RMI Jini
Web naming
XML JSON

ID2010

Java RMI

Remote Method Invocation

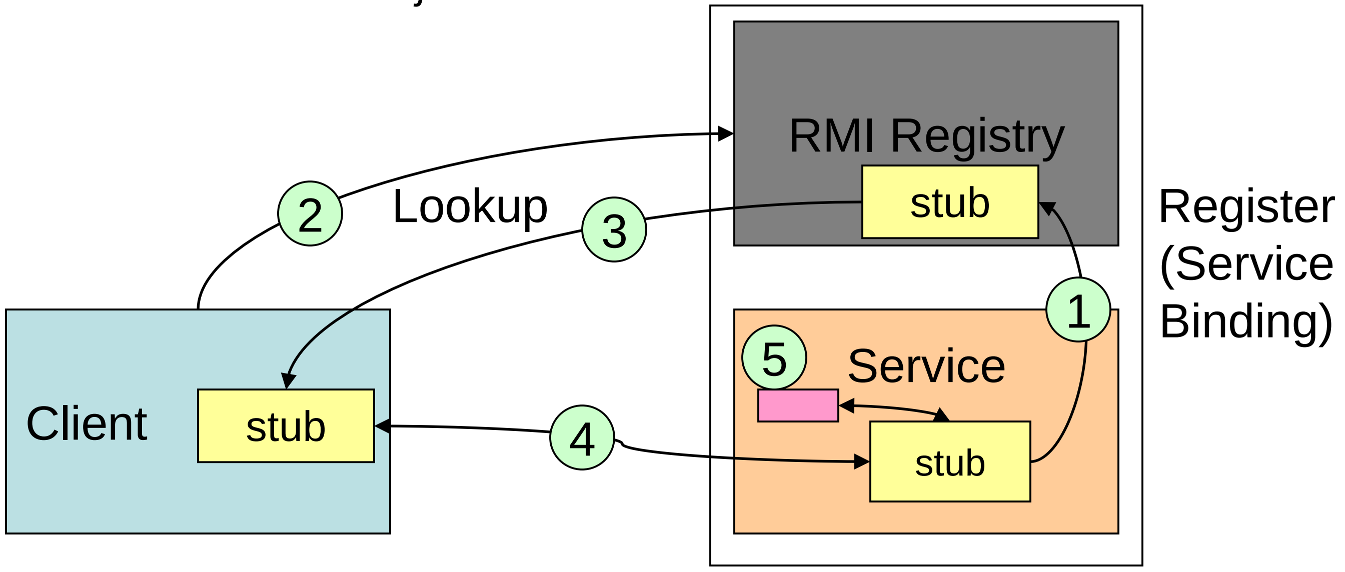
Remote Method Invocation

- RMI is remote procedure call for Java
- A service
  - implements a well-known interface
  - registers with the RMI registry
- A client
  - locates the host and the RMI registry
  - retrieves an object implementing the interface
  - makes method calls on the object
  - method calls are transferred to the service

Java RMI Registry

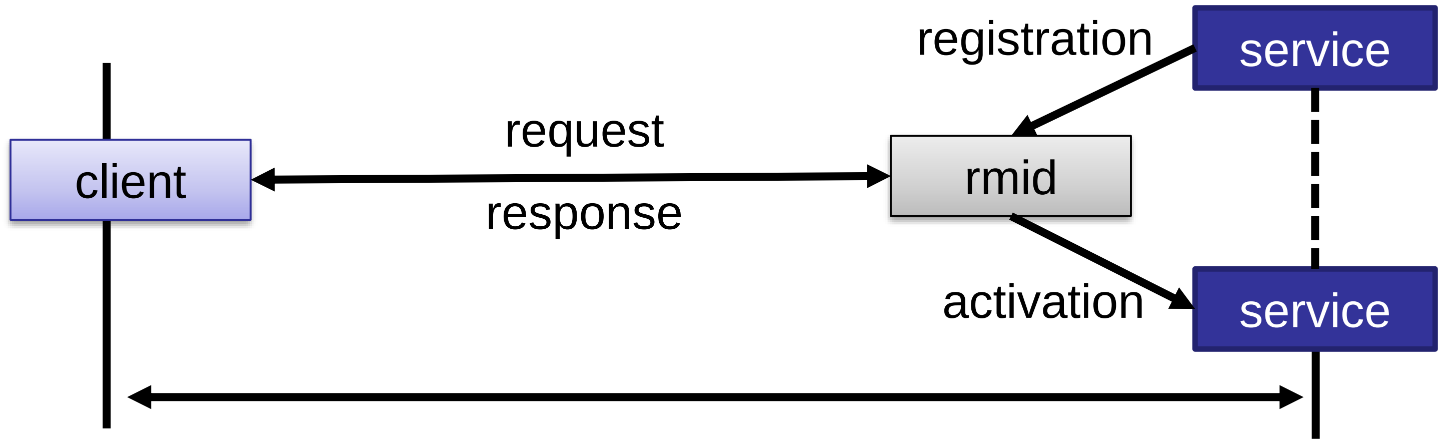
- Clients get the service location directly from the registry
- RMI Registry is known to both RMI Client and RMI Server

http://download.oracle.com/javase/tutorial/rmi/TOC.html



Java RMID

- Java RMI Activation Daemon
- An activatable service registers with rmid
- rmid listens for requests for the service
- A request is serviced by
  - activating the service
  - connecting the client with the service



