Ingeniería de Sistemas y Computación

Arquitectura de Software
Facilidad de Prueba

Universidad de los Andes

Agenda del día

- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- Patrones de diseño detallado

Introducción

- **Facilidad de Prueba (Testability)**
  - Se refiere a la facilidad con la que el software puede demostrar sus fallas a través de la ejecución de pruebas
  - Se refiere a la *probabilidad*, asumiendo que el sistema tiene al menos una falla, de que éste vuelva a fallar en su *próxima* ejecución de pruebas

Introducción

- Al menos el 40% del costo del desarrollo de sistemas de información está asociado a actividades de prueba
- Para que un sistema sea fácil de probar, debe ser posible:
  - *Controlar* las *entradas* y el *estado interno* de cada uno de sus componentes
  - *Observar* las *salidas* de cada componente

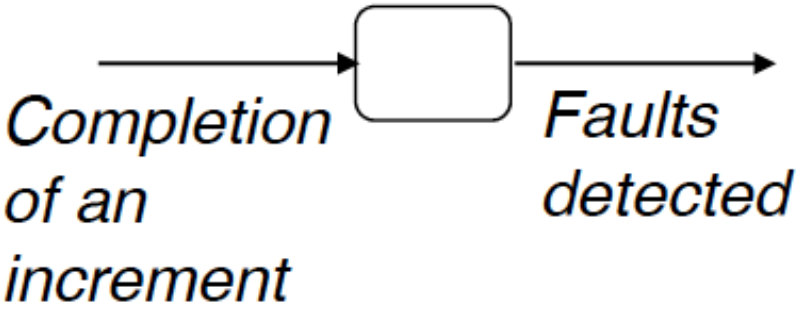
Introducción

- Las pruebas pueden ser hechas por:
  - Desarrolladores
  - Personal de pruebas del grupo de desarrollo (Testers)
  - Administrador del sistema
  - Verificador del sistema
  - Personal de pruebas del cliente (Client Tester)
  - Usuario del sistema
- Artefactos que podrían ser probados:
  - Porción de código fuente
  - Sección del diseño del sistema
  - Sistema completo

Agenda del día

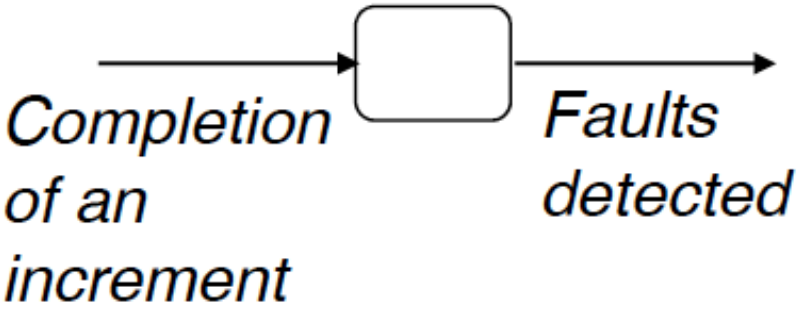
- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- Patrones de diseño detallado

Tácticas arquitecturales para favorecer Facilidad de Prueba



- Las tácticas facilitan la ejecución de pruebas al sistema cuando se *completa* un *incremento de desarrollo* de software
- Un proceso de ejecución de pruebas requiere que las *entradas* sean proporcionadas al software en prueba y que las *salidas* sean capturadas
  → *Test Harness*

Tácticas arquitecturales para favorecer Facilidad de Prueba



- Categorías de tácticas:
  - Proporcionar entradas y capturar salidas
  - Monitoreo interno

