Using Jini in non-Java devices

K. Shankari

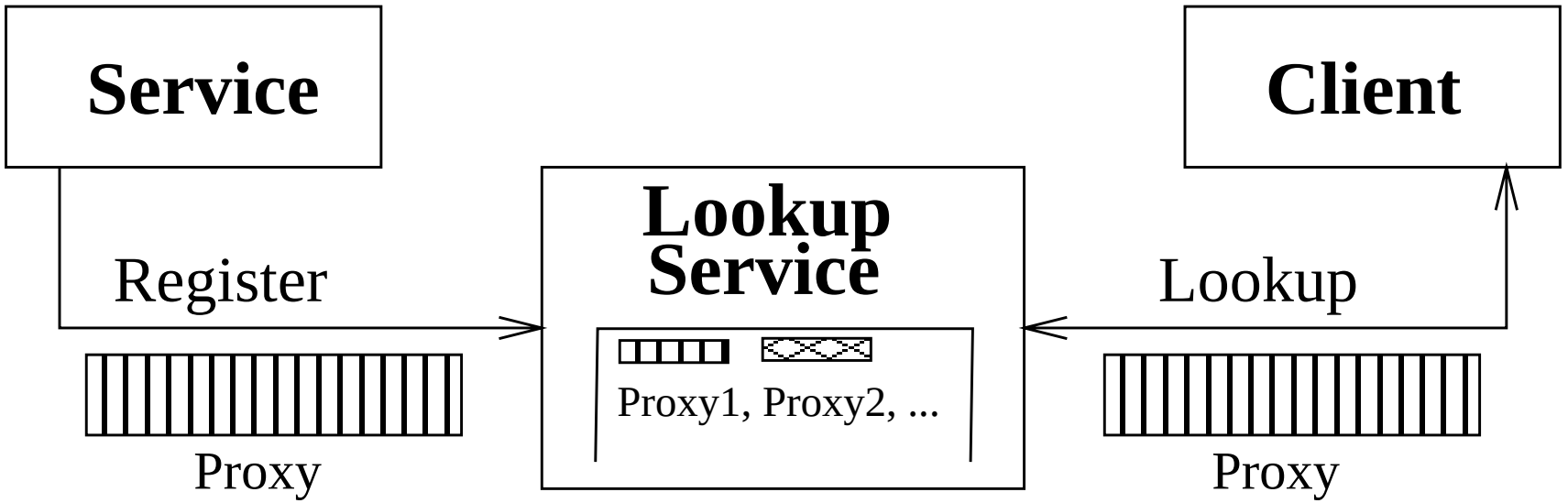
University of California, Santa Cruz

The device connection problem

or, "I have some good news and some bad news"

Good news: Several appliances with microprocessors in them
Idea: Connect them together and make intelligent home!
Bad news: Device installation and configuration problems magnified
Good news: Jini is one option for providing true plug and play
Bad news: Many applicances are too small to run Jini
Good news: We've found a work around

Jini Glossary



Jini: Set of protocols for interconnection, built on Java
Service: A software or hardware component which can be used by other components.
Lookup Service: Maintains central list of services.
Client: Program which utilizes the service.
Proxy: Jini equivalent of driver.

