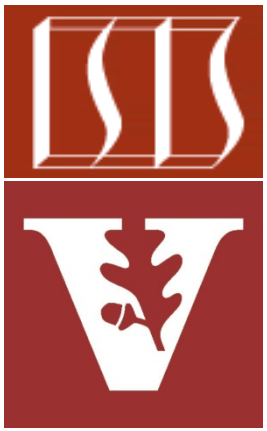
Android Services & Local IPC: The Activator Pattern (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



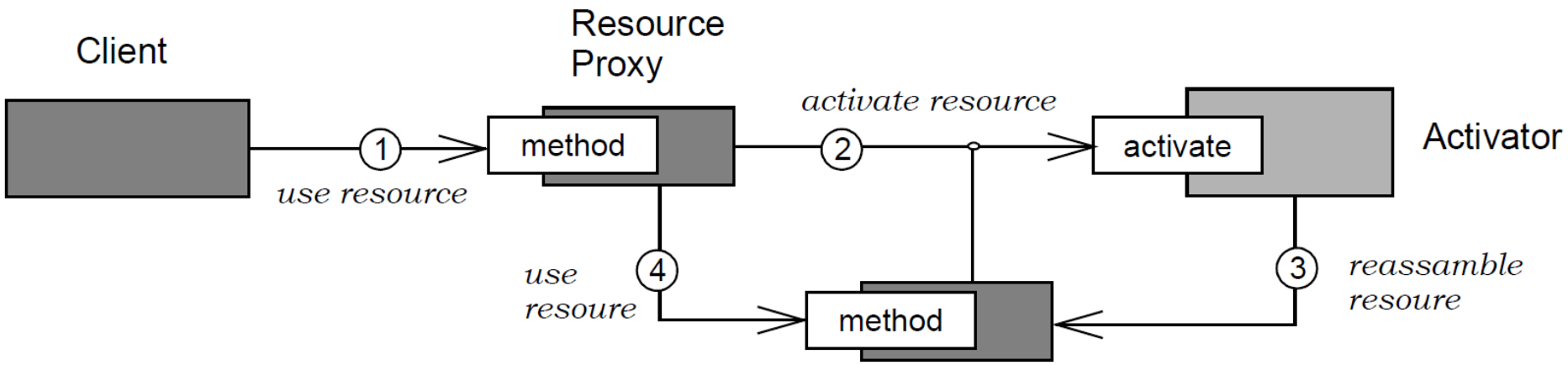
Professor of Computer Science

Institute for Software Integrated Systems

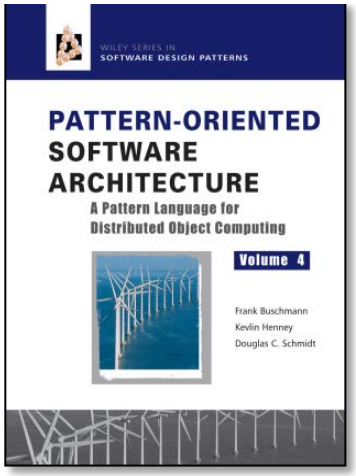
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand the *Activator* pattern



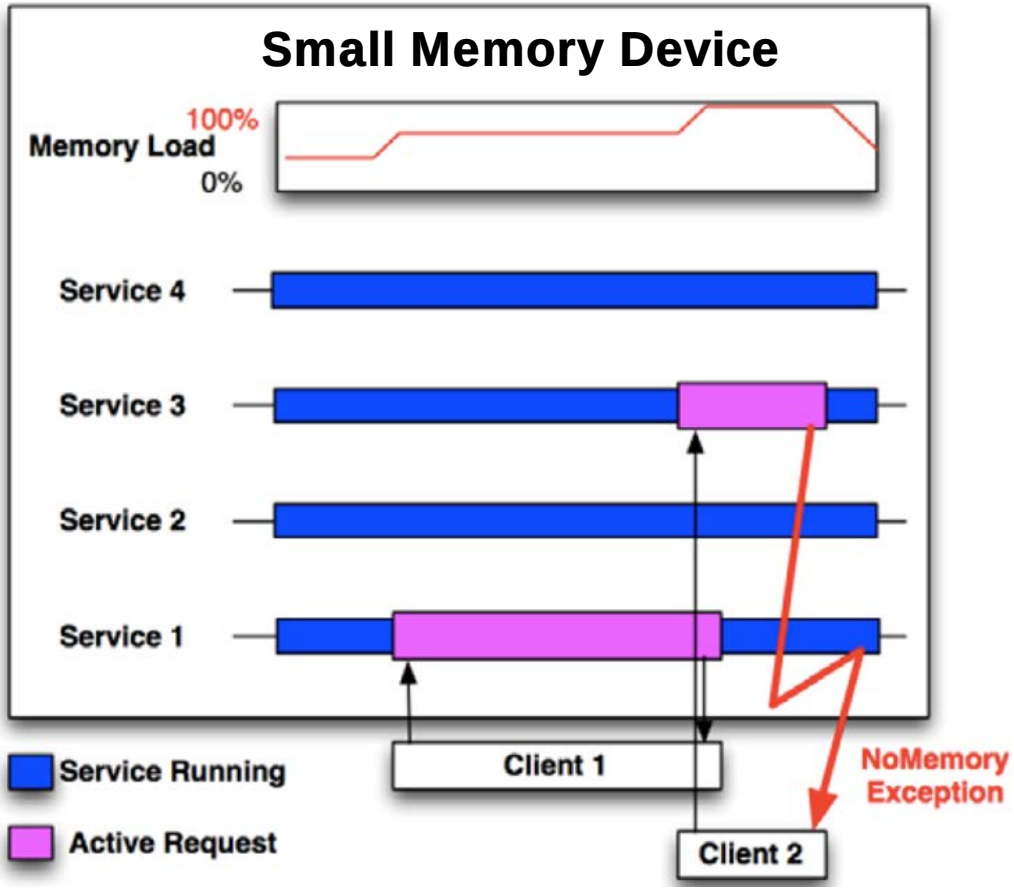

Efficiently and Transparently Automating Scalable On-demand Activation and Deactivation of Services with the Activator Pattern



Challenge: Minimizing Resource Utilization

Context

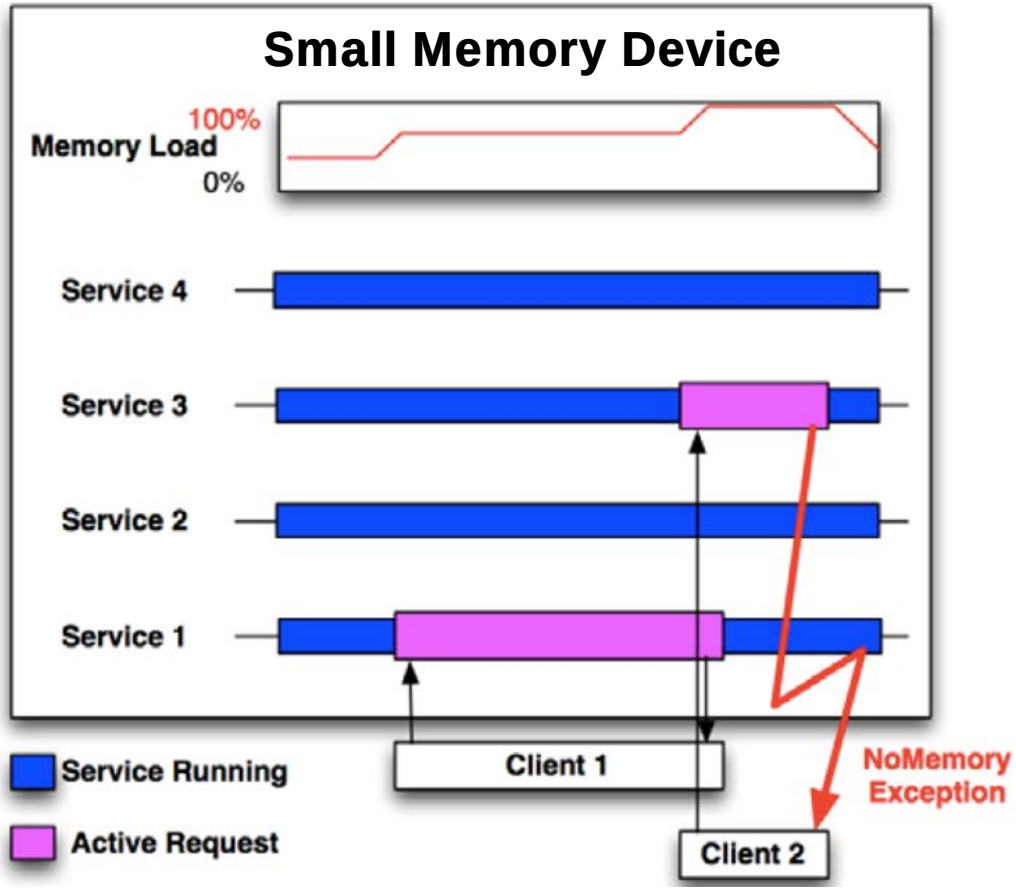
- Resource constrained & highly dynamic environments
  - Random-access memory (RAM) is a valuable resource in any software environment



Challenge: Minimizing Resource Utilization

Context

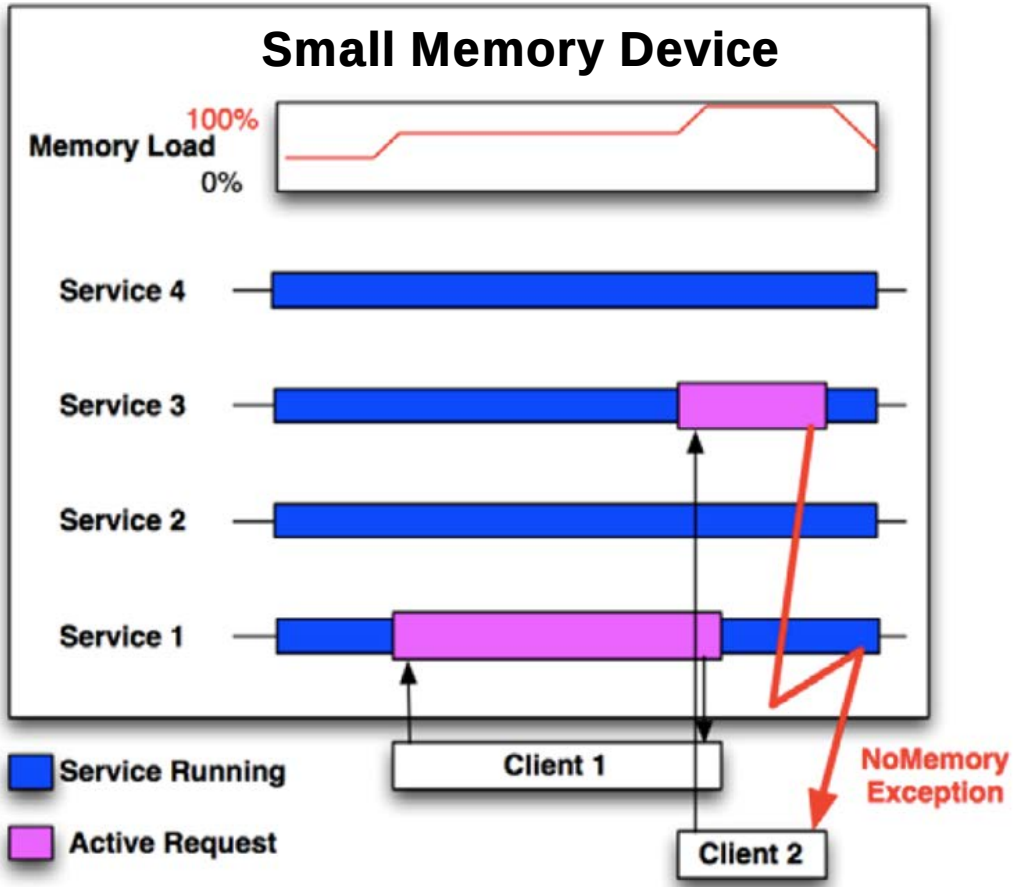
- Resource constrained & highly dynamic environments
  - Random-access memory (RAM) is a valuable resource in any software environment
  - It's even more valuable on a mobile OS like Android where physical memory is often constrained



