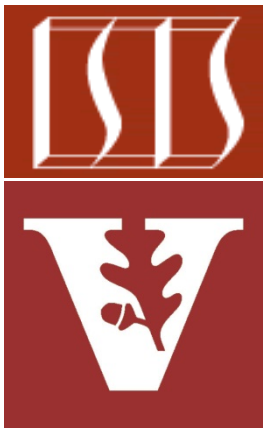
Android Services & Local IPC: The Activator Pattern (Part 2)

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



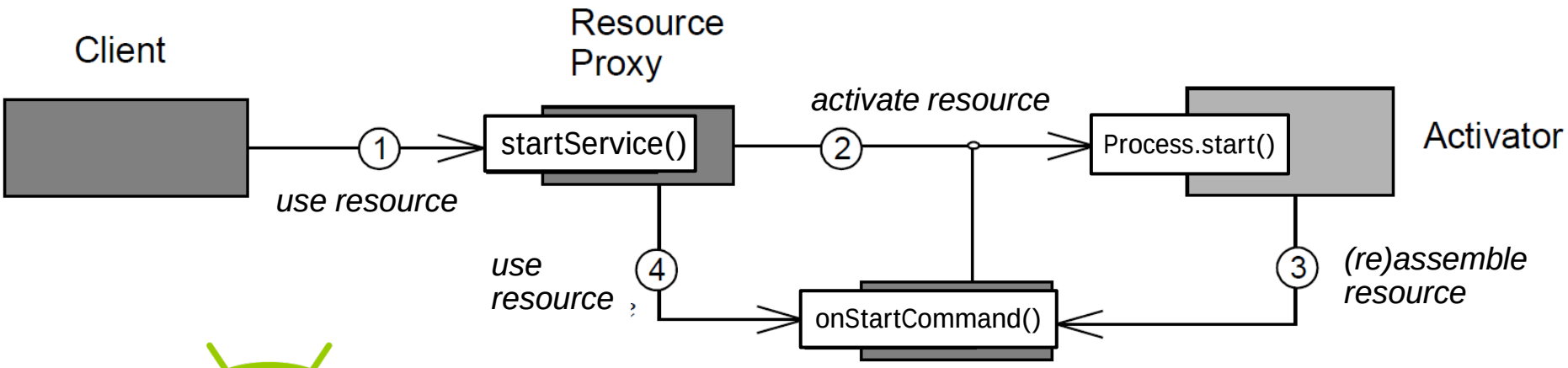
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand how the *Activator* pattern is applied in Android



Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
  - Encapsulate each distinct unit of app functionality into a self-contained service

```
public abstract class Service
       extends ContextWrapper
       implements
          ComponentCallbacks2
{
  public abstract IBinder
    onBind(Intent intent);

  ...
}
```

frameworks/base/core/java/android/app/Service.java has the source code

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
  - Encapsulate each distinct unit of app functionality into a self-contained service
  - Examples of service identifier representations include URLs, IORs, TCP/IP port numbers & host addresses, Android Intents, etc.

Intent Element	Purpose
Name	Optional name for a component
Action	A string naming the action to perform or the action that took place & is being reported
Data	URI of data to be acted on & the MIME type of that data
Category	String giving additional info about the action to execute

frameworks/base/core/java/android/content/Intent.java has the source code

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
  - Determine overhead of activating & deactivating services on-demand vs. keeping them alive for the duration of the system vs. security implications, etc.

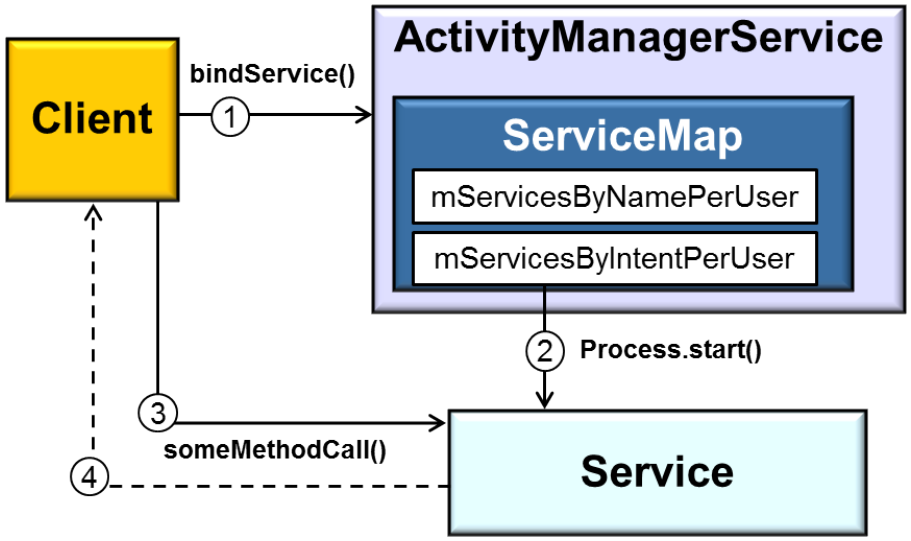
Android Service	Purpose
Media Playback Service	Provides "background" audio playback capabilities
Exchange Email Service	Send/receive email messages to an Exchange server
SMS & MMS Services	Manage messaging operations, such as sending data, text, & PDU messages
Alert Service	Handle calendar event reminders

packages/apps has source code for many Bound & Started Services

Activator POSA4 Design Pattern

Implementation

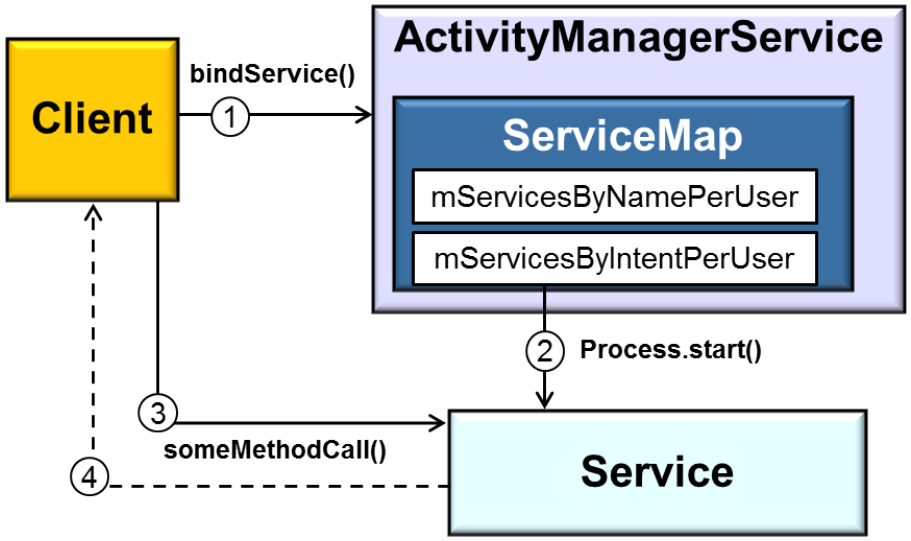
- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
    - e.g., an OS process/thread or middleware container



Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
  - Define service registration strategy
    - e.g., static text file or dynamic object registration



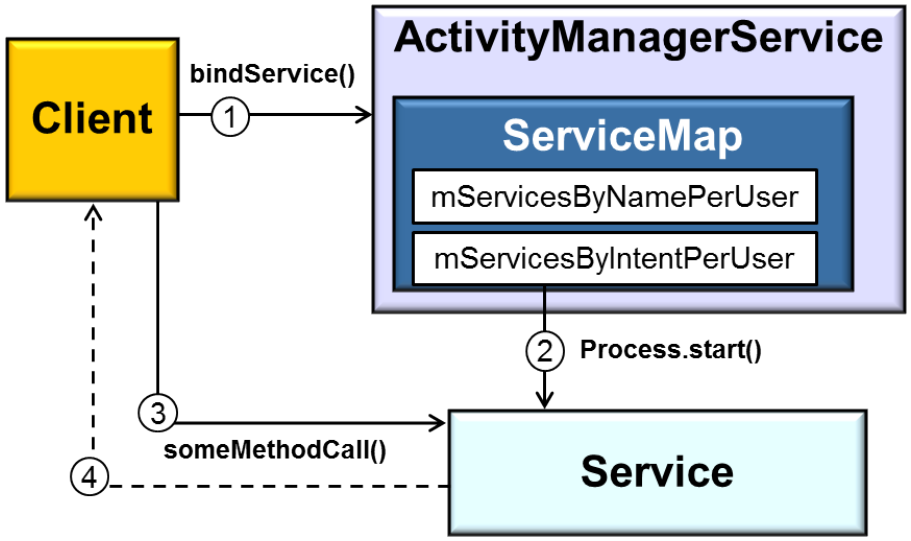

```
<service android:name=
"com.android.music.MediaPlaybackService"
   android:exported="false"/>
```

frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
  - Define service registration strategy
  - Define service initialization strategy
    - e.g., stateful vs. stateless services



Android Services are responsible for managing their own persistent state

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
  - Define service registration strategy
  - Define service initialization strategy
  - Define service deactivation strategy
    - e.g., service-triggered, client-triggered, or activator-triggered deactivation

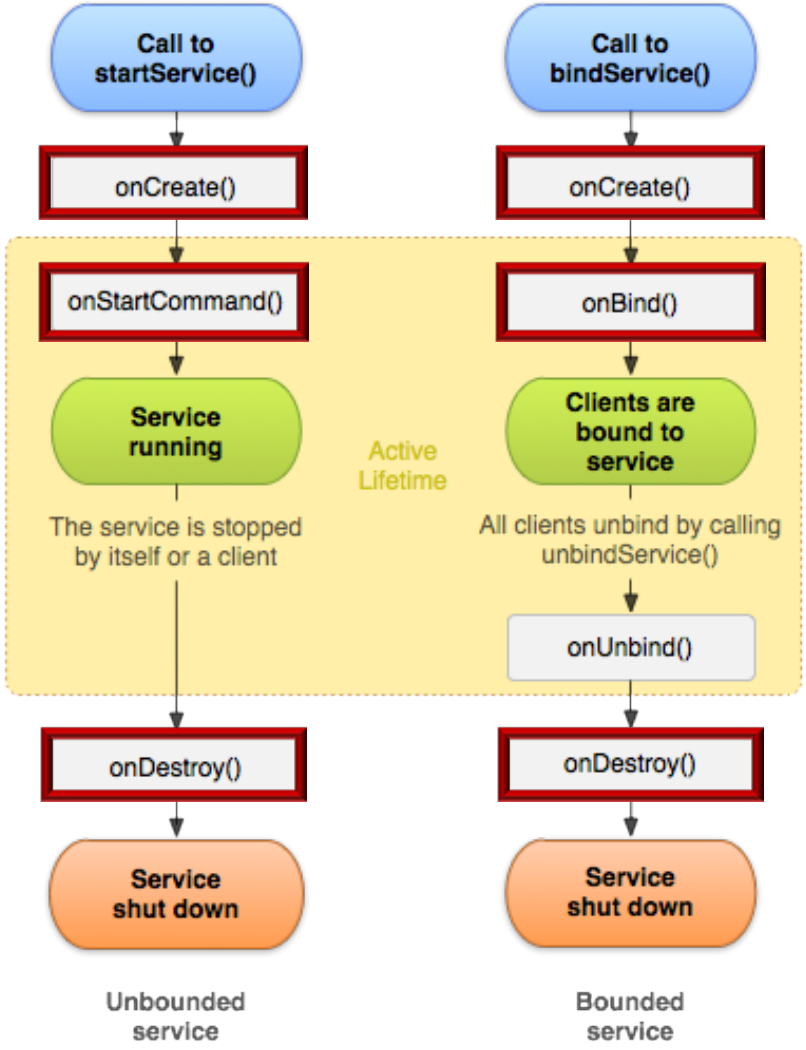
Started Service	Bound Service
• Service runs indefinitely & must stop itself by calling stopSelf() • A component can also stop the service by calling stopService() • When Service is stopped, Android destroys it	• Multiple clients can bind to same Service • When all of them unbind, the system destroys the Service • The Service does not need to stop itself an

developer.android.com/guide/components/services.html#Lifecycle has more

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
- Define interoperation between services & service execution context
  - Typically implemented via some type of lifecycle callback hook methods

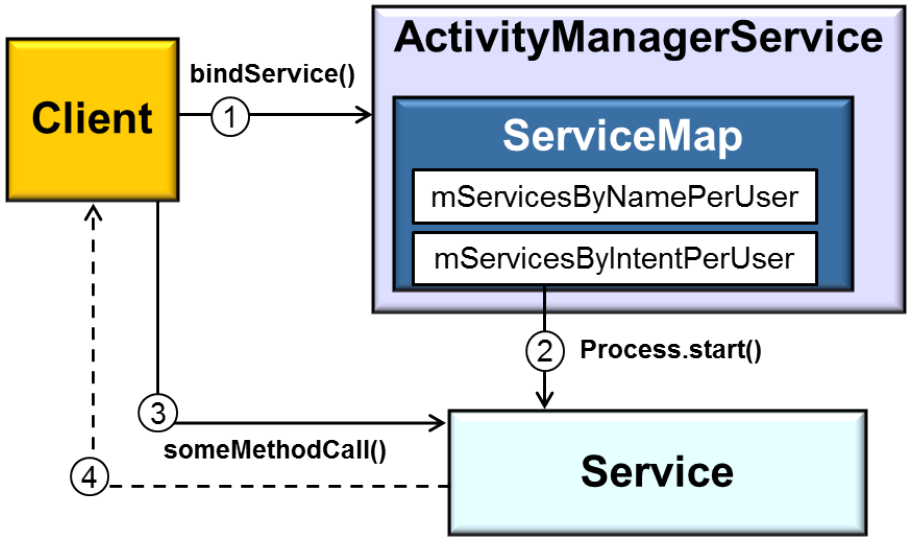


developer.android.com/guide/components/services.html#LifecycleCallbacks

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
- Define interoperation between services & service execution context
- Implement the activator
  - Determine the association between activators & services
    - e.g., singleton (shared) vs. exclusive vs. distributed activator

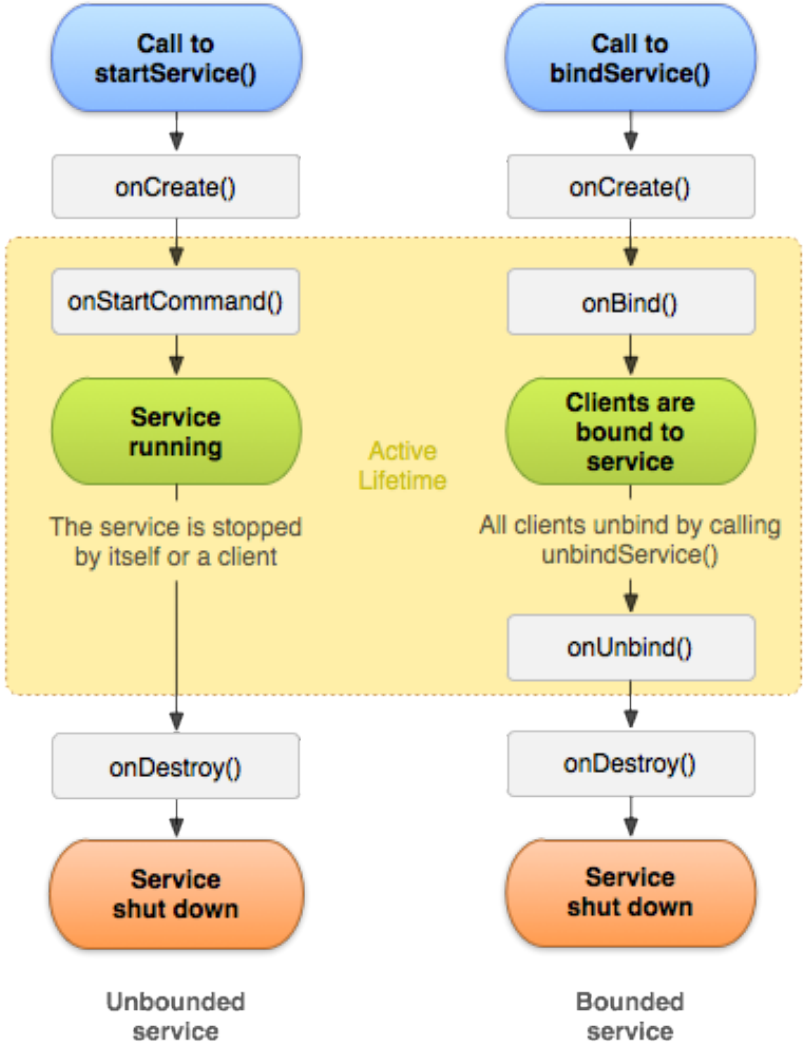


The Android Activity Manager Service is a singleton activator

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
- Define interoperation between services & service execution context
- Implement the activator
  - Determine the association between activators & services
  - Determine the degree of transparency
    - e.g., explicit vs. transparent activator