How Jini can help device configuration

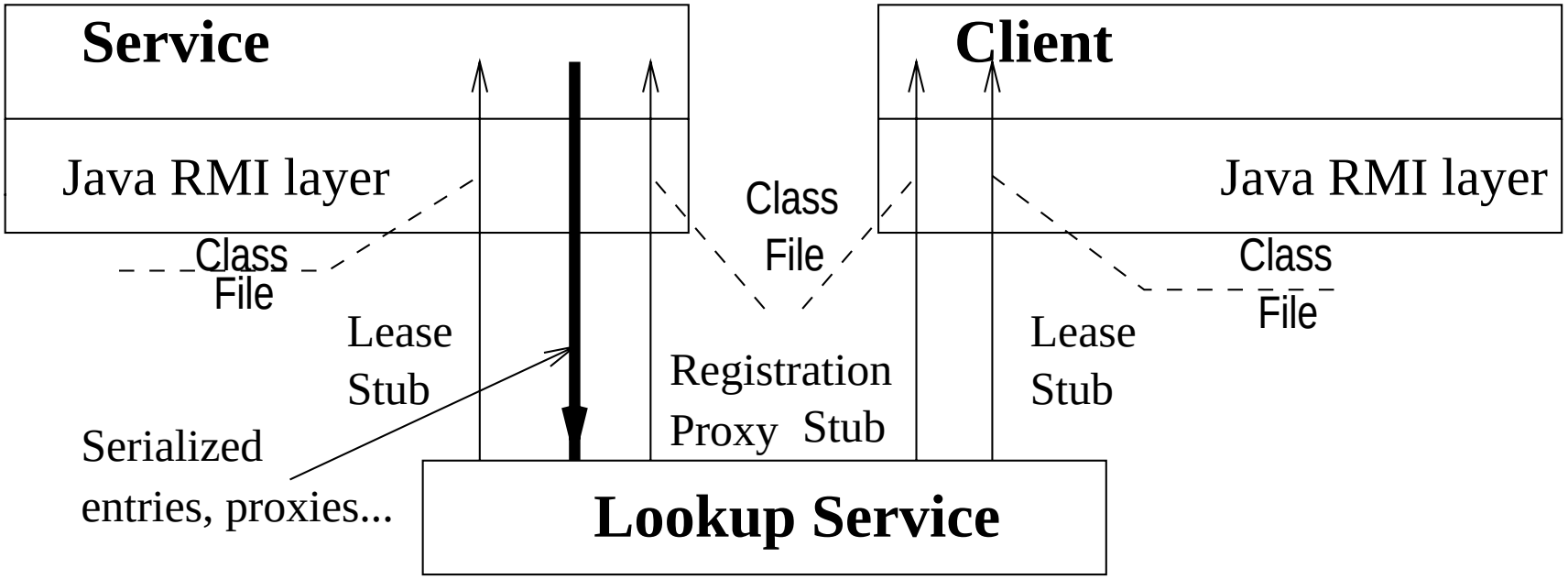
Installation: Drivers downloaded from code server when required.

Configuration: *Proxy* representing the device is downloaded into the client.

Obsolescence: Updating drivers on code server automatically installs them on clients when devices are used.

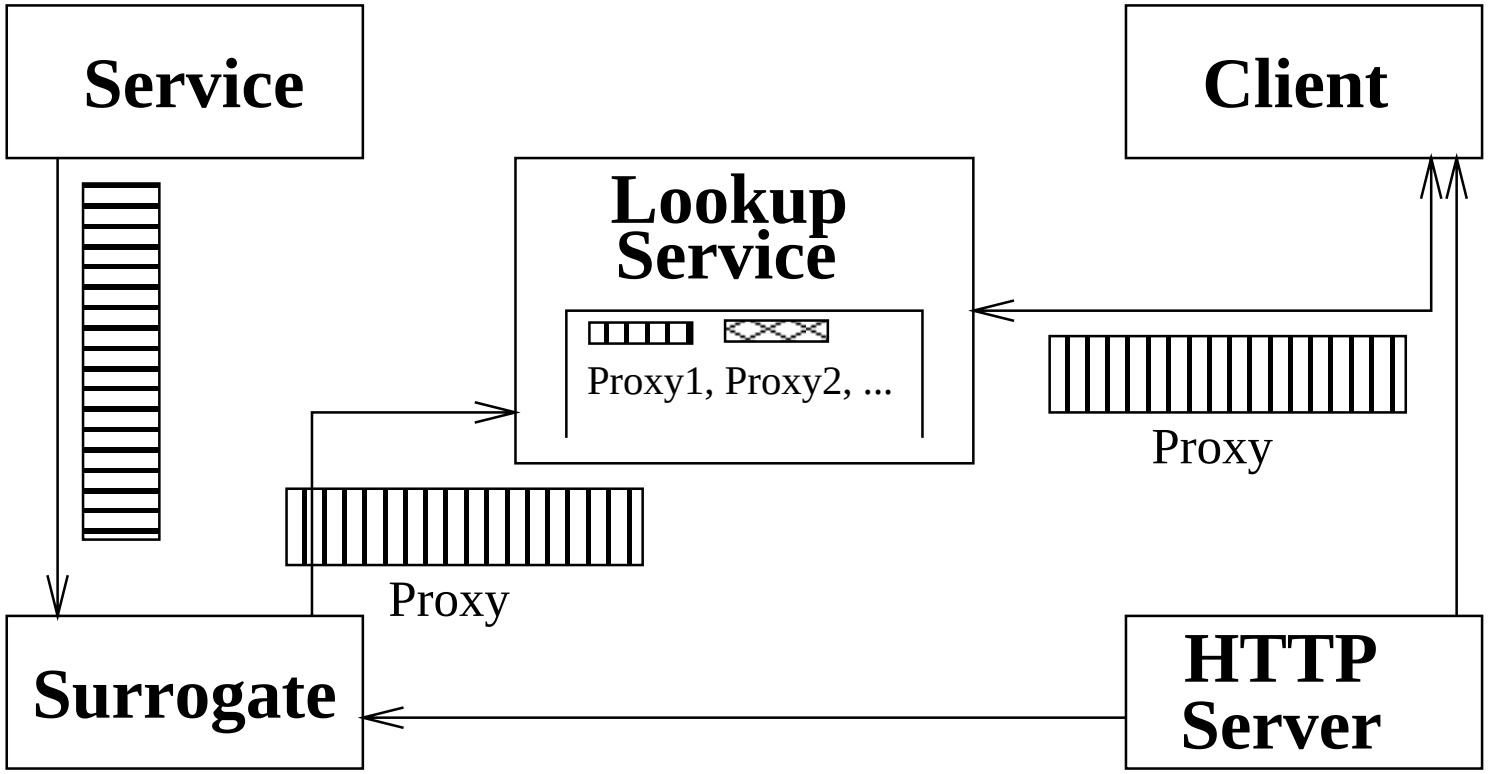
Different systems: Because of Java's platform independence, only one driver has to be written for all systems.

