

Jini

Kurzfassung als Kapitel für die Vorlesung
„Verteilte Systeme“

Friedemann Mattern

(unter Nutzung von Teilen von Andreas Zeidler und Roger Kehr)

Jini



- **Infrastructure** (“middleware”) for dynamic, cooperative, spontaneously networked systems
 - facilitates realization of distributed applications

Jini

- **Infrastructure** (“middleware”) for dynamic, cooperative, spontaneously networked systems
 - facilitates realization of distributed applications

- framework of APIs with useful functions / services
- helper services (discovery, lookup,...)
- suite of standard protocols and conventions

Jini

- **Infrastructure** (“middleware”) for dynamic, cooperative, spontaneously networked systems
 - facilitates realization of distributed applications

- services, devices, ... find each other automatically (“plug and play”)
- dynamically added / removed components
- changing communication relationships
- mobility

Jini

- **Infrastructure** (“middleware”) for dynamic, cooperative, spontaneously networked systems
 - facilitates realization of distributed applications
- Based on **Java** and implemented in Java
 - may use RMI (Remote Method Invocation)
 - typed (object-oriented) communication structure
 - requires JVM / bytecode everywhere
 - code shipping

Jini

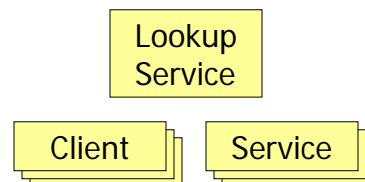
- **Infrastructure** (“middleware”) for dynamic, cooperative, spontaneously networked systems
 - facilitates realization of distributed applications
- Based on **Java** and implemented in Java
 - may use RMI (Remote Method Invocation)
 - typed (object-oriented) communication structure
 - requires JVM / bytecode everywhere
 - code shipping
- **Strictly service-oriented**
 - everything is a service (hardware / software / user)
 - Jini system is a federation of services
 - mobile proxy objects for service access

Service Paradigm

- (Almost) everything is a **service**
 - e.g. persistent storage, software filter, ...
- Jini's run-time infrastructure offers mechanisms for **adding, removing, finding, and using services**
- Services are defined by **interfaces** and provide their functionality via their interfaces
 - **services** are characterized by their **type** and their **attributes** (e.g. "600 dpi", "version 21.1")
- Services (and service users) "spontaneously" form a system ("**federation**")

Jini: Global Architecture

- **Lookup Service (LUS)**
 - main registry entity and brokerage service for services and clients
 - maintains information about available services
- **Services**
 - specified by Java interfaces
 - register together with **proxy objects** and attributes at the LUS
- **Clients**
 - know the Java interfaces of the services, but not their implementation
 - find services via the LUS
 - use services via proxy objects



Network Centric

- Jini is centered around the **network**
 - "the network is the computer"
- Network = hardware and software infrastructure
 - includes helper services
- View is "network to which devices are connected to", not "devices that get networked"
 - network always exists, devices and services are transient
- Set of networked devices is **dynamic**
 - components and communication relations come and go
- Jini supports dynamic networks and adaptive systems
 - adding and removing components should only minimally affect other components

Spontaneous Networking

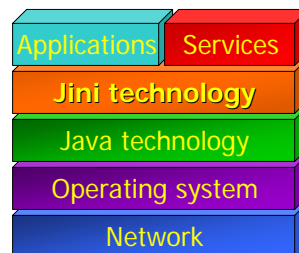
- Objects in an open, distributed, dynamic world find each other and form a **transitory community**
 - cooperation, service usage, ...
- Typical scenario: client wakes up (device is switched on, plugged in, ...) and asks for services in its vicinity
- Finding each other and establishing a connection should be **fast**, **easy**, and **automatic**

Some Fallacies of Common Distributed Computing Systems

- The **idealistic view**...
 - the network is reliable
 - latency is zero
 - bandwidth is infinite
 - the network is secure
 - the topology is stable
 - there is a single administrator
- ...**isn't true** in reality
 - Jini addresses some of these issues
 - at least it does not hide or ignore them