Challenge: Processing a Long-Running Action

Problem

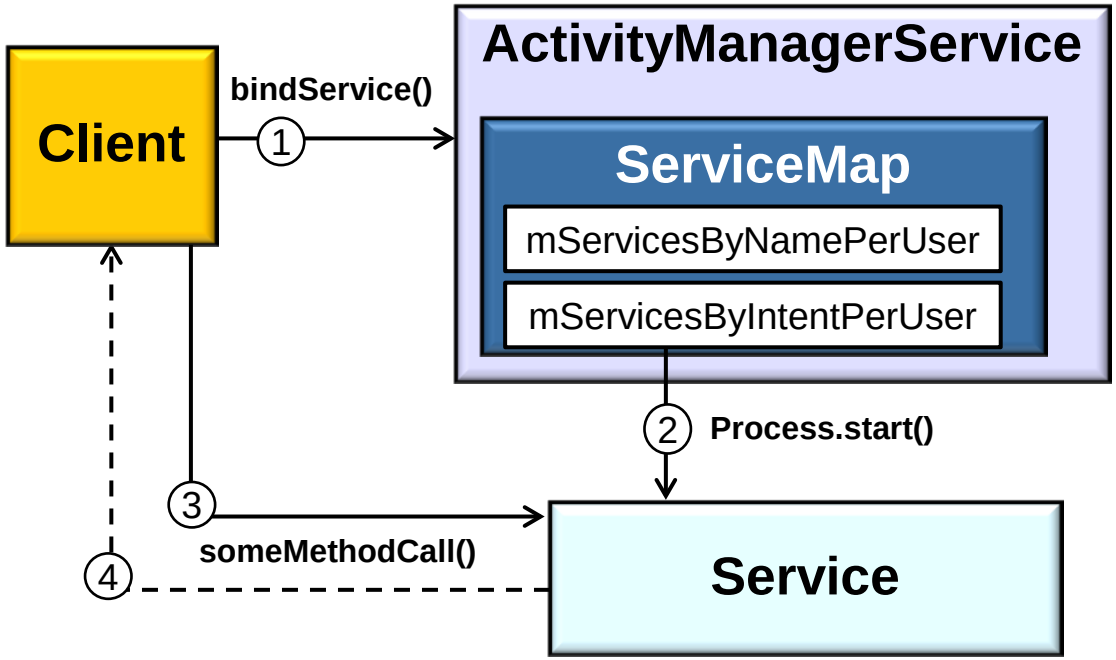
- It's not feasible to have all App Service implementations running all the time since this ties up end-system resources unnecessarily



Challenge: Processing a Long-Running Action

Solution

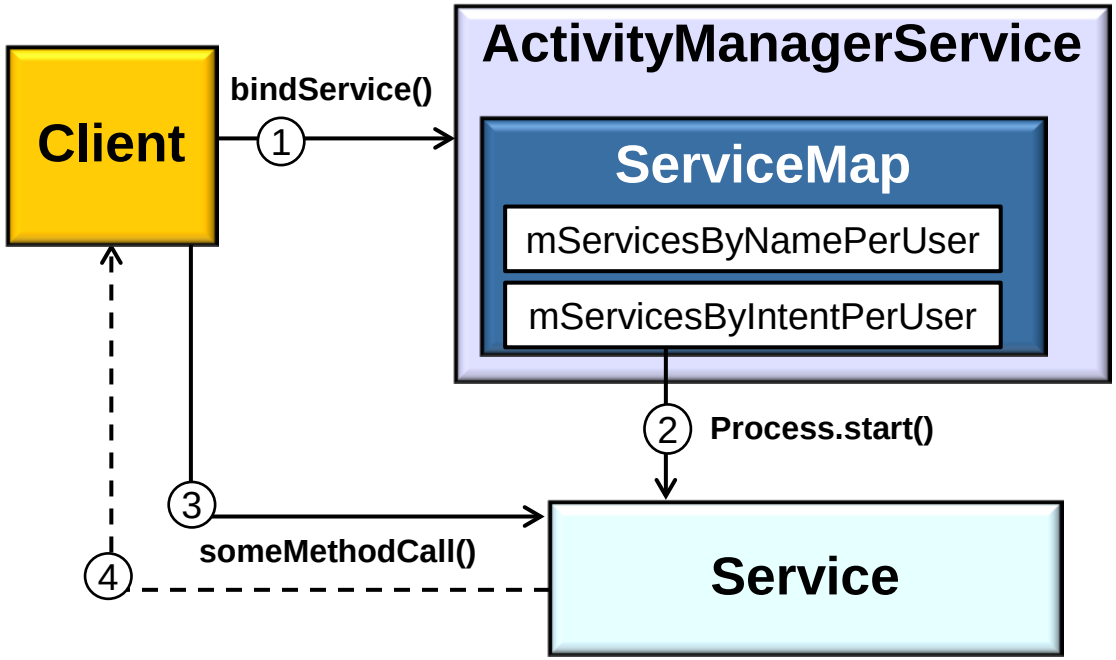
- Apply the *Activator* pattern to activate & deactivate Android Services automatically
  - If your app needs a Service to perform work in the background, don't keep it running unless it's actively performing a job



Challenge: Processing a Long-Running Action

Solution

- Apply the *Activator* pattern to activate & deactivate Android Services automatically
  - If your app needs a Service to perform work in the background, don't keep it running unless it's actively performing a job
  - Be careful to never leak your Service by failing to stop it when its work is done

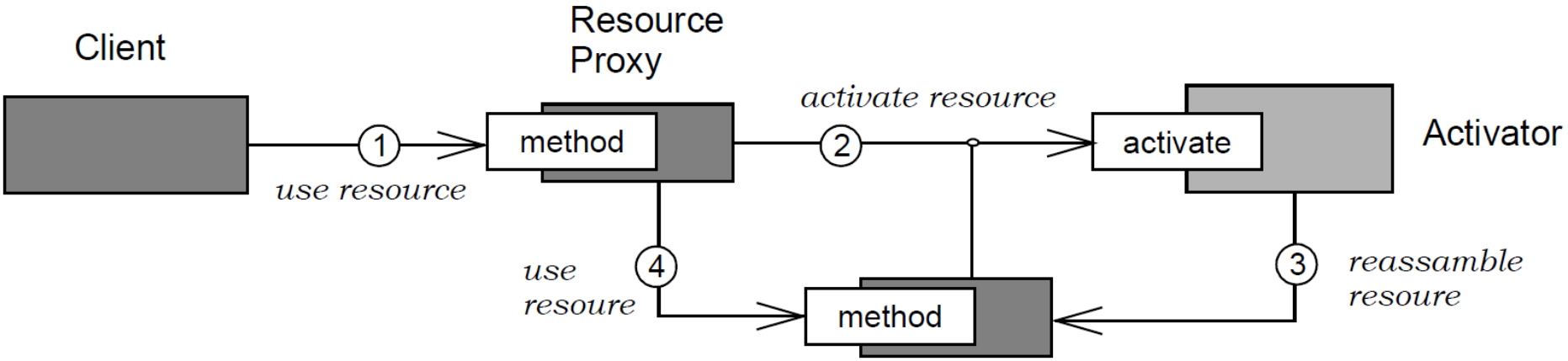


developer.android.com/training/articles/memory.html#Services has more info

Activator POSA4 Design Pattern

Intent

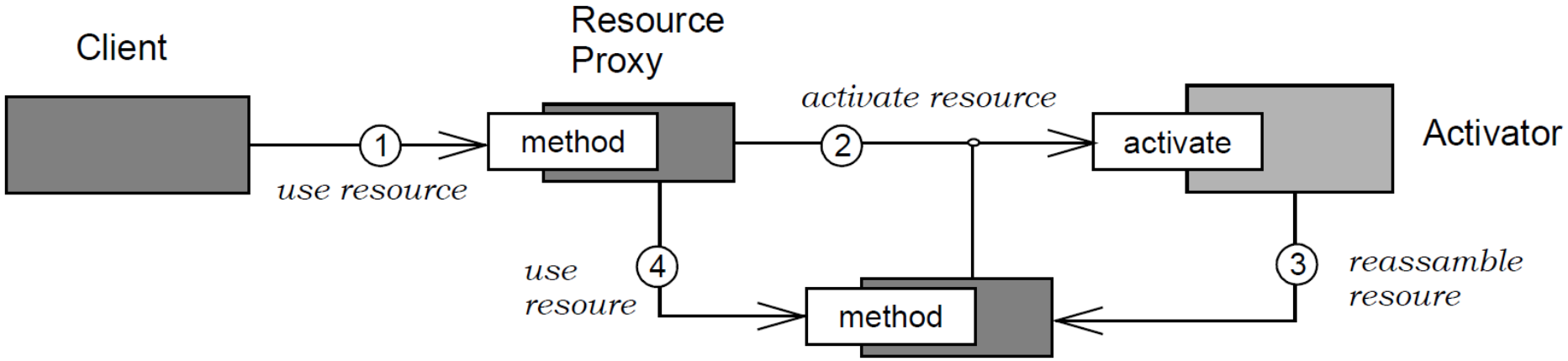
- *Activator* automates scalable on-demand activation & deactivation of service execution contexts to run services accessed by many clients without consuming excessive resources



Activator POSA4 Design Pattern

Applicability

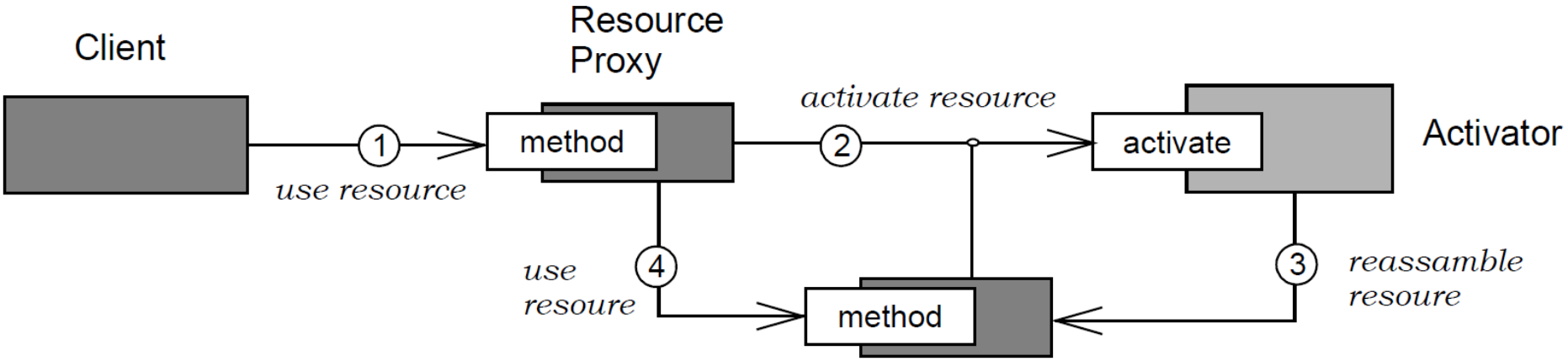
- When services in a system should only consume resources when they are accessed actively by clients



Activator POSA4 Design Pattern

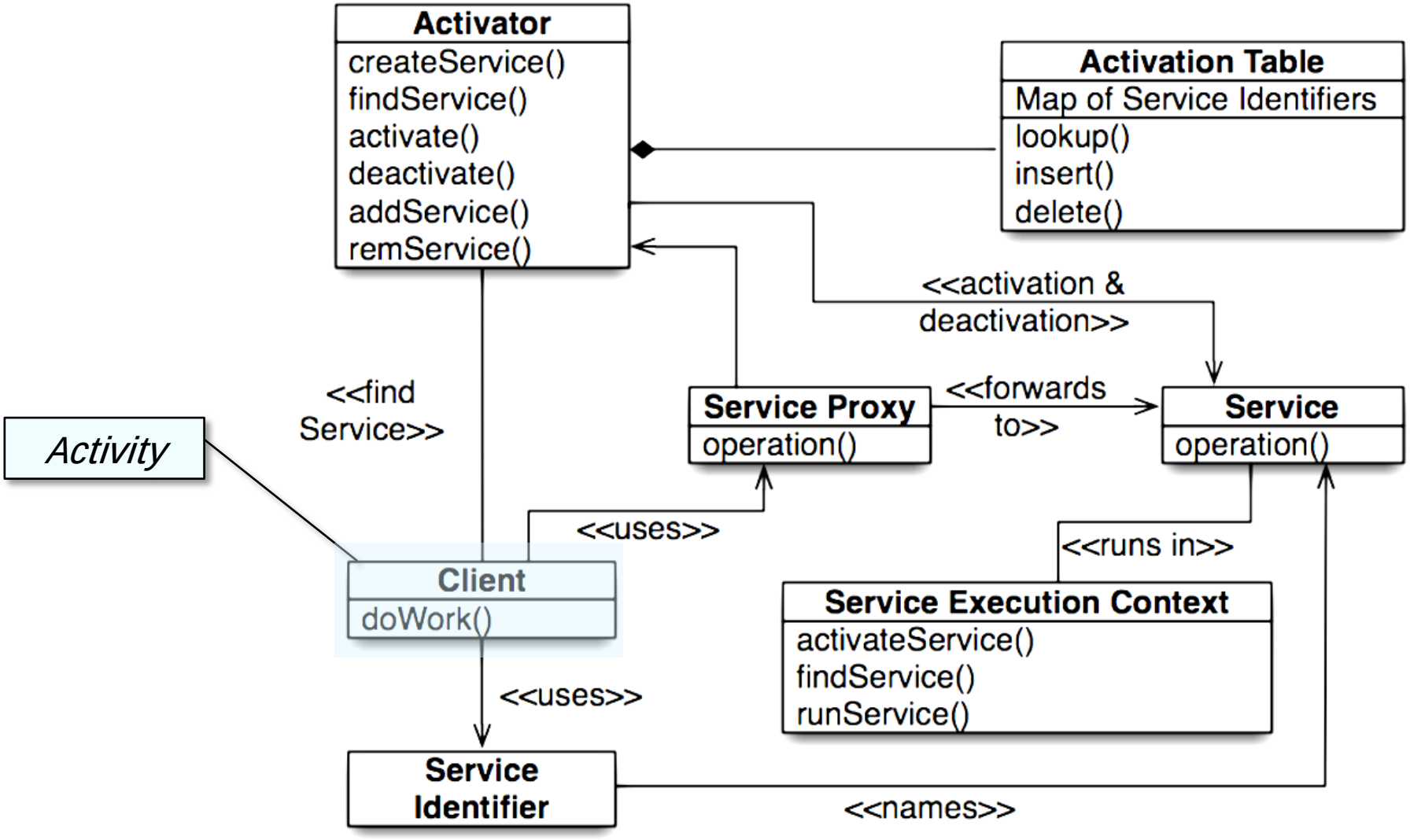
Applicability

- When services in a system should only consume resources when they are accessed actively by clients
- When clients should be shielded from where services are located, how they are deployed onto hosts or processes, & how their lifecycle is managed