So what's the problem here?



- ✔ Jini is over Java, and uses RMI for certain critical tasks.
- ✔ The current version of the JVM is around *20* Mb.
- ✔ **Needed:** Way to use Jini from a non-Java program

Obvious solution

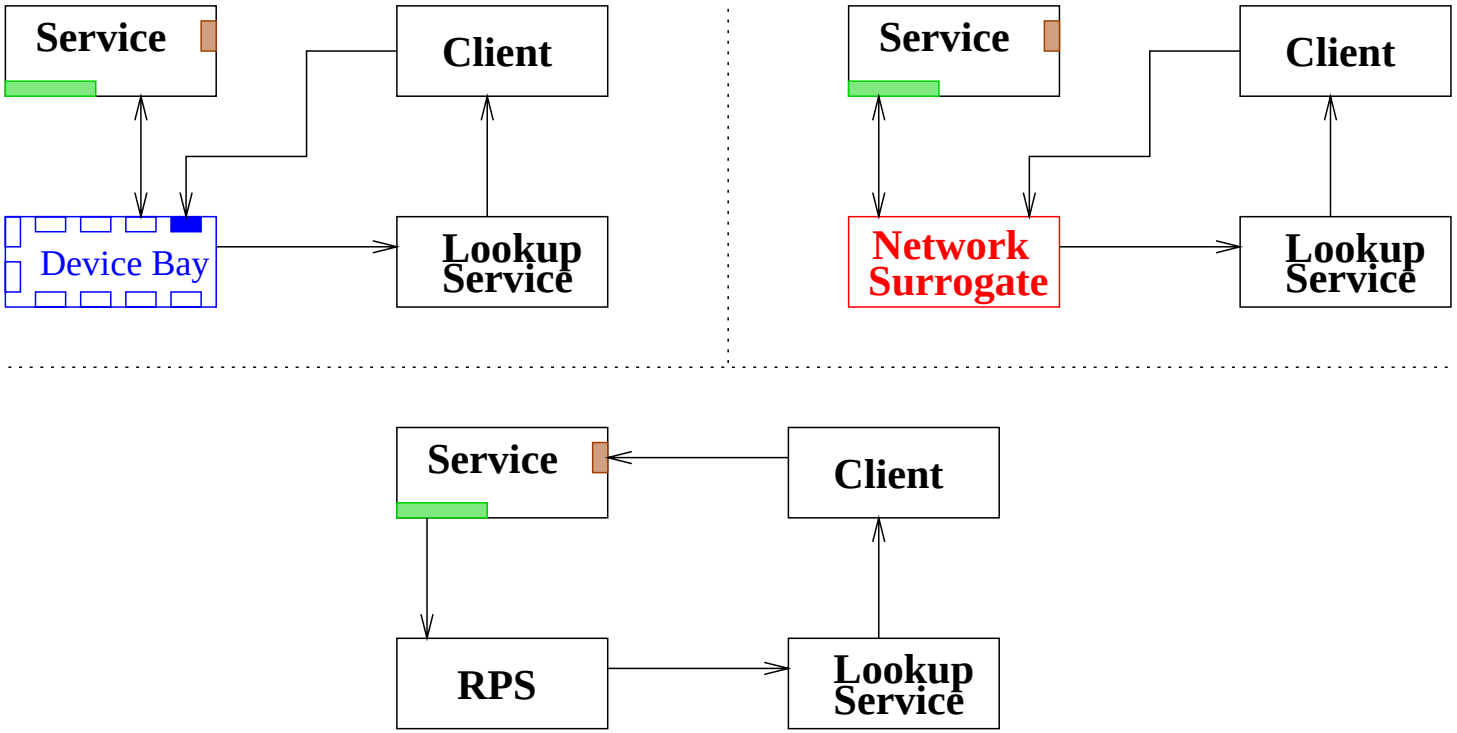


Observation: Service needs to use Java solely for interactions with Jini system.