RMI in Java

- Programming with native RMI is complex
  - The Hello example
  - `https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html`

Hello – service interface

```
package example.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
```

Hello – service interface

```
package example.hello;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Hello {

    public Server() {}

    public String sayHello() {
        return "Hello, world!";
    }

    public static void main(String args[]) {

        try {
            Server obj = new Server();
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);

            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Hello - client

```
package example.hello;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

    private Client() {}

    public static void main(String[] args) {

        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

RMI in Java

- Programming with native RMI is complex
  - The Hello example
  - https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html
- A simpler way to write and deploy services and clients
  - Jini/Apache River

Jini/Apache River

RMI Service Registration and Discovery

Java Jini (Apache River)

- Jini is a technology for building service-oriented architectures.
- Jini defines a programming model which exploits and extends Java technology.
- Jini is a generally stable, fault-tolerant, scalable, dynamic, and flexible solution.

Java Jini (Apache River)

- Services discover lookup servers and register with multiple properties
- Clients discover lookup servers and query them for services
- Discovery with multicast (LAN) or unicast (WAN)

Java RMI vs Jini

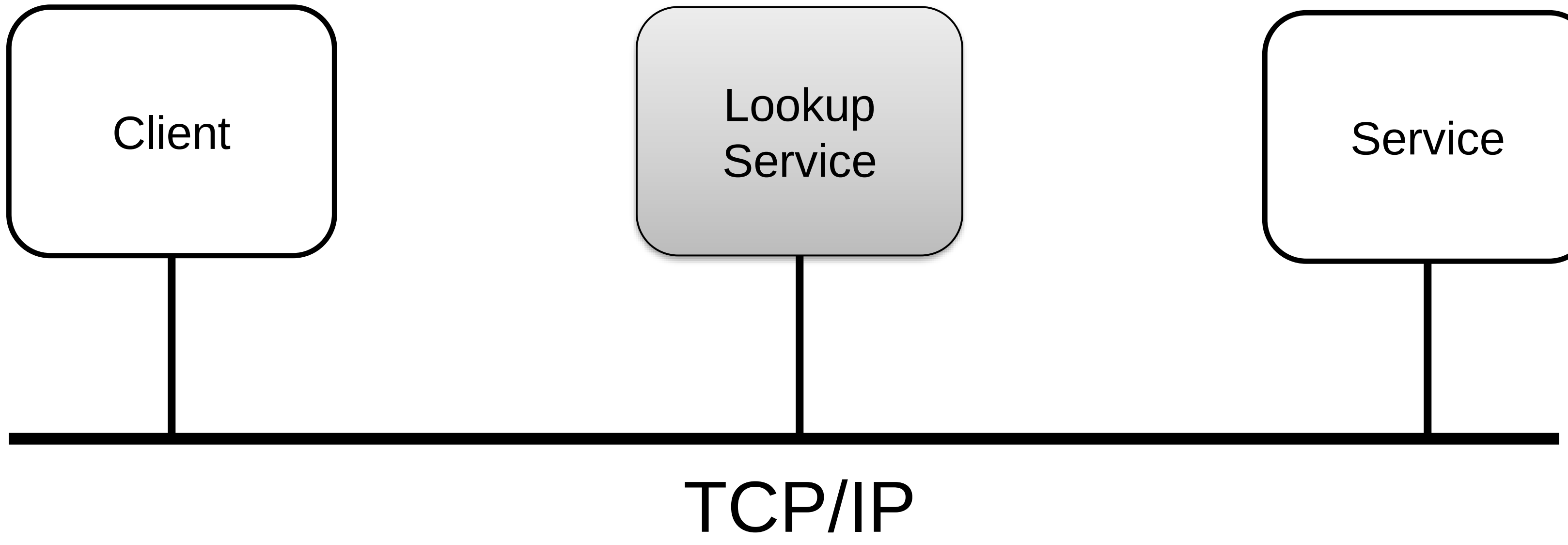
- RMI: registry is co-located with the service
- RMI: client must know the registry host
- RMI: service found by name

- *Jini: lookup services can be anywhere*
- *Jini: clients discover lookup services*
- *Jini: service found by name, attributes and interfaces*

Jini

Two ways to connect to a Lookup Service:

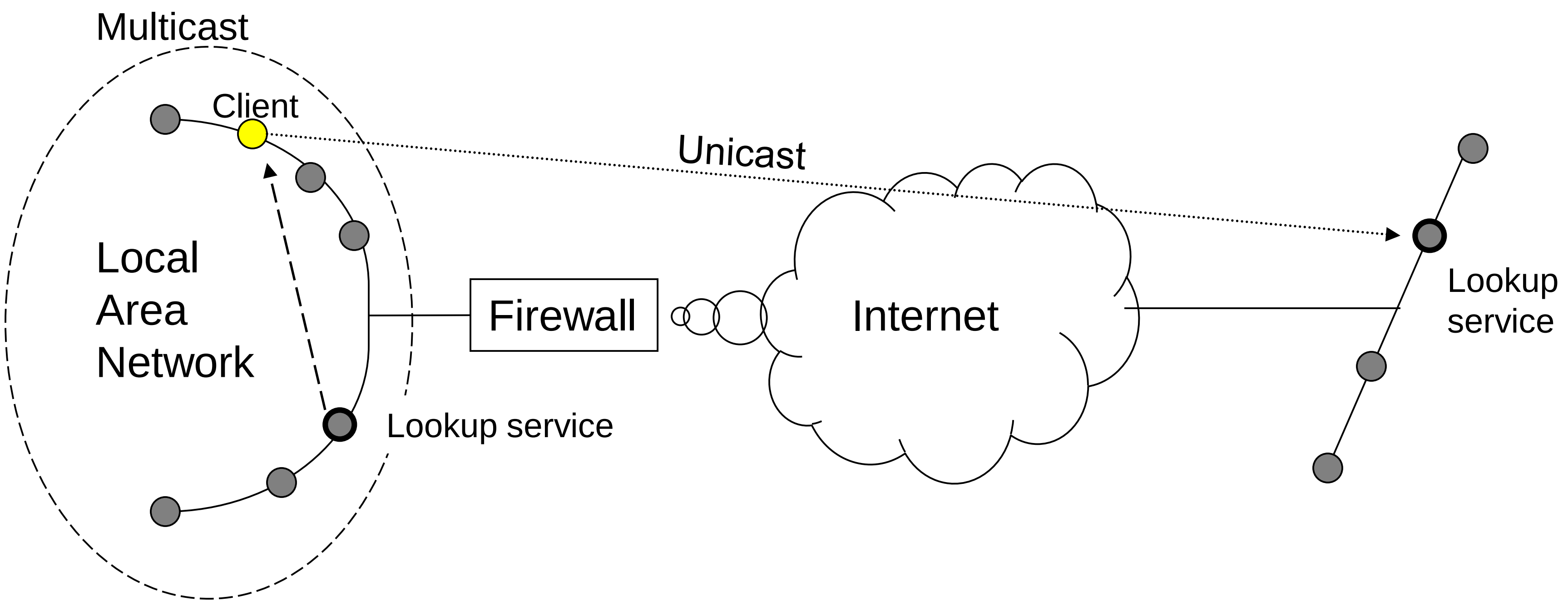
- **Multicast** to discover local lookup services
- **Unicast** to connect to a lookup service on a known host (e.g. a remote subnet)



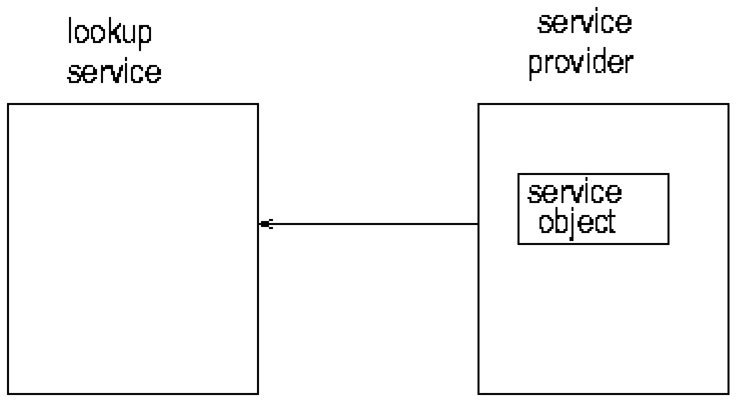
Jini

Two ways to connect to a Lookup Service:

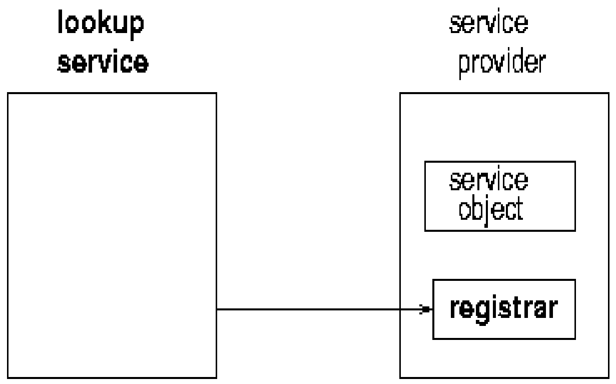
- **Multicast** to discover local lookup services
- **Unicast** to connect to a lookup service on a known host (e.g. a remote subnet)