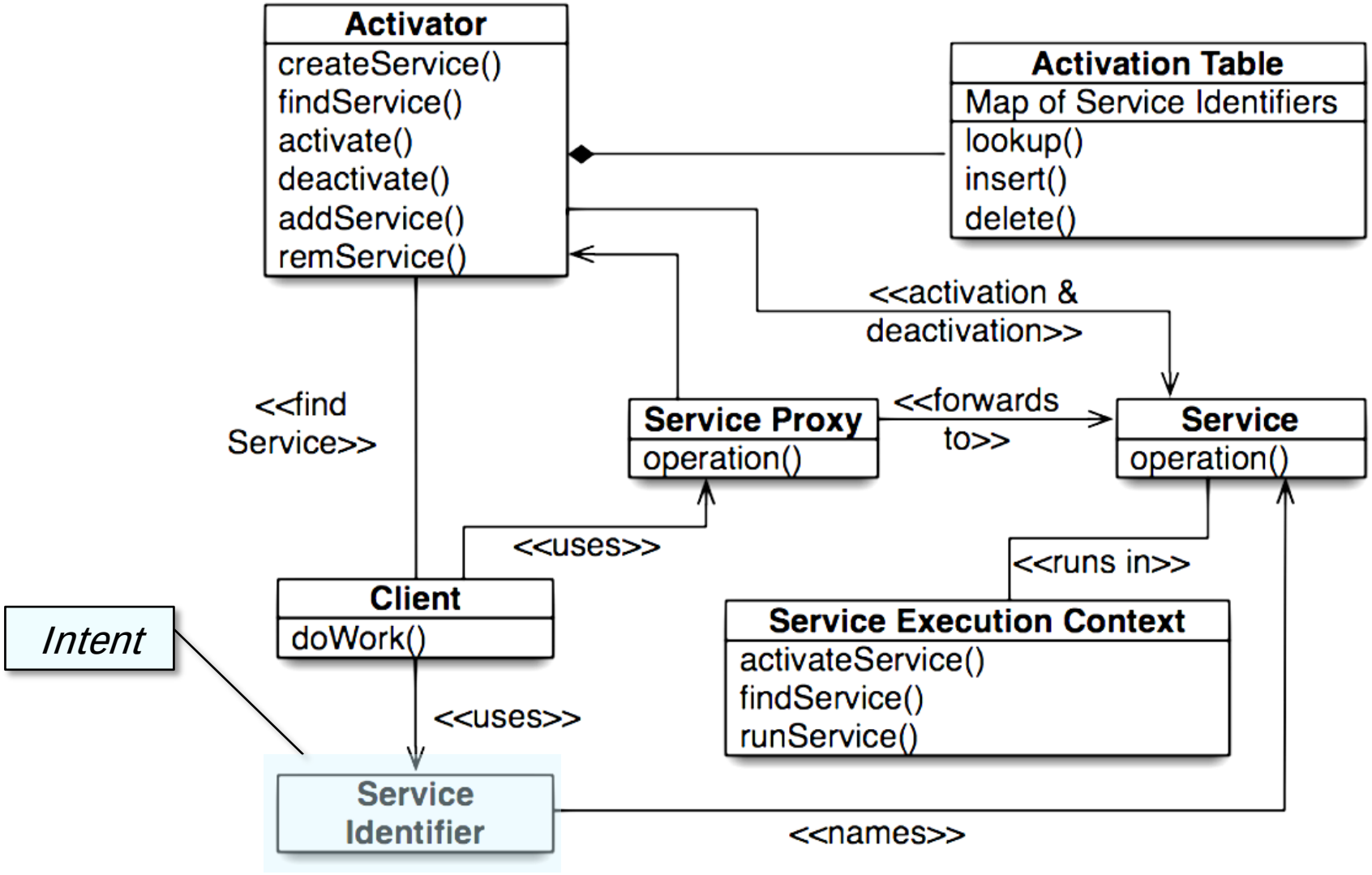
Activator POSA4 Design Pattern

Structure & Participants



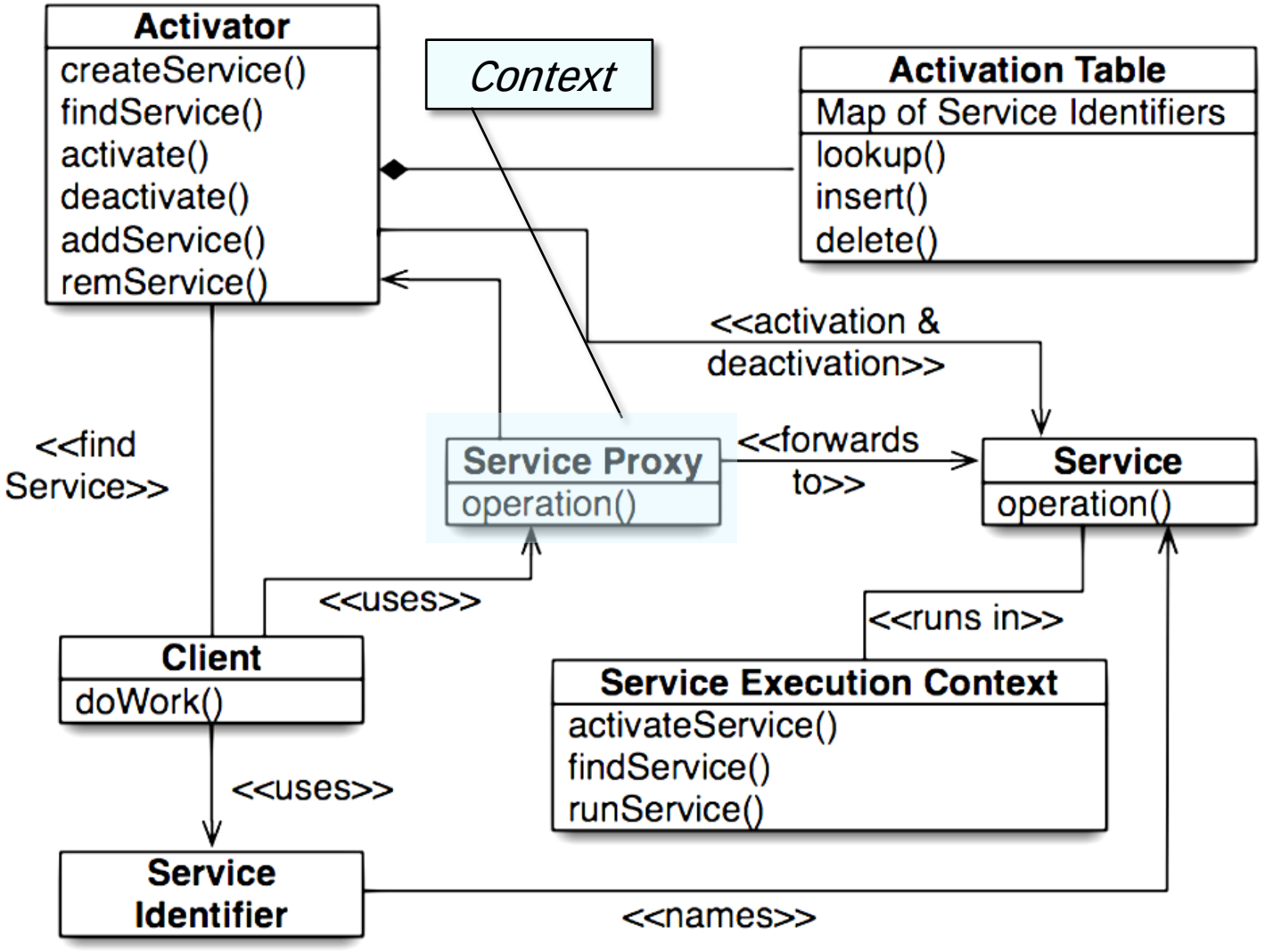
Activator POSA4 Design Pattern

Structure & Participants



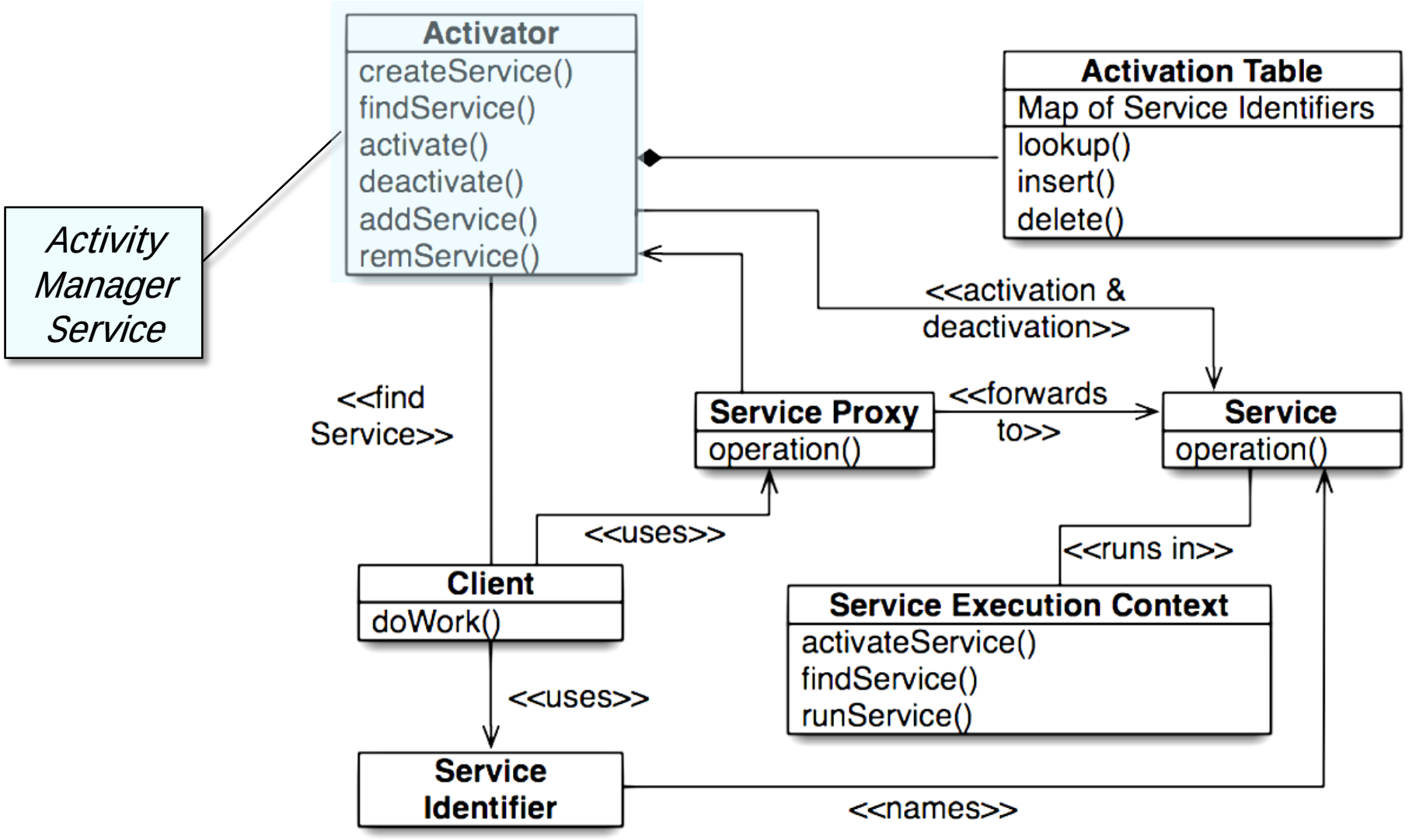
Activator POSA4 Design Pattern

Structure & Participants



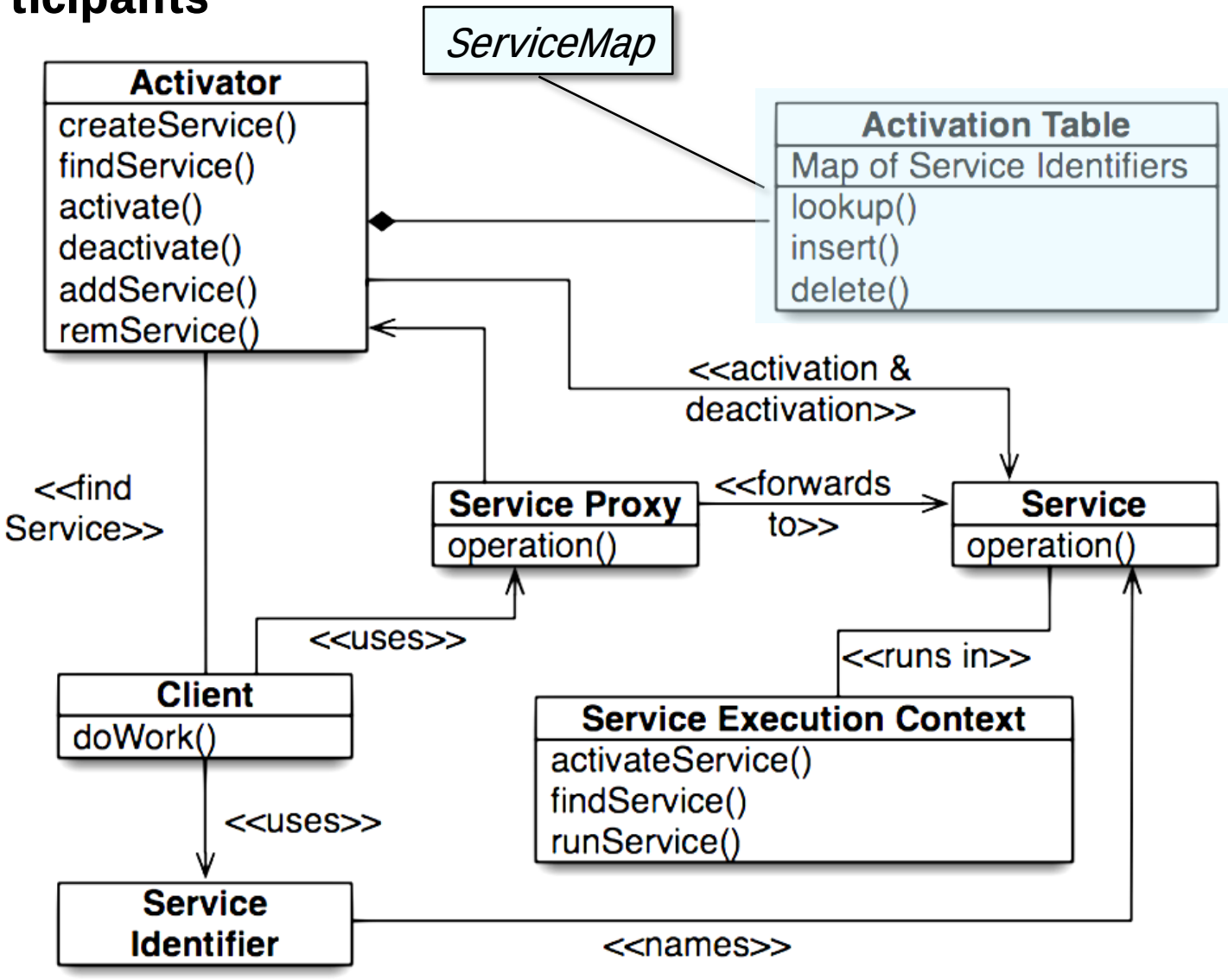
Activator POSA4 Design Pattern

Structure & Participants