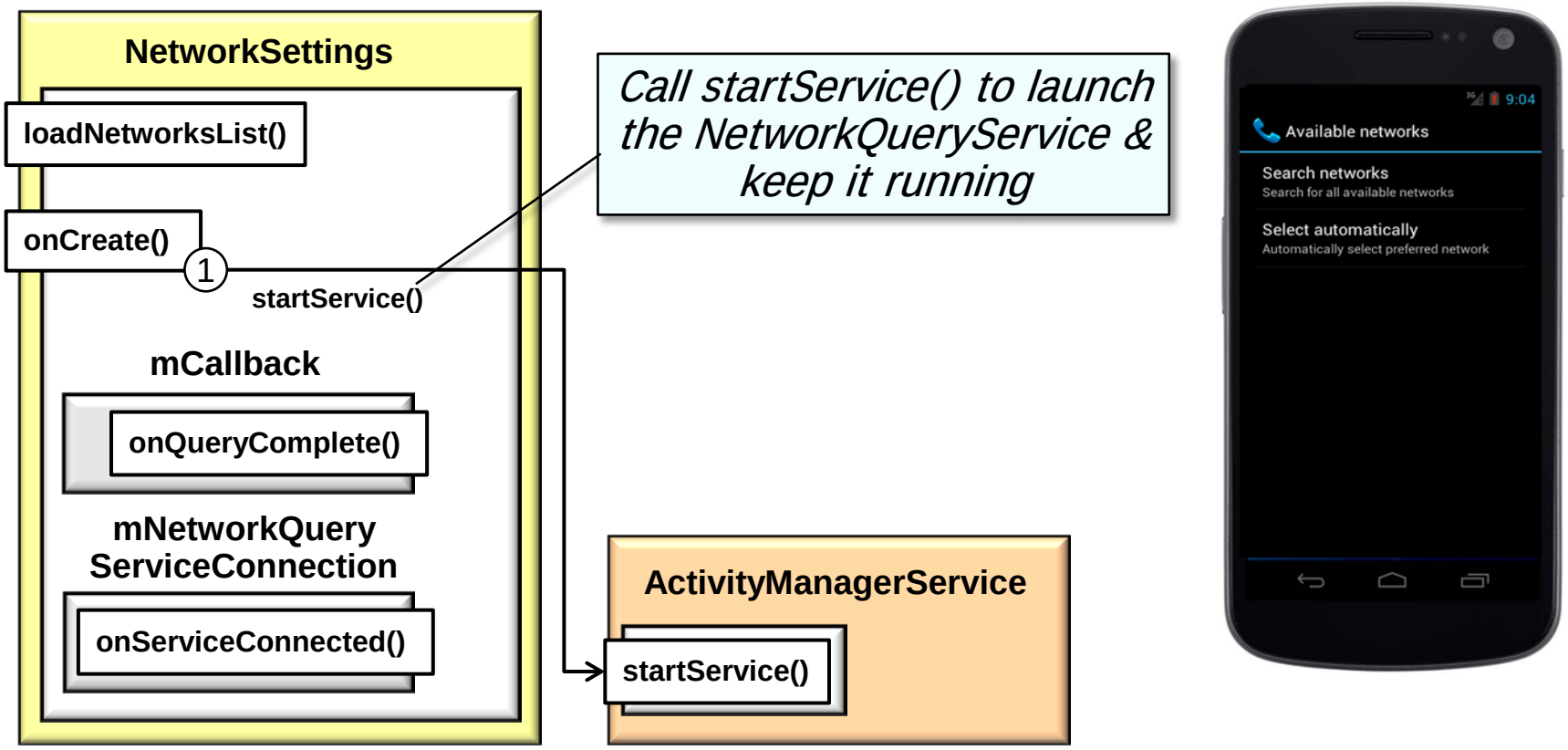


Android Started & Bound Services use an explicit activator model

Activator POSA4 Design Pattern

Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

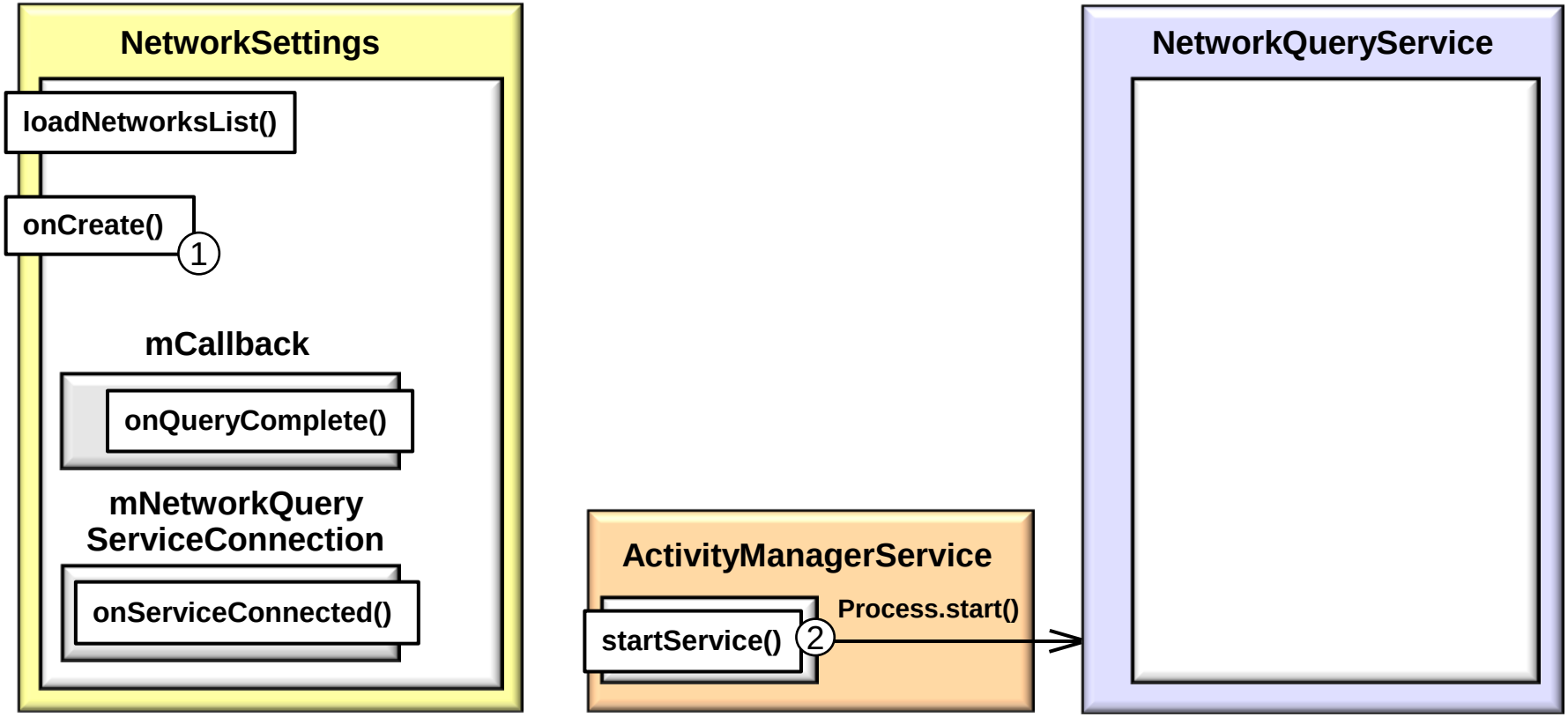


packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

Activator POSA4 Design Pattern

Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

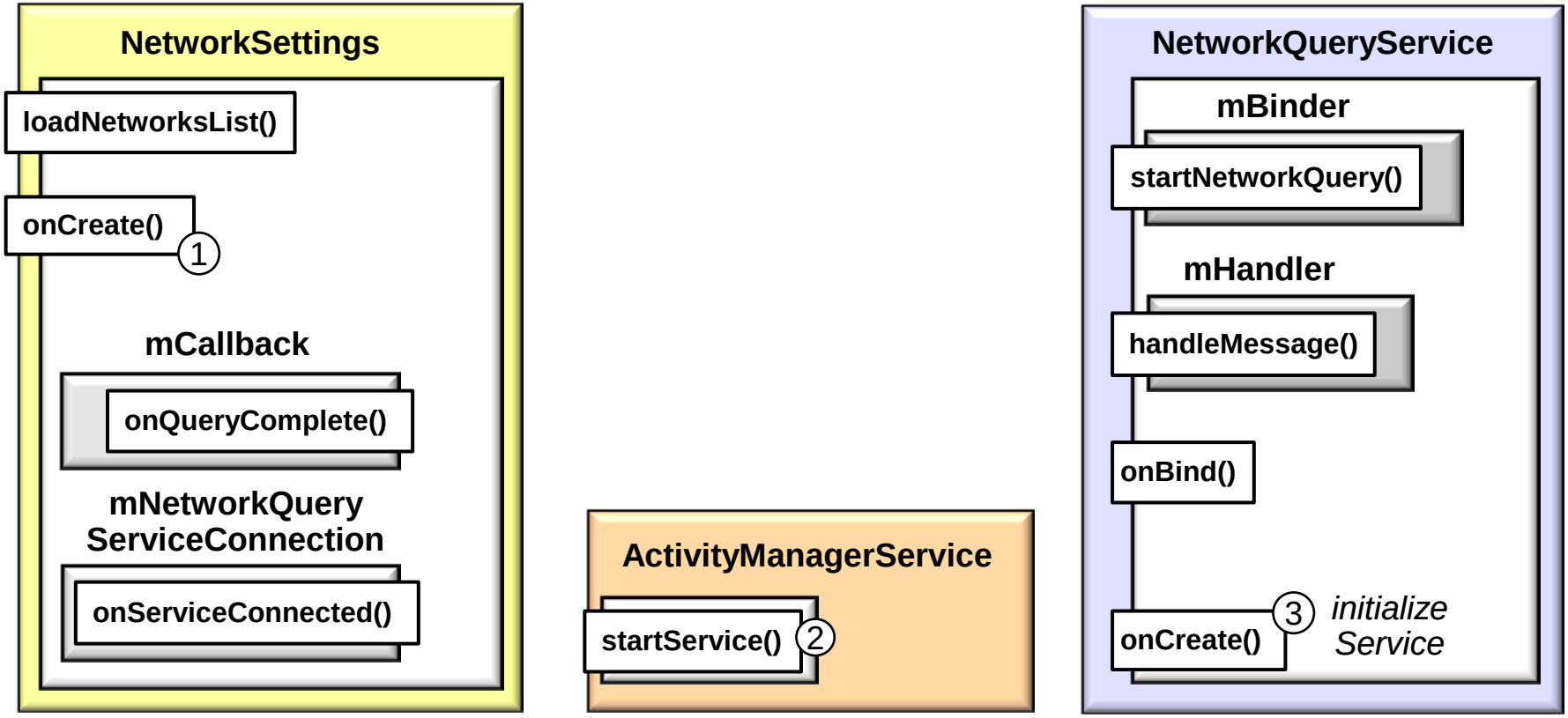


frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

Activator POSA4 Design Pattern

Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

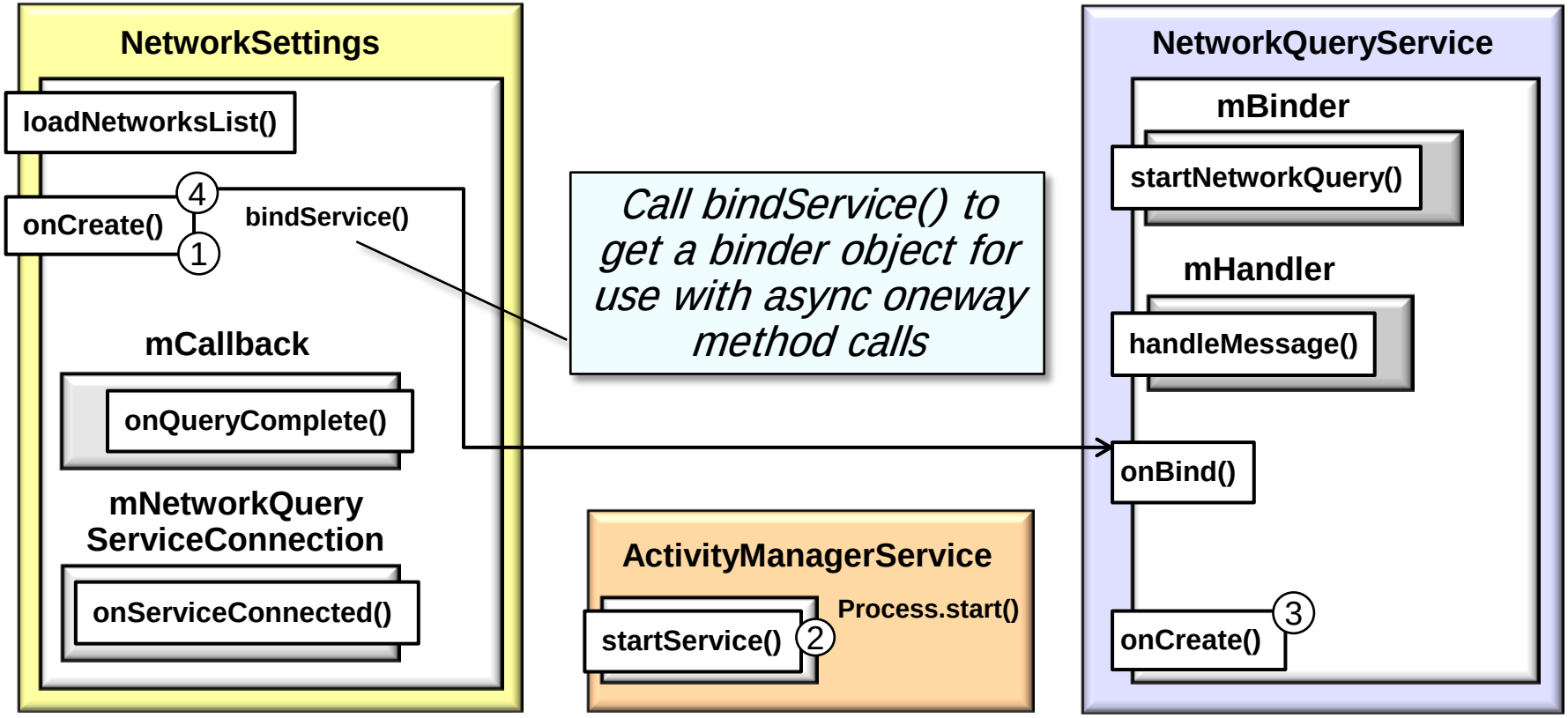


packages/apps/Phone/src/com/android/phone/NetworkQueryService.java has source

Activator POSA4 Design Pattern

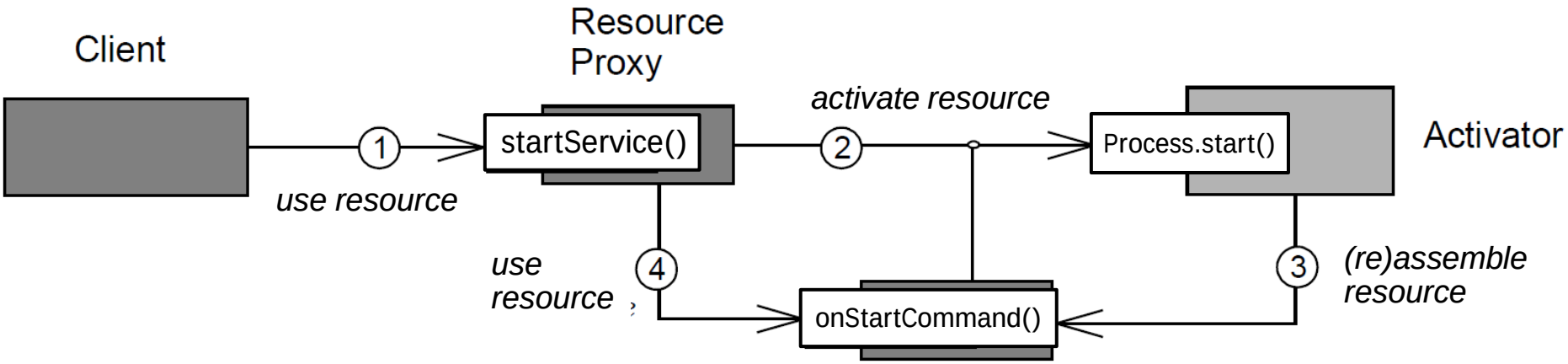
Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability



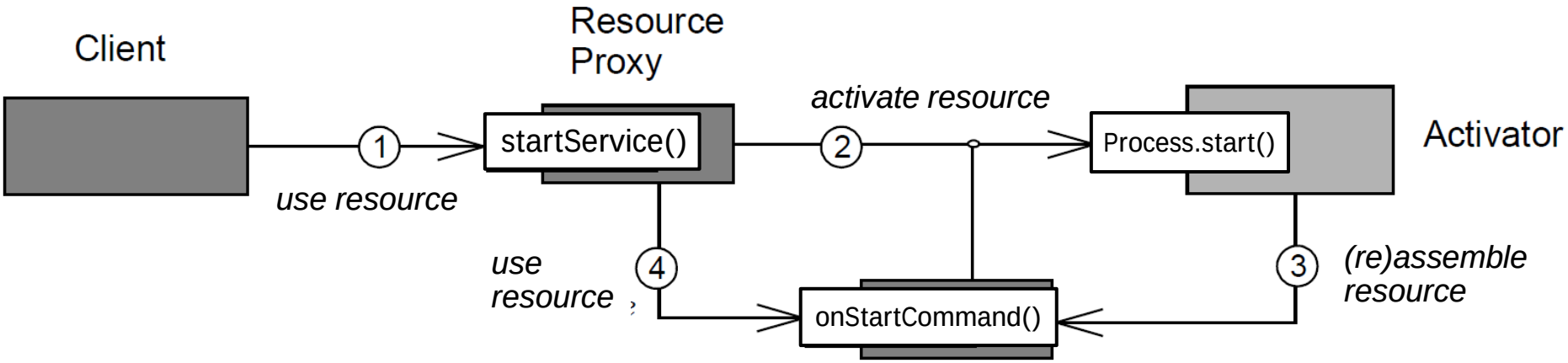
packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

Summary



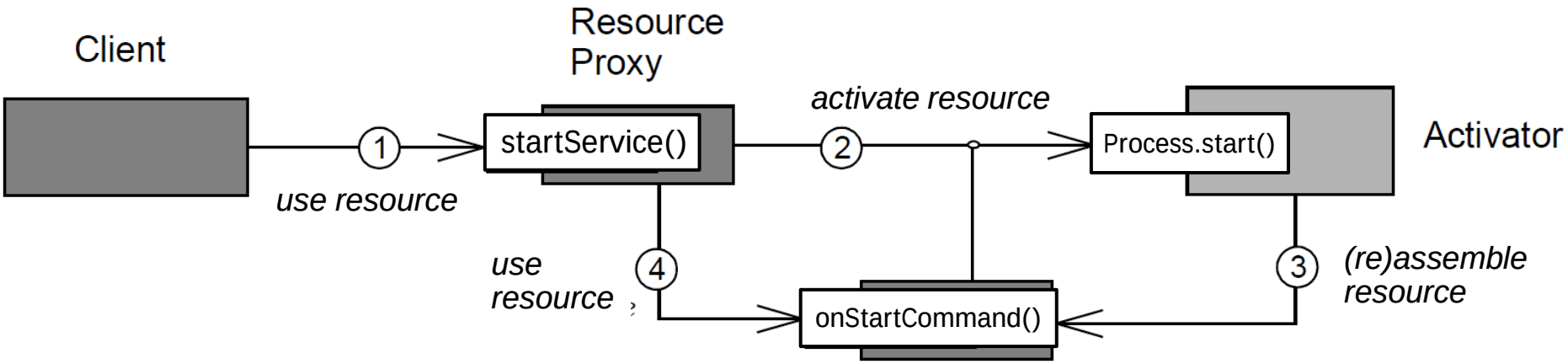
- The Android Started & Bound Services implement the *Activator* pattern

Summary



- The Android Started & Bound Services implement the *Activator* pattern
- These Services can process requests in background processes or threads
  - Processes can be configured depending on directives in the AndroidManifest.xml file

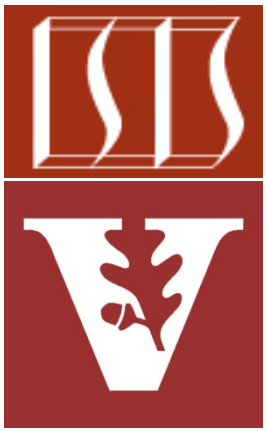
Summary



- The Android Started & Bound Services implement the *Activator* pattern
- These Services can process requests in background processes or threads
  - Processes can be configured depending on directives in the AndroidManifest.xml file
  - Threads can be programmed using the Android Intent Service framework, as discussed next

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

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



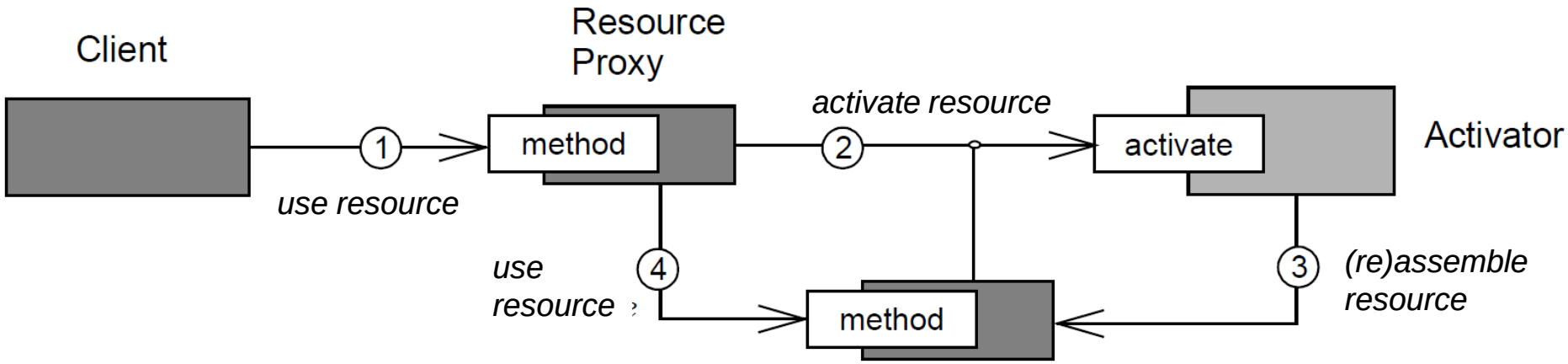
Professor of Computer Science

Institute for Software Integrated Systems

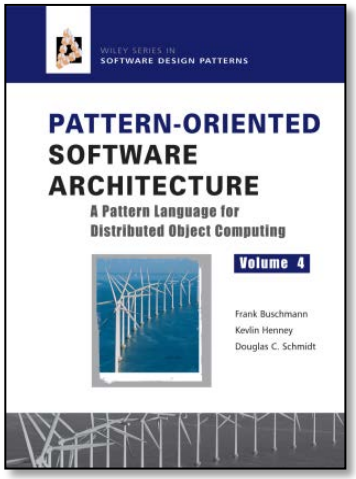
Vanderbilt University
Nashville, Tennessee, USA

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

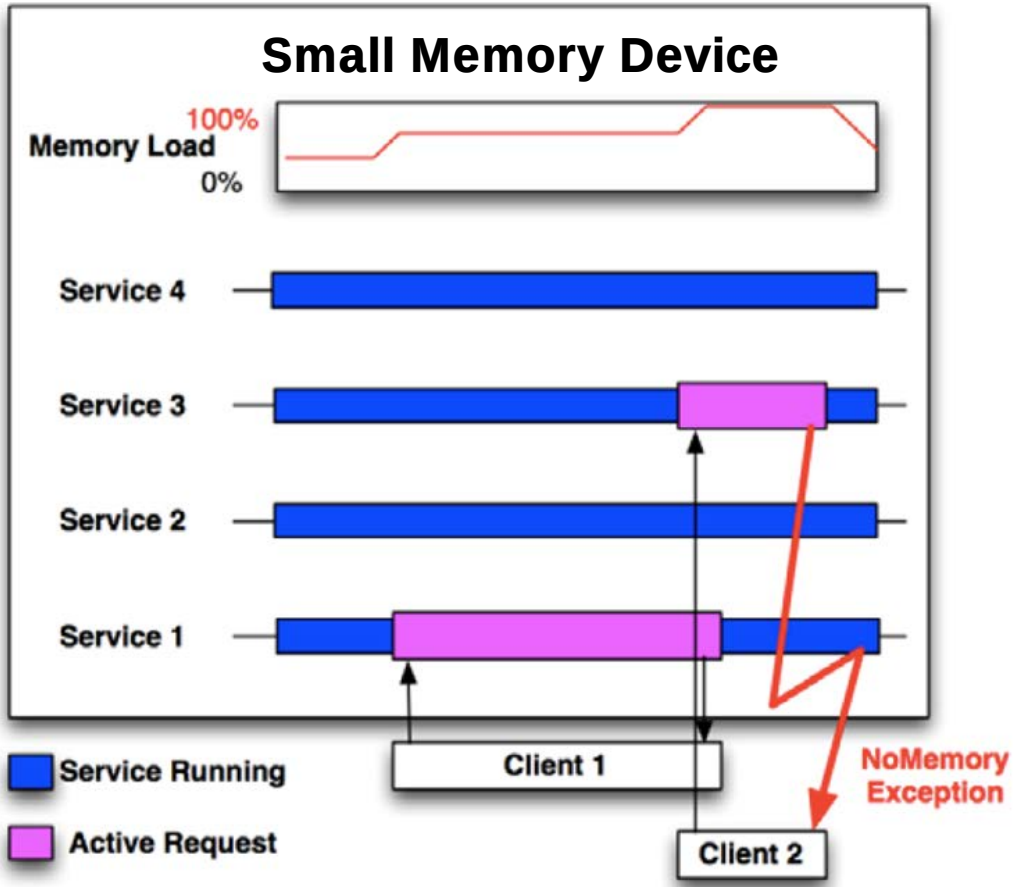


www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf

Challenge: Minimizing Resource Utilization

Context

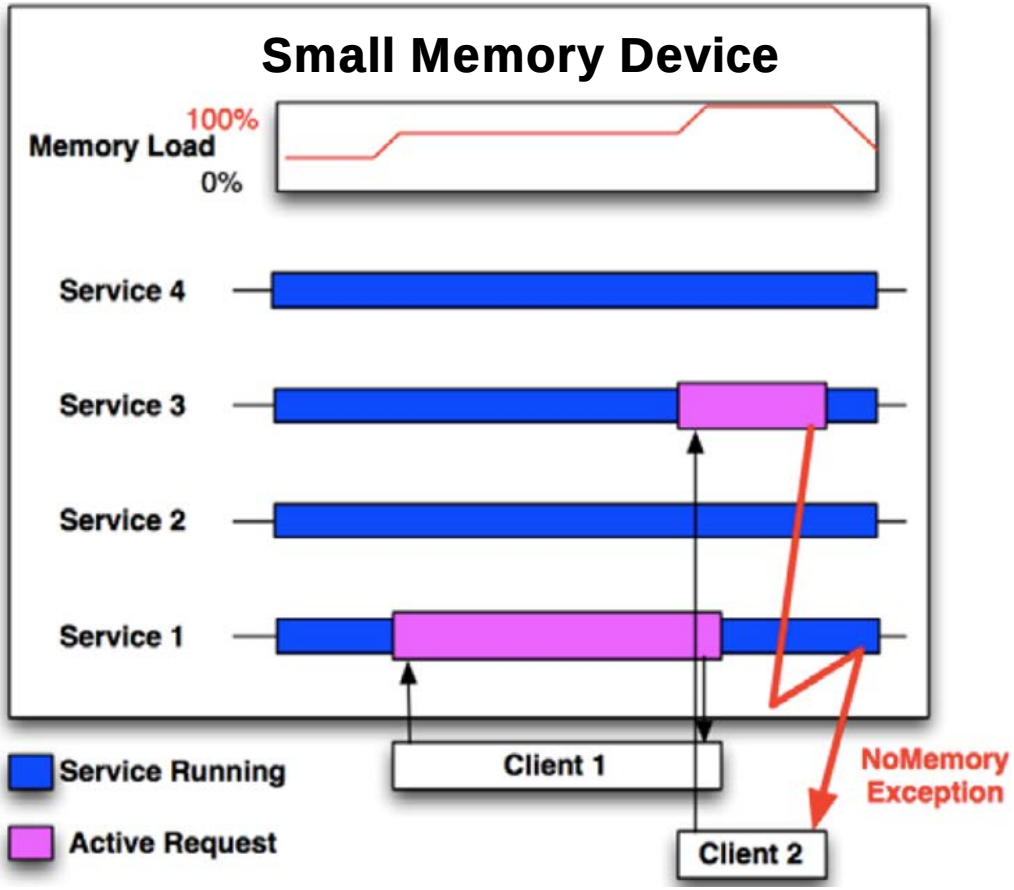
- Resource constrained & highly dynamic environments
  - Random-access memory (RAM) is a valuable resource in any software environment



