The UNIVERSITY *of* EDINBURGH
SCHOOL *of* INFORMATICS

CS4/MSc

Distributed Systems

Jane Hillston

`jeh@inf.ed.ac.uk`

Room 1604

(Lecture 15: Naming and Discovery, 27th November 2003)

Naming and Discovery Services

As we have seen, the notion of service is crucial in a distributed system, to allow processes to manage access to shared resources of various forms. An aspect of this which we have not previously considered in detail is how a process can find the service that it requires, particularly in a dynamic environment.

At the most basic level we might have the name of a service and need to find out more about it — most crucially its location, but sometimes other properties also. A **name service** stores a collection of bindings between **names** and **attributes**. It offers an interface that allows the attributes associated with a name to be retrieved, as well as to bind attribute and name pairs. In recent years requirements on name services have included *unification*, *integration* and *scalability*.

A directory service also stores (name, attribute) pairs, but allows the collection to be queried on the attributes rather than the names.

Name spaces

- A **name space** is a collection of all valid names recognised by a particular service. Here validity is a characteristic of the presentation of the name. Any particular valid name may or may not be **bound** by the naming service.
- Most collections of names are internally structured, typically hierarchically, and names themselves may be structured, using **contexts**, to represent their location within the structure; e.g. UNIX file names.
- Hierarchic name spaces allow names to be interpreted within different contexts, meaning that the same name can have different meanings in different contexts. This has great benefits for scalability.
- When a hierarchic name space is used **relative names** may be used, as for example, in UNIX files or URLs.
- An **alias** may be used, particularly within a hierarchic name space, to allow a simple name to be substituted for a more complicated one. It may also be used to provide transparency.

Naming domains

- Large name spaces may be highly distributed and too large to be managed centrally. In this case the space may be divided administratively into a number of **naming domains**.
- The responsible authority then has control over the names which are **bound** within that domain.
- For example within DNS a domain's name is a common suffix of all the domain names within it: e.g. `inf.ed.ac.uk` is the sub-domain managed by the School of Informatics, while `ed.ac.uk` is the naming domain managed by the University of Edinburgh. Thus, administratively responsibility may be devolved to a sub-domain within a naming domain.
- Naming data belonging to different naming domains are usually stored by distinct name servers managed by the corresponding authorities.
- This means that name resolution may need to be carried out iteratively and across different servers.

Name resolution (1)

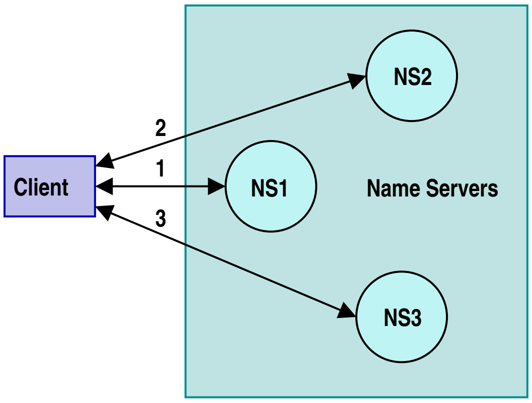
- The process of accessing the attributes associated with a name is termed **name resolution**.
- Typically name resolution is iterative:
  - a name is presented to a naming context;
  - the naming context either returns an appropriate set of primitive attributes (if the name can be resolved directly) or
  - the naming context returns a further naming context and a derived name which can be resolved by that context.

  Thus each context descends a level through the hierarchy.
- At best, the use of an alias introduces another level of resolution, as the alias is resolved to the corresponding name. At worst it can introduce cycles into the resolution calls.

Name resolution (2)

- When the name space is very large and/or it has a very large number of potential users, the name service will not be based on a single server for obvious reasons (performance, scalability, single-point-of-failure...)
- However, partitioning the data means that the local name server cannot satisfy all resolution requests, and must seek the help of other name servers.
- Finding the appropriate name server is termed **navigation**, and differing strategies can be employed for this.
- For example, the navigation may be under the control of the client, which may contact a number of name servers either iteratively, or via multicast; or the navigation may be under the control of the first name server contacted by the client.
- Access to name servers in different administrative domains may be restricted and this can influence the choice of strategy.