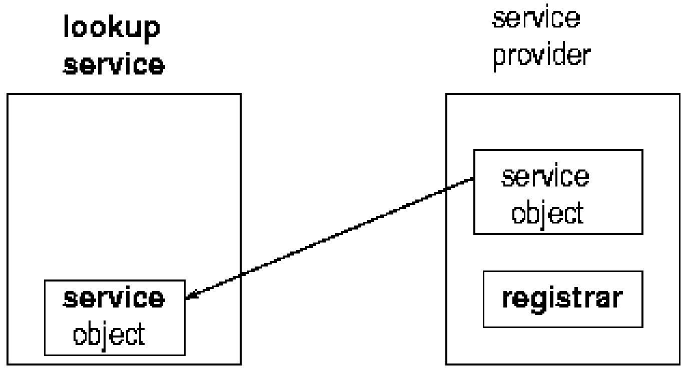
Jini (Registering a Service)



(1) Multicast/Unicast to discover the LUS

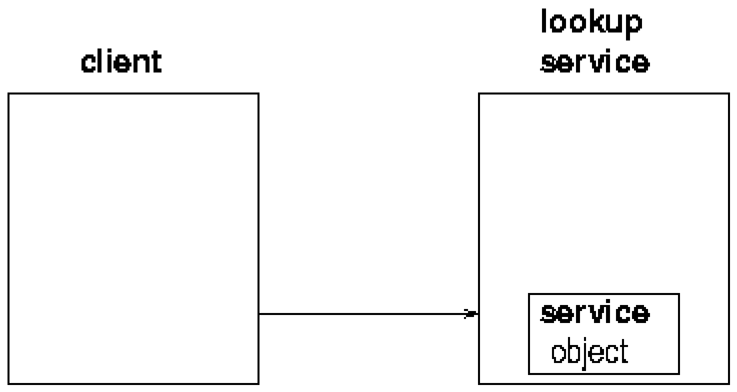


(2) Retrieve a registrar object from the LUS

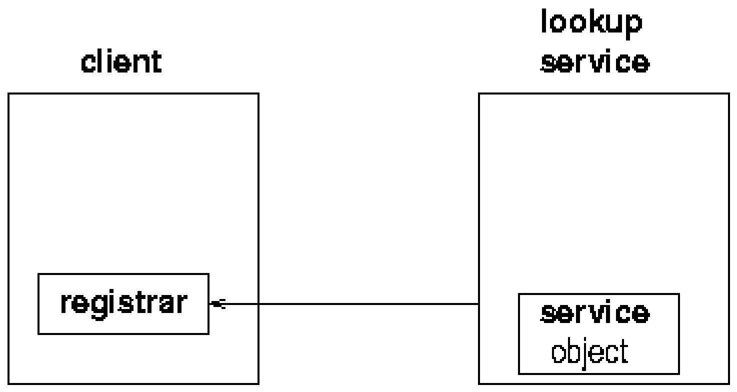


(3) Use registrar to send a service object to the LUS

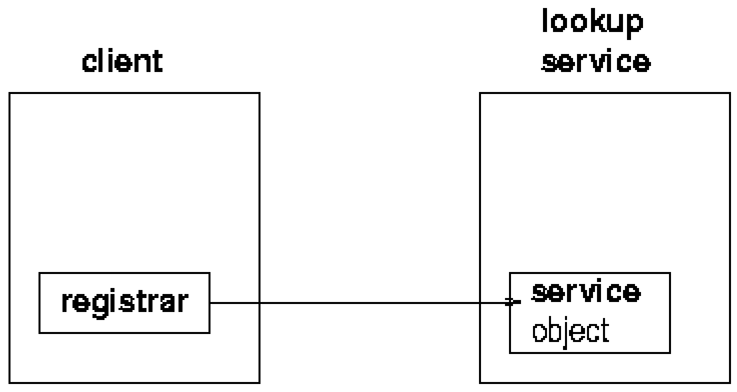
Jini (Discover a service)



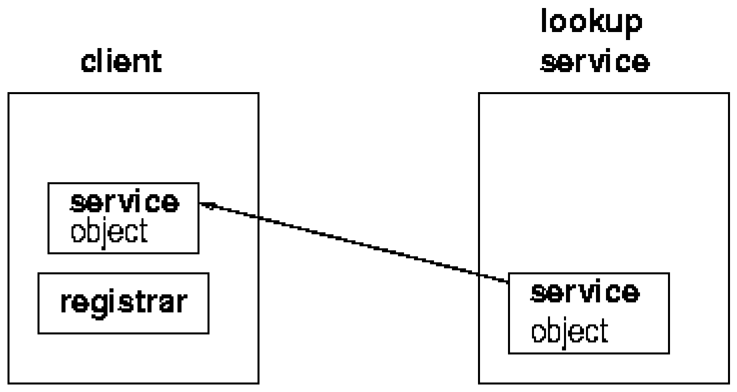
(1) Multicast/Unicast to connect to LUS



(2) Retrieve a registrar object from LUS



(3) Search service using registrar



(4) Retrieve a service object (proxy)