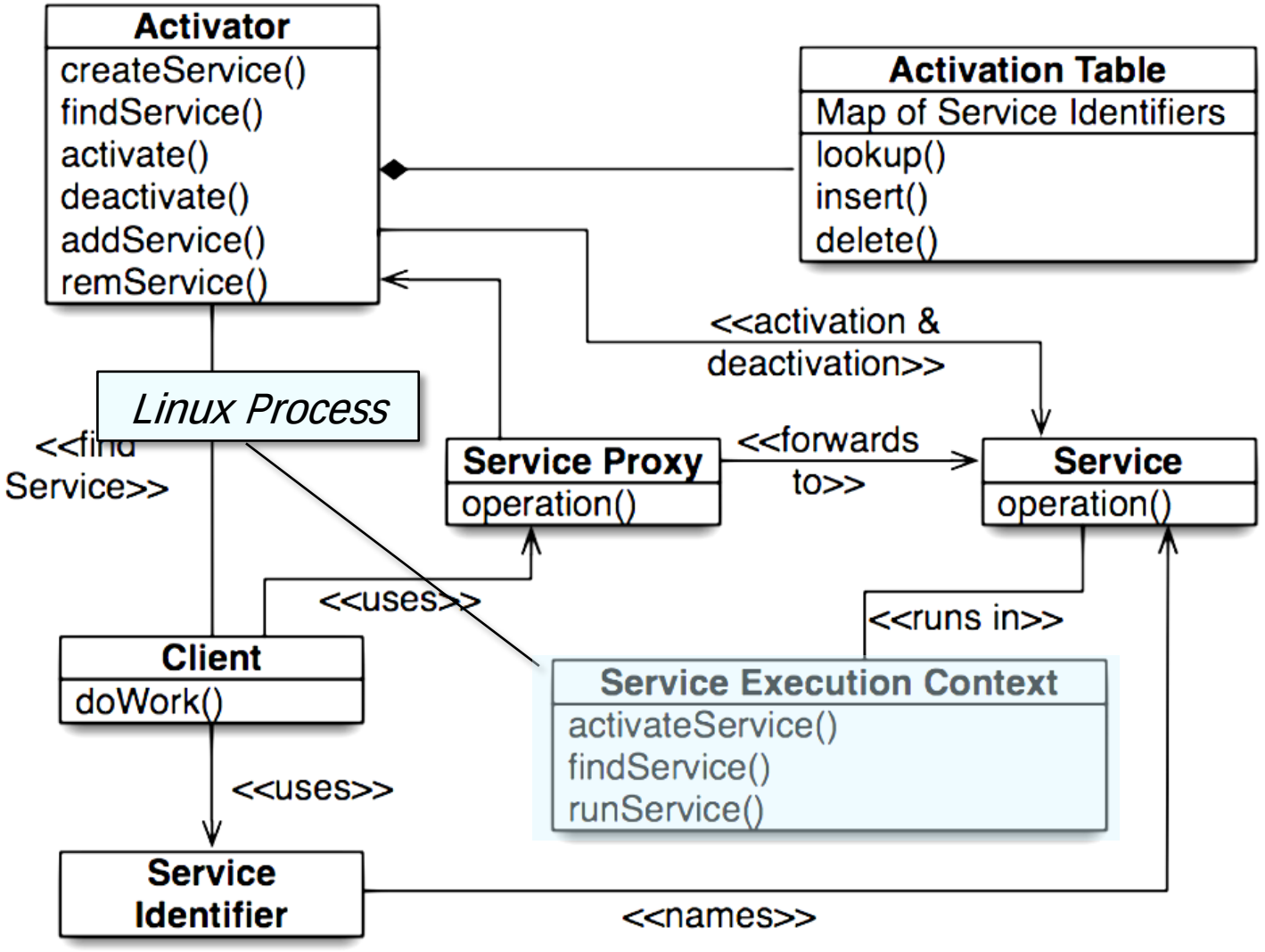
Activator POSA4 Design Pattern

Structure & Participants



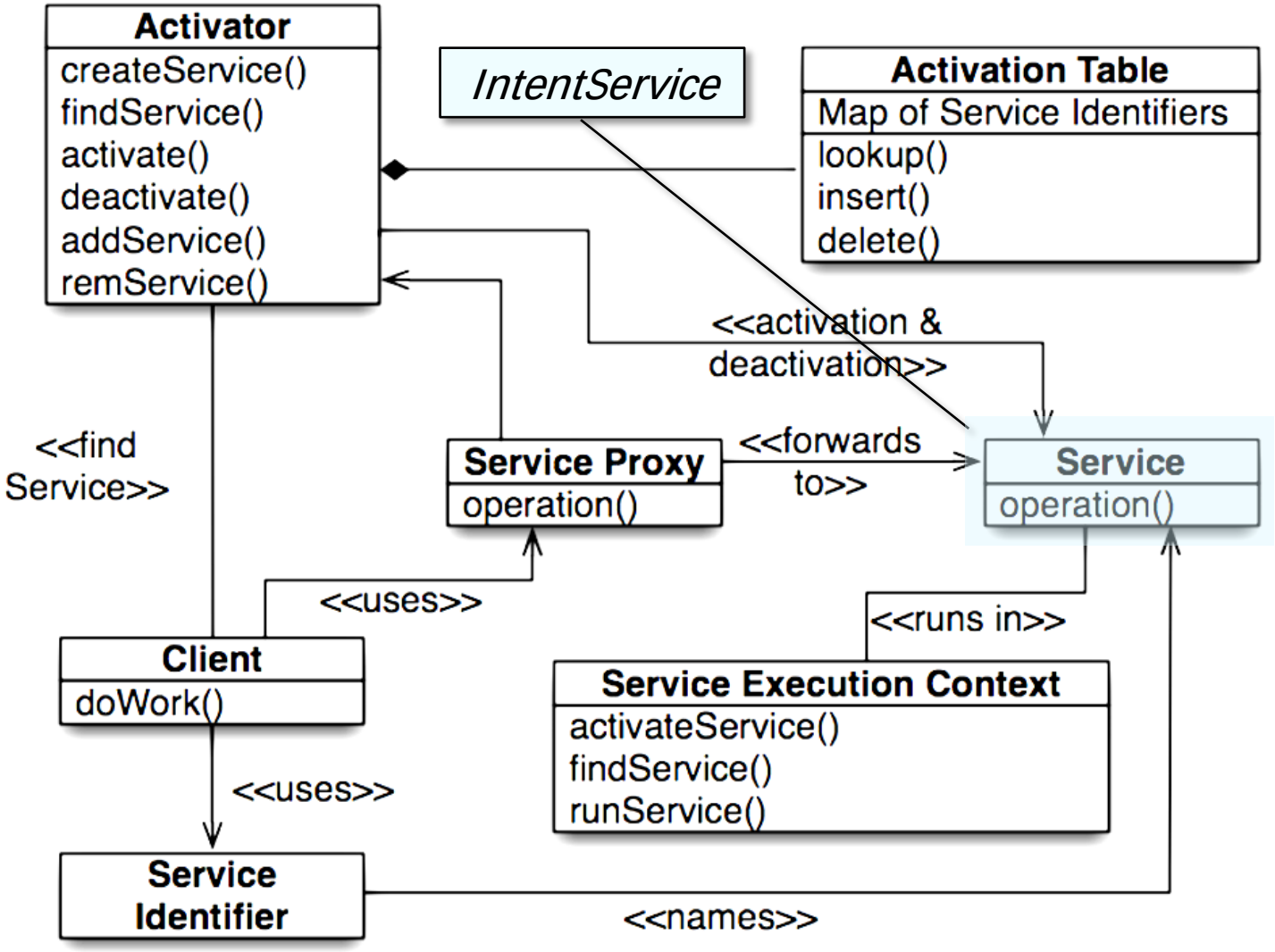
Activator POSA4 Design Pattern

Structure & Participants



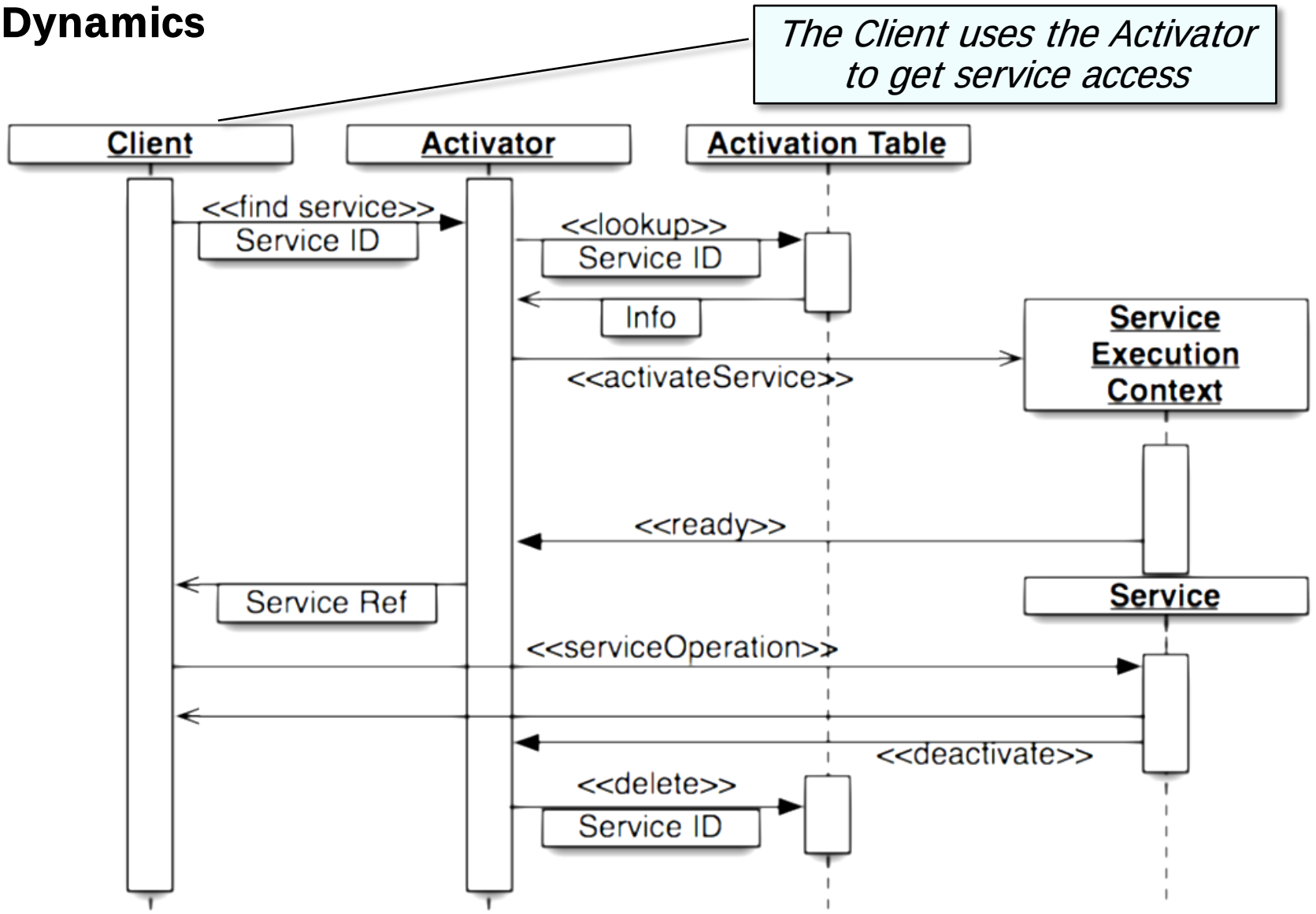
Activator POSA4 Design Pattern

Structure & Participants



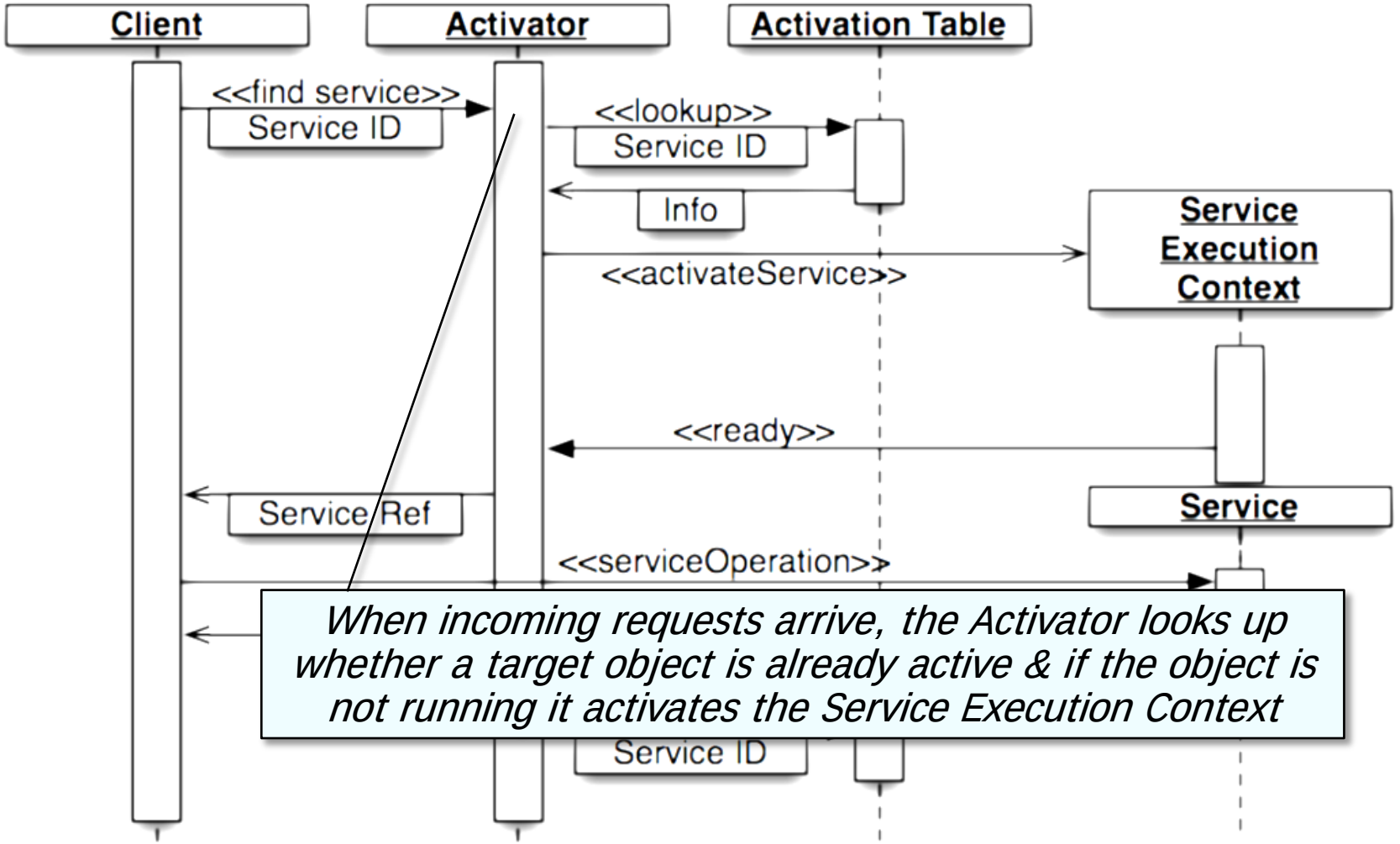
Activator POSA4 Design Pattern

Dynamics