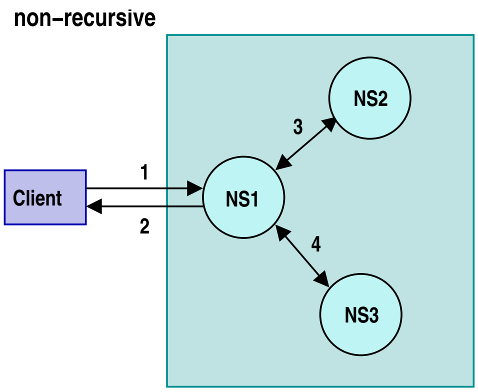
Client-controlled navigation

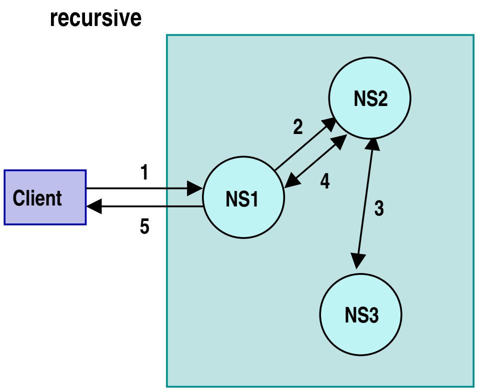


Using **iterative navigation**, a client presents a name to the local name server, which attempts to resolve it. If the local name server has the name, it returns the result immediately. If it does not, it will suggest another name server which will be able to help. Resolution proceeds at the new server, with further navigation as necessary until the name is located or found to be unbound.

Using **multicast navigation**, a client presents the name to a group of name servers via multicast. Only the server that can carry out the resolution responds. This has the disadvantage that there is no response if the name is not bound.

Server-controlled navigation





An alternative approach is to allow a name server to coordinate the search. This server communicates by multicast or iteratively with its peers, as though it were a client (**non-recursive** server-controlled navigation). In **recursive** server-controlled navigation, the client again contacts a single server. If the server cannot resolve the name, it contacts a peer storing a prefix of the name, which in turn attempts to resolve it.

The Domain Name System (DNS)

The DNS was designed in the late 1980s to provide a naming database to be used across the Internet. It is used primarily for computers — the attributes stored are generally IP addresses — but in principle any type of object can be named and bound.

The system is effective because it provides a hierarchical partition of the database, the naming data is replicated and caching is used extensively.

It is primarily used for two forms of query:

Host name resolution: this involves finding the IP address for a given host name, e.g. `clootie.inf.ed.ac.uk`. Many applications (web browsers, ftp, telnet, etc.) will make a DNS query to convert a given host name before establishing connection via the returned IP address and a standard port.

Mail host location: Email software uses the DNS to resolve domain names into the IP addresses of mail hosts.

DNS Servers

- DNS naming data is divided into **zones**.
- Each zone has at least two name servers that provide *authoritative* data for the zone.
- The zone contains names and attributes for a domain, except for any separately administered sub-domains: e.g. a zone containing `ed.ac.uk` does not contain information about `inf.ed.ac.uk`. The zone name servers will know the name servers which hold authoritative data about the sub-domains.
- System administrators enter the data for a zone into a **master file**. A **primary** server reads zone data directly from the master file. A **secondary** server downloads zone data from a primary server, and checks periodically for updates. Both types of server are considered to be authoritative.
- The management parameters, such as caching and replication policy, will be set within each zone.