Solution: Can delegate Jini interactions to Java surrogate

Options for non-Java devices

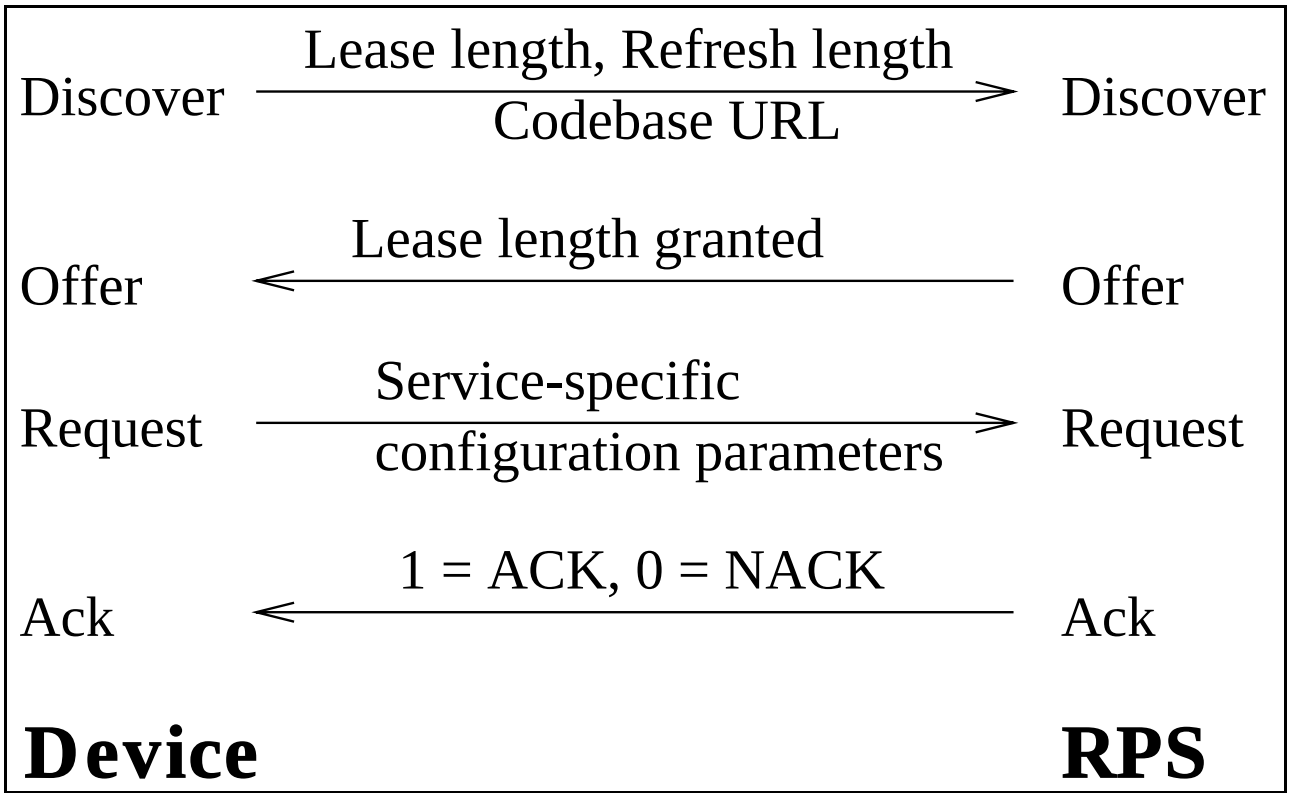
- Device Bay = poor scalability + poor flexibility
- Network Surrogate = poor flexibility + poor robustness
- CORBA (using the RMI-IIOP bridge) = high complexity
- RPS (Registration Proxy Server) = our choice