Bird's-Eye View on Jini

- Jini consists of a number of **APIs**
- Is an extension to the **Java** platform dealing with distributed computing
- Is an **abstraction layer** between the application and the underlying infrastructure (network, OS)
 - Jini is a kind of "**middleware**"



Jini's Use of Java

- Jini **requires JVM** (as bytecode interpreter)
 - homogeneity in a heterogeneous world
 - is this realistic?
- But: devices that are **not "Jini-enabled"** or that do not have a JVM can be managed by a **software proxy** which resides somewhere in the net

run protocols
for discovery and
join; have a JVM

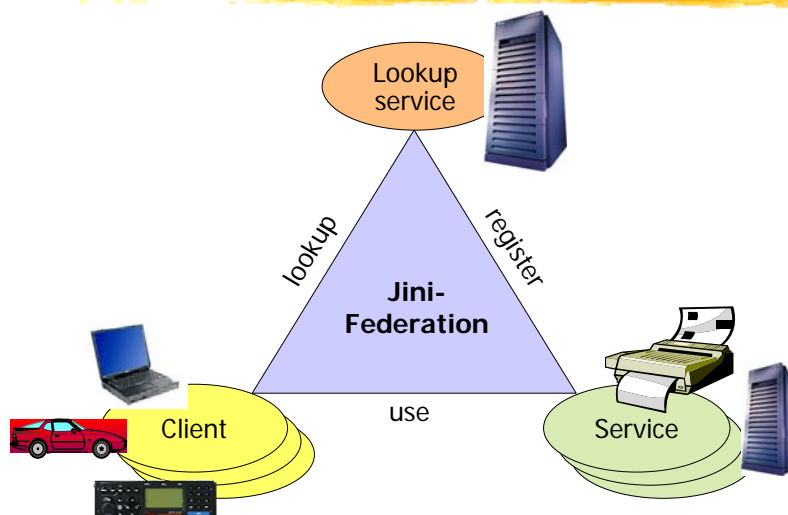
Jini Infrastructure

- Main components are:
 - **lookup service** as repository / naming service / trader
 - **protocols** based on TCP/UDP/IP
 - discovery & join, lookup of services
 - **proxy objects**
 - transferred from service to clients
 - represent the service locally at the client
- Goal: **spontaneous networking** and formation of federations without prior knowledge of local network environment
- Problem: How do service providers and clients **learn about their local environments?**

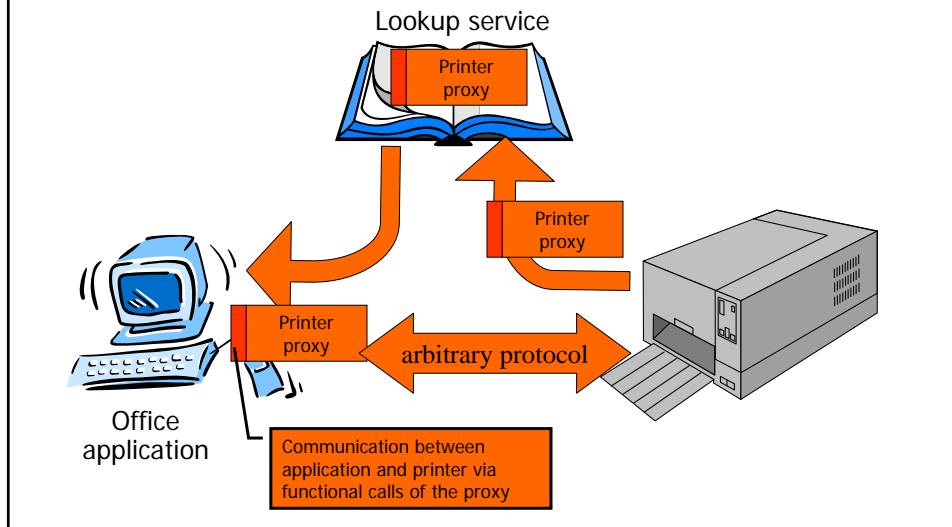
Lookup Service (LUS)