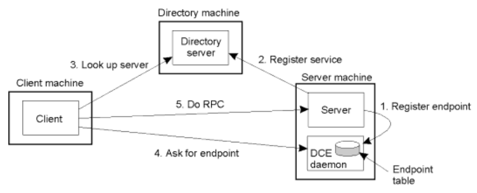
JINI

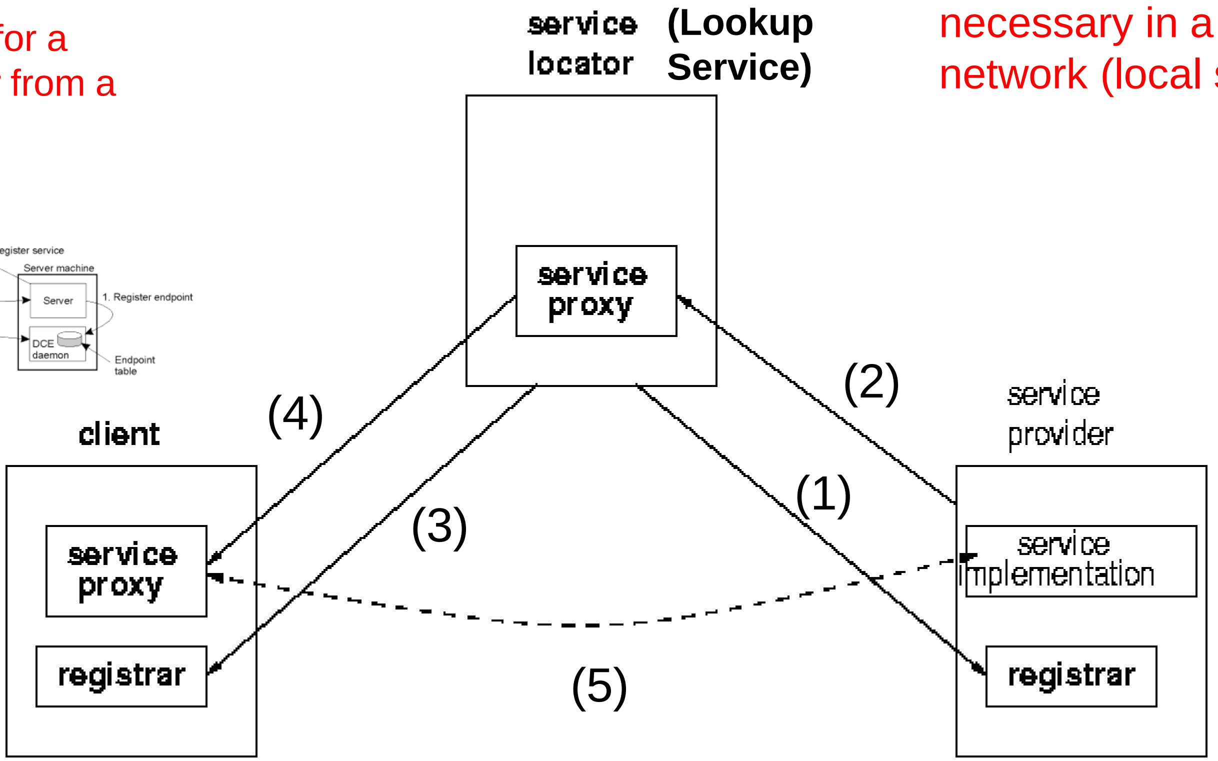
2) Dynamic Service Discovery:

- Multicast to discover local lookup service automatically
- Unicast is used for a service to register from a remote subnet

1) Simplified Running Environment:

- Only one Jini lookup service is necessary in a local Jini network (local subnet).





