Distributed Systems

Naming and Directory Services

Tenshi Hara
tenshi.hara@tu-dresden.de
https://www.rn.inf.tu-dresden.de/hara
Outline

- Definitions
- Requirements
- Basic Terms & Name Structures
- Implementation Techniques
- System Examples
- Summary
Definitions – Naming & Directory Services

- **Naming Service**
  - Mapping of logical names to physical addresses or object references
  - Logical names given by user (location independent)
  - Service returns the addresses (location dependent)

- **Directory Service**
  - Management of attributes of the named instances
  - Attribute-based search for named instances
  - Similar to ‘Yellow Pages’

Requirements:
- Mapping of logical names/attributes to addresses
- Registration of logical names and addresses
- Special system support
  → Distributed Directory Service
Requirements to the name structure

- unique names within the system
- multi-stage names (‘user@example.com’) 
- attributed names (‘host_x:CPU=Dual, PRT=Ipr, LOC=rz’) 
- group names (‘Workgroup’ ➔ [‘jdoe’,‘pmiller’,‘asmith’])
- flexible binding with roles and groups; e.g., Role: Customer Services Representative. → Name bound to currently available employee
- alias names
Requirements to Directory Services

- **scalability, reliability, fault tolerance:**
  - distributed, decentralized name management (e.g., one server per department of a company)
  - distributed protocol for name interpretation
  - replication of name tables (efficiency, fault tolerance)

- **performance:**
  - result caching → reduce amount of queries needed
Basic Terms & Name Structures

- Address: explicit, physical object notation (‘#346’)  
- Name: logical object notation, mostly location independent (‘printer’)  
- Interpretation: mapping of name to address (‘printer at #346’)  
- Context:  
  - Interpretation of a component of a multi-part name; e.g., user@host: user is interpreted in the context of host  
  - Disambiguation of same name in different namespaces  
- Namespace: set of contexts; e.g., ‘<user>@<computer-node>’  
- Relative names: interpretation dependant on specific context  
- Absolute names: context independent
Hierarchical name space:

- Context: "root"
- Context: "Customer"
- Context: "Private Customer"
- Context: "Business Customer"

Example (Unix, NFS): `/Customer/PrivateCustomer/CustomerX’

Flat name space:

- Context "user names"

Example (local operating system): ‘CustomerX’
Context & Name Interpretation (for `Customer_X@host_X`)
Context & Name Interpretation: Approach to interpretation

- Interpretation of each name component via related context
- Result: mapping of components onto an address + a new context
- Assignment of different contexts to different name servers

⇒ decentralised, distributed interpretation
Context & Name Interpretation: Combination of Namespaces (Junctions)

- Combination of several namespaces into one
- Management of individual namespace by dedicated directory service
- Further directory service called with transition to a subordinate namespace
- E.g., interpretation of global file name
  - First, internet domain name server finds appropriate fileserver,
  - Then further name interpretation takes place within the fileserver
Basic conceptions

- hierarchical name spaces
- Assignment of one or several contexts to each name server
- each name server knows the sub-ordinate and super-ordinate servers
- distributed interaction of the name servers for name interpretation
  - no expensive broadcast necessary
  - name servers work relatively autonomous
Example of a name space with related name servers:

- Name Server S1: Context "Company Names"
- Name Server S2: Context "Online Trader"
- Name Server S3: Context "Supplier"
- Name Server S4: Context "Warehouse Management"
- Name Server S5: Context "Customer Management"
- Name Server S6: Context "Ordering"

- interpretation of name dependent on contacted name server
- e.g., name in context of warehouse management on S4
Example of a name space with related name servers:

Query processing: query S4 for ‘ordering.supplier.com’ (S6)
- Direct – Query sent directly to S6
- Indirect – Query sent first to different server
  - Chaining: e.g., S4→S2→S1→S3→S6, then result sent back along chain
  - Referral: client given address of next server to query → if client already knows servers, this can improve response time with multicast
Caching – Goals and Approaches