Our implementation

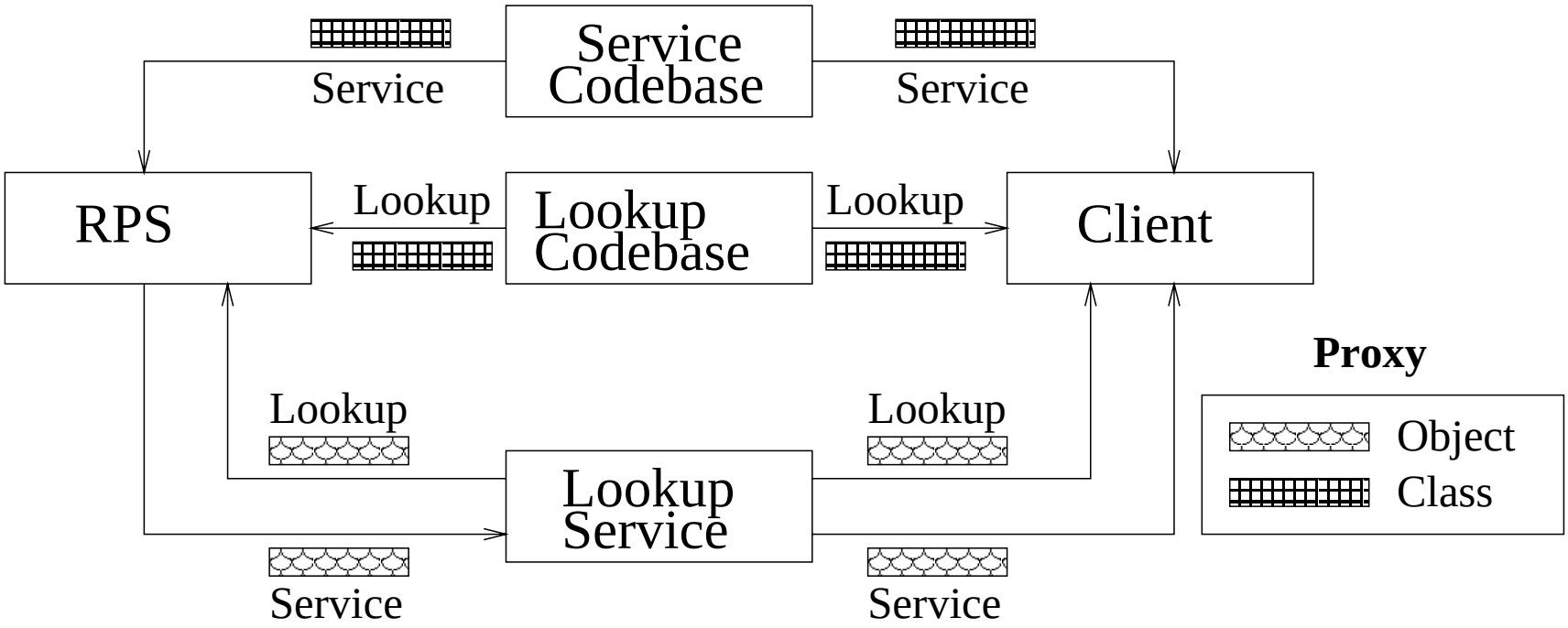
- Protocol developed for service contacting the RPS to register itself.
- Protocol verified using CFSM techniques.
- `MessageService` – returns fixed string when contacted.
- Implemented on NS486SXF running Nanokernel from UCSC as OS.
- Implementation issues -
  - Code download while obtaining the proxy
  - Pinging the embedded device

Communication Protocol



- ⇨ Based on Dynamic Host Configuration Protocol (DHCP)
- ⇨ 2 stages - one to find, second to choose
- ⇨ First stage requires UDP broadcast

Communication with Jini system



RPS:

- downloads proxy class and instantiates it
- configures it using parameter string from service
- registers proxy with lookup service