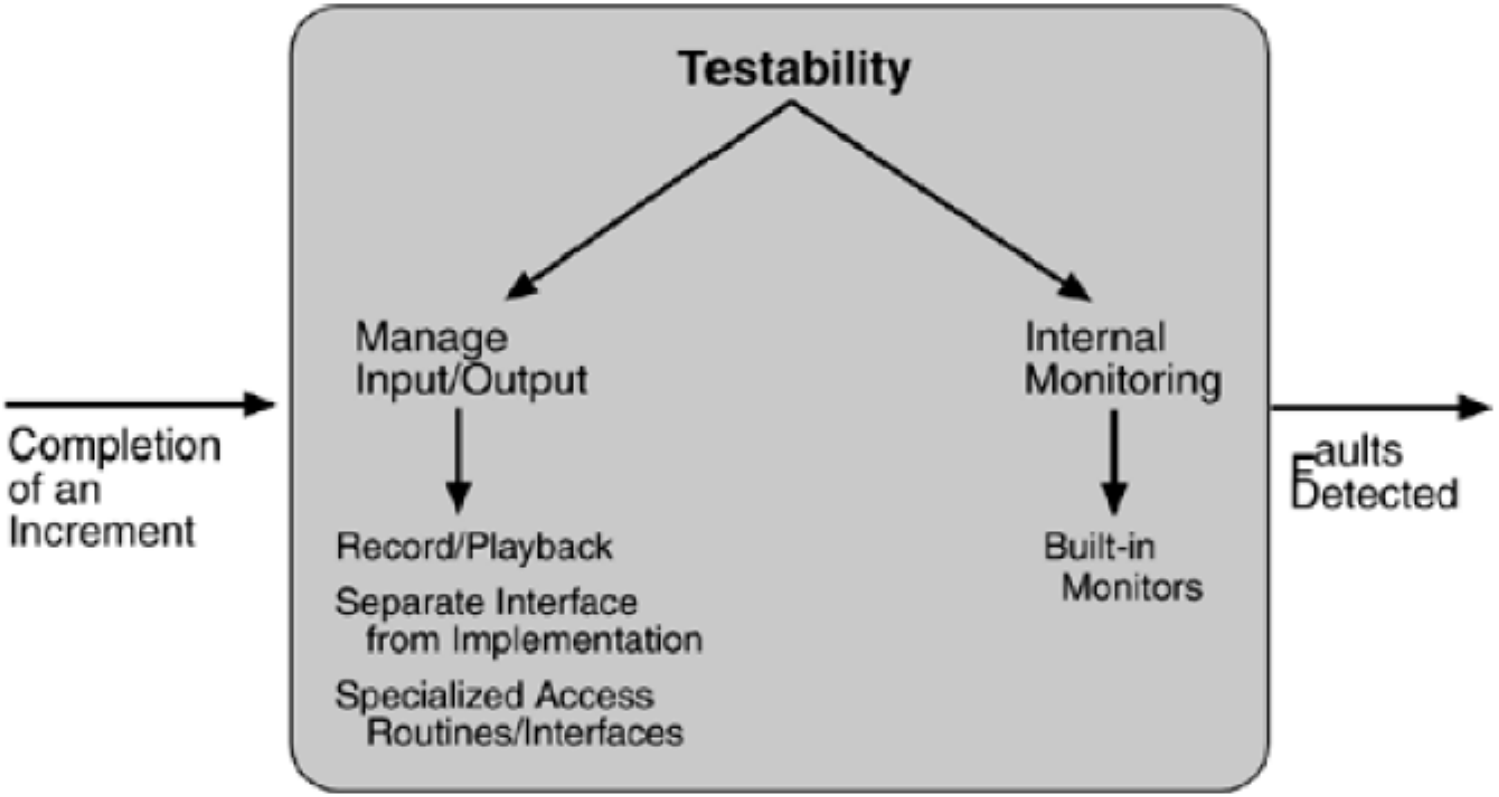
Entradas/Salidas

- Grabar/Reproducir
  - Captura y almacenamiento de información ingresada en la interfaz del sistema → *Repositorio*
  - Reproducción de información almacenada como entrada al software de pruebas → *Test Harness*
- Separar la interfaces de su implementación
  - Sustituir implementaciones de acuerdo a los propósitos de prueba
- Rutas/Interfaces de accesso especializadas
  - Vías para acceder los componentes permitiendo la especificación de datos de prueba independientes de la ejecución normal

Monitoreo Interno

- Objetivo
  - Identificar el estado interno de un componente para soportar el proceso de prueba
- Actividades
  - Proporcionar acceso al estado, rendimiento, seguridad, u otra información de los componentes
  - Registrar dicha información
  - Notificar a las personas y/o sistemas si los valores de lectura están por fuera de los rangos esperados