- Performance improvement via re-use of precedent query results
  - complete names (specific, fast resolution)
  - partial name (more general, more requests)

- Caching via clients of the Directory Service:
  - parts of an interpreted name and address of the interpreting name server of subordinate level
  - example of a cache-record: ("OnlineTrader: CustomerMngt", "S5")
    → allows direct querying of the lowest subordinate server (here ‘S5’)

- Caching via name server directly
  - records: name context and addresses of all servers on the same level
  - reduction of one level

- Timestamps to recognise and discard obsolete cache entries
Caching - Example

Server Cache: "Supplier", "S3"

Name Server S1: Context "Firm_names"

Name Server S2: Context "OnlineTrader"

Name Server S3: Context "Supplier"

Name Server S4: Context "WarehouseMngt"

Name Server S5: Context "CustomerMngt"

Name Server S6: Context "Ordering"

Server Cache: "OnlineTrader", "S2"

Client Cache: "OnlineTrader: CustomerMngt", "S5"
Context Replication

- **Goals**: fault tolerance, locality of queries, load balancing

- **Approach**:
  - implementation of a context via several name servers
  - selection of an alternative server after timeout by a query

- **Consistency between replicas**
  - Assumption: name modifications occur rarely
    - Toleration of temporary inconsistencies between replicas
    - simple update process
  - Modifications performed on primary server
    - then notification of replicas
  - Convergence to consistence state within reasonable timeframe
    - no strict consistence → possibility of obsolete results
    + update process does not require concurrent availability of all replicas
Context Replication – Example

- S2 and S3 replicate each others’ contexts
- reduces need for forwarding
  - e.g., query for ‘OnlineTrader’ sent to S3 processed without further forwarding
  - i.e., no need for S3→S1→S2 because ‘OnlineTrader’ replicated on S3 and S2
Context Replication – Fine-grained Replication

- Selection of contexts to be replicated
- many can be replicated on one server, or fewer may be replicated due to
  - update expense
  - performance
  - effort for maintaining consistency

Administration Area

Server 1
/Firm_names
/Firm_names/
  OnlineTrader/...
/Firm_names/
  OnlineTrader/
  CustomerMgmt/...
/Firm_names/
  Supplier/...

Server 2
/Firm_names/
  Supplier/...

Server 3
/Firm_names/
  OnlineTrader/...
/Firm_names/
  CustomerMgmt/...
/Firm_names/
  OnlineTrader/
  WarehouseMngt/...
Relatively extensive administration tasks
- Installation and monitoring of server processes
- Definition of name space structure
- Setting of access control mechanisms
- Replication control of name records
- Reconfiguration of name spaces

Tools
- simple command interface
- Vendor-specific tools (also remote management)
- Lack of orientation towards standards
Existing Systems

- Internet Domain Name System (DNS)
- X.500/LDAP Standard – Products:
  - Novell eDirectory
  - Microsoft Active Directory
- Java Naming & Directory Interface (JNDI)
**Internet Domain Name System (DNS)**

- **Naming service** of the Internet
  - Mapping of domain names to IP addresses
  - Queries for services such as SIP server or RTP gateway

- Hierarchical namespace – organisation: contexts (domains)

- Domain Name – path along/composition of contexts
  - e.g., order.supplier.com.

- Name assignment by ICANN (Internet Corporation for Assigned Names and Numbers) → in general, delegated to authorised companies

- No single name server for DNS namespace (no single point of failure)

- Organised into non-overlapping zones (different from domain)
Example domain name: order.supplier.com.

Cached Secure DNS (DNSsec) Zone
(TLD DNS Servers all know each other)

Root "."