Challenge: Minimizing Resource Utilization

Context

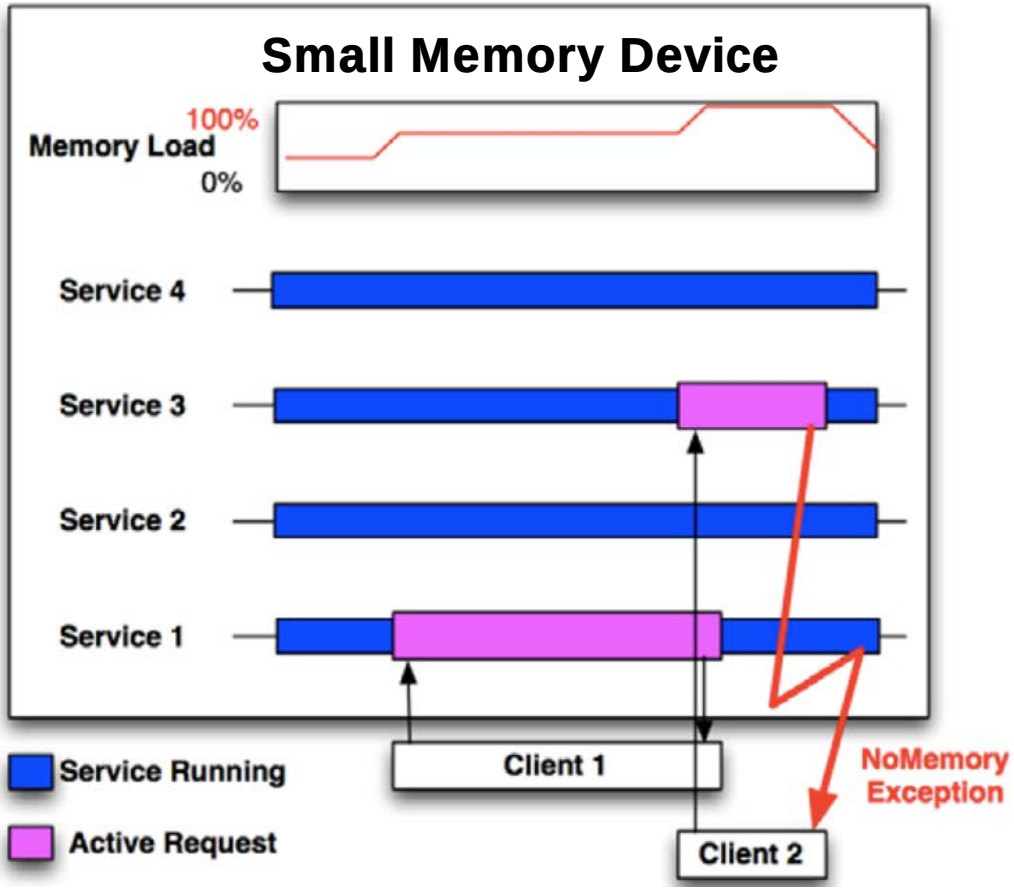
- Resource constrained & highly dynamic environments
  - Random-access memory (RAM) is a valuable resource in any software environment
  - It's even more valuable on a mobile OS like Android where physical memory is often constrained



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

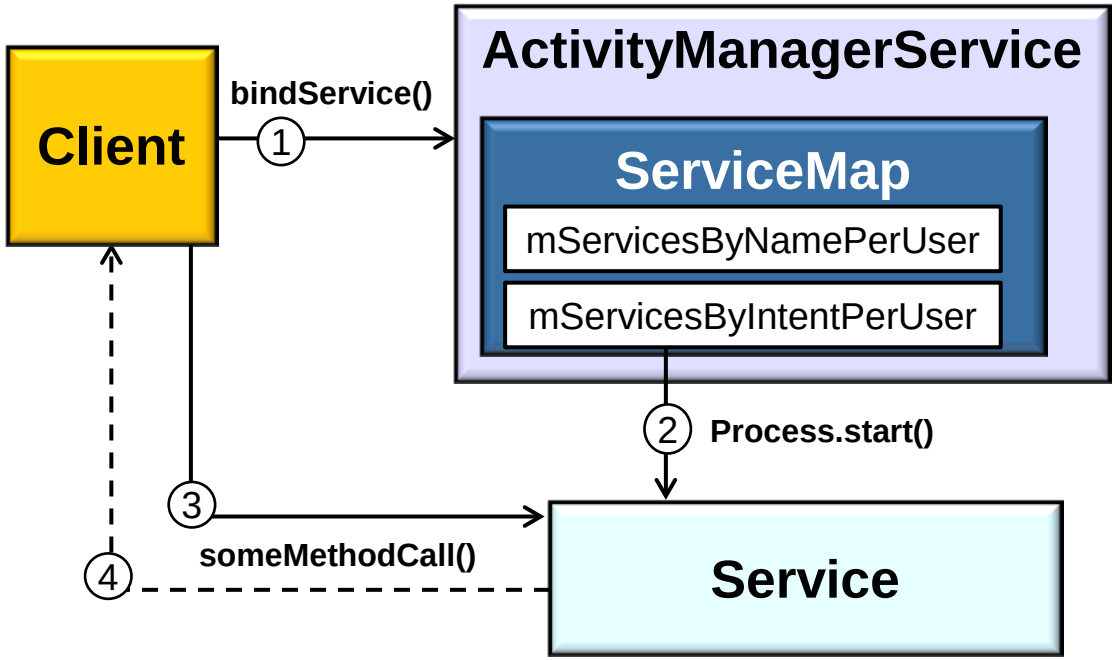


developer.android.com/training/articles/memory.html has more info

Challenge: Processing a Long-Running Action

Solution

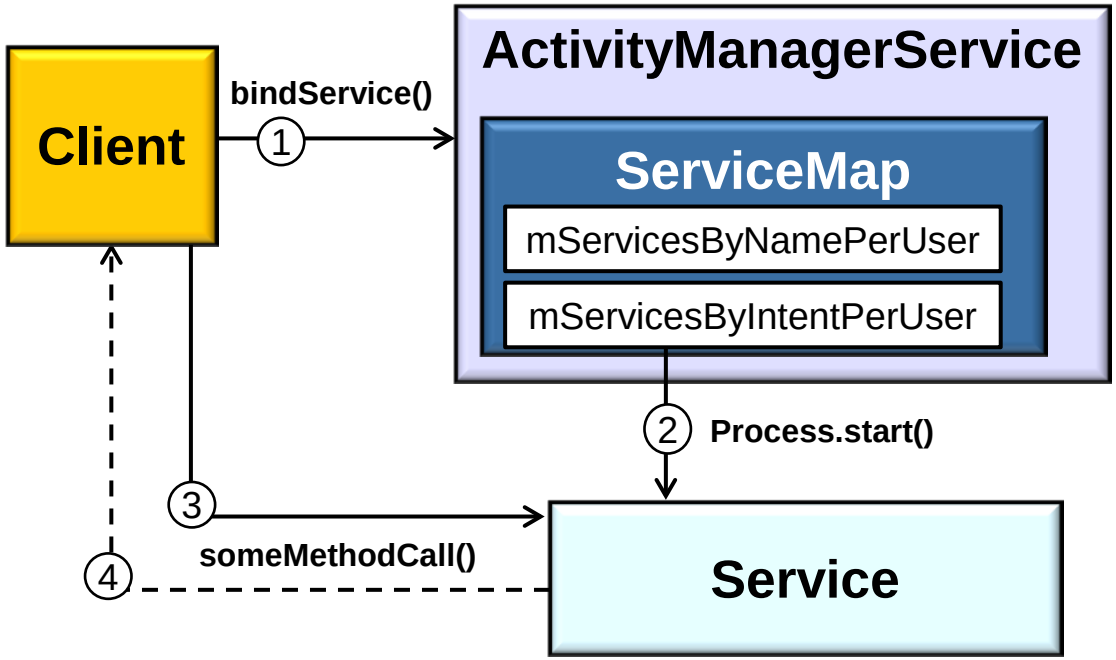
- Apply the *Activator* pattern to activate & deactivate Android Services automatically
  - If your app needs a Service to perform work in the background, don't keep it running unless it's actively performing a job



Challenge: Processing a Long-Running Action

Solution

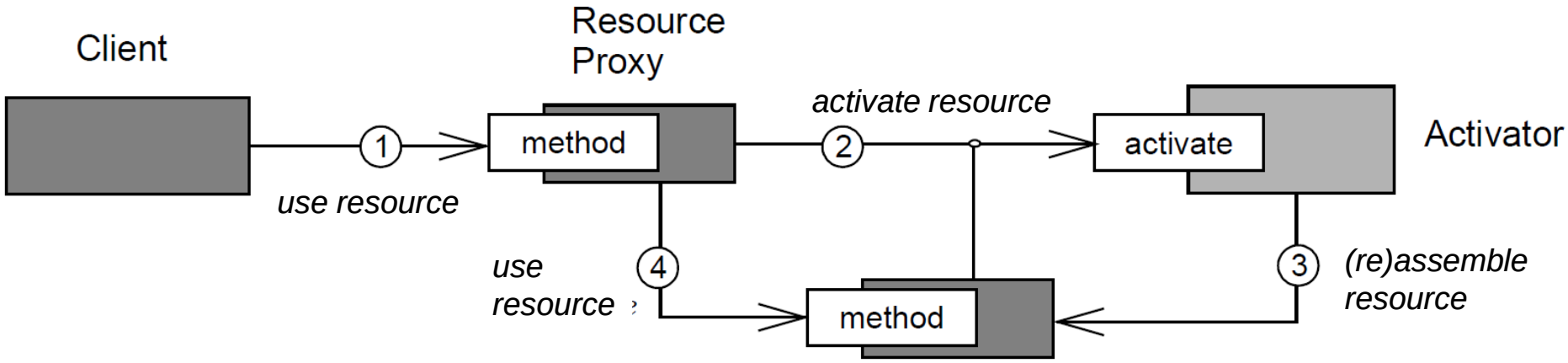
- Apply the *Activator* pattern to activate & deactivate Android Services automatically
  - If your app needs a Service to perform work in the background, don't keep it running unless it's actively performing a job
  - Be careful to never leak your Service by failing to stop it when its work is done



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

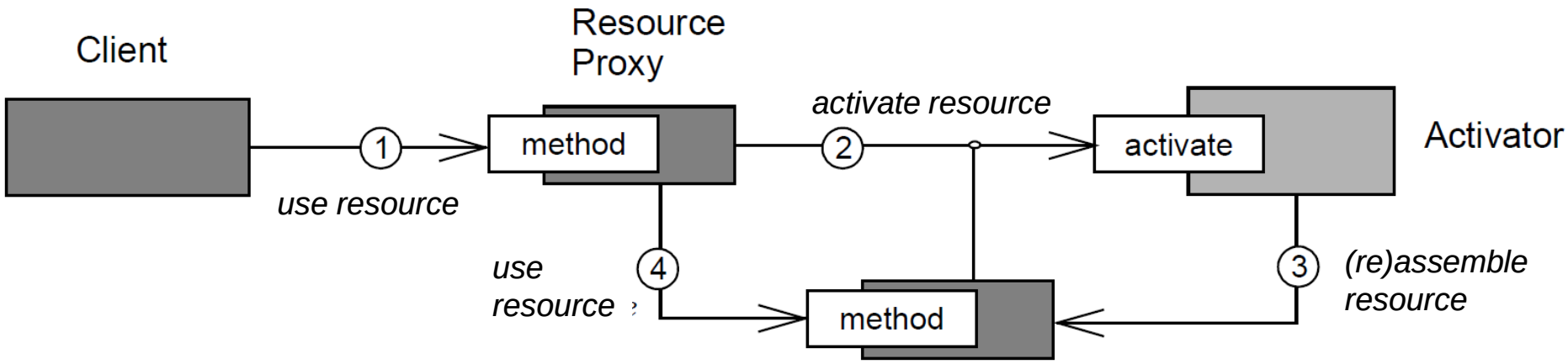


www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf has more info

Activator POSA4 Design Pattern

Applicability

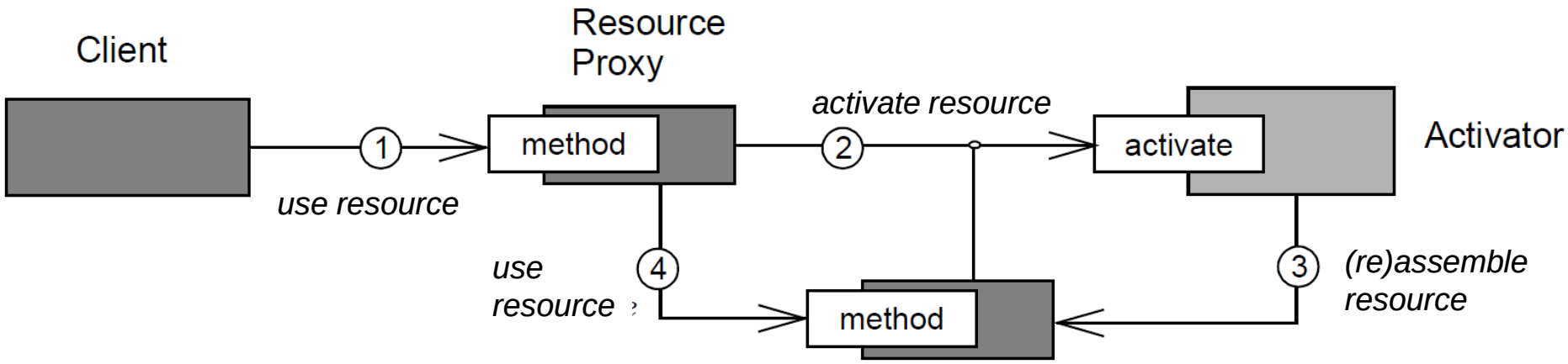
- When services in a system should only consume resources when they are accessed actively by clients