Resumen


Testability
Manage Input/Output
Internal Monitoring
Completion of an Increment
Faults Detected
Record/Playback
Separate Interface from Implementation
Specialized Access Routines/Interfaces
Built-in Monitors

Universidad de los Andes

Agenda del día

- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- Patrones de diseño detallado

Universidad de los Andes

Patrones arquitecturales

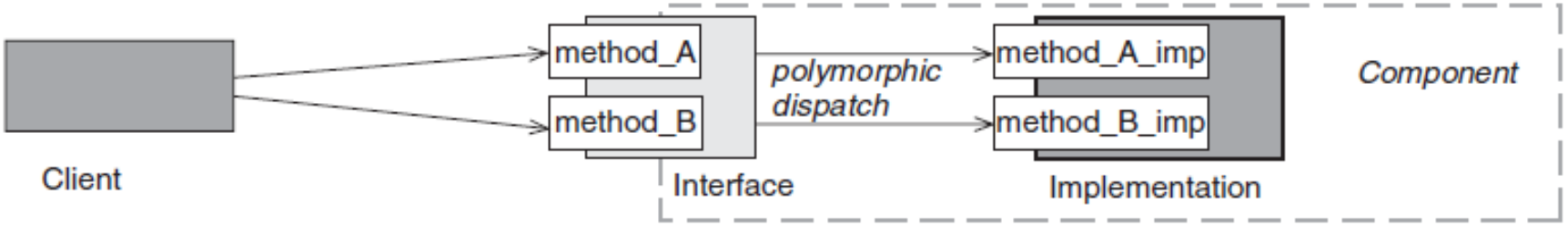
- Inclusión de interfaces
  - Interfaces explícitas
  - Extensión de interfaces
  - Interfaces introspectivas
  - Proxy
  - Delegados de negocio
  - Fachada
- Control de la aplicación
  - Procesador de comandos

Universidad de los Andes

Interfaces explícitas (Explicit Interface)

- Problem
  - Direct access to the full component implementation would make clients dependent on component internals, which ultimately increases application internal software coupling
- Solution
  - Separate the declared interface of a component from its implementation
  - Export the interface to the clients of the component, but keep its implementation private and location-transparent to the client

Interfaces explícitas (Explicit Interface)

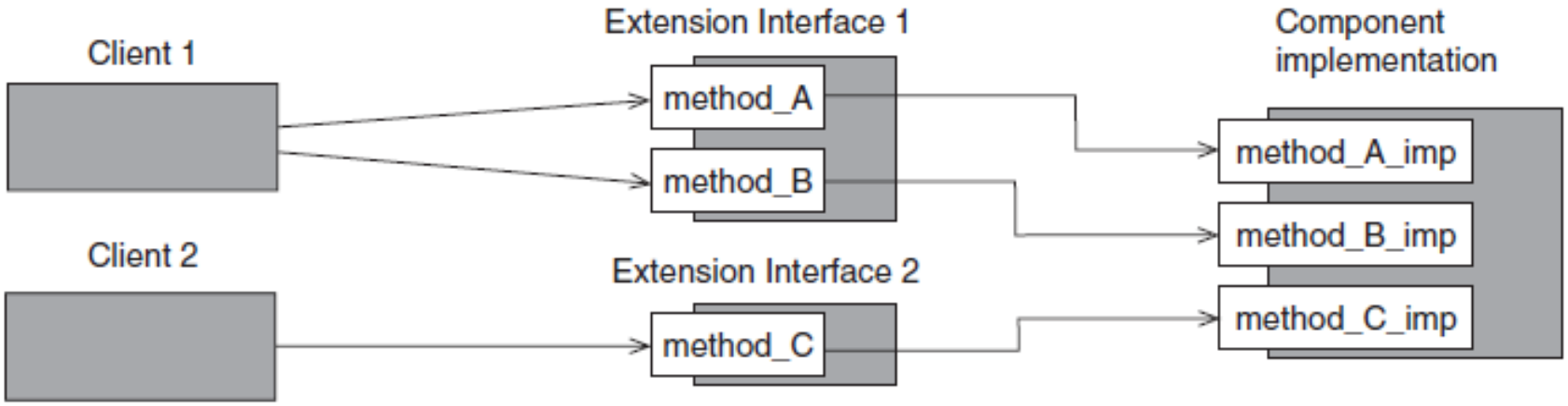


Universidad de los Andes

Extensión de interfaces (Extension Interface)