RPS - Jini system interaction

RPS:

- ☆ downloads proxy class file and instantiates proxy
- ☆ passes device configuration string to proxy
- ☆ performs discovery, finds Lookup services
- ☆ downloads proxy for Lookup service using RMI
- ☆ changes codebase to service codebase
- ☆ registers service proxy with Lookup service proxy
- ☆ changes codebase back to codebase

Client:

- ☆ performs discovery, finds Lookup services
- ☆ downloads proxy for Lookup service using RMI
- ☆ finds service proxy by *lookup* procedure
- ☆ downloads service proxy object from Lookup service
- ☆ downloads service proxy class from service codebase
- ☆ uses service by invoking methods on service proxy

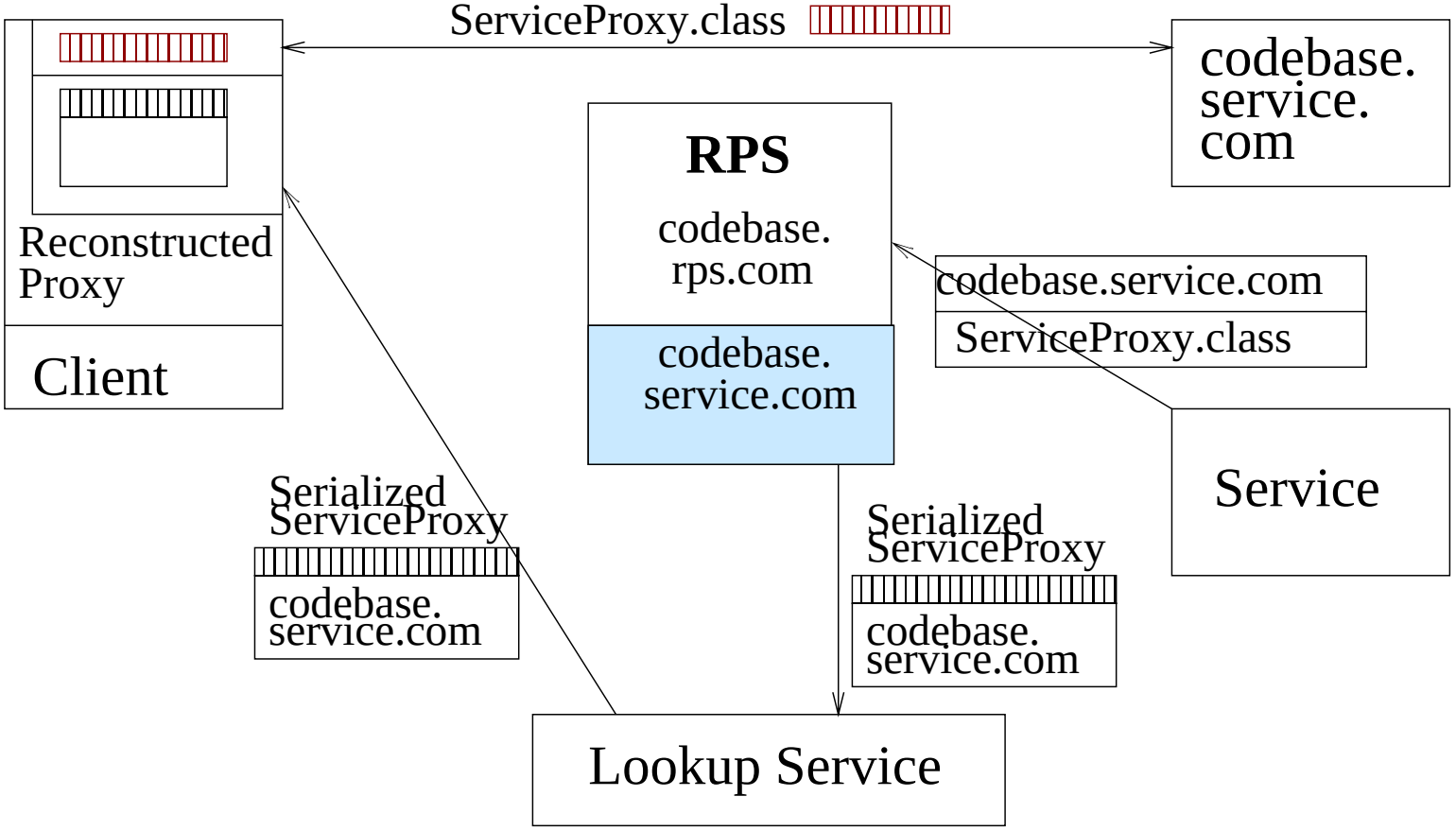
Properties verified using CFSMs

Method from Bochmann(1991) and Sunshine (1979).