Activator POSA4 Design Pattern

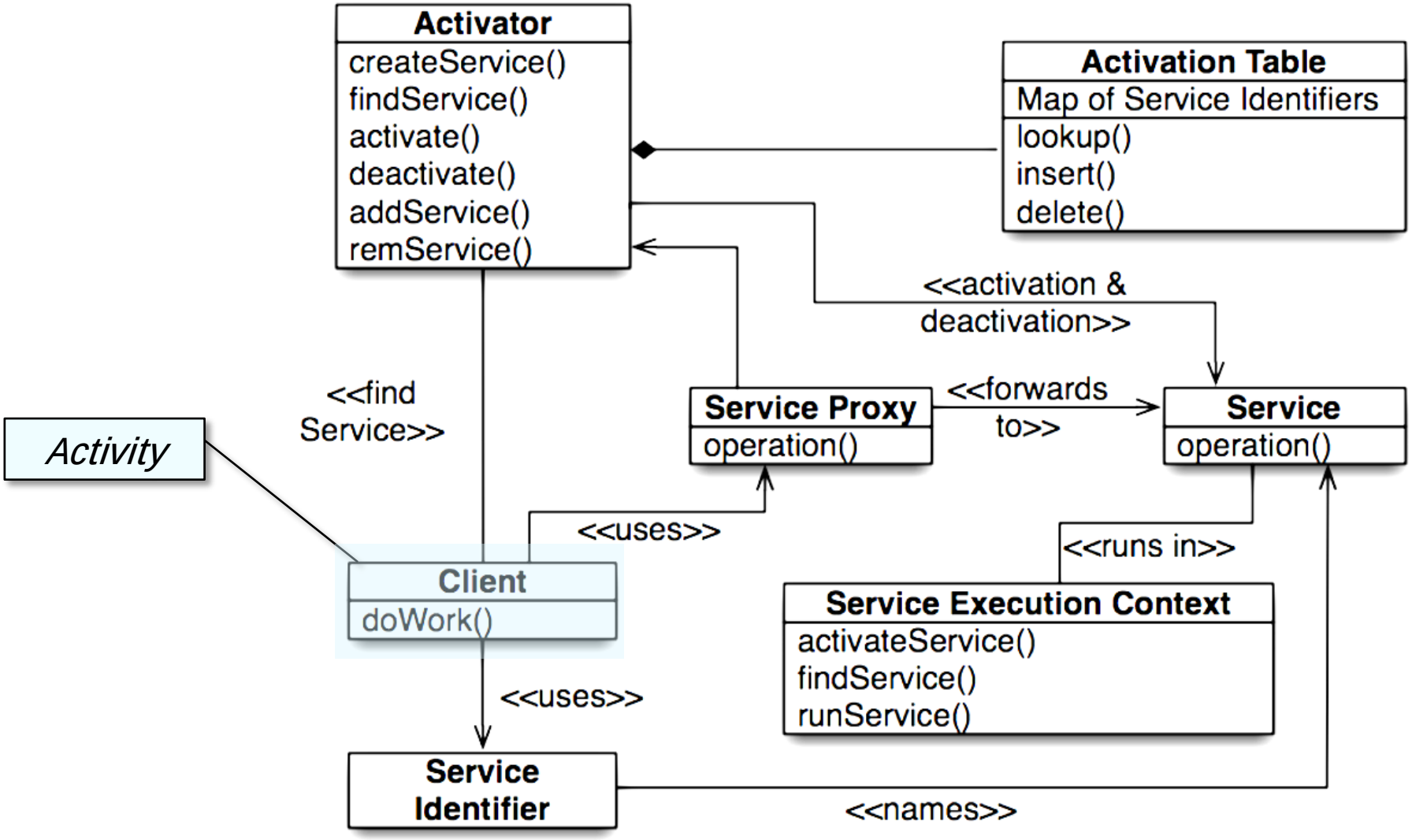
Applicability

- 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



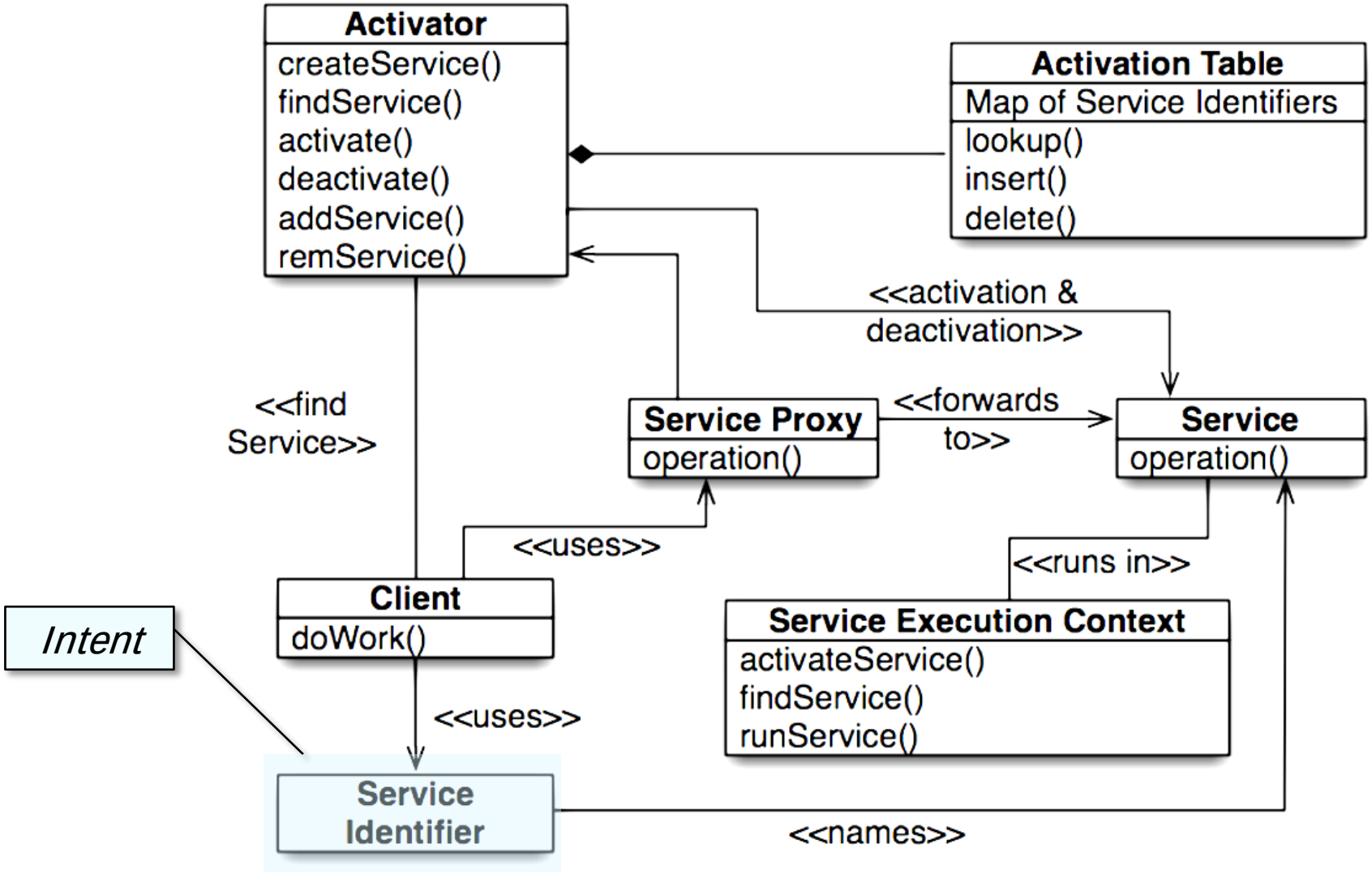
Activator POSA4 Design Pattern

Structure & Participants



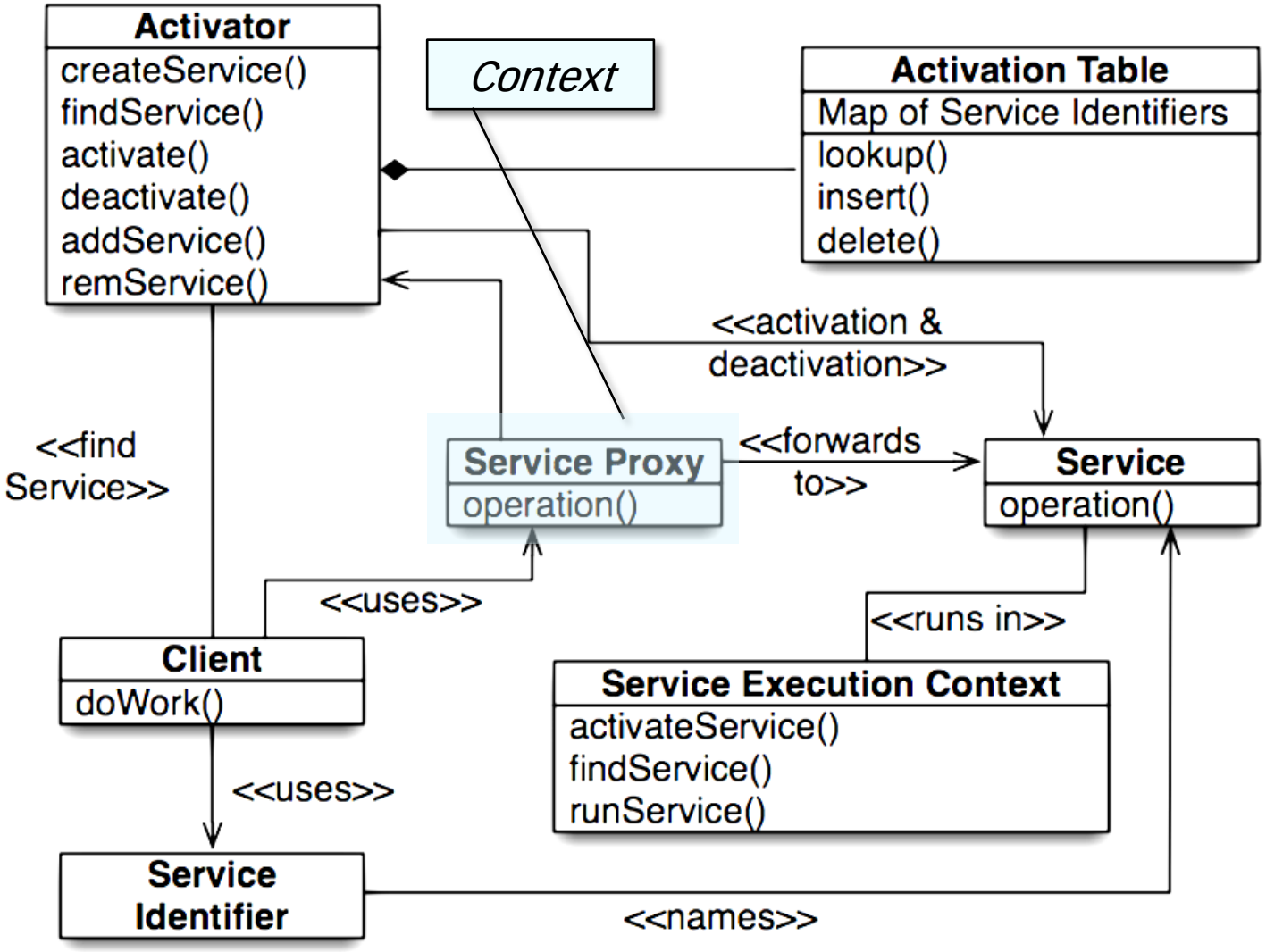
Activator POSA4 Design Pattern

Structure & Participants



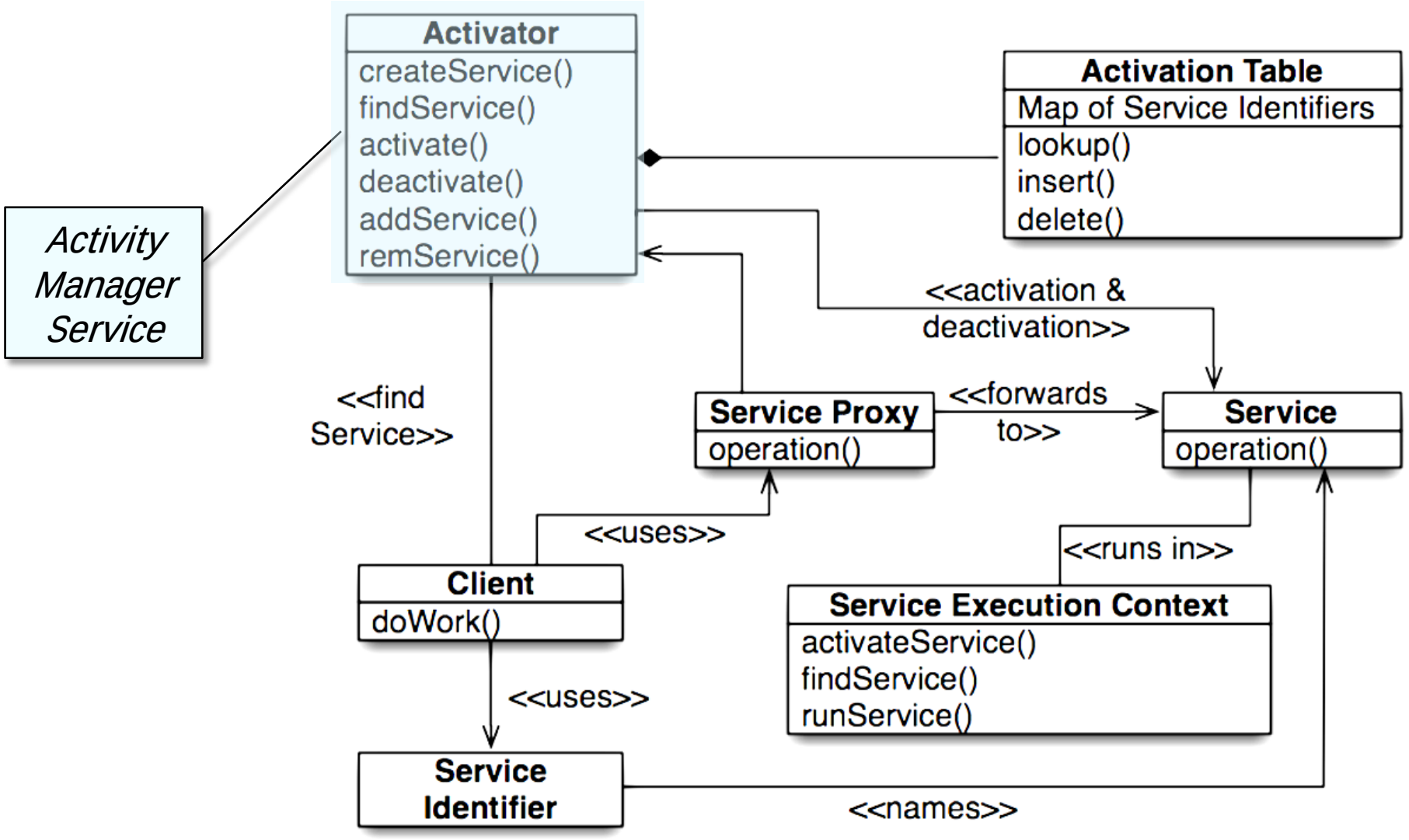
Activator POSA4 Design Pattern

Structure & Participants



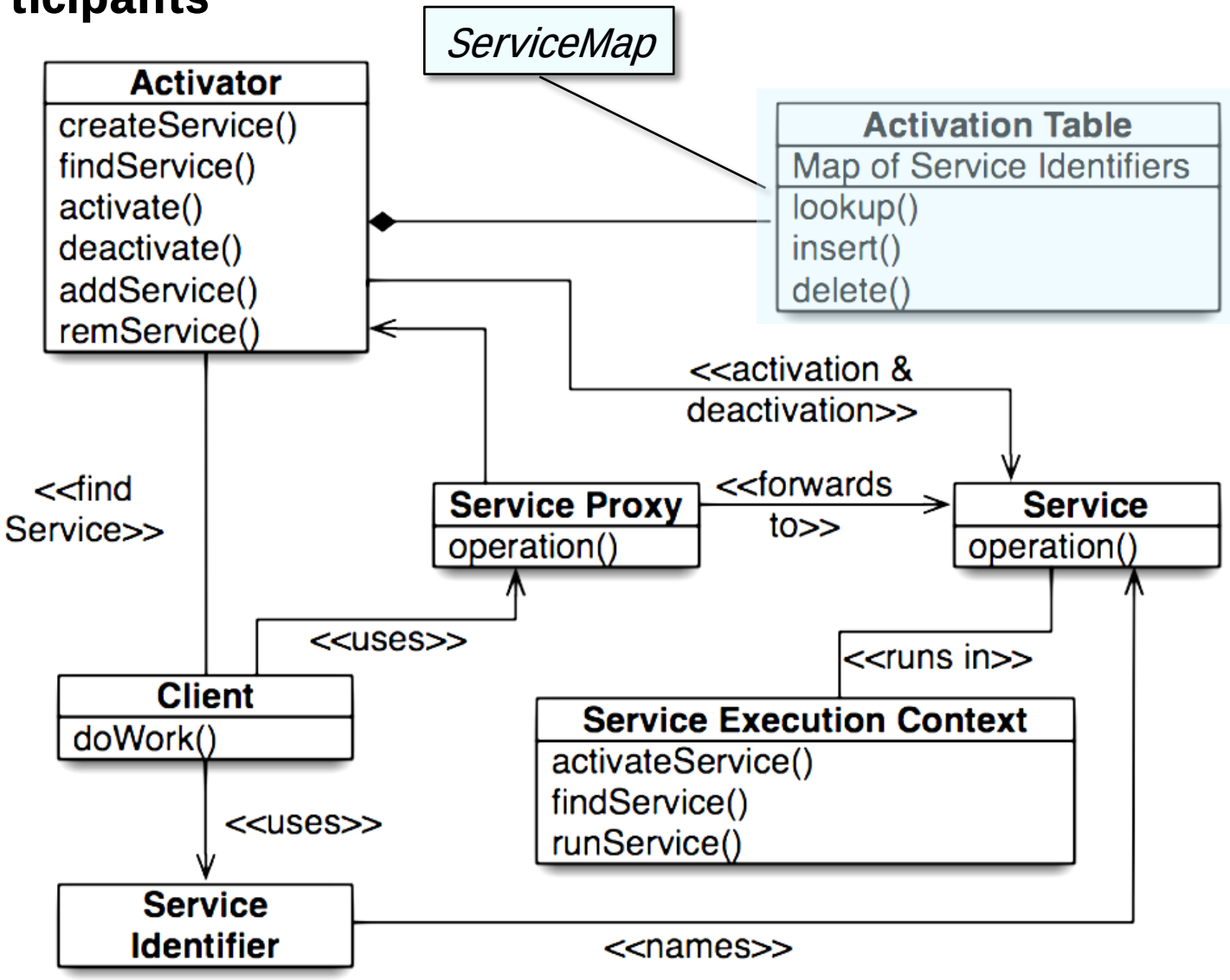
Activator POSA4 Design Pattern

Structure & Participants



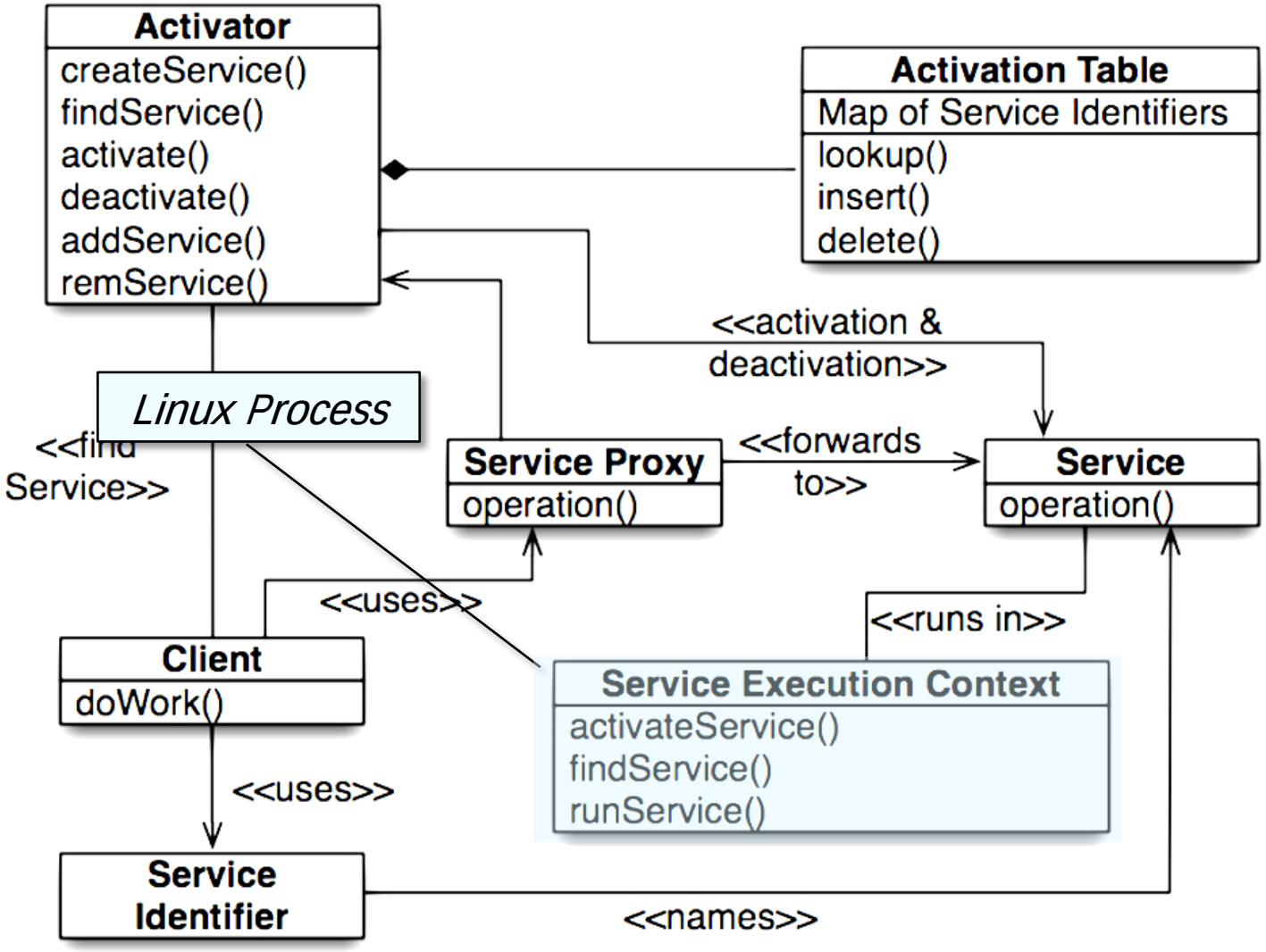
Activator POSA4 Design Pattern

Structure & Participants



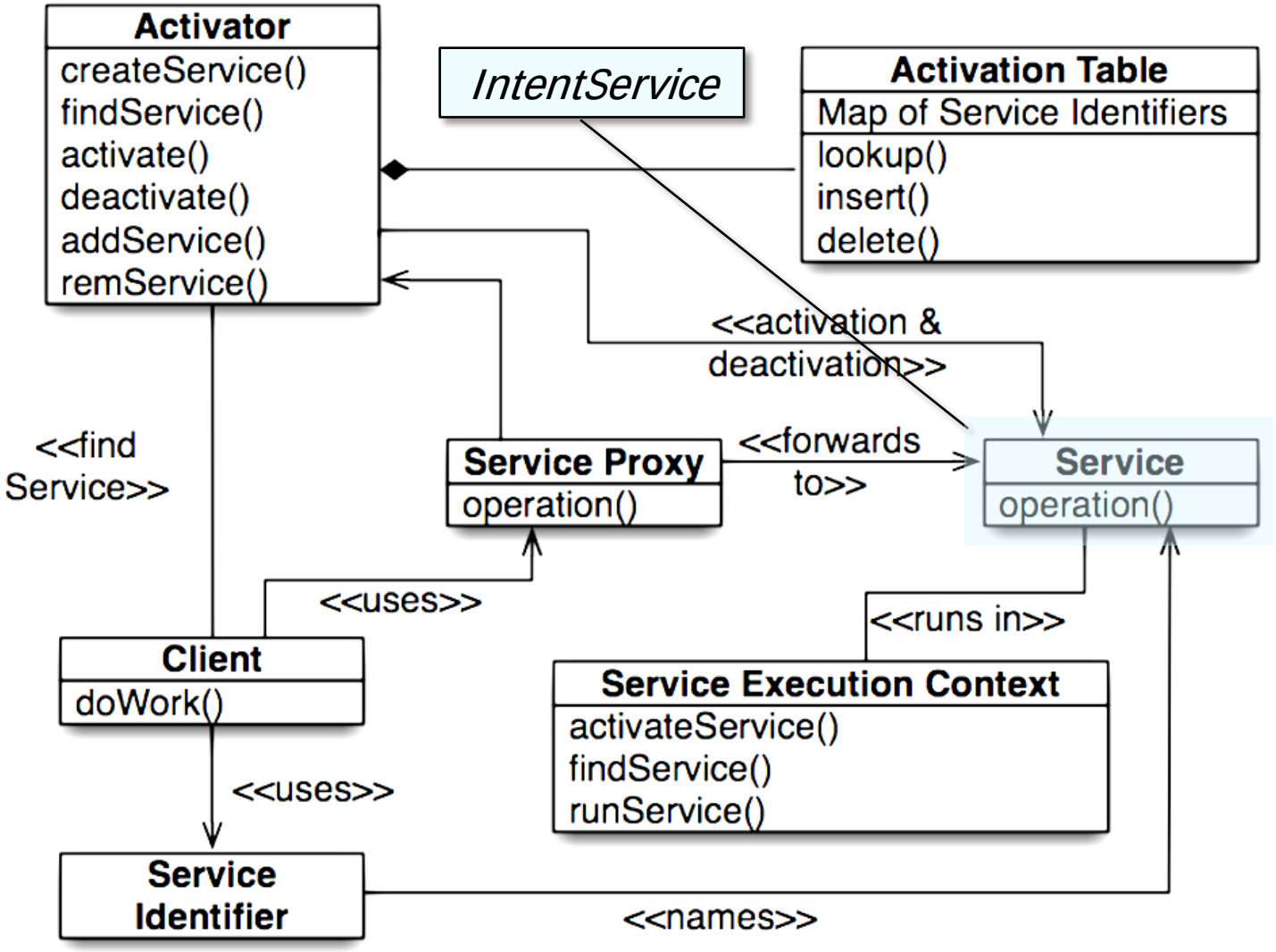
Activator POSA4 Design Pattern

Structure & Participants



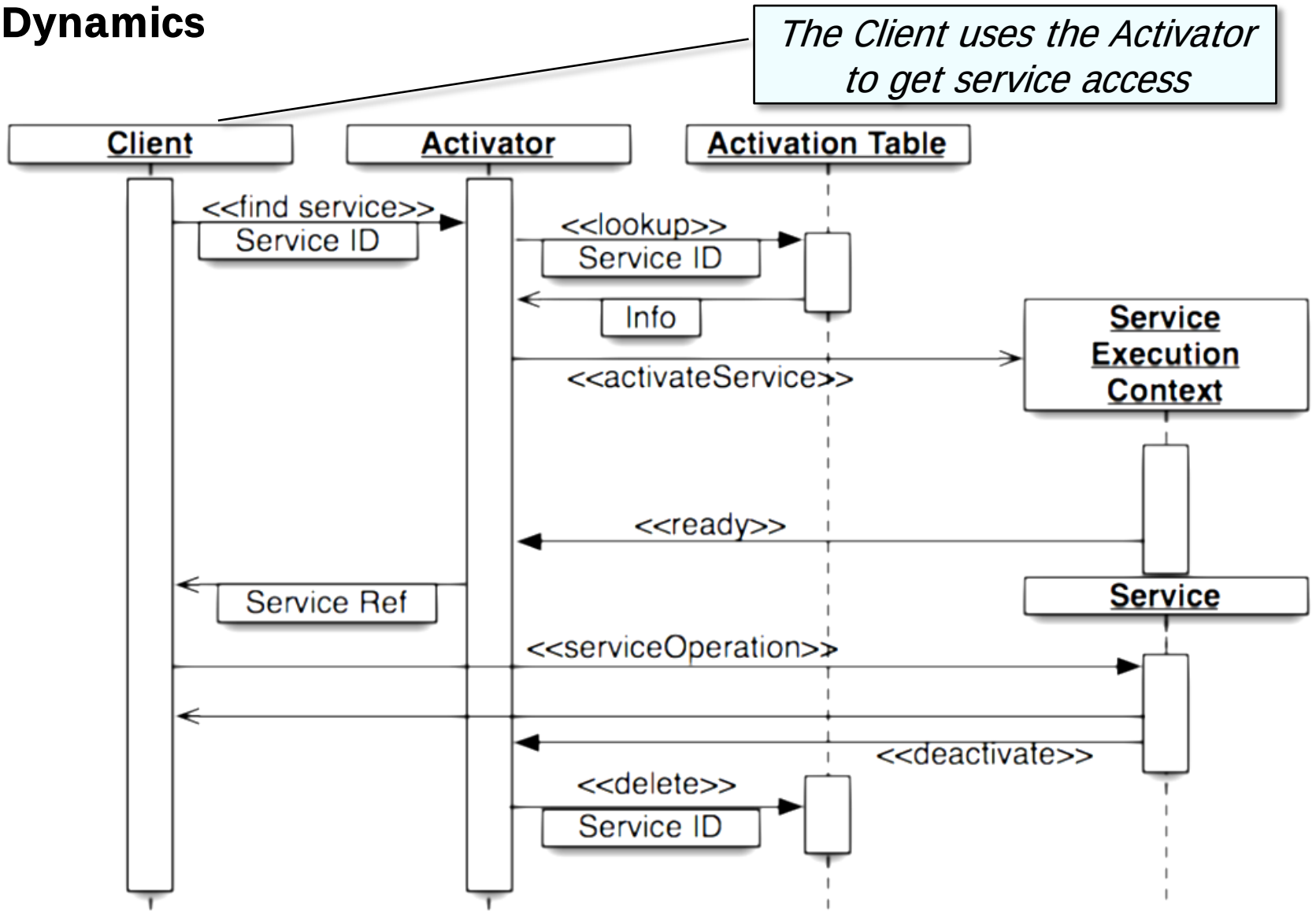
Activator POSA4 Design Pattern