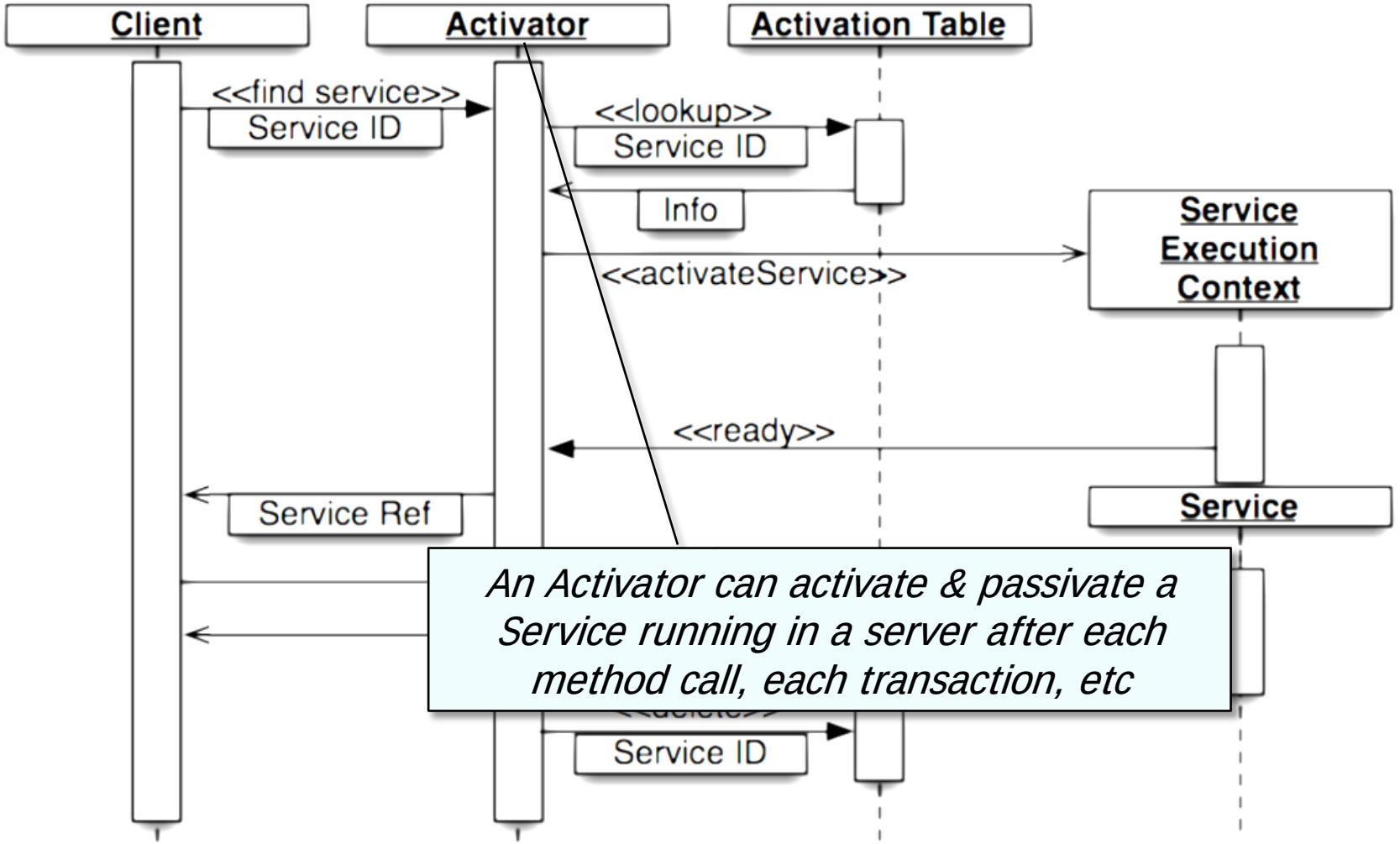
Activator POSA4 Design Pattern

Dynamics



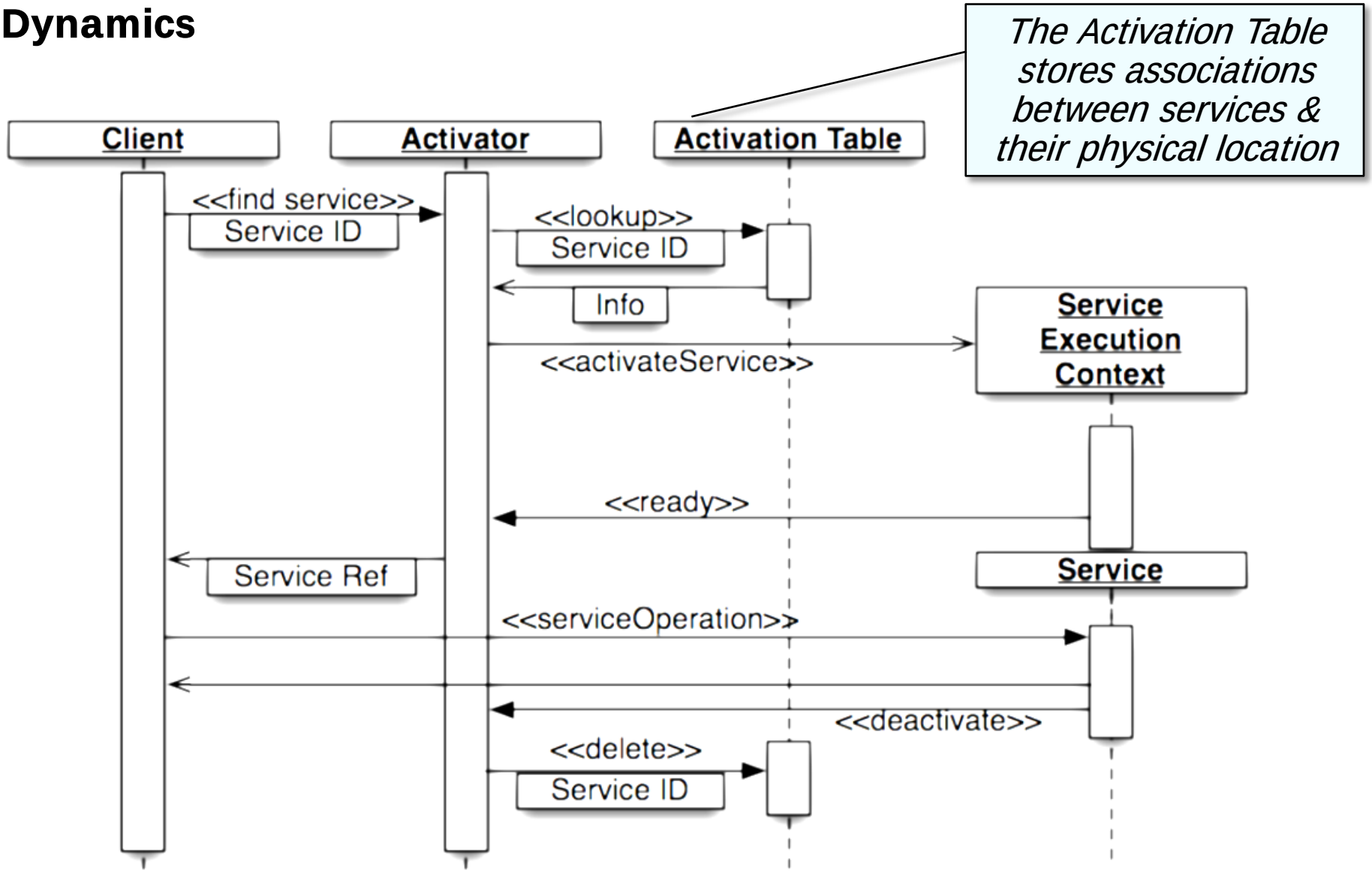
Activator POSA4 Design Pattern

Dynamics



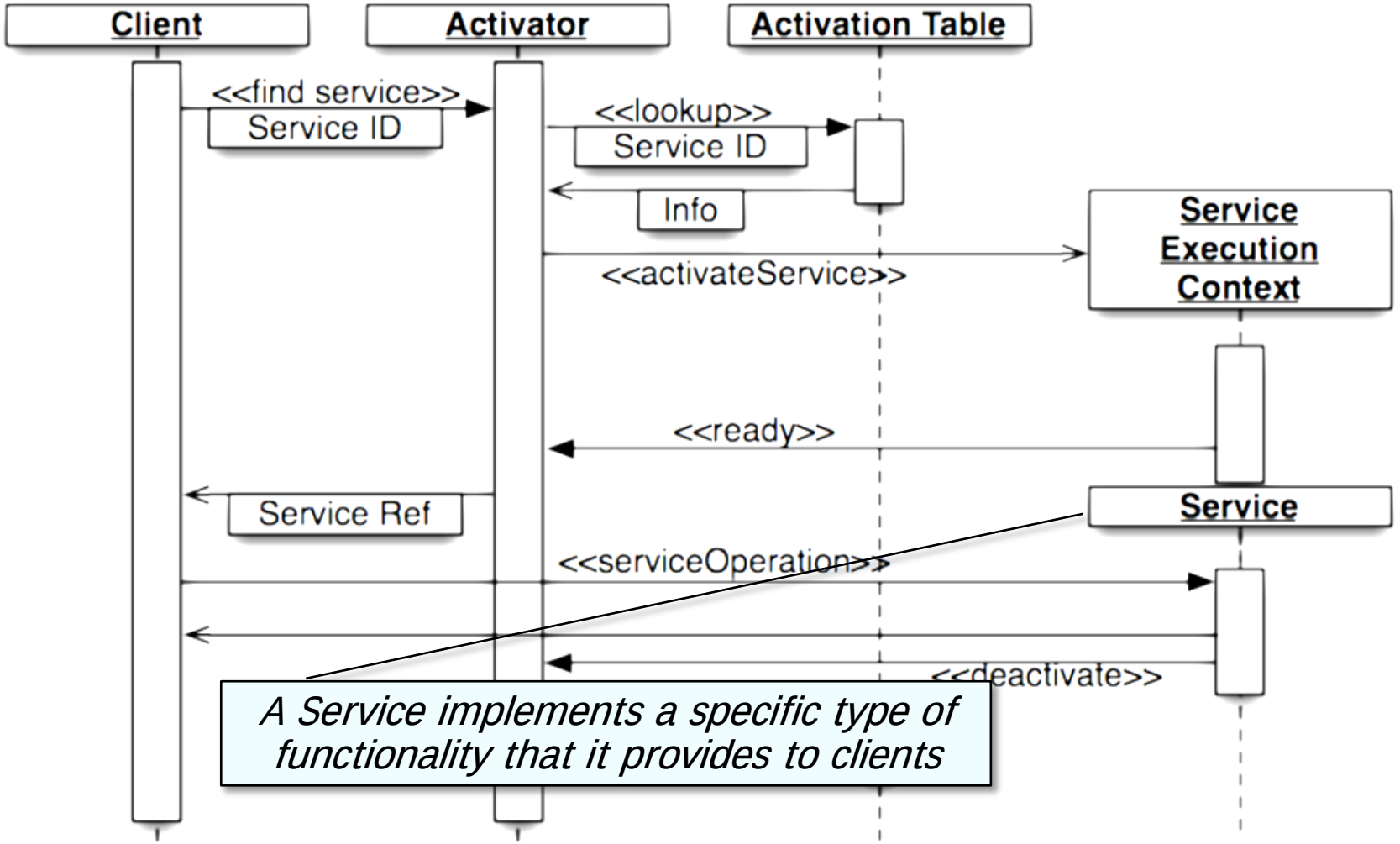
Activator POSA4 Design Pattern

Dynamics



Activator POSA4 Design Pattern

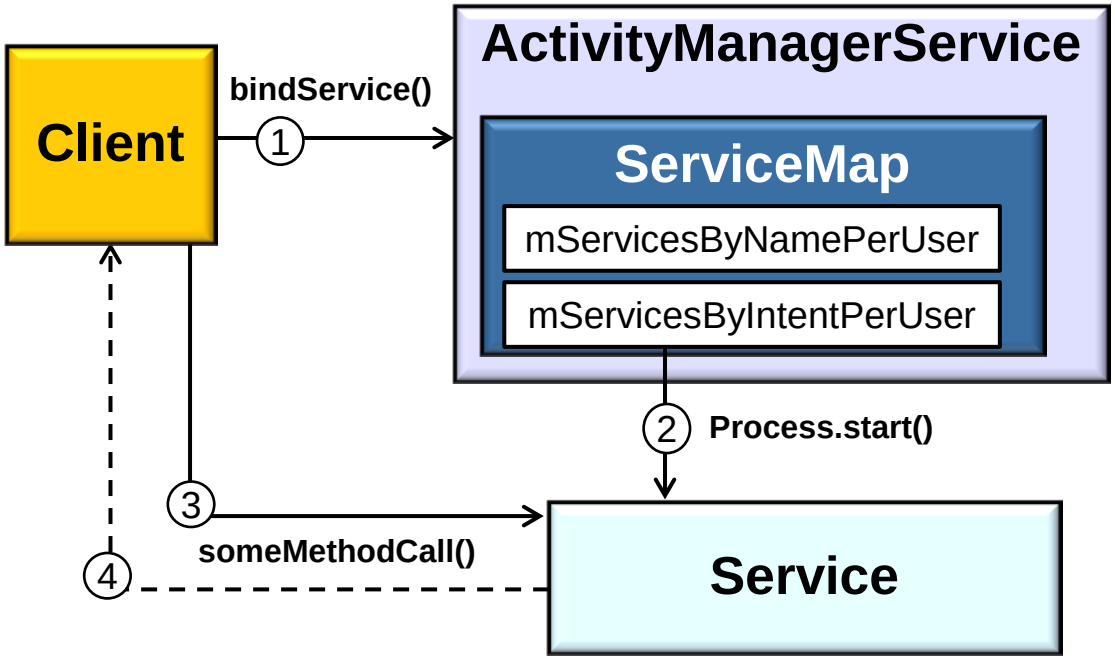
Dynamics



Activator POSA4 Design Pattern

Consequences

+ More effective resource utilization