Self-synchornization: Can reach normal operation from any state.
No deadlock: All states have at least one outgoing edge
No livelock: No "bad" loops in the system
Live state: State reachable from all "normal" states

All these properties verified for this protocol.

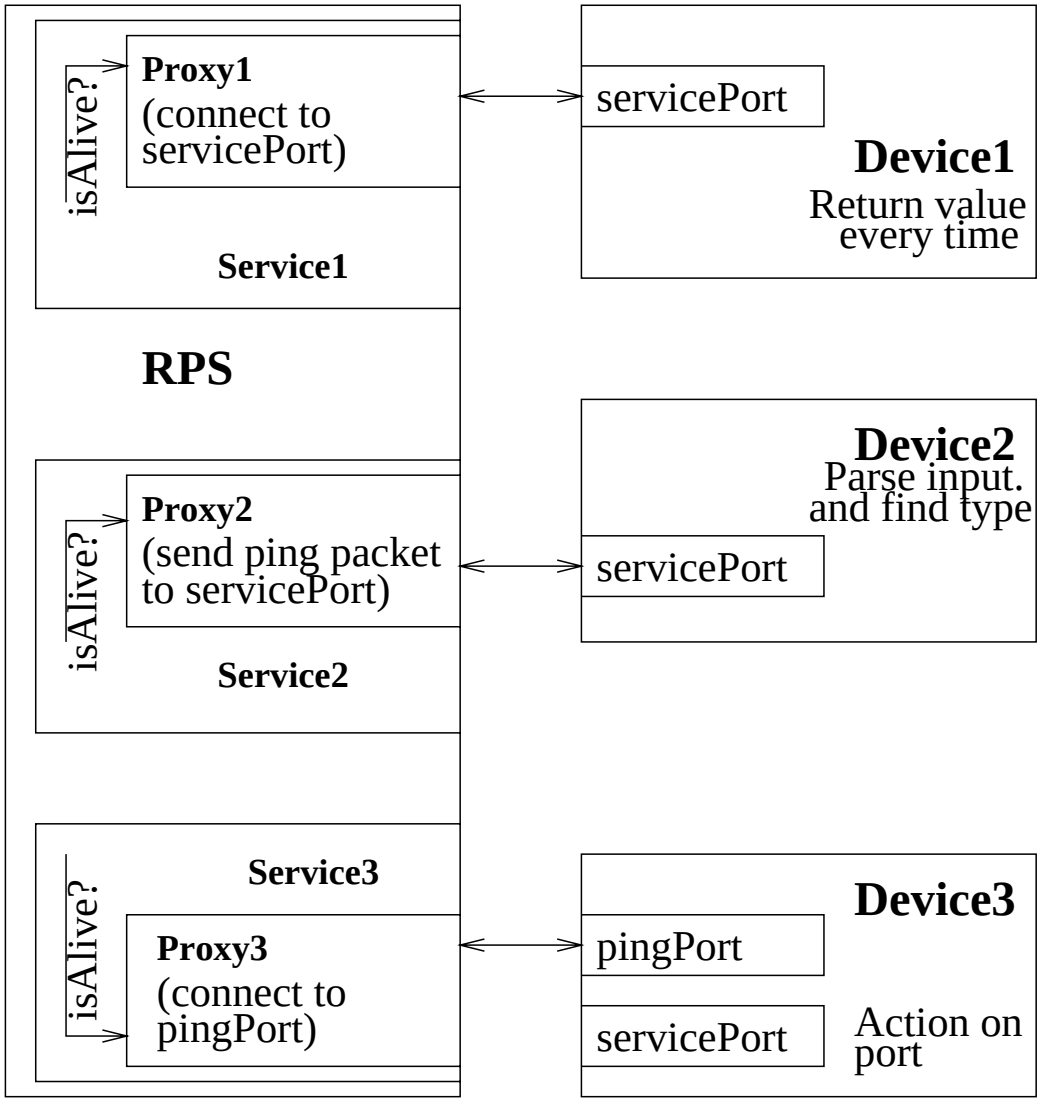
Code Download Issues



Change codebase in RPS to service codebase

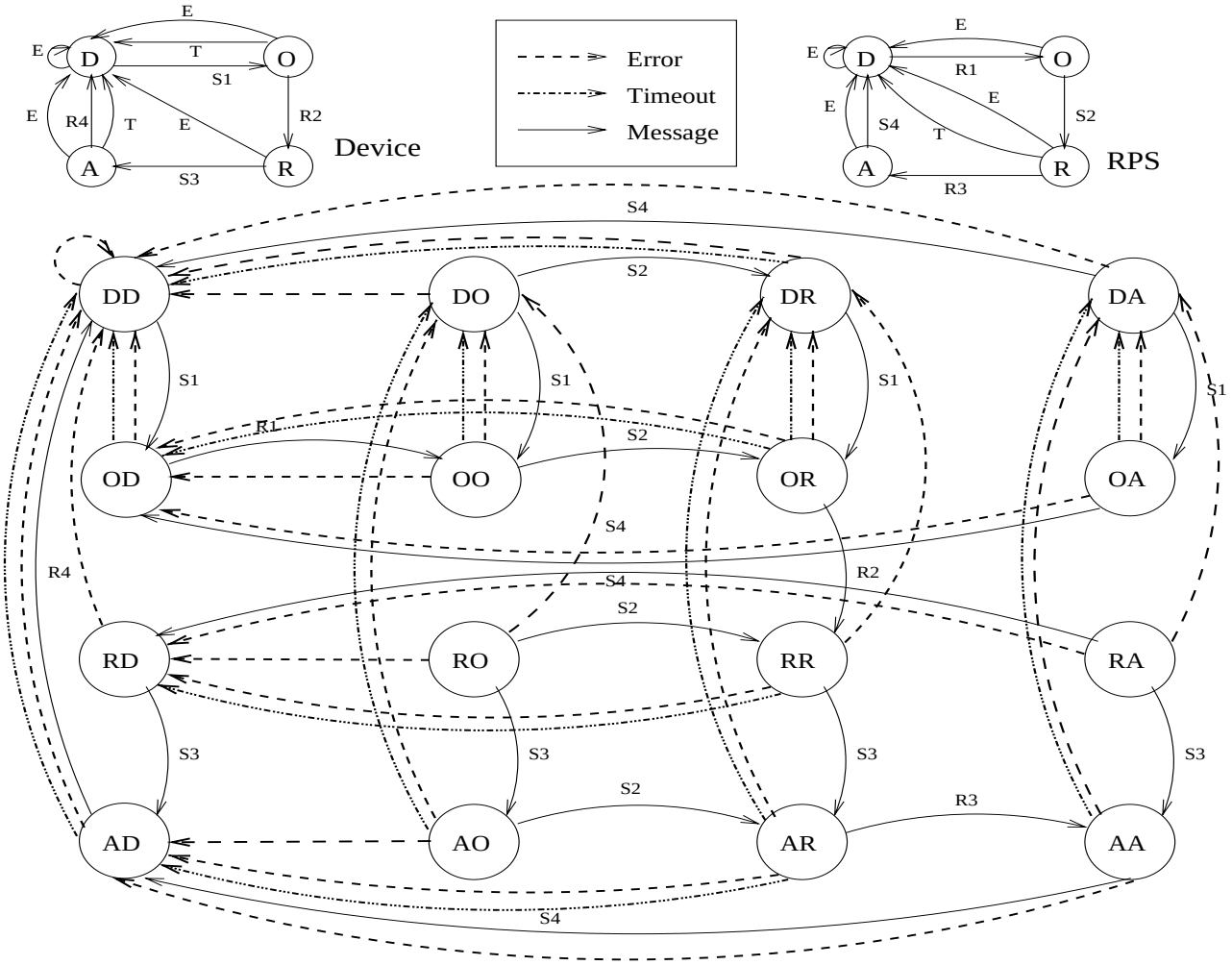
Side effect: Add security property to policy file

Ping Methods



- Different methods suitable for different configurations
- The `isAlive` method in the proxy allows different ping methods used in the same RPS.

Formal Verification



Conclusions

- Jini = set of protocols
- Built on Java
- Handles many of the issues required to plug and use devices
- Difficult to get it to work on embedded devices

But,

- With our modifications, it can!