Structure & Participants



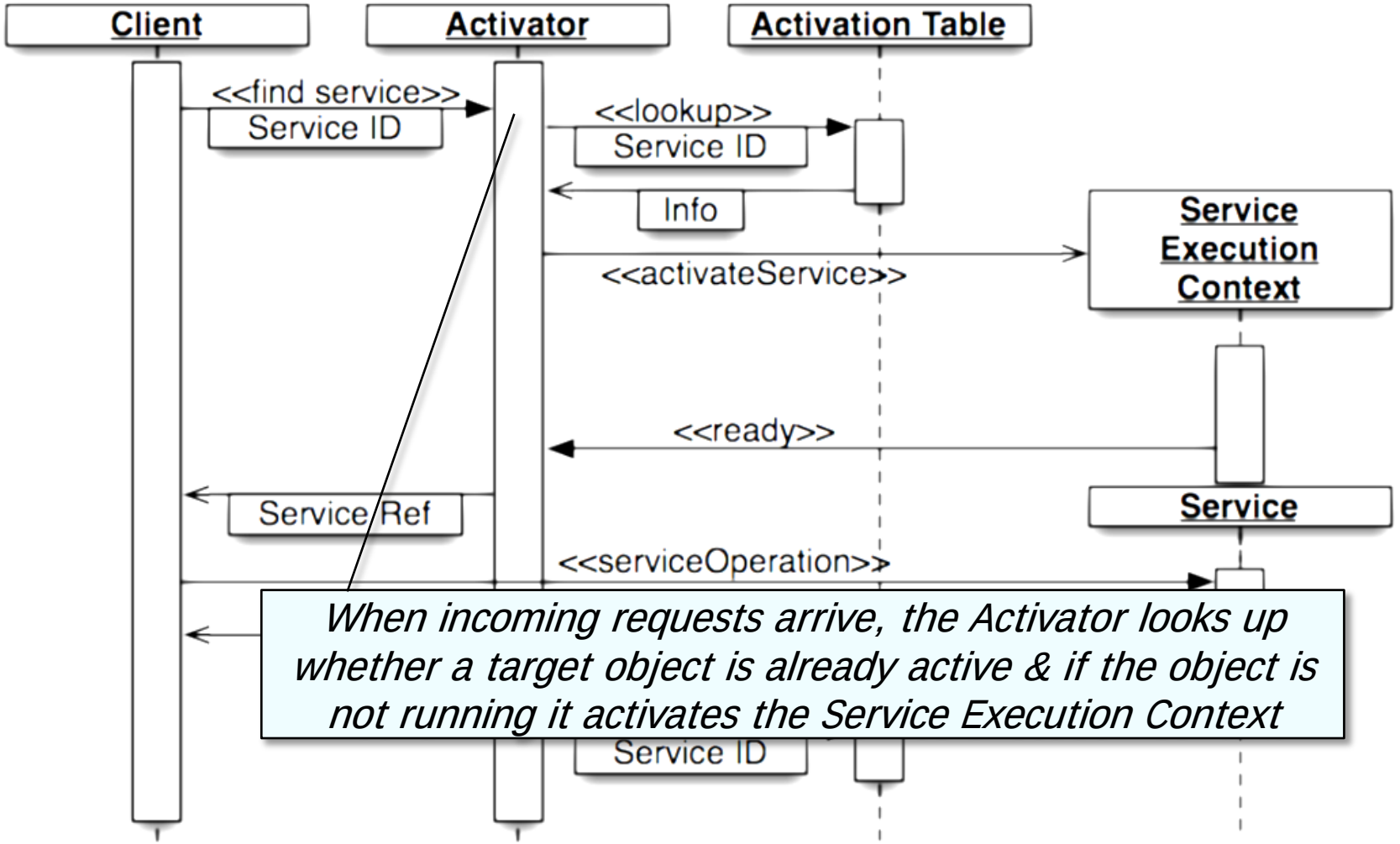
Activator POSA4 Design Pattern

Dynamics



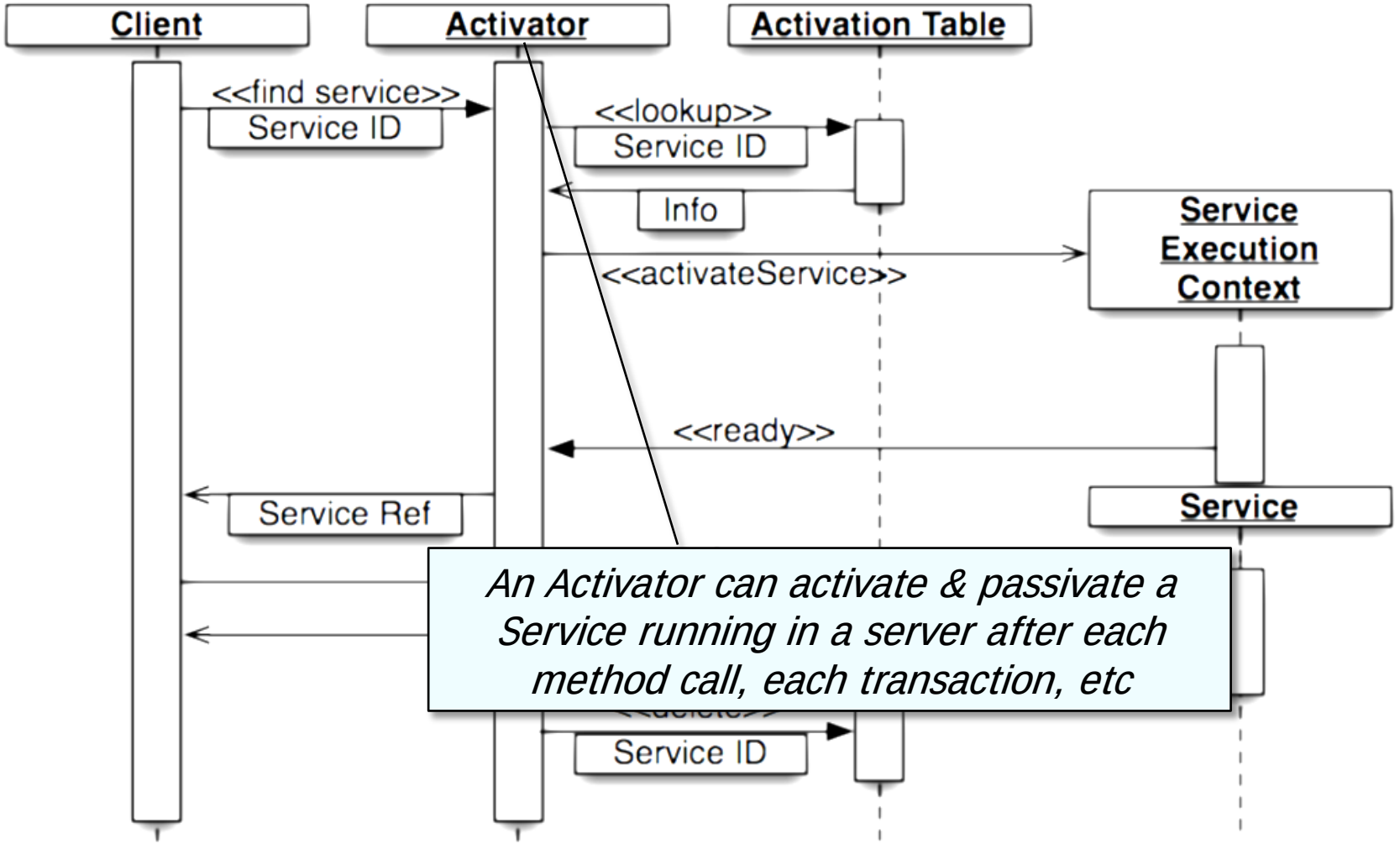
Activator POSA4 Design Pattern

Dynamics



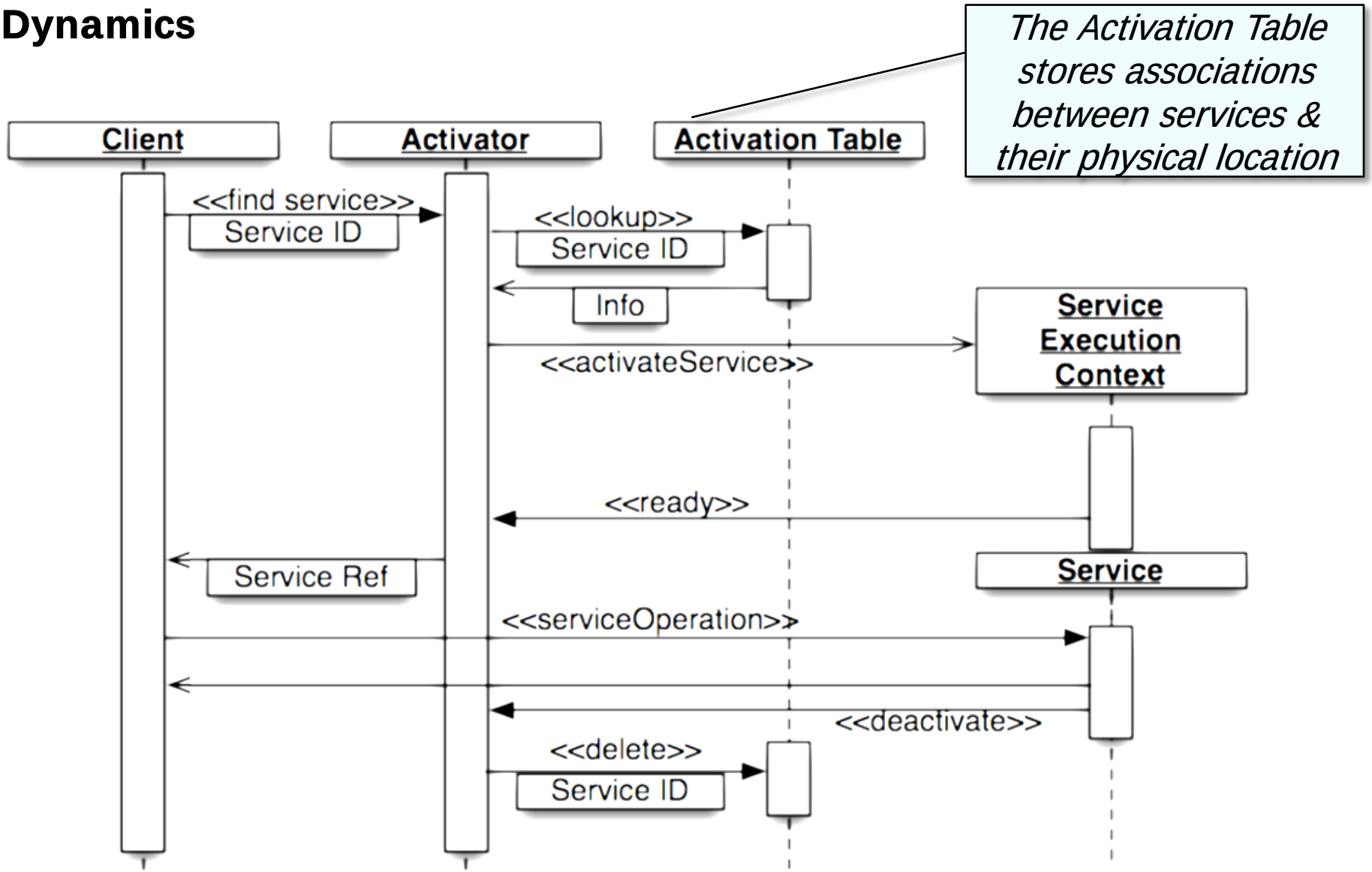
Activator POSA4 Design Pattern

Dynamics



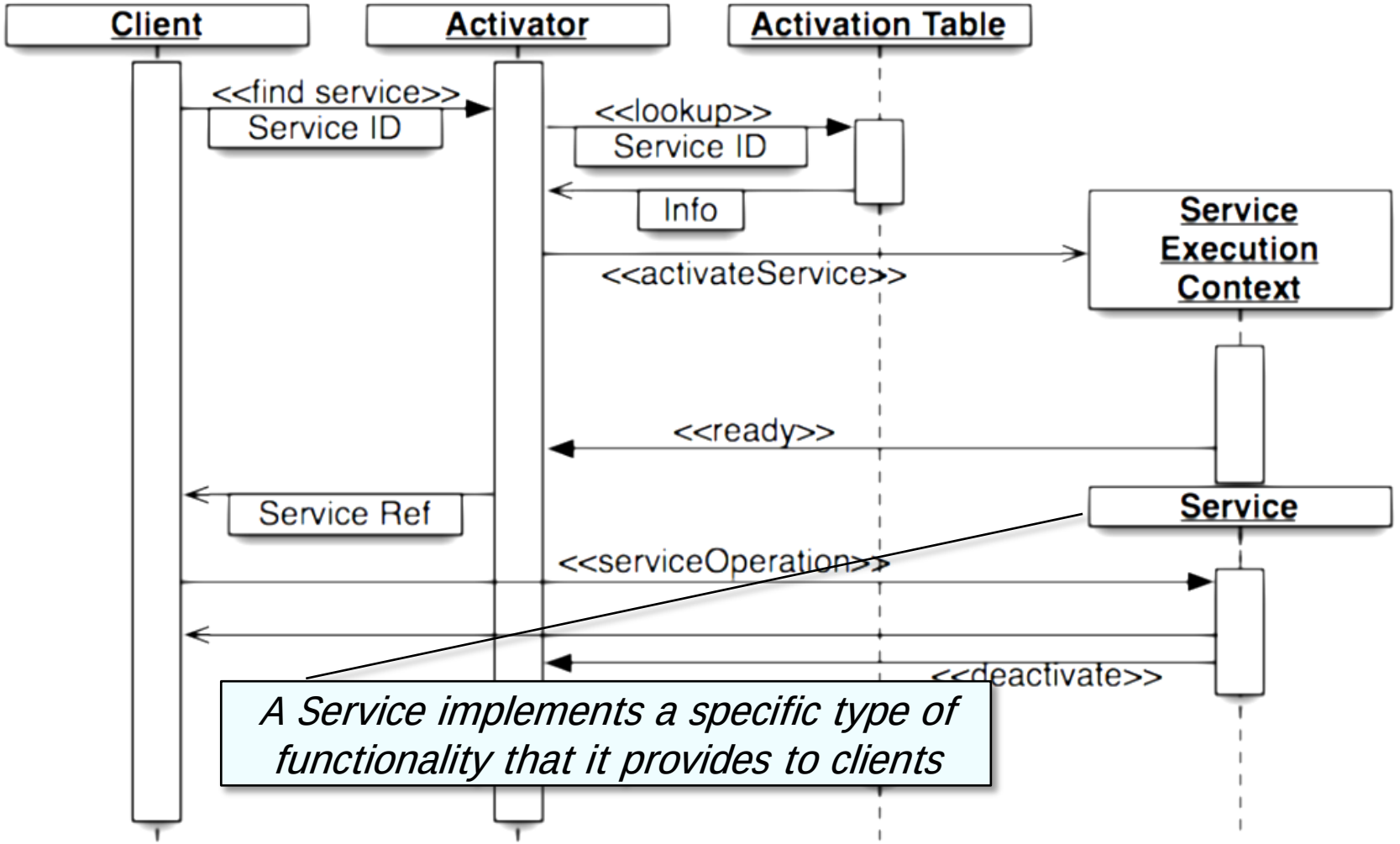
Activator POSA4 Design Pattern

Dynamics



Activator POSA4 Design Pattern

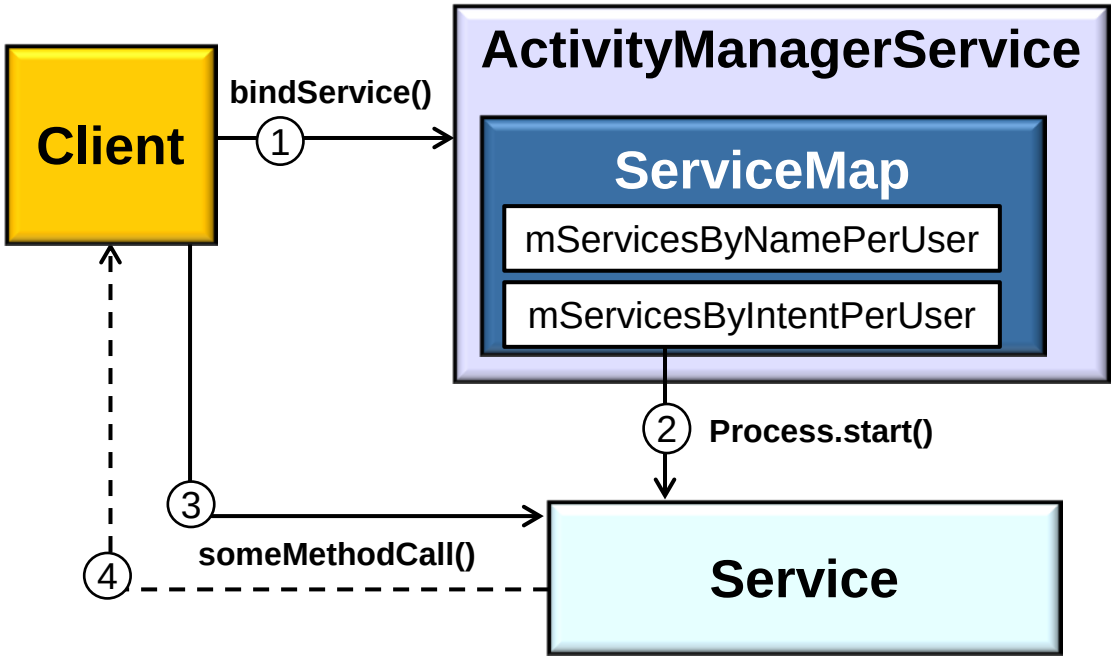
Dynamics



Activator POSA4 Design Pattern

Consequences

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

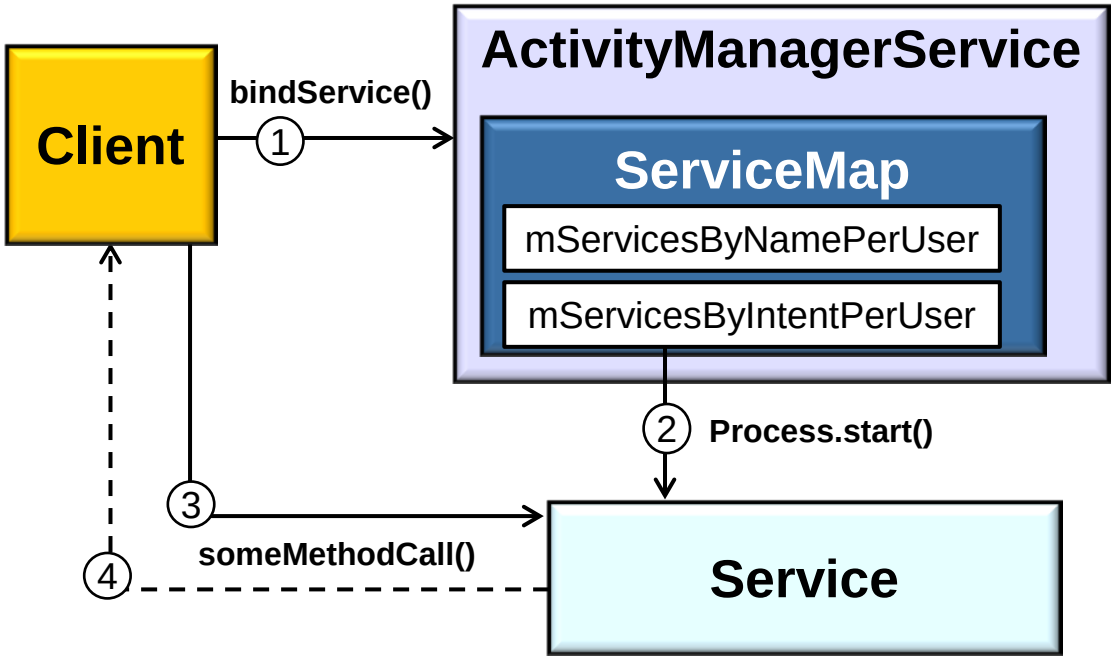


Activator POSA4 Design Pattern

Consequences

+ More effective resource utilization

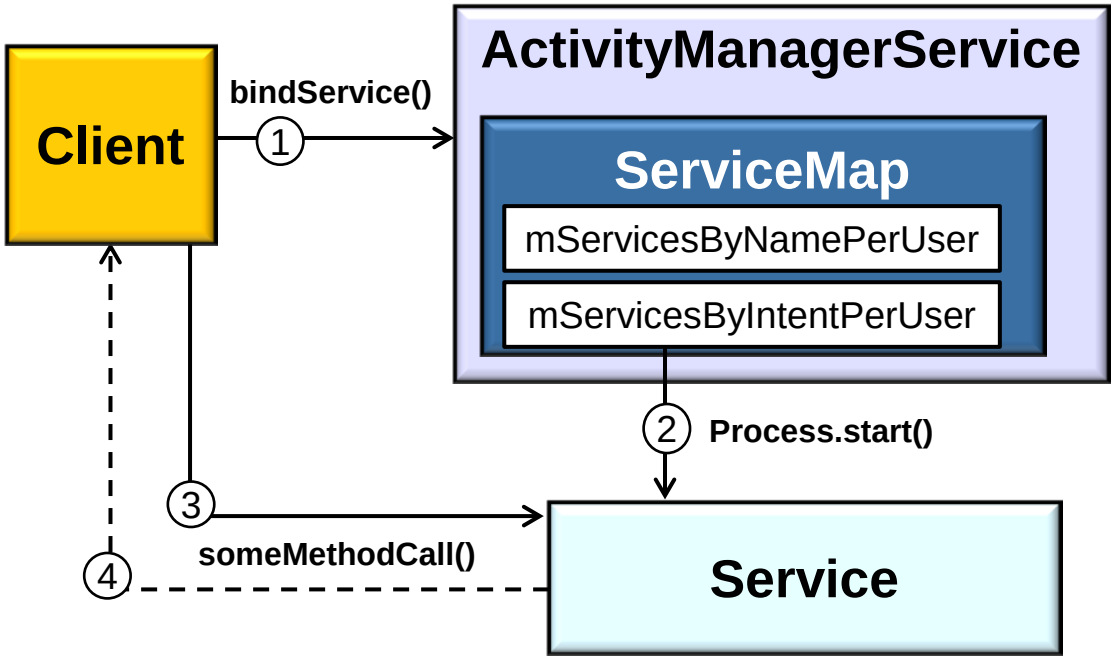
+ Coarse-grained concurrency
  - By spawning server processes to run on multi-core/CPU computers