Special: Infrastr. Top Level Domain

.it
.uk
.de
.com
.net
.org
.edu
... (iTLD)

General Top-Level Domains (gTLD)

.onlinetrader

.country-top-level-domains (ccTLD)

.order

.customer-management

.supplier

.customer-management

.warehouse-management
storing information with names
→ typically, host » IP address
→ in general, <Name, Lifespan, Class, Type, Value>

Lifespan: validity of entry (expires after lifespan)

Class: separation of records into different databases
• e.g., "IN" for Internet Information, "NS" for Name Server Delegation

Types: certain defined set of records for each class; e.g., "IN" class:
  o A  IPv4 address for given name
  o AAAA IPv6 address for given name
  o NS  Name Server for a particular domain
  o MX  Mail Exchange for a particular domain
  o CNAME Mapping of alias names to domain names
  o PTR Alias for an IP address (Pointer)
  o SRV Service Record (e.g., SIP) for a particular domain
<table>
<thead>
<tr>
<th>Domain Names</th>
<th>Lifespan</th>
<th>Class</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>86400</td>
<td>IN</td>
<td>SOA</td>
<td>Zone Data</td>
</tr>
<tr>
<td></td>
<td>86400</td>
<td>IN</td>
<td>NS</td>
<td>ns1</td>
</tr>
<tr>
<td></td>
<td>86400</td>
<td>IN</td>
<td>NS</td>
<td>ns2</td>
</tr>
<tr>
<td>warehousemngt</td>
<td>86400</td>
<td>IN</td>
<td>NS</td>
<td>ns3.warehousemngt</td>
</tr>
<tr>
<td>ns3.warehousemngt</td>
<td>86400</td>
<td>IN</td>
<td>A</td>
<td>203.0.113.3</td>
</tr>
<tr>
<td></td>
<td>86400</td>
<td>IN</td>
<td>MX</td>
<td>10 server17</td>
</tr>
<tr>
<td></td>
<td>86400</td>
<td>IN</td>
<td>MX</td>
<td>50 server18</td>
</tr>
<tr>
<td>www</td>
<td>86400</td>
<td>IN</td>
<td>A</td>
<td>203.0.113.100</td>
</tr>
<tr>
<td>mail1</td>
<td>86400</td>
<td>IN</td>
<td>CNAME</td>
<td>server17</td>
</tr>
<tr>
<td>mail2</td>
<td>86400</td>
<td>IN</td>
<td>CNAME</td>
<td>server18</td>
</tr>
<tr>
<td>server17</td>
<td>86400</td>
<td>IN</td>
<td>A</td>
<td>203.0.113.117</td>
</tr>
<tr>
<td>server18</td>
<td>86400</td>
<td>IN</td>
<td>A</td>
<td>203.0.113.118</td>
</tr>
<tr>
<td>external.mail</td>
<td>3600</td>
<td>IN</td>
<td>CNAME</td>
<td>mx.example.com</td>
</tr>
</tbody>
</table>

- **SOA**: Start of Authority record
- **NS**: Name Server record
- **MX**: Mail Exchange record
- **A**: Address record
- **CNAME**: Canonical Name record
- **ttl**: Time to Live

**Example Domain Name: example.com**

- **master.example.com**: Primary nameserver
- **example.com.serial**: Serial number
- **example.com.refresh**: Time to refresh
- **example.com.retry**: Time to retry
- **example.com.expire**: Time to expire
- **example.com.ttl**: Time to live

**Priority**: The priority of MX records determines the order of mail delivery.
DNS namespace is split up into zones
- non-overlapping
- not necessarily similar with domain

One or multiple name servers per zone
- primary name server maintains data base for zone
- secondary name servers sync with primary name server based on zone transfer