- Problem
  - The interface of a component is often affected, however, when its functionality is modified or extended, which can break the client code (in some cases even if the new functionality is not used)
- Solution
  - Let clients access a component only via specialized extension interfaces, and introduce one such interface for each role that the component provides

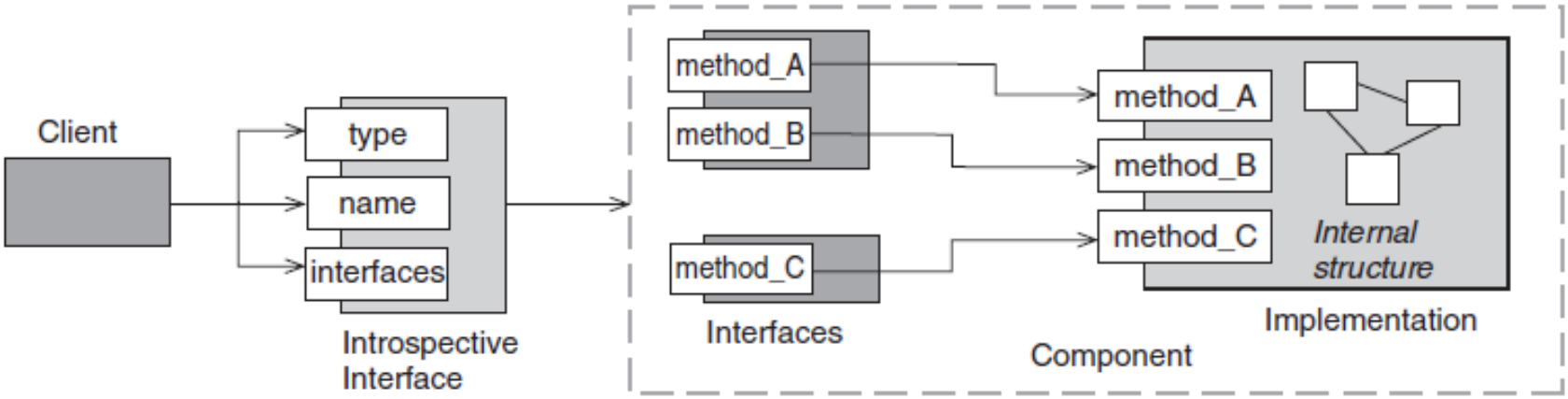
Extensión de interfaces (Extension Interface)



Interfaces introspectivas (Introspective Interface)

- Problem
  - Allowing clients to access information about component details directly could break component encapsulation and reduce dependency stability
- Solution
  - Introduce a special introspective interface for the component that allows clients to access information about its mechanisms and structure
  - Keep the introspective interface separate from the component's 'operational' interfaces

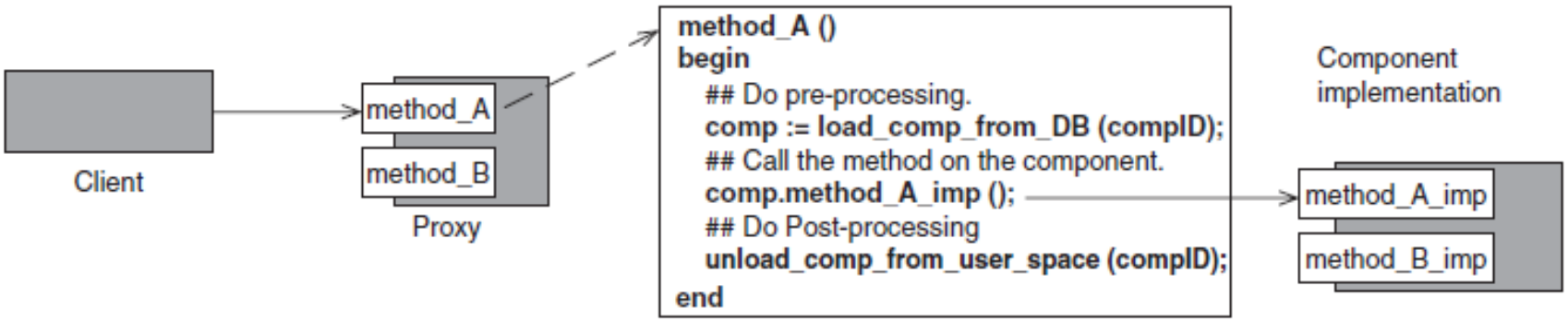
Interfaces introspectivas (Introspective Interface)