- Main component of every Jini federation
- **Repository** of services
- Similar to RMI registry or naming services of other middleware architectures
- Functions as a “help-desk” for services and clients
 - **registration of services** (services advertise themselves)
 - **distribution of services** (clients lookup and find services)
- Has mechanisms to **bring together services and clients**

Lookup Service



Example

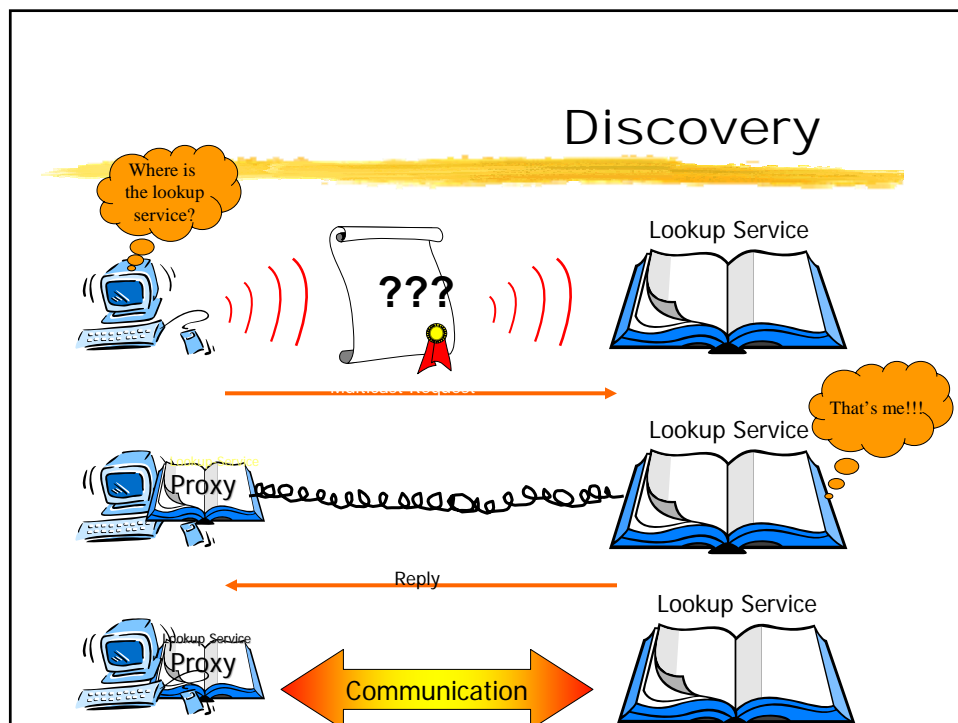


Lookup Service

- Uses **Java RMI** for communication
 - objects („proxies“) can migrate through the net
- Not only **name/address** of a service is stored (as in traditional naming services), but also
 - set of **attributes**
 - e.g.: printer(color: true, dpi: 600, ...)
 - **proxies**, which may be complex classes
 - e.g. user interfaces
- Further possibilities:
 - increase robustness by running **redundant lookup services**
 - responsibility can be distributed to a number of (logically separated) lookup services

Discovery: Finding a LUS

- Goal: Find a lookup service (without knowing anything about the network) to
 - advertise (register) a service
 - find (look up) an existing service
- Discovery protocol:
 - multicast to well-known address/port
 - lookup service replies with a serialized object (its proxy)
 - communication with LUS via this proxy