exemplary zone definition:
company branch ‘warehouse-management’ has its own zone
The 13 root name servers (a-m) are operated by 12 independent organisations. Replicated servers are organized in anycast groups; i.e., a set of spatially distributed root servers with the same content share an IP address. Name-scheme: [a-m].root-servers.net
The 13 root name servers (a-m) are operated by 12 independent organisations. Replicated servers are organized in anycast groups; i.e., a set of spatially distributed root servers with the same content share an IP address. Name-scheme: [a-m].root-servers.net
Domain name to be resolved: **ubicomplab.cs.washington.edu**
- **International Telecommunications Union (ITU) Standard for a directory service**
- **products based on X.500:**
  - Novell eDirectory
  - Microsoft Active Directory
- **hierarchical name structure (RFC 4512)**
  - information held in the directory is collectively known as the **Directory Information Base (DIB)**
  - basic unit of information is called **Directory Entry**
  - set of entries representing the DIB are organised hierarchically in a tree structure known as the **Directory Information Tree (DIT)**
- attribute-based description and search for entries
• defined by Object Classes
  • identify the set of attributes for entries of a class (required or optional)
  • ‘top’ – root class (abstract class)
  • multiple inheritance allowed
  • Three types of classes:
    o **Abstract**: can’t be instantiated, can only derive from other abstracts
    o **Structural**: derive (directly or indirectly) from ‘top’ by subclassing abstract or structural object classes
    o **Auxiliary**: add attributes to ‘base’ class, similar to Java interface

• Attributes are name/value pairs of defined type
  • single-valued or multiple-valued (arrays or lists)
  • syntax and matching rules for values
  • e.g., UID=12345 or OU=Engineering

• Directory entries can only have one structural class and may have any number of auxiliary classes
Directory Information Tree (DIT)

- C=uk
- C=de
- L=Dresden
- O=TU-Dresden
- O=supplier
- OU=Informatik
- OU=inf
- O=onlinetrader
- OU=warehousemngt
- OU=customermngt
- SN=Customer_X
- CN=Firm_Y
- L=Frankfurt
- L=Dresden
- OU=ordering
- OU=customermngt

Legend:
- Country (C)
- Organization (O)
- OrganizationalUnit (OU)
- Location (L)
- Alias
- organizationalPerson
- residentialPerson
- Device
Directory Information Tree (DIT)

Directory Entry: Customer_X

<table>
<thead>
<tr>
<th>object class</th>
<th>attribute type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>c</td>
<td>DE</td>
</tr>
<tr>
<td>organization</td>
<td>o</td>
<td>onlinetrader</td>
</tr>
<tr>
<td>organizationalUnit</td>
<td>ou</td>
<td>customermngt</td>
</tr>
<tr>
<td>residentialPerson</td>
<td>sn (person)</td>
<td>Customer_X</td>
</tr>
<tr>
<td>residentialPerson</td>
<td>street</td>
<td>Hauptstrasse 23</td>
</tr>
</tbody>
</table>

Country (C)  Organization (O)  OrganizationalUnit (OU)  Location (L)  Alias
organizationalPerson  residentialPerson  Device
The ‘residentialPerson’ object class is the basis of an entry that includes a person’s residence in the representation of the person. (Source: X.521 [X.521])

```plaintext
( 2.5.6.10 NAME 'residentialPerson'
    SUP person ; (superclass is person with attributes
                 ; surname and commonName)
    STRUCTURAL
    MUST l
    MAY ( businessCategory $ x121Address $
          registeredAddress $ destinationIndicator $
          preferredDeliveryMethod $ telexNumber $
          teletexTerminalIdentifier $ telephoneNumber $
          internationalISDNNumber $ facsimileTelephoneNumber $
          preferredDeliveryMethod $ street $ postOfficeBox $
          postalCode $ postalAddress $
          physicalDeliveryOfficeName $ st $ l )
)
```

...
Relative Distinguished Name (RDN)
- individual component of a DN
- relative to super ordinate entry
- formed from attributes of the entry
- must be unique within level of directory tree
- exemplary RDN: CN="Customer_X"+Street="2301 Wall St."