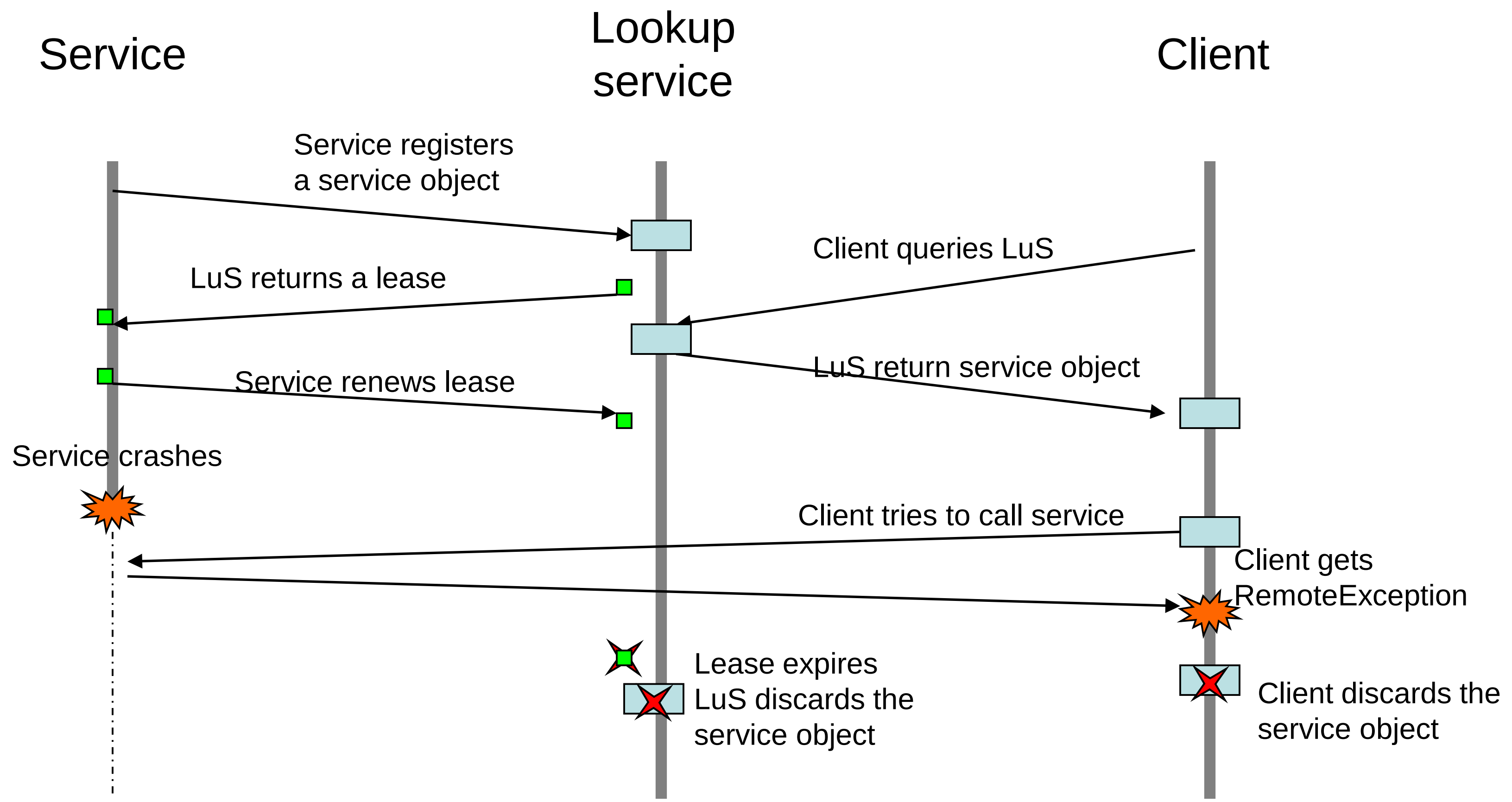
http://en.wikipedia.org/wiki/Jini

http://river.apache.org/about.html

Jini distributed garbage collection

- Leases (time-delimited promises)
- Example:
  - A service registers a service object on the lookup server in return for a lease
  - The service must renew the lease every five minutes *("I am still here")*
  - When the lease expires the LUS deletes the service object from its database

Jini – dead service


Service
Lookup service
Client
Service registers a service object
Client queries LuS
LuS returns a lease
LuS return service object
Service renews lease
Service crashes
Client tries to call service
Client gets RemoteException
Lease expires
LuS discards the service object
Client discards the service object

Jini/Apache River

- Local area networks evolved
  - Increased use of network partitions
  - Wireless access
- Multicast would not go through
- Unicast reintroduce the need for well-known names
- The advantage of clients being agnostic to specific hosts is lost

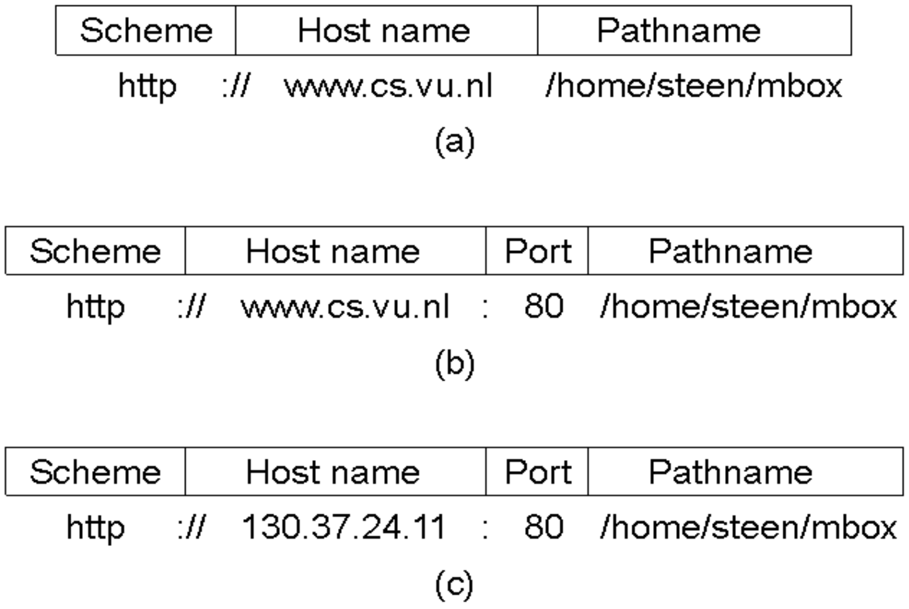
Web naming

URI URL URN

Web naming scheme

- Uniform Resource Identifiers (URI) come in two forms:
  - URL: Uniform Resource Locator
  - URN: Uniform Resource Name