  - Servers can be spawned "on-demand," thereby minimizing resource utilization until clients actually require them

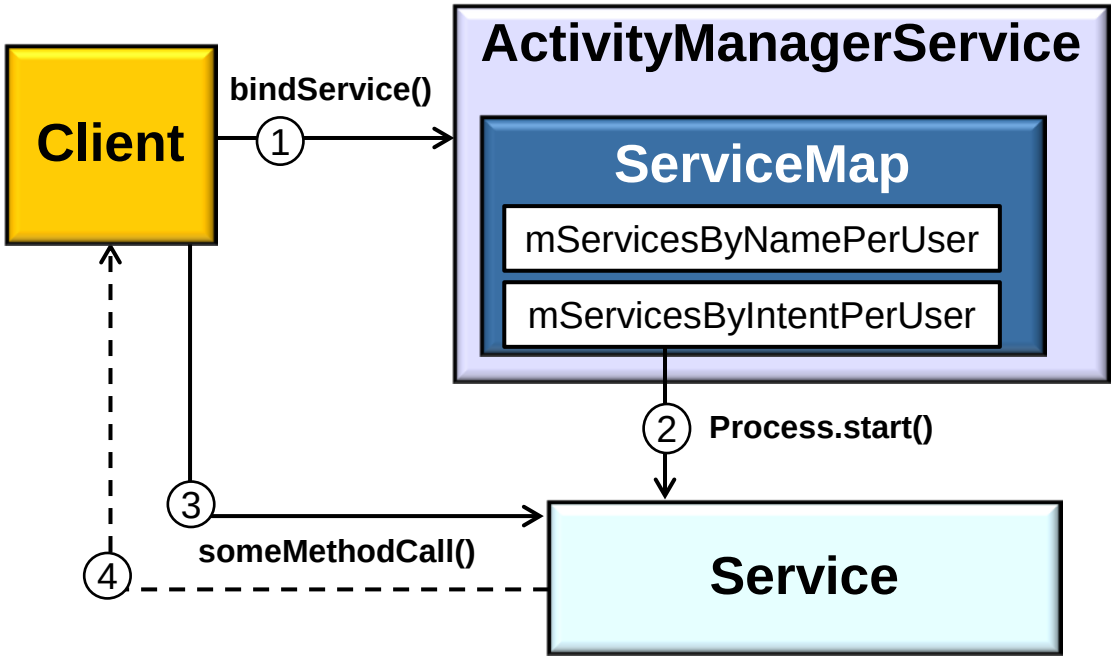


Activator POSA4 Design Pattern

Consequences

+ More effective resource utilization

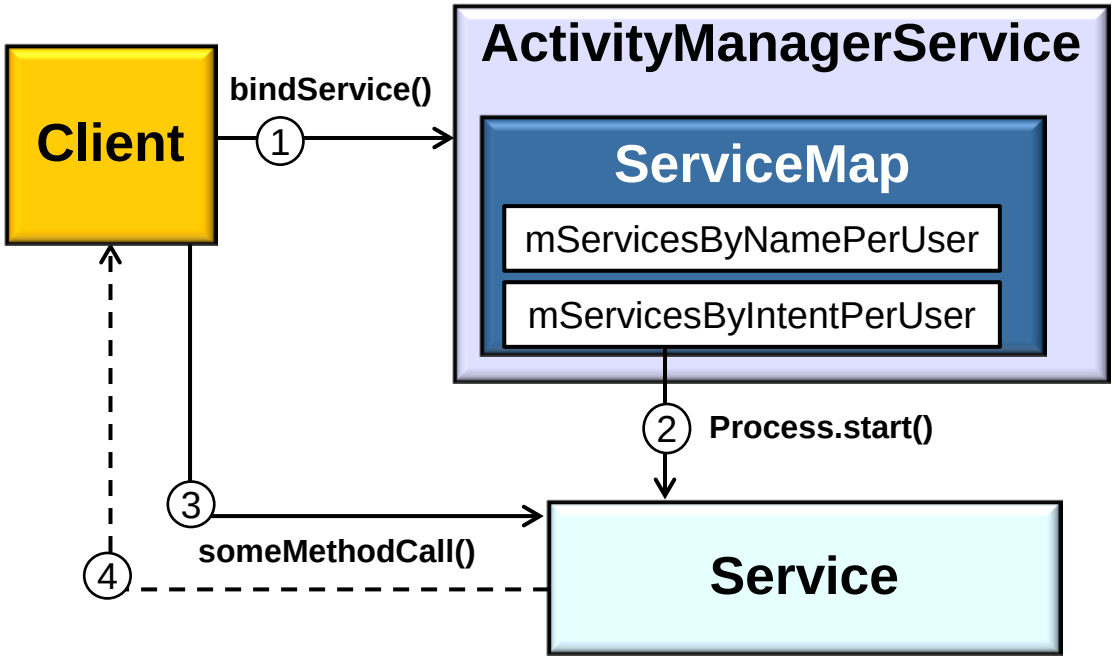
+ Coarse-grained concurrency
  - By spawning server processes to run on multi-core/CPU computers