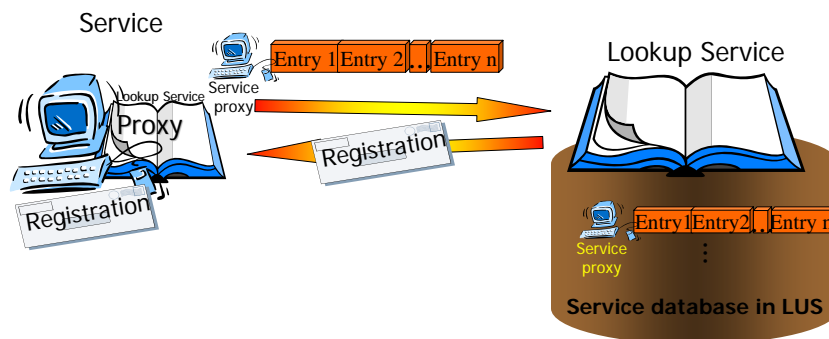
Multicast Discovery Protocol

- No information about the host network needed
- Active [search for lookup services](#)
- Discovery request uses multicast **UDP** packets
 - [multicast address](#) for discovery is 224.0.1.85
 - default [port number](#) of lookup services is 4160
 - recommended [time-to-live](#) is 15
 - usually does not cross [subnet boundaries](#)
- Discovery [reply](#) is establishment of a **TCP connection**
 - port for reply is included in multicast request packet

Join: Registering a Service

- Assumption: Service provider already has a proxy of the lookup service (→ discovery)
- It uses this proxy to [register its service](#)
- Gives to the lookup service
 - its [service proxy](#)
 - [attributes](#) that further describe the service
- Service provider can now be found and used in this Jini federation

Join



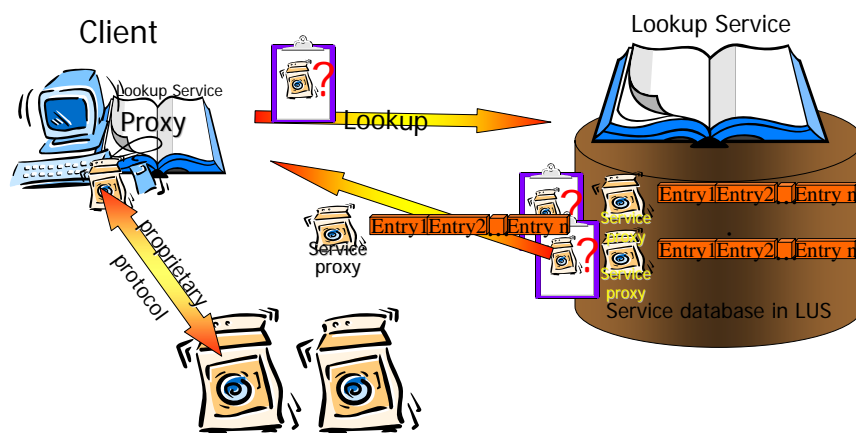
Join: More Features

- To join, a service supplies:
 - its **proxy**
 - its **ServiceID** (if previously assigned; “universally unique identifier”)
 - set of **attributes**
 - (possibly empty) set of specific **lookup services** to join
- Service waits a random amount of time after start-up
 - prevents packet storm after restarting a network segment
- Registration with a lookup service is bound to a **lease**
 - service has to **renew** its lease periodically

Lookup: Searching Services

- Client creates query for lookup service
 - in the form of a “service template”
 - matching by registration **number** of service and/or service **type** and/or **attributes** possible
 - attributes: only **exact matching** possible (no “larger-than”, ...)
 - **wildcards** possible („null“)
 - Via its proxy at the client, the lookup service returns zero, one or more **matches** (i.e., **server proxies**)
 - Selection of several matches usually done by client
-
- Client uses service by calling functions of the **service proxy**
 - Any “private” protocol between service proxy and service provider is possible

Lookup



Proxies

- Proxy object is **stored in the lookup service** upon registration
 - serialized object
 - implements one or more service interfaces
- Upon request, stored object is **sent to the client** as a local proxy of the service
 - client communicates with service implementation via its proxy: **client invokes methods of the proxy object**
 - proxy **implementation hidden** from client

Smart Proxies

- Parts of or whole functionality may be **executed by the proxy** at the client
- When dealing with large volumes of data, it usually makes sense to **preprocess** parts of or all the data
 - e.g.: compressing video data before transfer
- Partition of service functionality depends on service implementer's choice
 - client needs appropriate **resources**