DNS Clients

- A DNS client is called a **resolver**.
- It is normally implemented as library software, which accepts queries, formats them into messages expected under the DNS protocol and communicates with one or more name servers in order to satisfy them.
- The DNS architecture allows for both iterative and recursive navigation: the resolver will specify the navigation to be used when it contacts a name server. However recursive navigation is not necessarily supported by all servers.
- A simple request-reply protocol is used, typically using UDP packets on the Internet.
- The DNS protocol allows a resolver to submit multiple queries in a single request, and similarly, receive multiple replies within a single response.

Naming in CORBA (1)

As we saw earlier in the course, the CORBA Naming Service allows object implementations to be identified by name. Each binding in stored by the Name Service is a mapping between a textual name and an object reference. Contexts are provided to allow the name space to be hierarchically structured.

The use of textual names rather than object references has two major advantages:

- textual names are much easier for human users to deal with;
- the name used for an object can be persistent across different instantiations of the server, although the object references will change each time the object is instantiated.

In fact the Name Service allows names to represented in two formats: an internal format and an external "stringified" format. Furthermore a URL format may be used to fully specify the location, or location and identity, of an object.

URL CORBA references(1)

$$\underbrace{\texttt{corbaloc:}}_{a}\underbrace{\texttt{iiop:1.1@}}_{b}\underbrace{\texttt{host1.company2.com}}_{c}\underbrace{\texttt{/Test/objectA}}_{d}$$

a The first part is a scheme identifier, which may be `corbaloc` or `corbaname`.

b Optionally, the protocol and version number may be supplied. By default this is `iiop`. The alternative, `rir`, is used to signify that `resolve_initial_references`, the bootstrapping method of the ORB, should be used.

c This part is the object address which can be supplied as a DNS host name (as here) or as an IP address. A port number may also be given but when it is absent the default port 2809 is assumed.

d The final component of a location is the object key, consisting of the POA name and an object ID.

URL CORBA references (2)

A `corbaname` URL is a concatenation of a `corbaloc` URL and a stringified object name. The two parts are separated by a "#" character. Thus in the URL:

$$\texttt{corbaname::}\underbrace{\texttt{h3.com4.com:4050}}_{a}\underbrace{\texttt{/MyPOA/Context1}}_{b}\underbrace{\texttt{\#public/MyService}}_{c}$$

a This is the host and port number through which the context server can be reached: port number `4050` on the host `h3.com4.com`

b This is the naming context: `Context1` in the POA `MyPOA`

c This is the name of the object

The method to construct a URL reference is provided in the interface of `NamingContextExt` (`to_url()`) which also provides methods `to_string()` and `to_name()` to convert between names and their stringified form.

Name resolution in CORBA

Each name consists of a one or more `NameComponents`, each of which is an atomic string. Components are separated by "/", and may have an optional `kind` suffix. For example

```
outercontext.ctx/innercontext.ctx/MyServer.service
```

When a `NamingContext` is asked to resolve a name using the `resolve(n)` method it proceeds recursively:

- it resolves the first component of the name `n` to an object reference;
  - if there are no remaining components in the name, the object reference is returned;
  - otherwise, the object reference is narrowed to a `NamingContext` and the remaining components are passed as argument to the `resolve()` method of this `NamingContext`.

Directory Services

A directory service can be considered to be the dual of a name service — for a particular $< name, attribute >$ pair the *attribute* is used to lookup the *name*.

It can be argued that attribute-based lookup is more flexible:

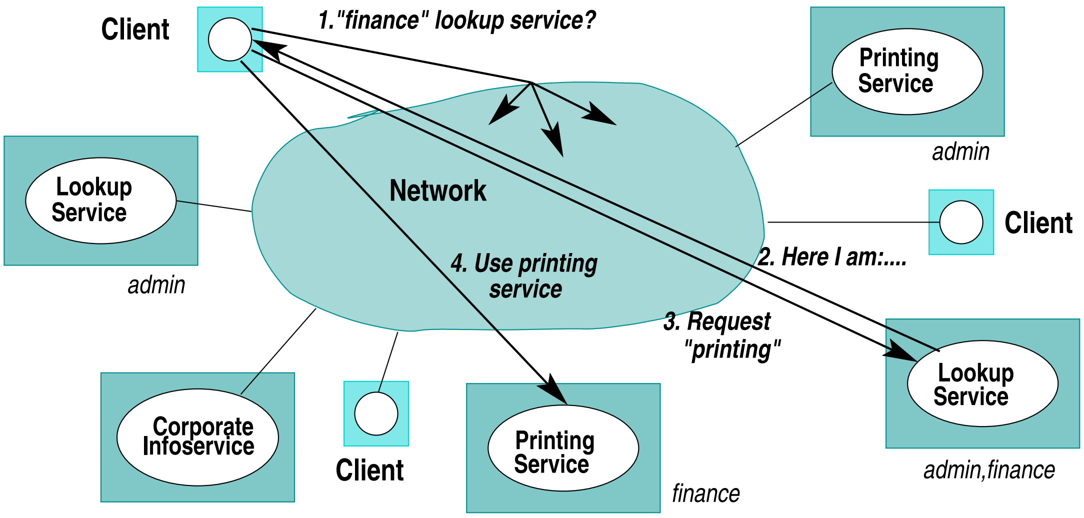
- clients can specify exactly the attributes required even when the name is not known;
- an equivalent, but differently named entity can be transparently substituted after the demise of a particularly named entity;
- the internal structure of organisation is not exposed when organisationally structured names are not used.

A **discovery service** is a directory service which registers information about a dynamically changing environment. For example, the DISCO discovery service allows a client to find details of available web services. Similarly a discovery service may provide brokerage in a spontaneous networking environment.

Discovery Services

- For example the purpose of a discovery service may be to accept and store details of services that are available on the network (**registration service**) and to respond to queries from clients about them (**lookup service**).
- Services are registered as ⟨**name, attributes**⟩ pairs and matching of requests to services will be done on the basis of the attributes. One of the attributes is a location which can be returned in response to a client request, allowing the client to subsequently to communicate with the service directly.
- The extent of a network covered by such a discovery service is called its **scope** and this is often defined in terms of connectivity to a particular network. This is generally appropriate because clients in a spontaneous network usually want to access services which are physically close by.
- **Jini** is a Java-based discovery system designed to be used for spontaneous networking. It assumes that JVMs run in all the hosts and communication is by means of RMI.

Jini discovery service



A Jini client discovering and using a printing service. The required service is within the **group** “finance”. To find the appropriate lookup service the client **multicasts** to the network. The lookup service responds with its unicast address for accepting requests.

Jini discovery service (2)

- Jini discovery makes use of both multicast (to find the lookup service) and iterative (to find the service it requires) navigation.
- The matching of client requests can be based on attributes or on Java typing: for example, a client can request a fax machine to which it has the corresponding Java interface.
- Clients and servers may listen to the discovery multicast address to find out about new lookup services which may be instantiated. When a new lookup service wants to announce themselves they send a datagram to the multicast group.
- Once the client has locates a lookup service it communicates directly with it using RMI.
- Registration with a lookup service is uses **leases**, so that a registration is for a limited period and must be renewed after that period. This is intended to ensure to protect clients from being given out of date information (for servers that have crashed).

CORBA trading service

The CORBA trading service is intended to allow a client to select from a number of *equivalent* object implementations, according to client-defined criteria.

A Trader is a repository of object references. Each object reference is part of a **service offer**. A service offer consists of an IDL interface type and a set of property values.

An Exporter is a server (or other software agent) which places a service offer into a trader. A service offer is stored as a **service type** consisting of a name, an interface type and a set of property specifications.

An Importer is a client which queries a trader in order to find a service. The importer's query includes a specification of the service type and a constraint expression over the properties of the service.

The role of the trader is to match a importer's criteria with an appropriate service offer.