Activator POSA4 Design Pattern

Consequences

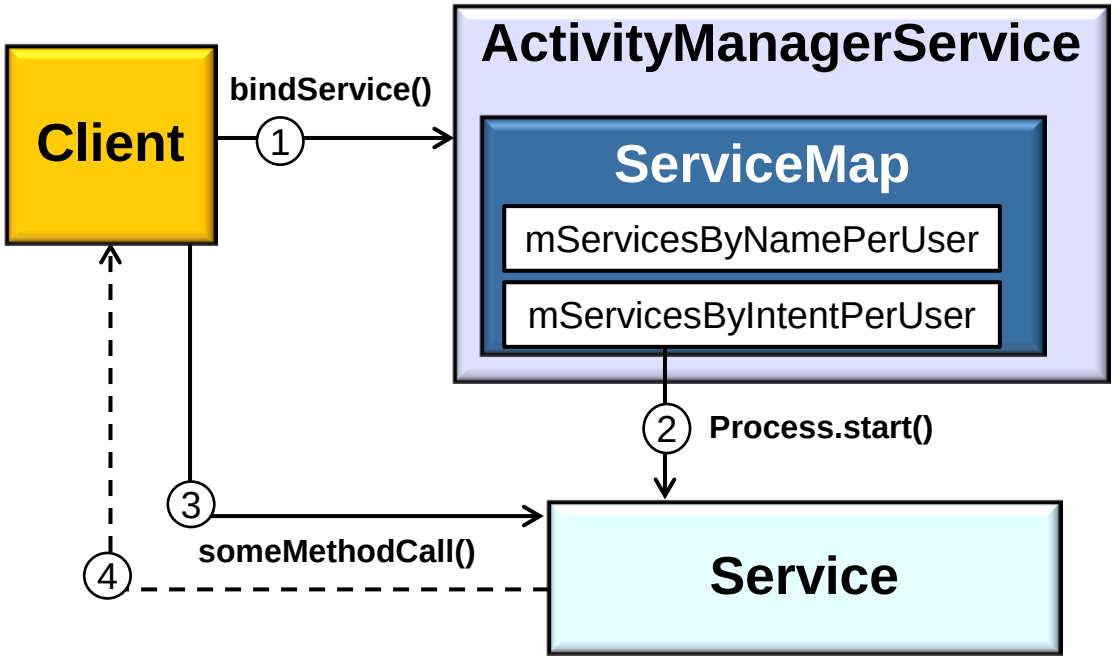
+ More effective resource utilization
+ Coarse-grained concurrency
+ Modularity, testability, & reusability
  - Application modularity & reusability is improved by decoupling server implementations from the manner in which the servers are activated



Activator POSA4 Design Pattern

Consequences

– Lack of determinism & ordering dependencies
  - Hard to determine or analyze the behavior of an app until its components are activated at runtime



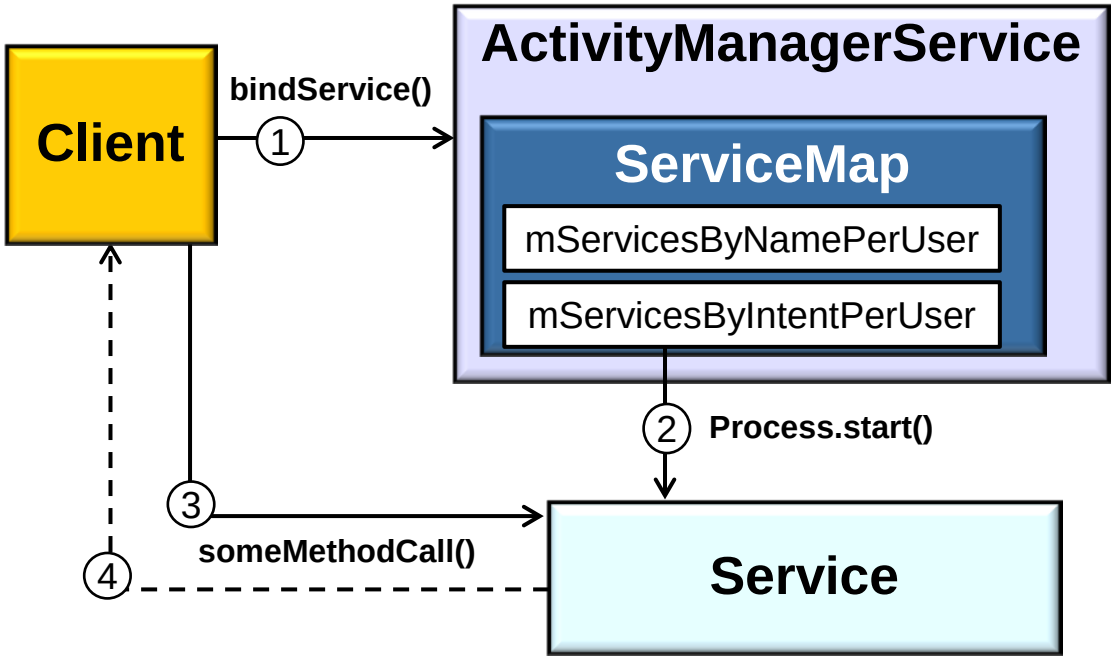
Activator POSA4 Design Pattern

Consequences

– Lack of determinism & ordering dependencies

– Reduced security & reliability

- An application that uses *Activator* may be less secure or reliable than an equivalent statically-configured application



Activator POSA4 Design Pattern

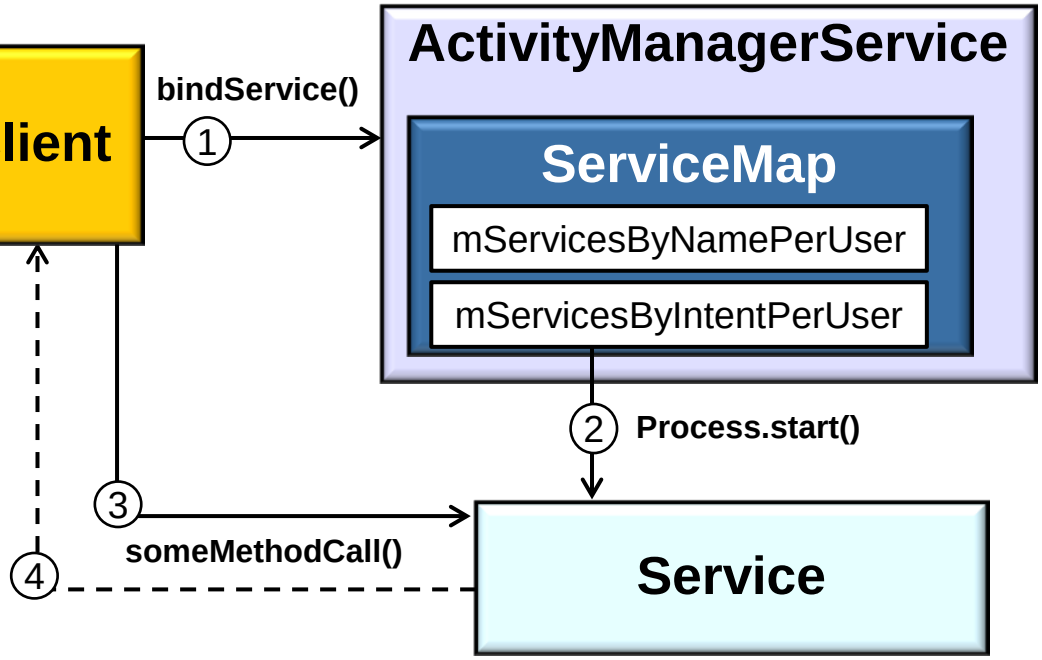
Consequences

– Lack of determinism & ordering dependencies

– Reduced security & reliability

– Increased run-time overhead & infrastructure complexity

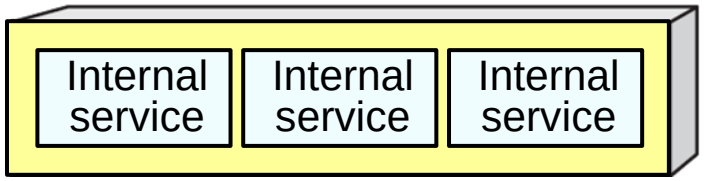
- By adding levels of abstraction & indirection when activating & executing components



Activator POSA4 Design Pattern

Known Uses

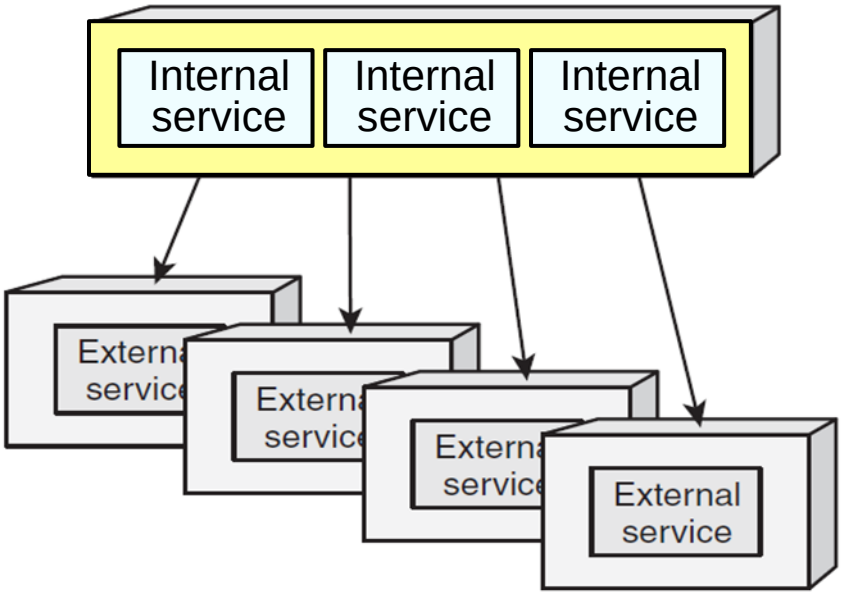
- UNIX Inetd “super server”
  - Internal services are fixed at static link time
    - e.g., **`ECHO`** & **`DAYTIME`**



Activator POSA4 Design Pattern

Known Uses

- UNIX Inetd “super server”
  - Internal services are fixed at static link time
  - External services can be dynamically reconfigured
    - e.g., **FTP**, **TELNET**, & **HTTP**

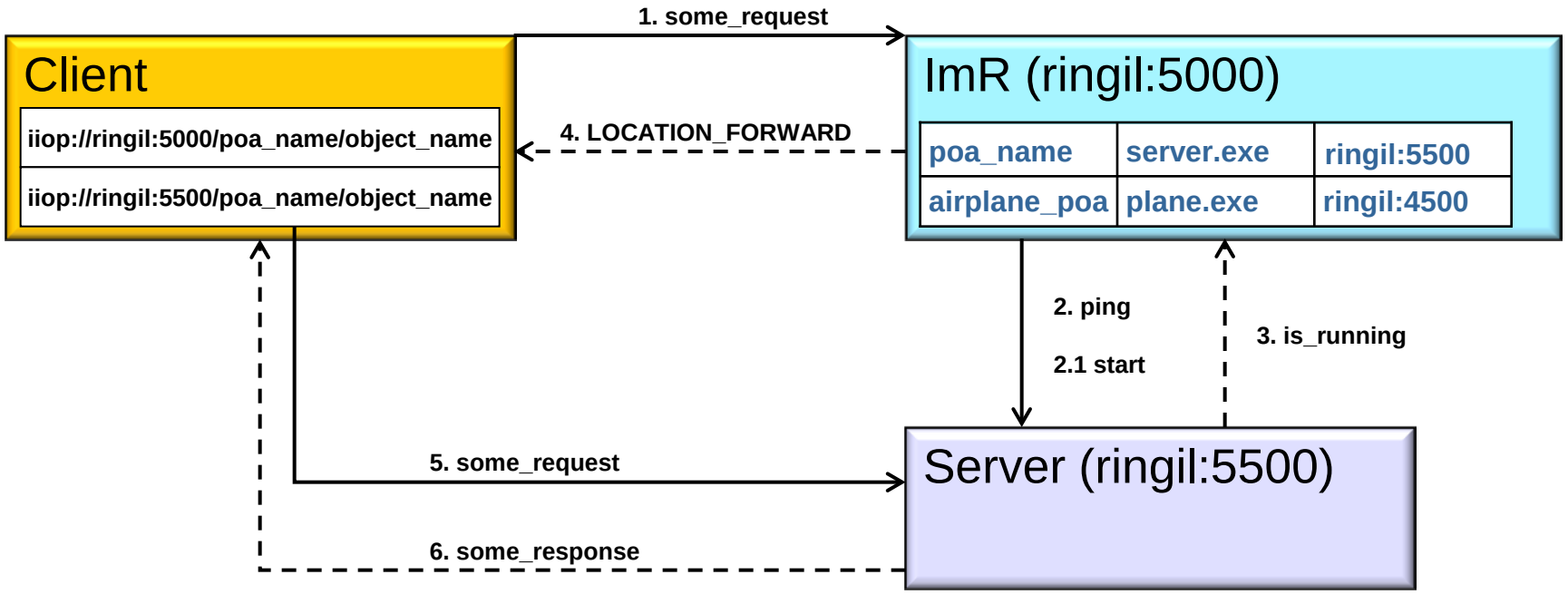


See en.wikipedia.org/wiki/Inetd for more on Inetd

Activator POSA4 Design Pattern

Known Uses

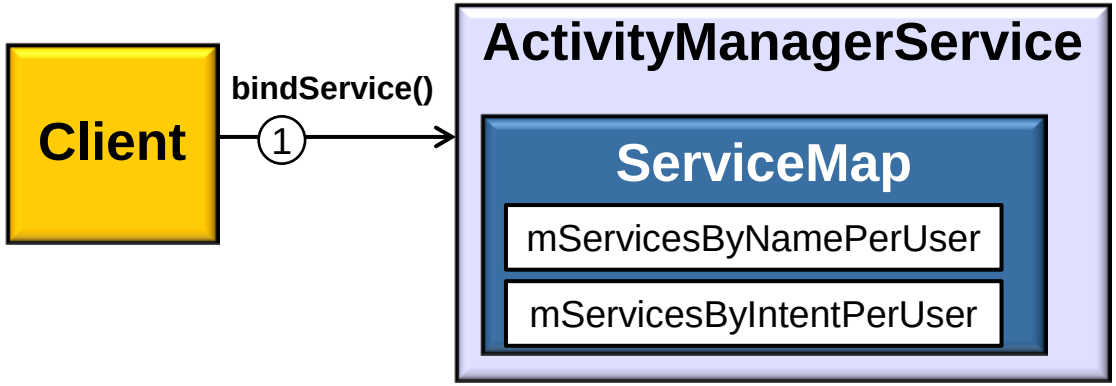
- UNIX Inetd "super server"
- CORBA Implementation Repository