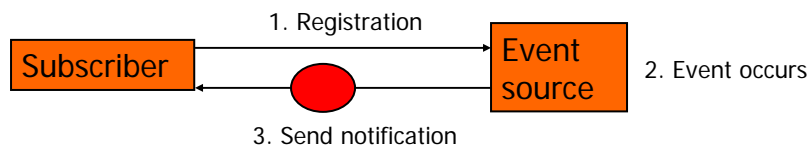


Leases

- Leases are **contracts** between two parties
- Leases introduce a notion of **time**
 - resource usage is restricted to a certain time frame
- Repeatedly express interest in some resource:
 - I'm **still interested** in X
 - renew lease periodically
 - lease renewal can be denied
 - I **don't need** X anymore
 - cancel lease or let it expire
 - lease grantor can use X for something else

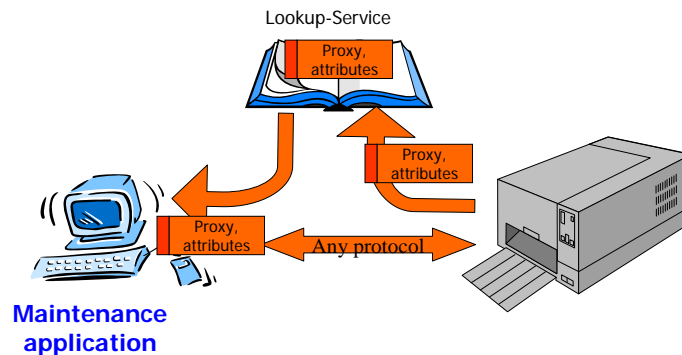
Distributed Events

- Objects on one JVM can **register interest** in certain events of another object on a different JVM
- **"Publisher/subscriber"** model



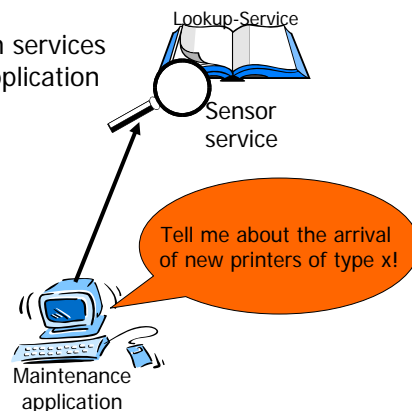
Distributed Events – Example

- Example: printer is **plugged in**
 - printer **registers** itself with local lookup service
- **Maintenance application** wants to update software



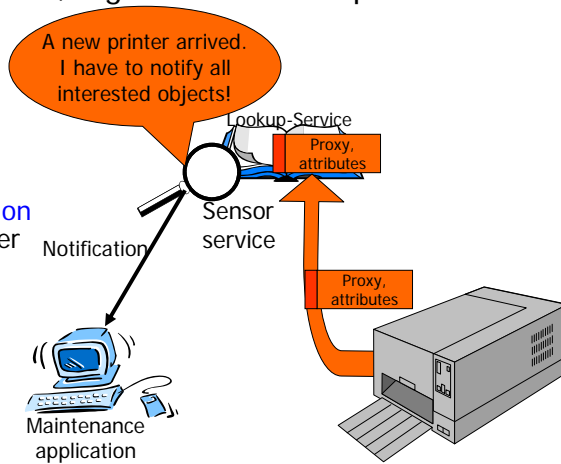
Distributed Events – Example

- Maintenance application is **run on demand**, search for printers is “outsourced”
 - “**sensor service**” looks for certain services on behalf of the maintenance application
 - application **registers for events** showing the arrival of certain types of printers
 - sensor observes the lookup service
 - **notifies application** as soon as matching printer arrives via distributed events



Distributed Events – Example

- Example: **printer arrives**, registers with lookup service
 - printer performs **discovery and join**
 - sensor finds new printer in lookup service
 - checks if there is an **event registration** for this type of printer
 - **notifies** all interested objects
 - **maintenance application** retrieves printer proxy and updates software



Jini Issues and Problem Areas

- **Security**
 - important especially in dynamic environments
 - services use other services on behalf of the user
 - principals, delegation
 - Simply rely on Java security?
- **Scalability**
 - does Jini scale to a global level?
- **Java centric**
- **Similar, non-Java-based systems**
 - UPnP, Bluetooth SDP, SLP, HAVi, Salutation, e-speak, HP Chai,...
 - open, Internet-scale infrastructures (e.g., Web services)