Uniform Resource Locators



Scheme | Host name | Pathname
http :// www.cs.vu.nl /home/steen/mbox
(a)
Scheme | Host name | Port | Pathname
http :// www.cs.vu.nl : 80 /home/steen/mbox
(b)
Scheme | Host name | Port | Pathname
http :// 130.37.24.11 : 80 /home/steen/mbox
(c)

Often-used structures for URLs.

a) Using only a DNS name.

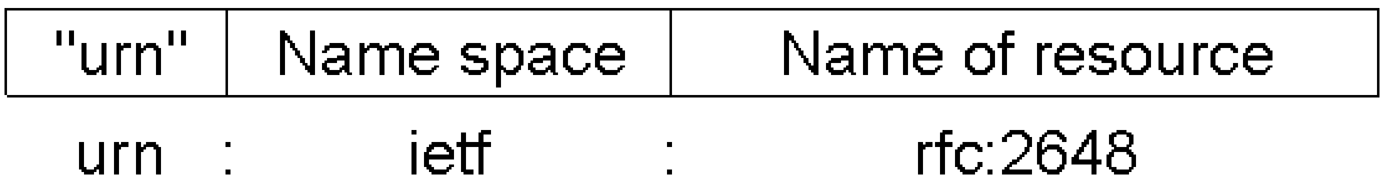
b) Combining a DNS name with a port number.

c) combining an IP address with a port number.

Common schemes

Name	**Used for**	**Example**
http	HTTP	http://www.cs.vu.nl:80/globe
https	Secure HTTP	https://fli.cs.edu/login.php
mailto	email	mailto:info@company.com
file	Local file	file:///etc/fstab

Uniform Resource Names

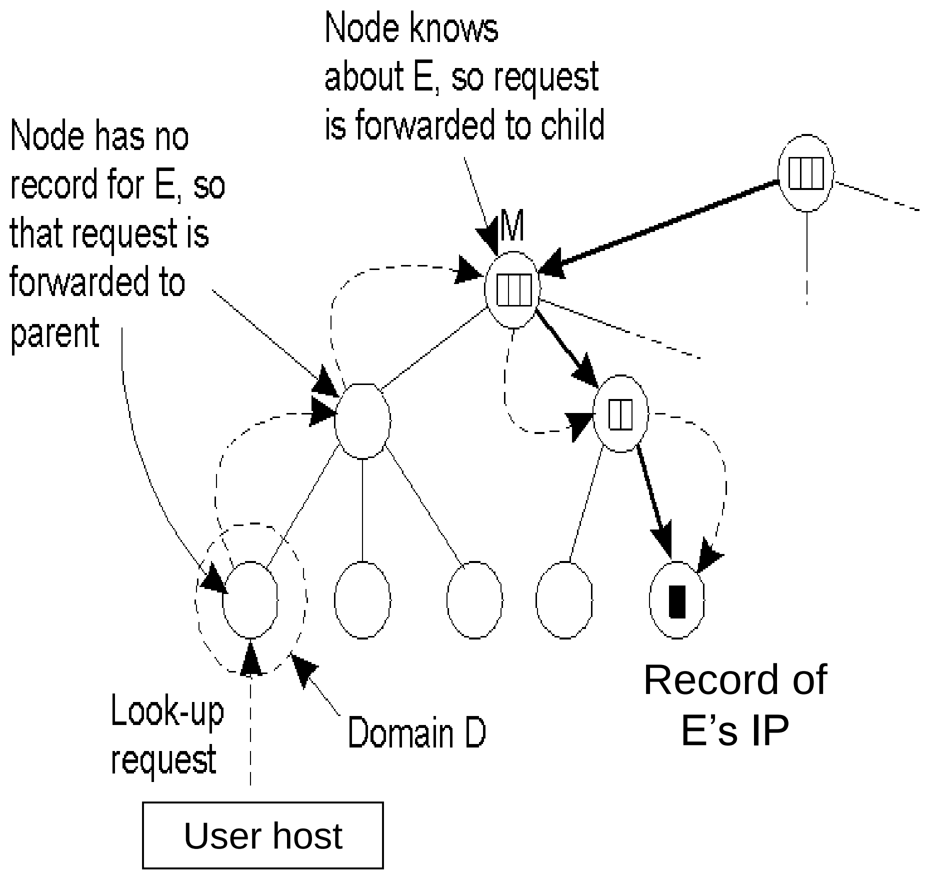


"urn"		Name space		Name of resource
urn	:	ietf	:	rfc:2648
urn	:	isbn	:	0-13-349945-6

- Three parts: Scheme : Name space identifier : Name of resource
- The name space identifier determines the syntactic rules for the third part. The third part may have different structure depending on the name space identifier. So URNs are not publicly resolvable.
- In contrast to URLs, URNs are location-independent which means URNs usually are not related to any specific entity (only used as a name space).