Proxy

- Problem
  - It is often impractical, or even impossible, to access the services of a component directly, for example because we must first check the access rights of its clients, or because its implementation resides on a remote server
- Solution
  - Encapsulate all component housekeeping functionality within a separate surrogate of the component (the proxy) and let clients communicate only through the proxy rather than with the component itself

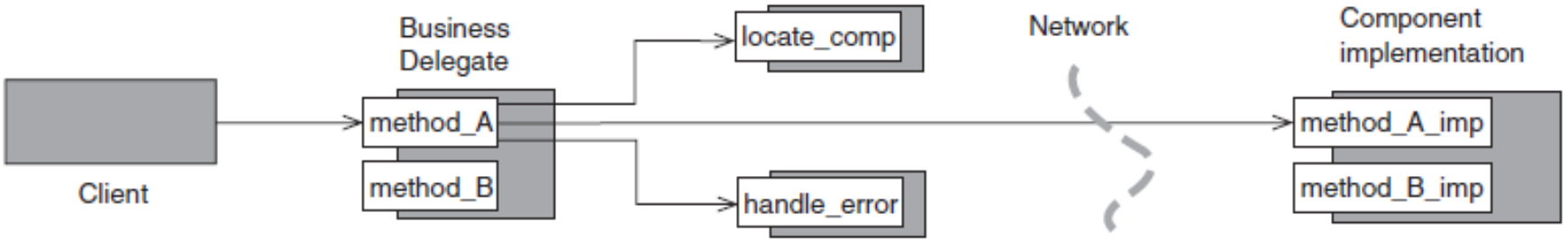
Proxy


Client
method_A
method_B
Proxy
method_A ()
begin
Do pre-processing.
comp := load_comp_from_DB (compID);
Call the method on the component.
comp.method_A_imp ();
Do Post-processing
unload_comp_from_user_space (compID);
end
Component implementation
method_A_imp
method_B_imp

Delegado de negocio (Business Delegate)

- Problem
  - Accessing remote components differs significantly from accessing local components. Ideally, however, clients should not need to care whether the components they use are collocated or remote
- Solution
  - Introduce a business delegate for each remote component that can be created, used, and disposed of like a collocated component, and whose interface is identical to that of the component it represents
  - Let the business delegate perform all networking tasks transparently for clients using the component

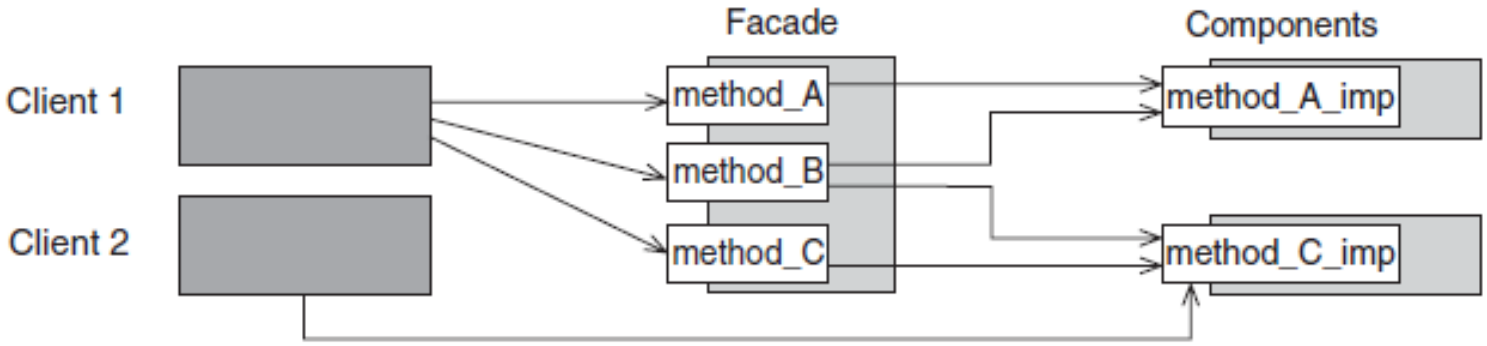
Delegado de negocio (Business Delegate)