Activator POSA4 Design Pattern

Consequences

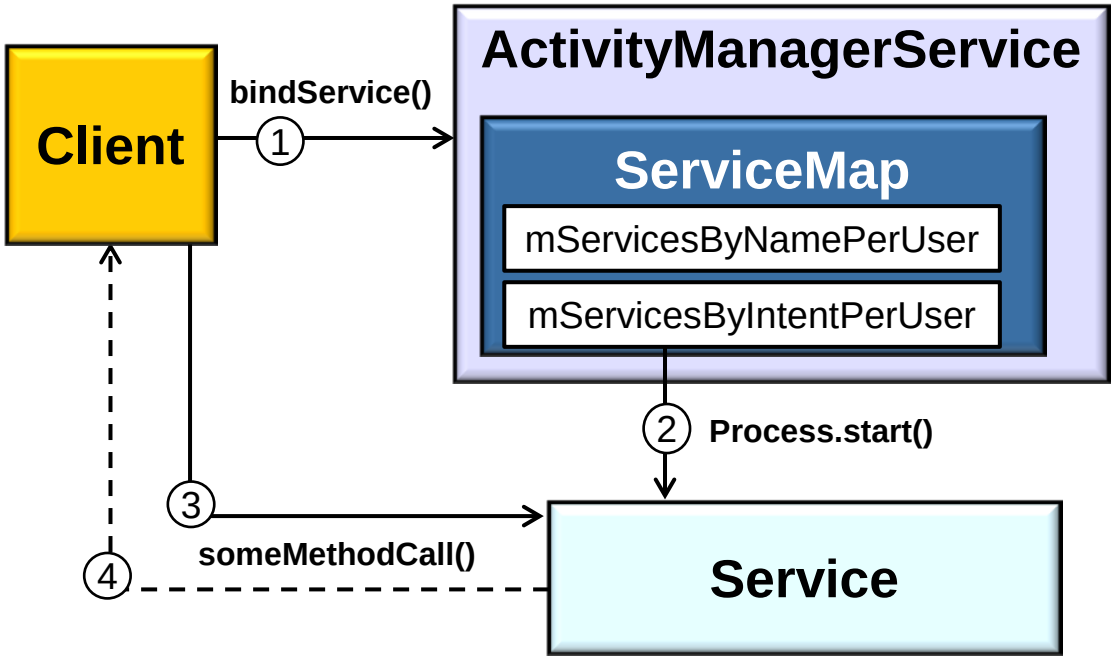
+ More effective resource utilization
+ Coarse-grained concurrency
+ Modularity, testability, & reusability
  - Application modularity & reusability is improved by decoupling server implementations from the manner in which the servers are activated



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



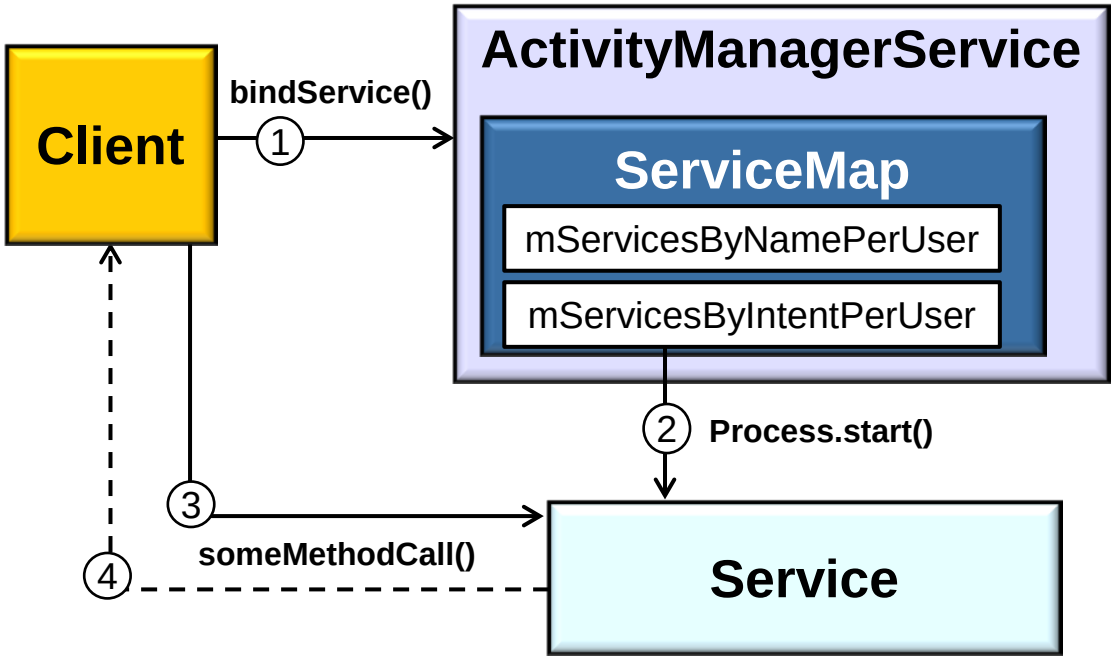
Activator POSA4 Design Pattern

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

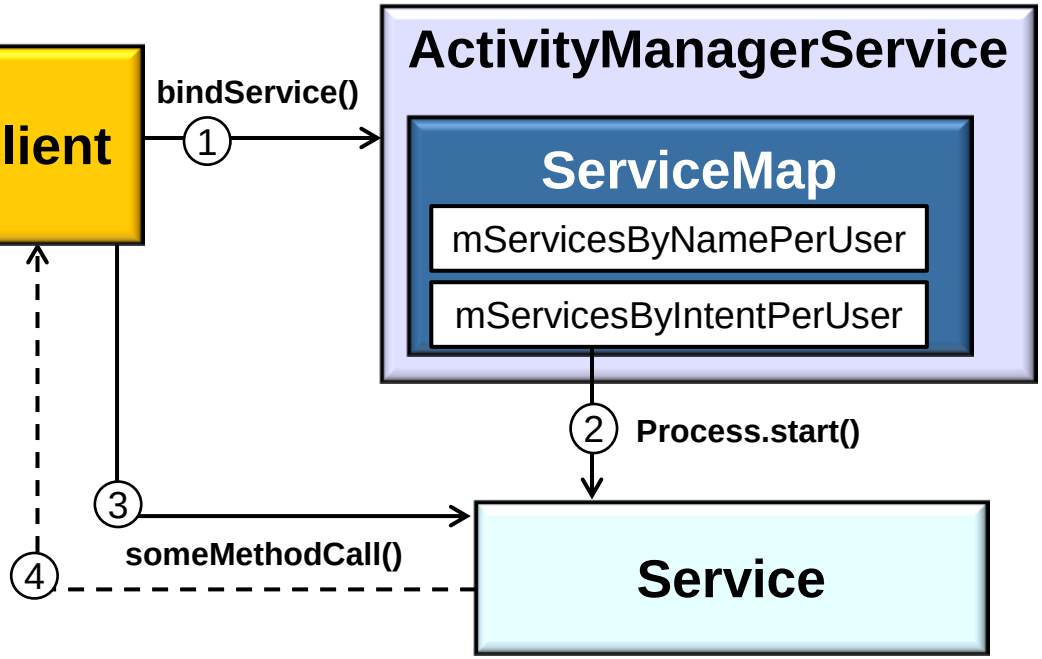
Consequences

– Lack of determinism & ordering dependencies

– Reduced security & reliability

– Increased run-time overhead & infrastructure complexity

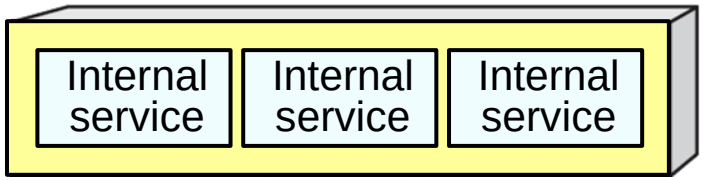
- By adding levels of abstraction & indirection when activating & executing components



Activator POSA4 Design Pattern

Known Uses

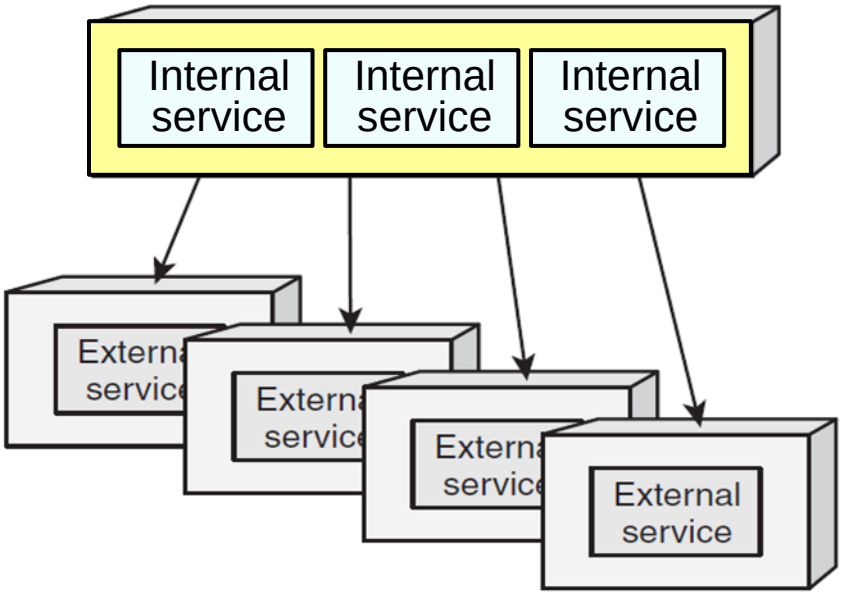
- UNIX Inetd “super server”
  - Internal services are fixed at static link time
    - e.g., **`ECHO`** & **`DAYTIME`**



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**

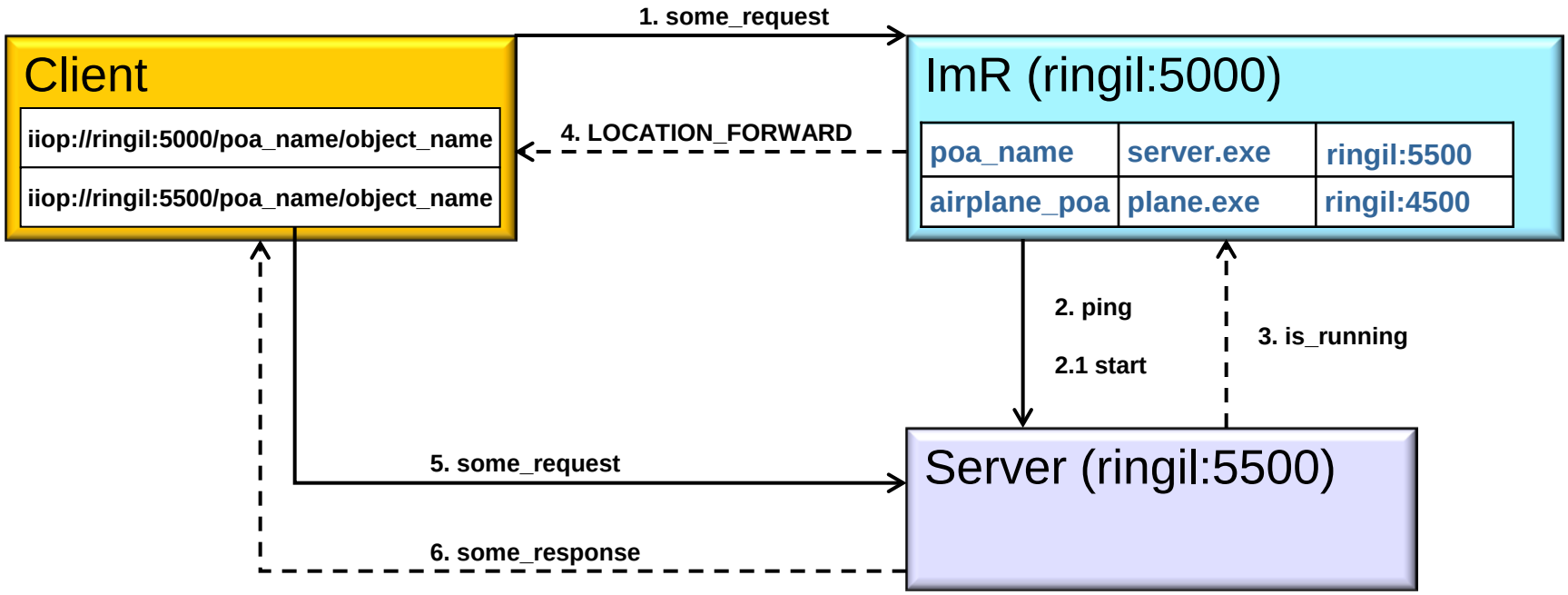


See en.wikipedia.org/wiki/Inetd for more on Inetd

Activator POSA4 Design Pattern

Known Uses

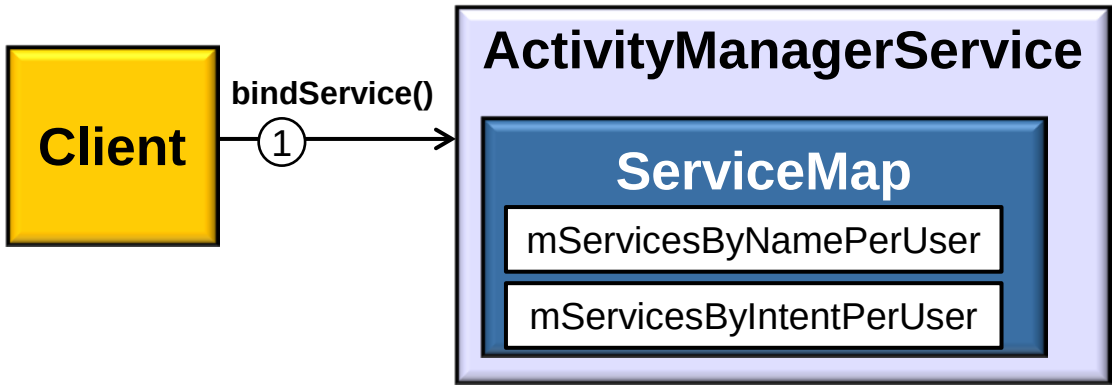
- UNIX Inetd “super server”
- CORBA Implementation Repository



Activator POSA4 Design Pattern

Known Uses

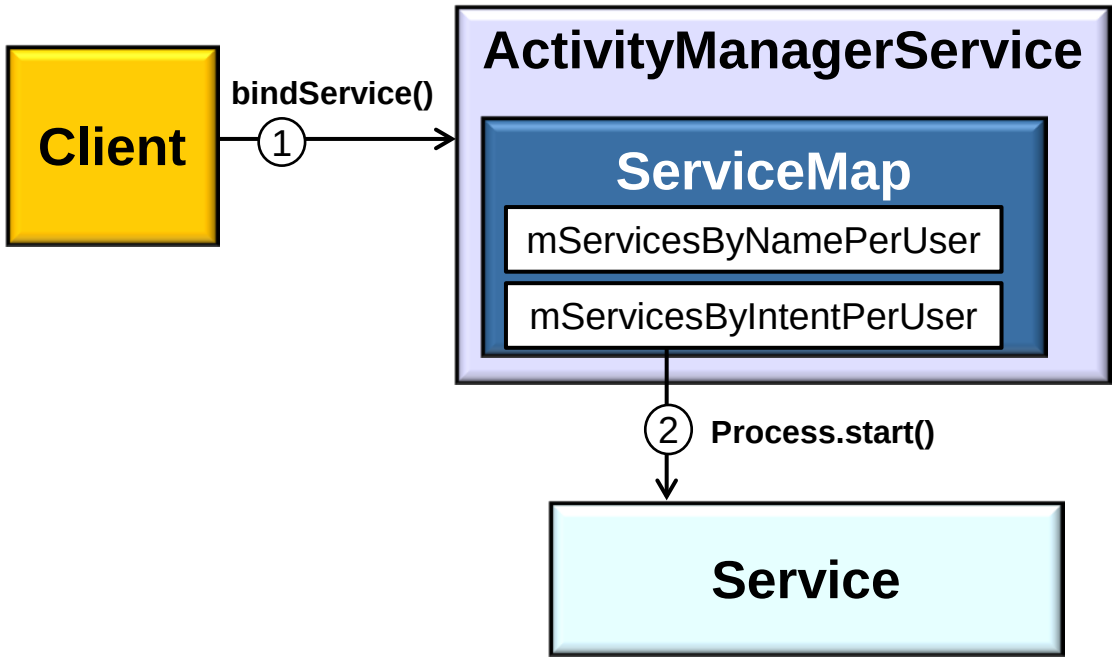
- UNIX Inetd “super server”
- CORBA Implementation Repository
- Android ActivityManagerService



Activator POSA4 Design Pattern

Known Uses

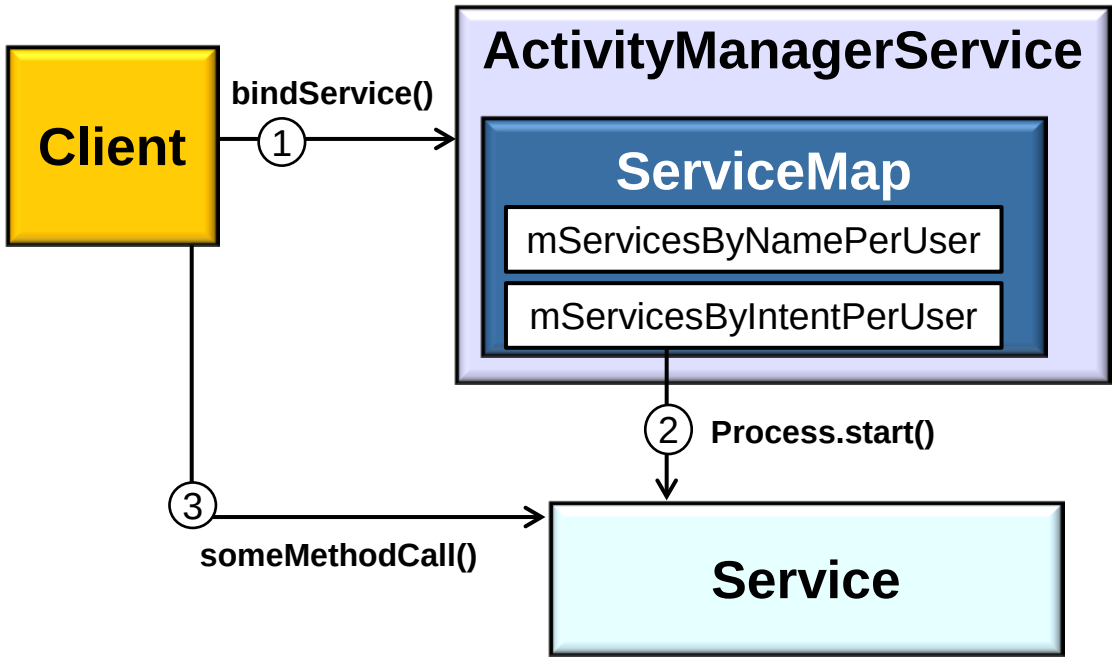
- UNIX Inetd “super server”
- CORBA Implementation Repository
- Android ActivityManagerService



