Alice.L.Flintstone
Alice.Flintstone
Alice Flintstone
A. Flintstone

surname
Flintstone

telephoneNumber
+44 986 33 4604

uid
alf

mail
alf@dcs.gormenghast.ac.uk
Alice.Flintstone@dcs.gormenghast.ac.uk

roomNumber
Z42

userClass
Research Fellow

Hierarchical Implementations: LDAP (1)

- Distributed directory services: combine structured naming with attribute-based naming (e.g., Microsoft Active Directory).
- Many such systems are based on LDAP (Lightweight Directory Access Protocol) derived in turn from OSI X. 500 Directory Service.
- LDAP Directory Service: collection of records (directory entries).
- Each record is a collection of (attribute, value) pairs.
- Each attribute has an associated type and it can be single-valued or multiple-valued (arrays and lists).

Hierarchical Implementations: LDAP (2)

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
CommonName	CN	Main server
Mail_Servers	—	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	—	130.37.20.20
WWW_Server	—	130.37.20.20

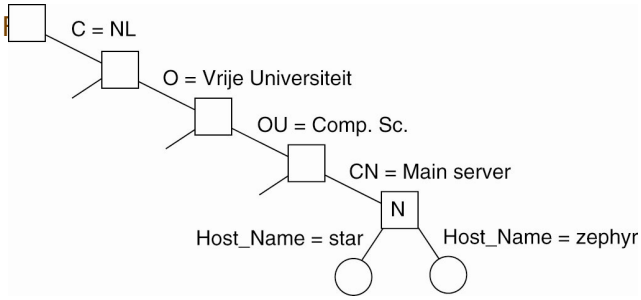
A simple example of an LDAP directory entry using LDAP naming conventions.

Hierarchical Implementations: LDAP (3)

- **Directory Information Base** (DIB): the collection of all directory entries in an LDAP directory service.
- Each record is uniquely named (so that it can be looked up).
- Globally unique name: sequence of naming attributes in each record.
- Each naming attribute is called a **Relative Distinguished Name** (RDN).
- In the previous example *Country*, *Organization* and *OrganizationalUnit* can be used to form the globally unique name:
  /C=NL/O=Vrije Universiteit/OU=Comp. Sc.
  (analogous to the DNS name *nl.vu.cs*).
- Listing RDNs in sequence leads to a hierarchy of directory entries: **Directory Information Tree** (DIT).

59

Hierarchical Implementations: LDAP (4)



Each node:
- represents a directory entry;
- may also represent a "traditional directory", having several children.

Hierarchical Implementations: LDAP (5)

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

(b)

Two directory entries having *Host_Name* as RDN.

Hierarchical Implementations: LDAP (6)

- The distinction between LDAP record and traditional directory is reflected into the lookup operations:
  - read: it reads a single record given its pathname in the DIT;
  - list: it returns the names of all outgoing edges of a given node in the DIT.

read("/C=NL/O=Vrije Universiteit/ OU=Comp. Sc./CN=Main Server")

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
CommonName	CN	Main server
Mail_Servers	—	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	—	130.37.20.20
WWW_Server	—	130.37.20.20

list("/C=NL/O=Vrije Universiteit/ OU=Comp. Sc./CN=Main Server")

star
zephyr

Hierarchical Implementations: LDAP (7)

- Implementing LDAP systems is similar to DNS systems (LDAP supports more lookup operations).
- In large-scale directory systems **DIT** is usually **partitioned** and **distributed** across servers called Directory Service Agents (DSAs) -- analogous to name servers of DNS zones.
- Clients are called Directory User Agents (DUAs) -- analogous to name resolvers.
- **LDAP** supports **more facilities to lookup** through a DIB.
- For instance we can specify to search for all main servers at the Vrije Universiteit using the following query:

answer=search("&(C=NL)(O=Vrije Universiteit)(OU=*)(CN=Main server)")

Hierarchical Implementations: LDAP (8)

- In general **searching** in a directory service is an **expensive operation** (many DSAs and leaf nodes to access).
- A further step is to allow several trees to coexist and to be cross-linked. This is the approach of Microsoft Active Directory where **forests** of LDAP domains can be implemented.
- To minimize the scalability problems, there is a **global index** server which can address to single LDAP domains.
- Moreover, it is possible to **combine LDAP with DNS**, allowing the root of the domain (domain controller in Active Directory) to be known under a DNS name.