Fachada
(Facade)

- Problem
  - Complex services are often provided by a group of components, each of which can offer its own self-contained services to clients.
  - If clients that want to invoke a complex service must maintain explicit relationships to each component in the group, however, they become dependent on the group's internal structure
- Solution
  - Specify a single point of access for the component group that mediates client requests to the appropriate components in the group

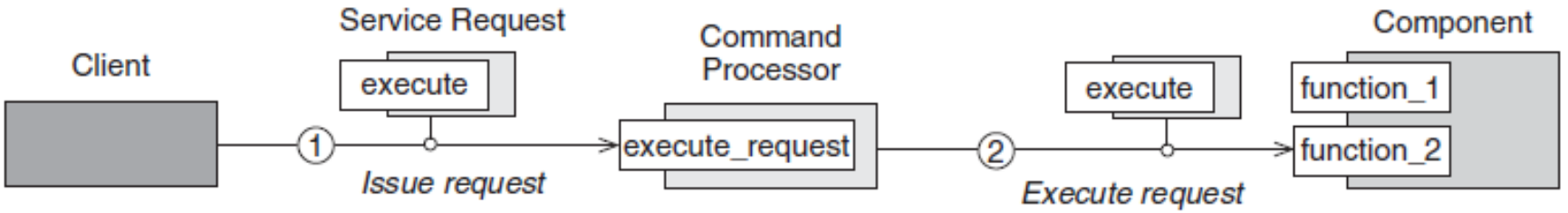
Fachada
(Facade)



Procesador de comandos (Command Processor)

- Problem
  - If an application can receive requests from multiple clients, they may need to manage the execution of these requests, for example to handle request scheduling, logging, and undo/redo
  - An individual client, however, generally has no knowledge about when and under what conditions its requests execute
- Solution
  - Introduce a command processor to execute requests to the application.
  - The command processor acts on behalf of the clients and within the constraints of the application

Procesador de comandos (Command Processor)