Activator POSA4 Design Pattern

Known Uses

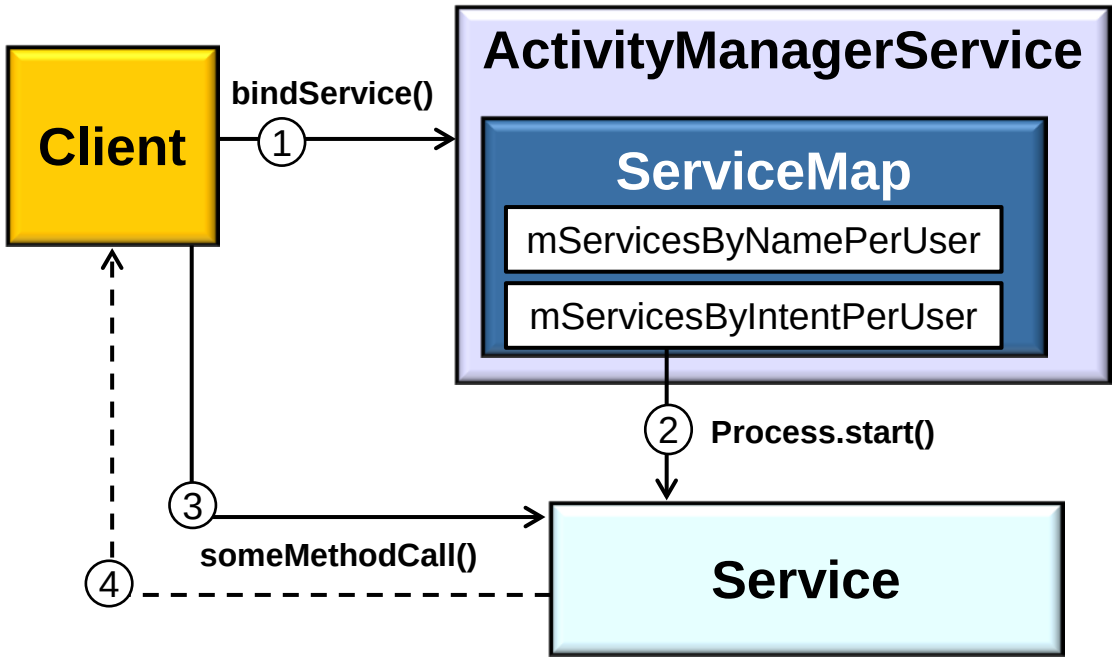
- UNIX Inetd “super server”
- CORBA Implementation Repository
- Android ActivityManagerService



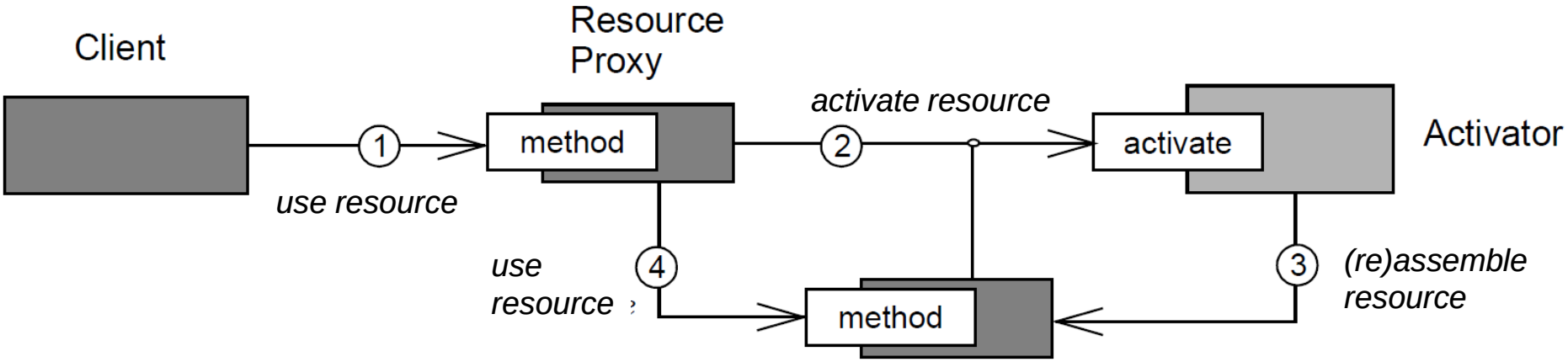
Activator POSA4 Design Pattern

Known Uses

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

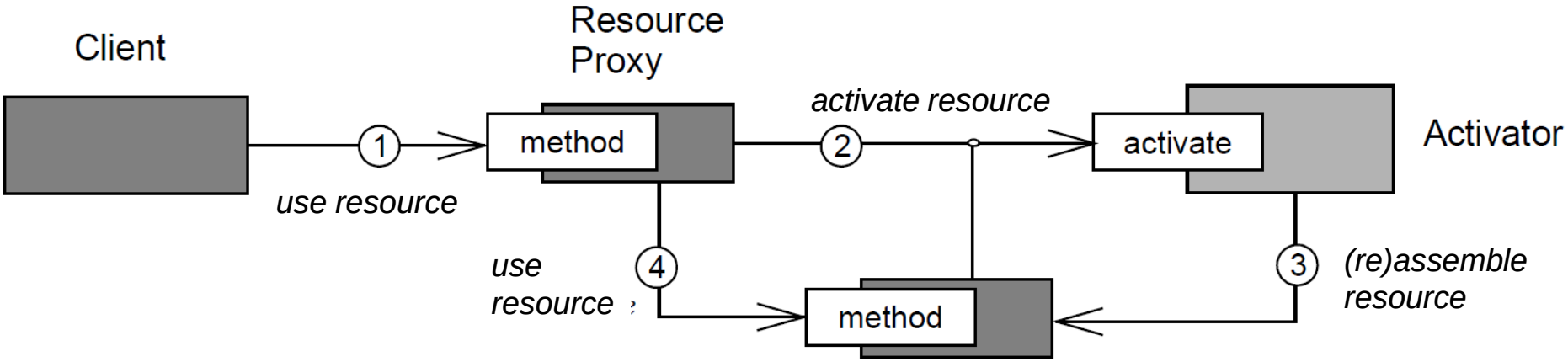


Summary



- *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

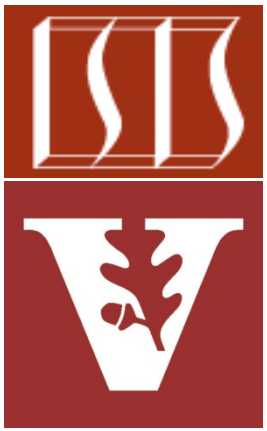


- *Activator* frees clients from the responsibility of (re)activating the resources they use
- *Activator* also ensures that (re)activating a resource incurs minimal overhead because it maintains information about how to optimize this process
  - e.g., an activator could reload the resource's persistent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

www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf has more info

Android Services & Local IPC: The Activator Pattern (Part 2)

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



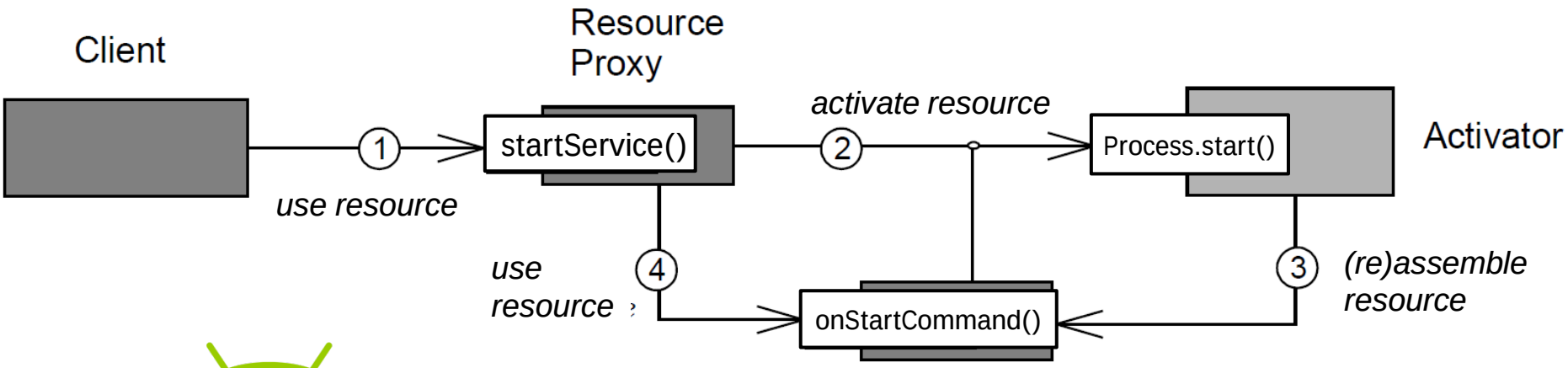
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand how the *Activator* pattern is applied in Android



Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
  - Encapsulate each distinct unit of app functionality into a self-contained service

```
public abstract class Service
       extends ContextWrapper
       implements
          ComponentCallbacks2
{
  public abstract IBinder
    onBind(Intent intent);

  ...
}
```

frameworks/base/core/java/android/app/Service.java has the source code

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
  - Encapsulate each distinct unit of app functionality into a self-contained service
  - Examples of service identifier representations include URLs, IORs, TCP/IP port numbers & host addresses, Android Intents, etc.

Intent Element	Purpose
Name	Optional name for a component
Action	A string naming the action to perform or the action that took place & is being reported
Data	URI of data to be acted on & the MIME type of that data
Category	String giving additional info about the action to execute

frameworks/base/core/java/android/content/Intent.java has the source code

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
  - Determine overhead of activating & deactivating services on-demand vs. keeping them alive for the duration of the system vs. security implications, etc.

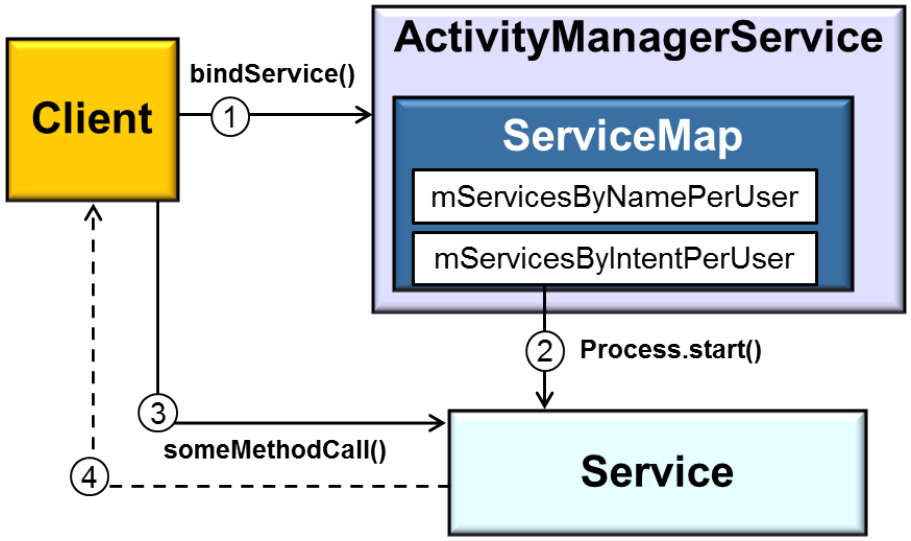
Android Service	Purpose
Media Playback Service	Provides "background" audio playback capabilities
Exchange Email Service	Send/receive email messages to an Exchange server
SMS & MMS Services	Manage messaging operations, such as sending data, text, & PDU messages
Alert Service	Handle calendar event reminders

packages/apps has source code for many Bound & Started Services

Activator POSA4 Design Pattern

Implementation

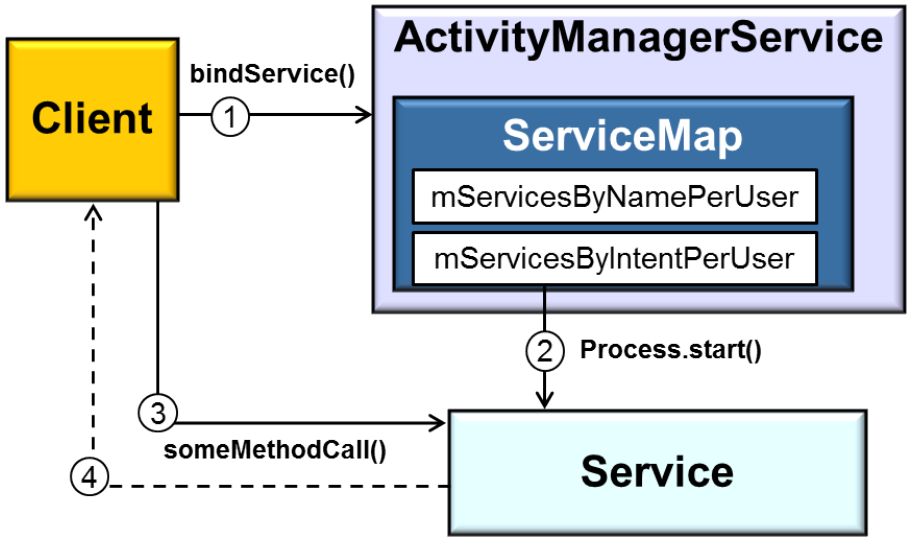
- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
    - e.g., an OS process/thread or middleware container



Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
  - Define service registration strategy
    - e.g., static text file or dynamic object registration



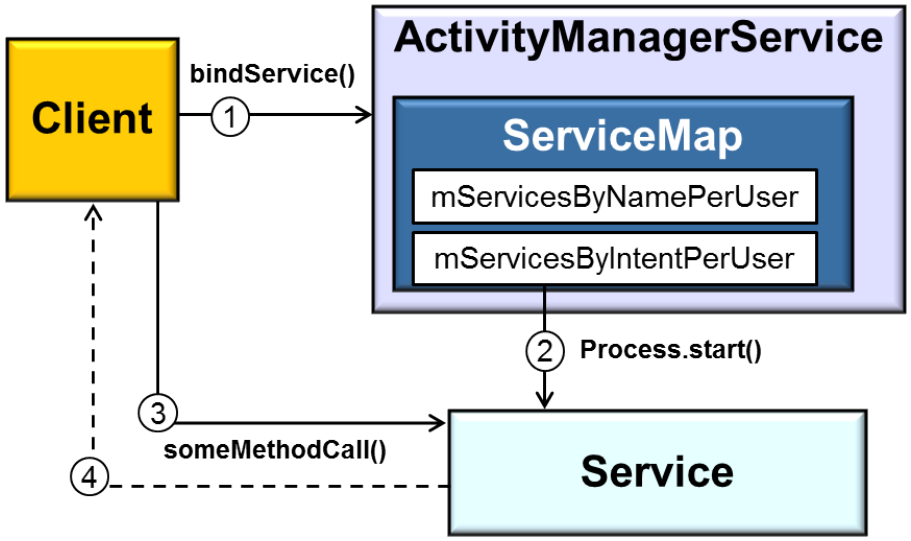

```
<service android:name=
"com.android.music.MediaPlaybackService"
   android:exported="false"/>
```

frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
  - Define service registration strategy
  - Define service initialization strategy
    - e.g., stateful vs. stateless services



Android Services are responsible for managing their own persistent state

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
  - Define service execution context representation
  - Define service registration strategy
  - Define service initialization strategy
  - Define service deactivation strategy
    - e.g., service-triggered, client-triggered, or activator-triggered deactivation

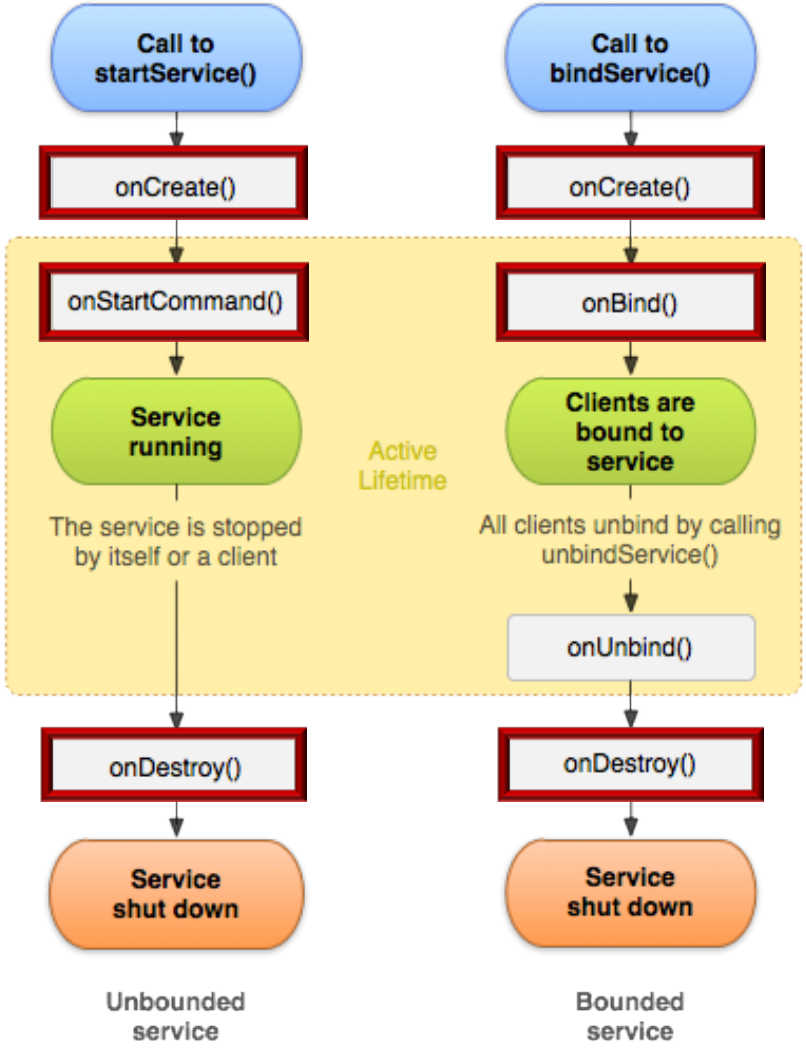
Started Service	Bound Service
• Service runs indefinitely & must stop itself by calling stopSelf() • A component can also stop the service by calling stopService() • When Service is stopped, Android destroys it	• Multiple clients can bind to same Service • When all of them unbind, the system destroys the Service • The Service does not need to stop itself an

developer.android.com/guide/components/services.html#Lifecycle has more

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
- Define interoperation between services & service execution context
  - Typically implemented via some type of lifecycle callback hook methods

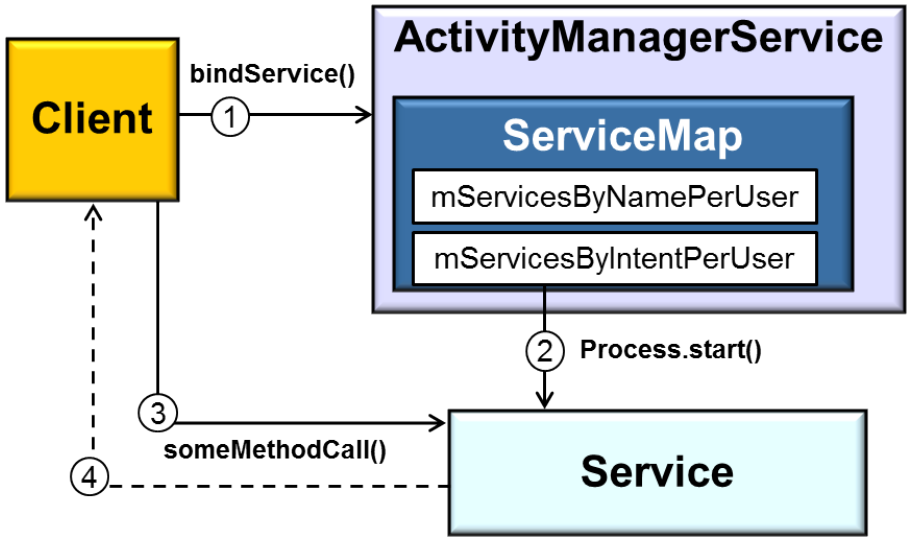


developer.android.com/guide/components/services.html#LifecycleCallbacks

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
- Define interoperation between services & service execution context
- Implement the activator
  - Determine the association between activators & services
    - e.g., singleton (shared) vs. exclusive vs. distributed activator

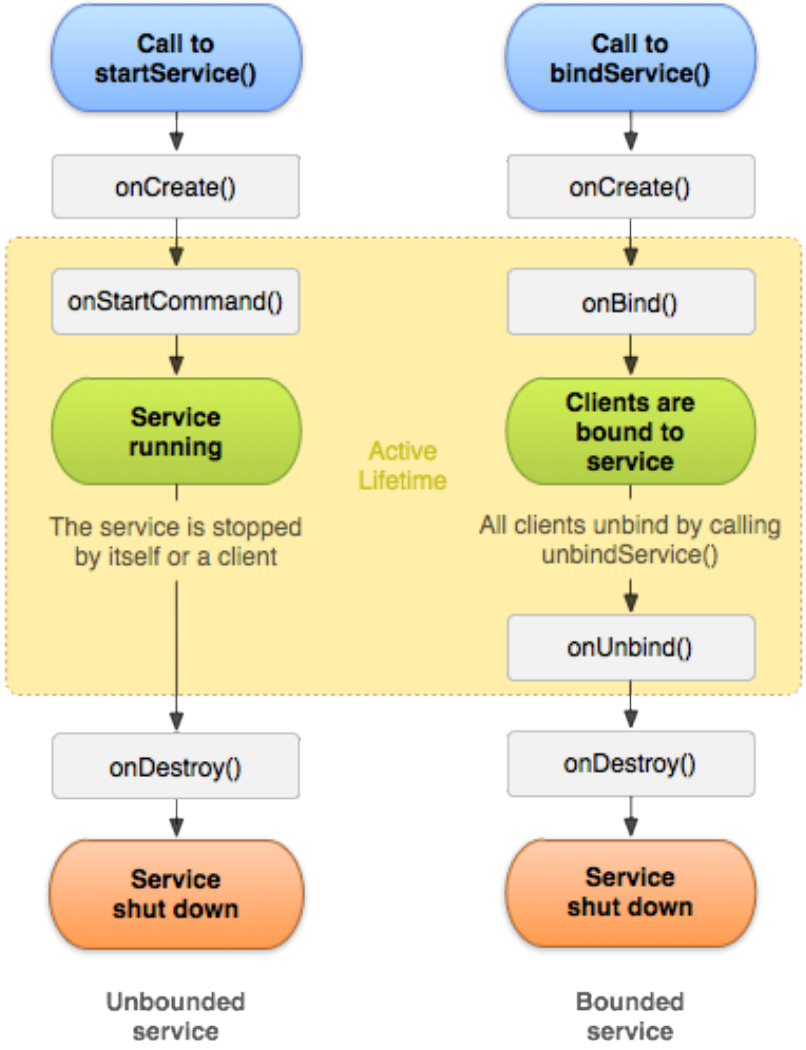


The Android Activity Manager Service is a singleton activator

Activator POSA4 Design Pattern

Implementation

- Define services & service identifiers
- Identify services to activate & deactivate on demand
- Develop service activation & deactivation strategy
- Define interoperation between services & service execution context
- Implement the activator
  - Determine the association between activators & services
  - Determine the degree of transparency
    - e.g., explicit vs. transparent activator