Locating URL (Name Resolving)

- Domain Name System (HostName -> IP)
  - Each computer has to be assigned an IP address and DNS server IP address manually or through DHCP server.
  - DNS Request (nslookup) sends the hostname to the specified DNS server.
  - The DNS server returns the IP if it knows it, otherwise, the request is forwarded to upper-layer DNS server.


Node knows about E, so request is forwarded to child
Node has no record for E, so that request is forwarded to parent
M
Look-up request
Domain D
Record of E's IP
User host

XML and JSON

XML namespaces

XSLT

JavaScript Object Notation

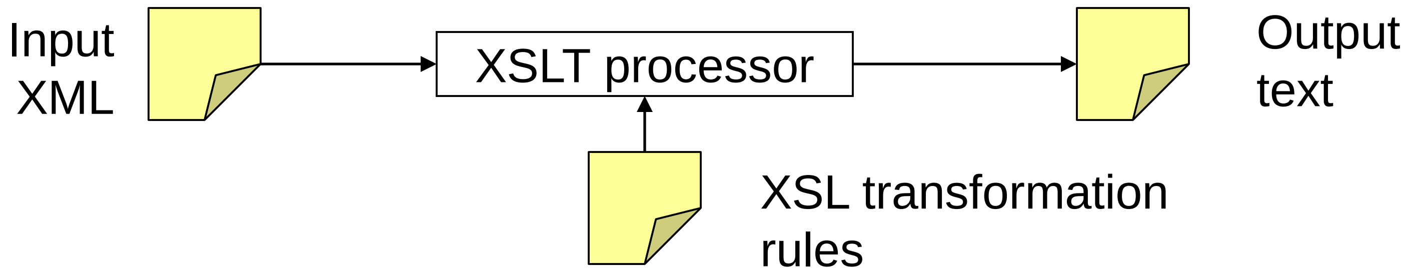
Data Representation / XML (1)

- Extensible Markup Language (XML) is a standard format for interchanging structured documents.
- XML is designed to describe data
- HTML was designed to present data.
- Anyone can use XML to define data in any tree-based structure.
- To be able to distinguish different structures, XML Name Spaces are used to enable different structures of data to co-exist in one document.

```
<?xml version="1.0" ?>
<note xmlns:note=“http://tv.com/note.xml”>
  <note:to>Tove</note:to>
  <note:from>Jani</note:from>
  <note:heading>Reminder</note:heading>
  <note:body>Don't forget me this
weekend!</note:body>
 </note>
```

Data Representation /XML (2)

- Extensible Stylesheet Language Transformations (XSLT) is a language for converting XML document from one structure to another.
- The XSL transformation rules are written in XML
- The XSLT processor transforms the input XML to the output



Data Representation /XML (3)

- XML has a substantial overhead
  - opening and closing tags: <author>Franz Kafka</author>
- Short, high-volume messages are penalized harder
- Compression schemes exist but require additional effort

Data Representation /JSON(1)

- JavaScript Object Notation
  - www.json.org
  - Created in 2002-ish
  - Standardized in 2017
    - RFC 8259, ISO/IEC 21778:2017
- A sequence of objects, arrays, and values
- UTF-8 encoding

JSON syntax

- object
  - { }
  - { "string" : value , ... }
- array
  - [ ]
  - [value , ...]
- value
  - "string", number, object, array
  - `true, false, null`

Data Representation /JSON(2)

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Name and value pairs
Names are strings
Values shown: objects, strings, array

Data Representation /JSON(3)

JSON

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

XML

```
<menu>
  <id>file</id>
  <value>File</value>
  <popup>
    <menuitem>
       <value>New</value>
       <onclick>CreateNewDoc()</onclick>
    </menuitem>
    <menuitem>
       <value>Open</value>
       <onclick>OpenDoc()</onclick>
    </menuitem>
    <menuitem>
       <value>Close</value>
       <onclick>CloseDoc()</onclick>
    </menuitem>
  </popup>
</menu>
```

Data Representation /JSON(4)

- Supported by several programming languages
- Relatively easy to parse
- Special rules for exotic Unicode characters

Data Serialization Formats

- https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats
- Text:
- CSV, Extensible Data Notation, JSON, OpenDDL, XML-RPC, YAML
- Binary:
- Apache Avro, Apache Parquet, ASN.1, Binn, BSON, CBOR CDR, D-BUS, EXI, FlatBuffers, Fast Infoset, FHIR, Ion, Java serialization, MessagePack, OGDL, Pickle, Property list, Protocol Buffers, RLP, S-expressions, Structured Data eXchange Formats, Apache Thrift, UBJSON, XDR
- Hybrid:
- Bencode, Netstrings, SOAP, XML

End