Distinguished Name (DN) (comparable with DNS domain name)
- composite name (CN) for individual entry from RDN
- put together in order starting with the directory tree root
- unique within whole directory tree (absolute name)
- exemplary DN: SN="Customer_X"+Street="2301 Wall St.", OU=customermngt,O=onlinetrader,C=de,DC=example,DC=com
- DC (Domain Component) represent the top of an LDAP tree that uses DNS to define its namespace
- Lightweight Directory Access Protocol (LDAP)
  - Simplified protocol to access X.500 directories
- Quasi-standard to access Directory Services, based on TCP/IP
- Widespread vendor support, widely used in practice
- Front-end for heterogeneous Directory Servers (e.g., X.500, Microsoft Active Directory, Novell eDirectory)
- Typical operations for name search and name manipulation → bind, search, compare, add, delete, modify
- Optimisation of access functions
- Automatic forwarding of queries between Directory Servers with distributed information repositories
- Authentication, Encryption, and Integrity for Directory Accesses
Example: Standalone LDAP Server

**Binding**
- Session establishment with LDAP-Server → client specifies host and TCP/IP-Port of server
- Authentication: anonymous session or authentication with username/password

**Search Request**
- **Specify base:** part that is searched
- **Specify search filter:** C=de+OU=customermgmt

**Search Result**
- SearchResultEntry 1: DN: SN=Customer_X,OU=customermgmt,O=onlinetrader,C=de
- SearchResultEntry 2: DN: OU=customermgmt,L=Dresden,O=supplier,C=de

**Unbinding**
- Session with server is closed
- Replication across several directory servers
- Name interpretation: chaining, referral or multicast protocols
- Communication:
  - Directory System Protocol (DSP)
    - interaction between directory servers
    - (DSA = Directory System Agents)
  - Directory Information Shadowing Protocol (DISP)
    - Definition of replication between master- & slave-server
  - Directory Operational Binding Management Protocol (DOP)
    - Definition of administrative communication between two DSAs
  - Directory Access Protocol (DAP)
    - client access to directory servers
    - communication between DUA (Directory User Agent) and DSA
    \[\rightarrow\] simplified DAP: Lightweight Directory Access Protocol (LDAP)
Directory System Protocol (DSP)
Directory Information Shadowing Protocol (DISP)
Directory Operational Binding Management Protocol (DOP)
Directory Access Protocol (DAP)
Lightweight Directory Access Protocol (LDAP)
Java Naming & Directory Interface

- Directory service interface for Java applications
- used together with RMI and search for application servers within the binding process
- designed independently of directory service implementations
- abstract interfaces to access different directory services
  - JNDI API – access to directory information for applications
    - functions for event handling
    - LDAP (v3) extensions
  - JNDI SPI (Service Provider Interface)
    → integration of diverse directory service implementations
- abstract interfaces can be integrated into composite namespaces
JNDI Context Interface – basic operations for name contexts

```java
public interface Context {
    public Object lookup(Name name) throws NamingException;
    public void bind(Name name, Object obj)
        throws NamingException;
    public void rebind(Name name, Object obj)
        throws NamingException;
    public void unbind(Name name) throws NamingException;
    public void rename(Name old, Name new) throws NamingException;
    public NamingEnumeration listBindings(Name name)
        throws NamingException;
    ...
    public Context createSubcontext(Name name)
        throws NamingException;
    public void destroySubcontext(Name name)
        throws NamingException;
    ...
}
```
JNDI Directory Interface for

- examining and updating attributes
- searching the directory

```java
public interface DirContext extends Context {
    public Attributes getAttributes(Name name)
        throws NamingException;
    public Attributes getAttributes(Name name, String[] attrIds)
        throws NamingException;
    ...
    public void modifyAttributes(Name name, int modOp,
        Attributes attrs) throws NamingException;
    public void modifyAttributes(Name name,
        ModificationItem[] mods) throws NamingException;
    ...
}
```
Summary

Distributed Naming & Directory Services:

- Important for search and management of resources
- Mapping of names to addresses
- Hierarchically structured name spaces
- Multi-level mapping of names
- Distributed name servers – fault tolerant and scalable

Important Standards and Quasi-Standards:

- ISO/OSI 9594 / ITU X.500: directory service with additional attribute management
- Internet Domain Name System: Capable, worldwide available example of naming service
- LDAP/JNDI: interfaces for portable implementations
References

- Albitz, P.: DNS and BIND. O’Reilly 2006
- Mockapetris, P.: Domain names - implementation and specification IETF RFC1035, 1987