Activator POSA4 Design Pattern

Known Uses

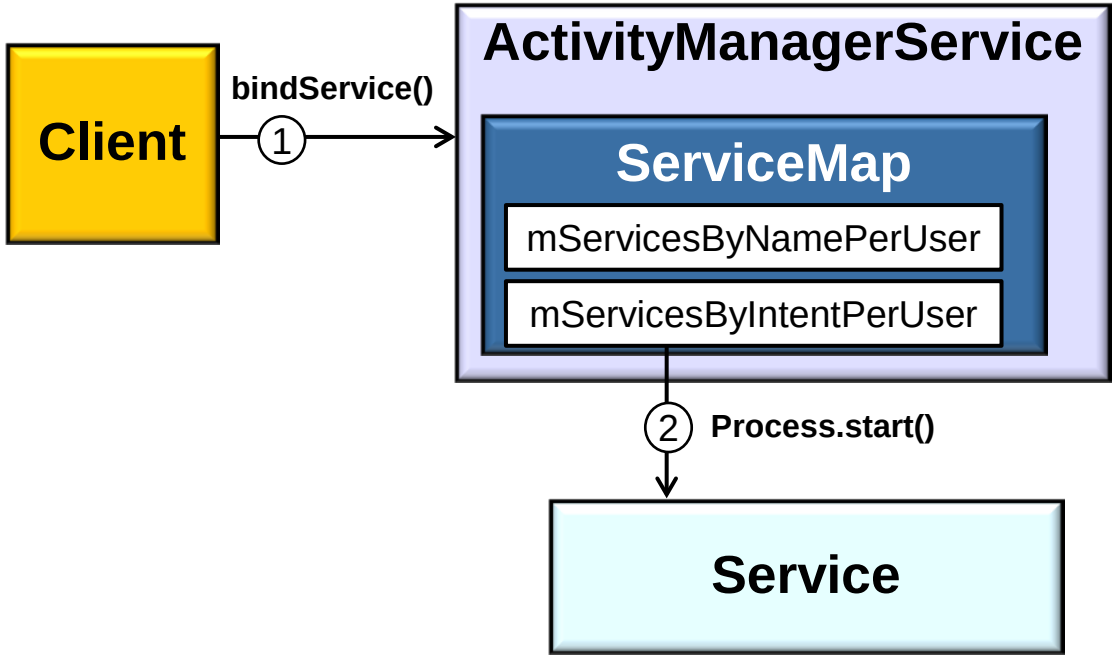
- UNIX Inetd “super server”
- CORBA Implementation Repository
- Android ActivityManagerService



Activator POSA4 Design Pattern

Known Uses

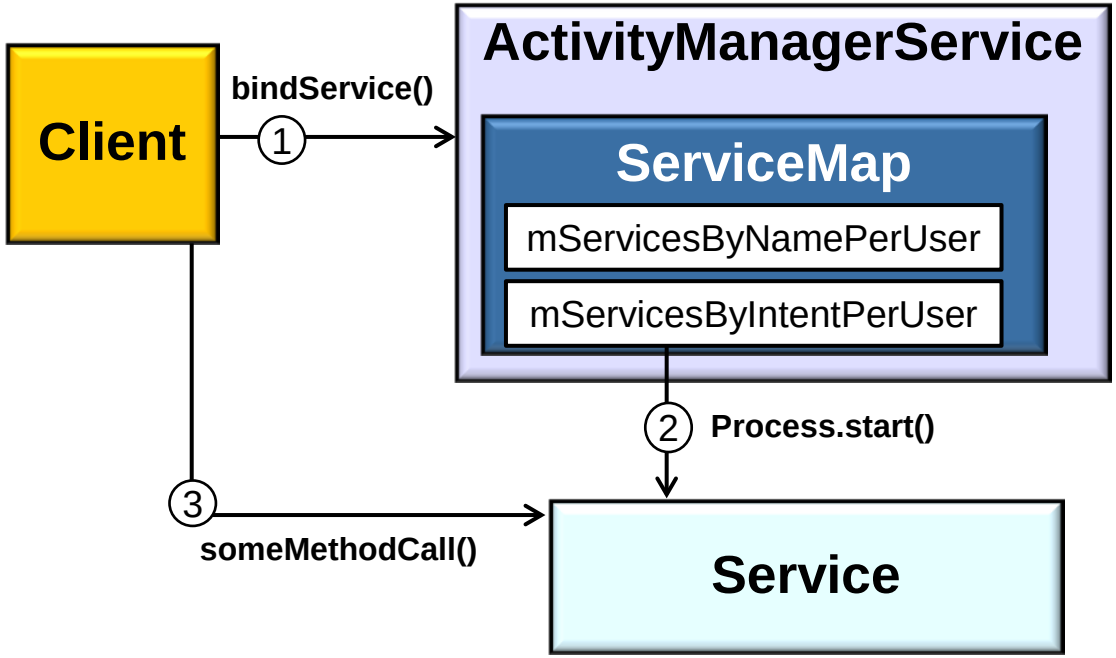
- UNIX Inetd “super server”
- CORBA Implementation Repository
- Android ActivityManagerService



Activator POSA4 Design Pattern

Known Uses

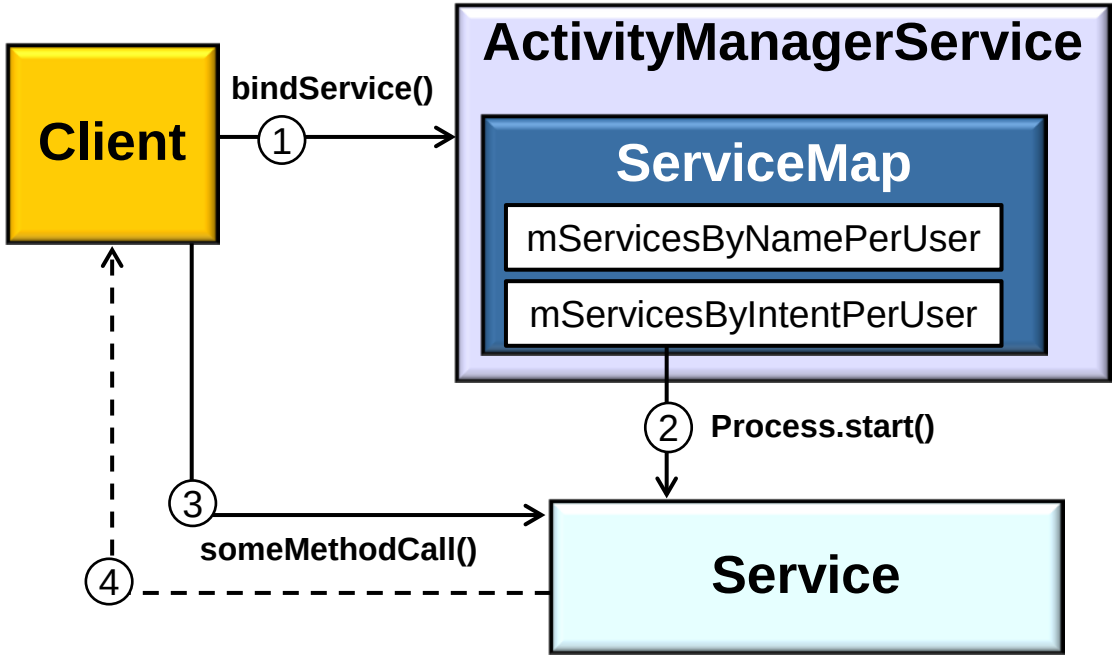
- UNIX Inetd “super server”
- CORBA Implementation Repository
- Android ActivityManagerService



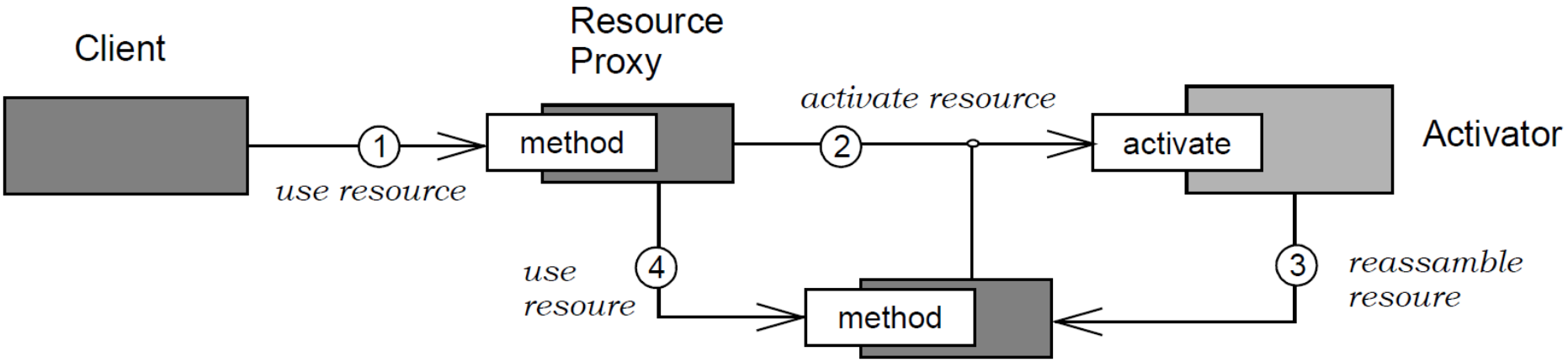
Activator POSA4 Design Pattern

Known Uses

- UNIX Inetd “super server”
- CORBA Implementation Repository
- Android ActivityManagerService

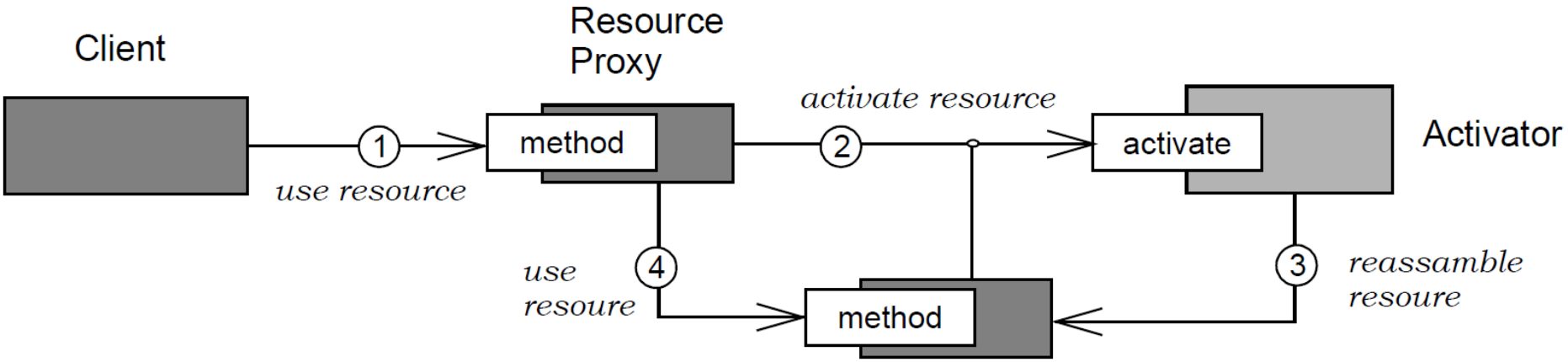


Summary



- *Activator* frees clients from the responsibility of (re)activating the resources they use
  - It appears to them as if all resources were always (virtually) available

Summary



- *Activator* frees clients from the responsibility of (re)activating the resources they use
- *Activator* also ensures that (re)activating a resource incurs minimal overhead because it maintains information about how to optimize this process
  - e.g., an activator could reload the resource's persistent state & reacquire the needed computing resources in parallel, thereby speeding resource initialization