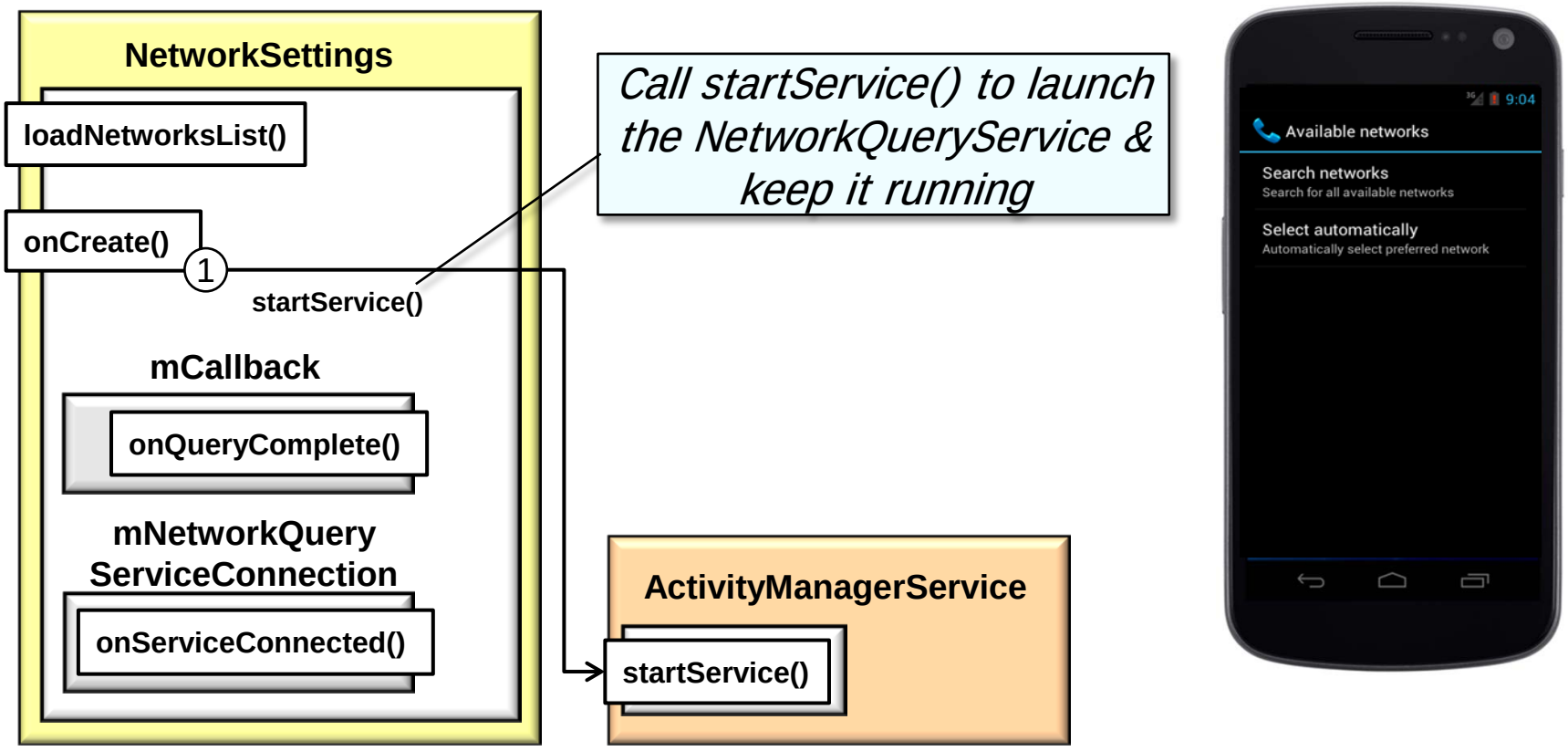


Android Started & Bound Services use an explicit activator model

Activator POSA4 Design Pattern

Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

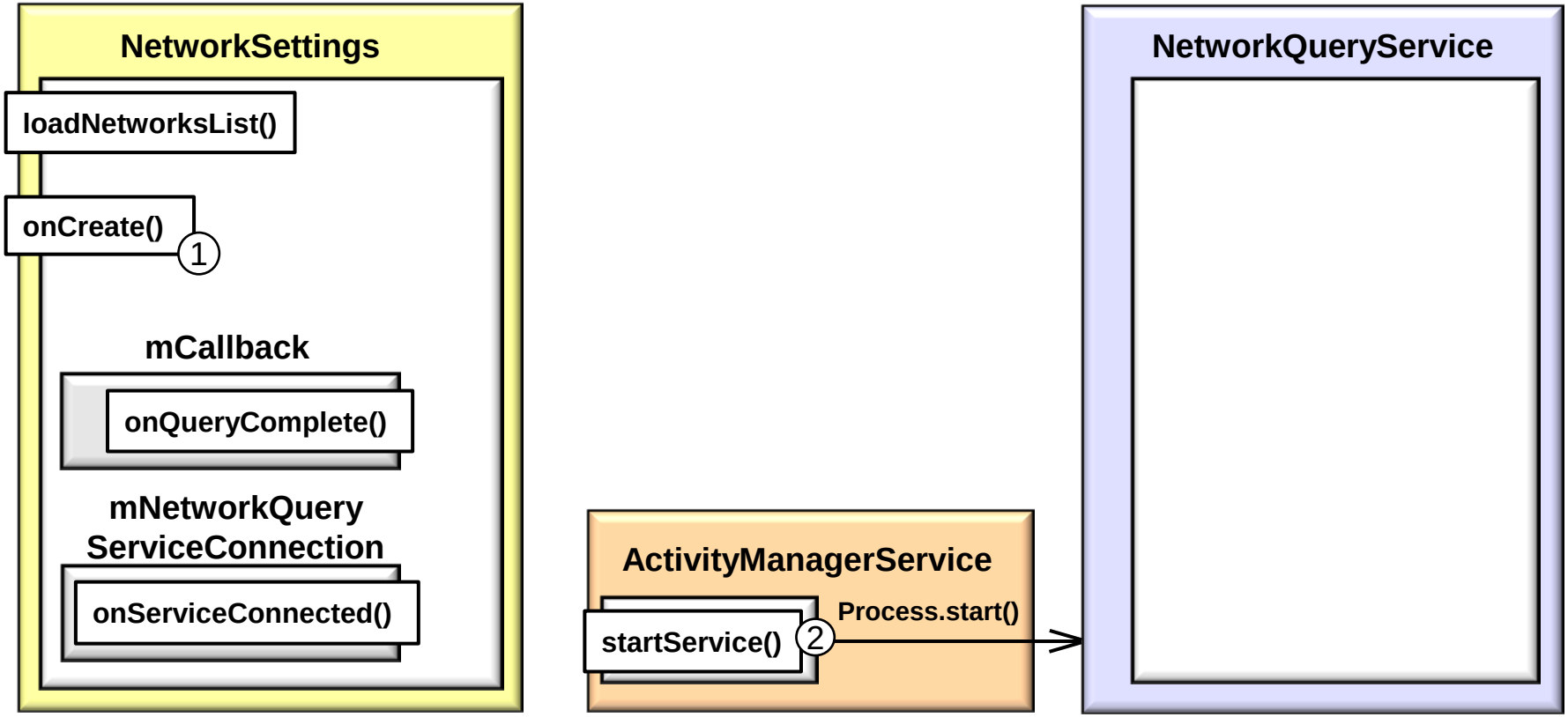


packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

Activator POSA4 Design Pattern

Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

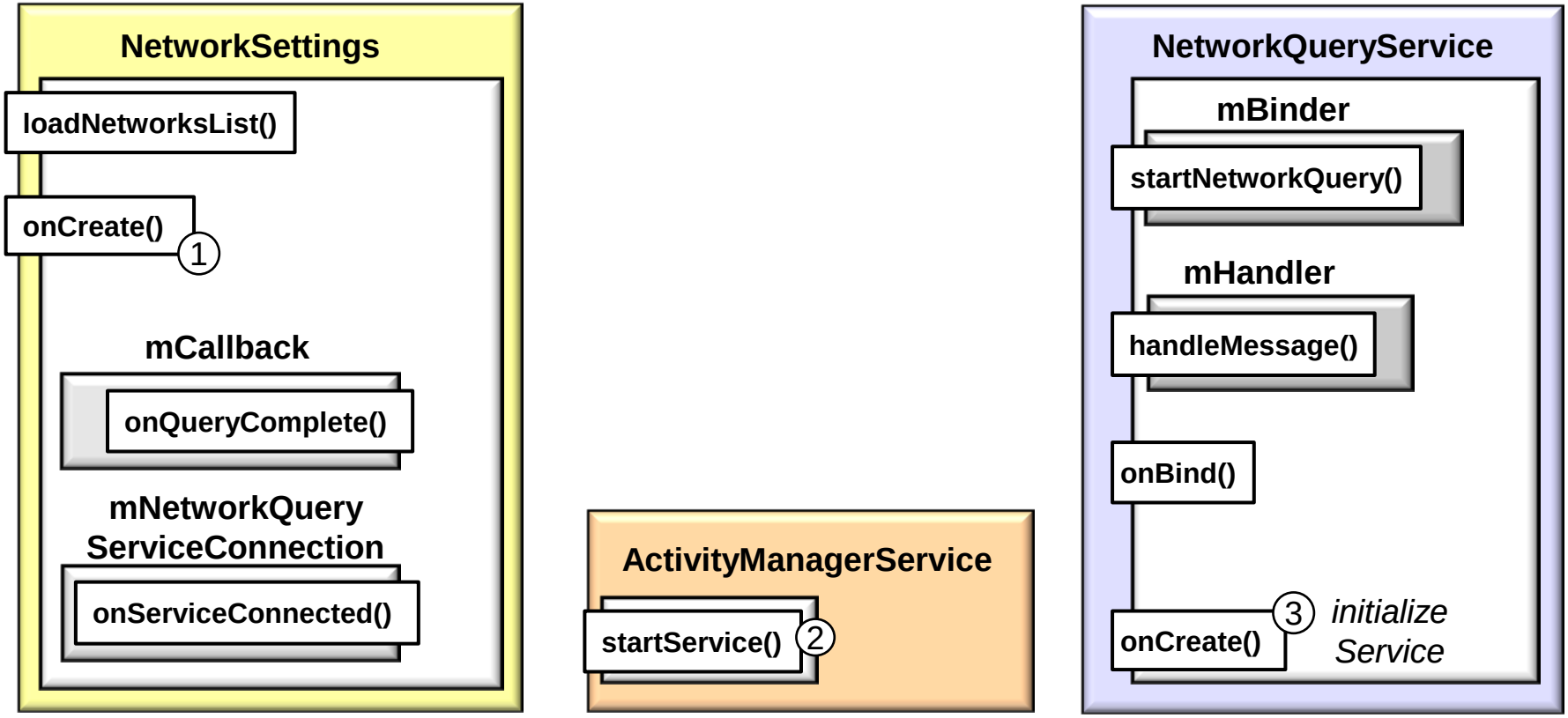


frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

Activator POSA4 Design Pattern

Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability

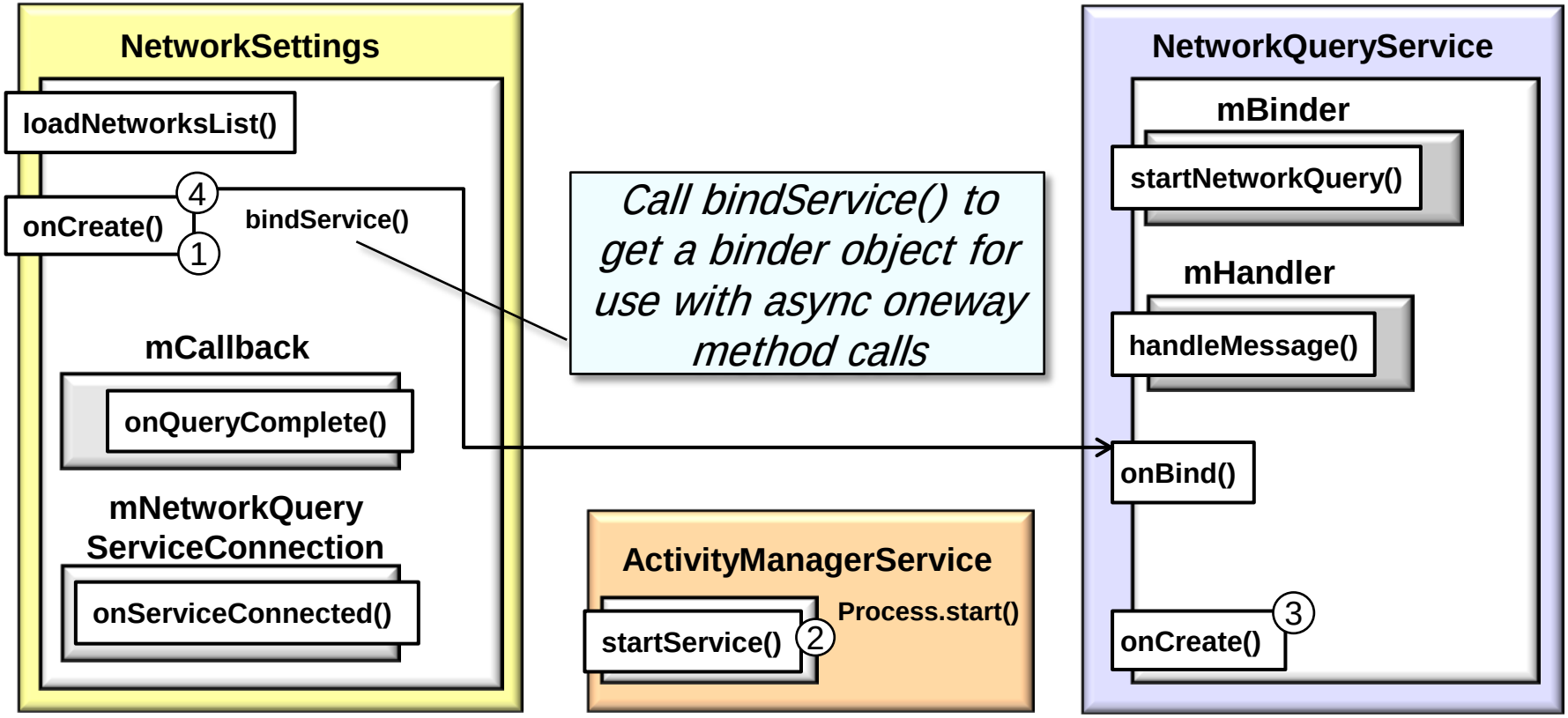


packages/apps/Phone/src/com/android/phone/NetworkQueryService.java has source

Activator POSA4 Design Pattern

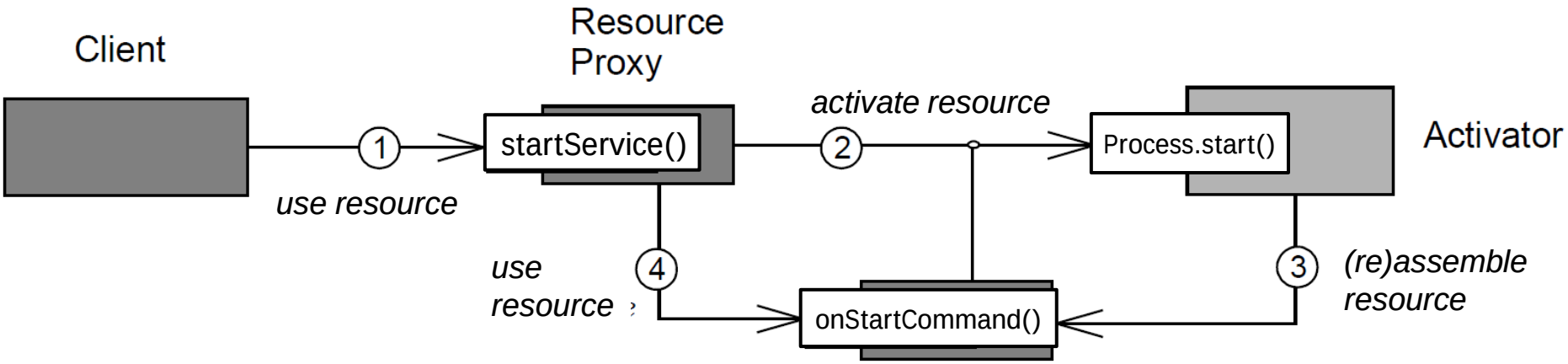
Applying the Activator pattern in Android

- The NetworkSettings Activity uses the *Activator* pattern to launch the NetworkQueryService to assist in querying the network for service availability



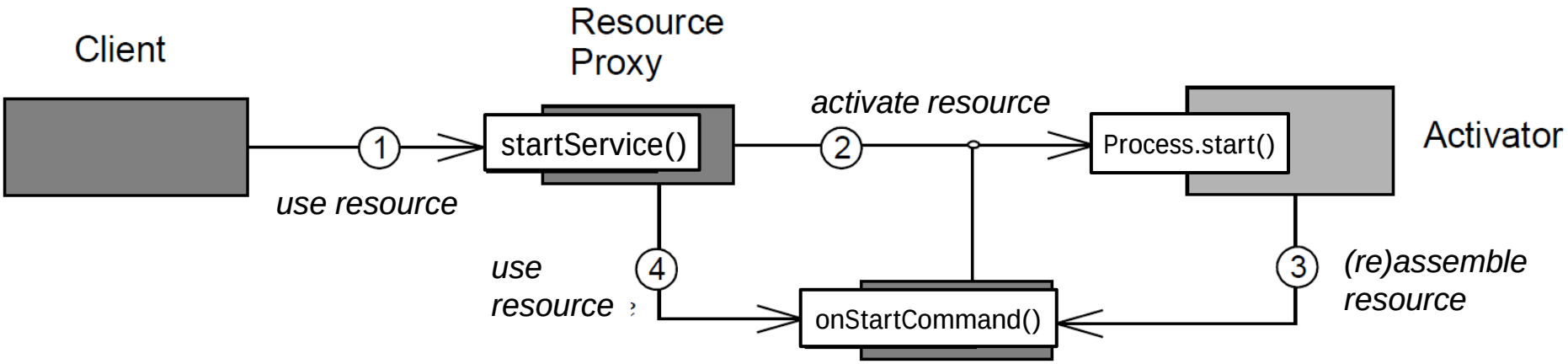
packages/apps/Phone/src/com/android/phone/NetworkSetting.java has source code

Summary



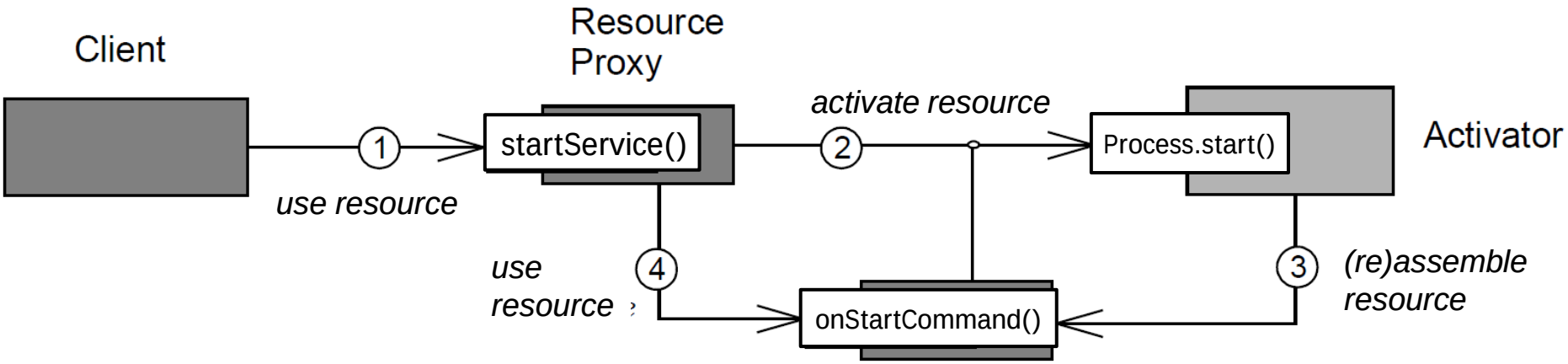
- The Android Started & Bound Services implement the *Activator* pattern

Summary



- The Android Started & Bound Services implement the *Activator* pattern
- These Services can process requests in background processes or threads
  - Processes can be configured depending on directives in the AndroidManifest.xml file

Summary



- The Android Started & Bound Services implement the *Activator* pattern
- These Services can process requests in background processes or threads
  - Processes can be configured depending on directives in the AndroidManifest.xml file
  - Threads can be programmed using the Android Intent Service framework, as discussed next