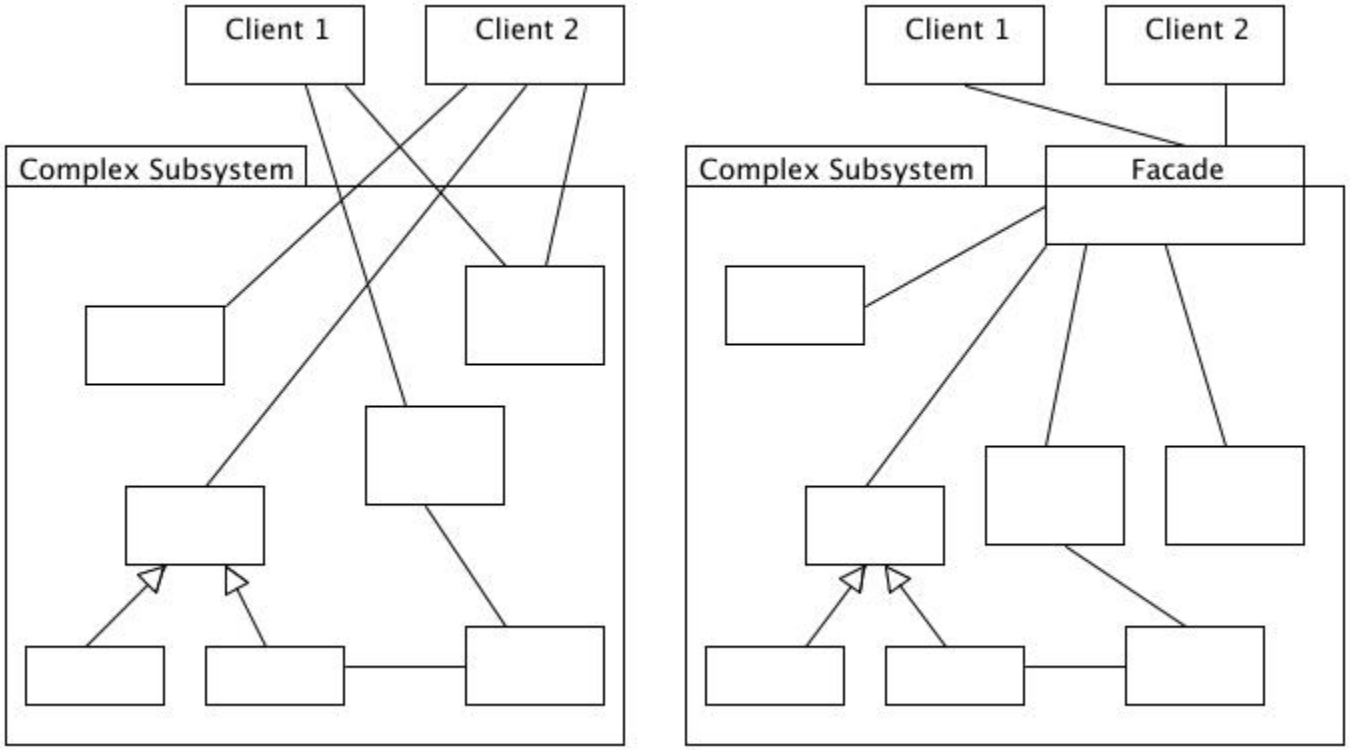


Agenda del día

- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- **Patrones de diseño detallado**

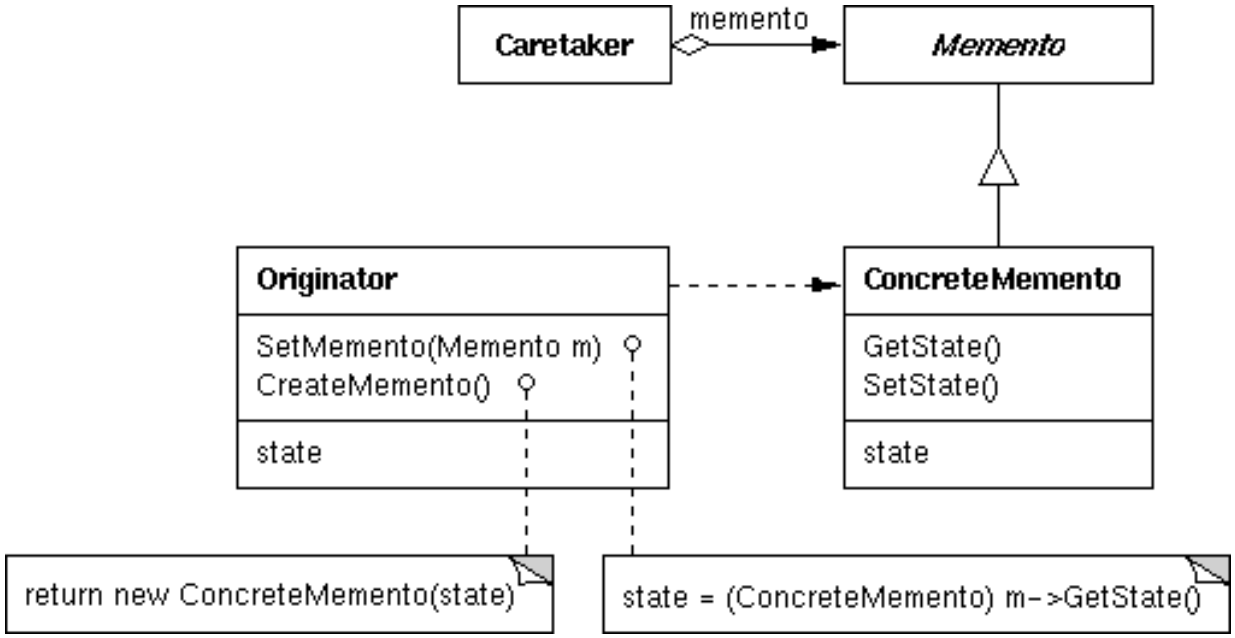
Patrón fachada (Facade)

- Simplifica el acceso a un conjunto de elementos por medio de un objeto que los encapsula y se comunica con ellos



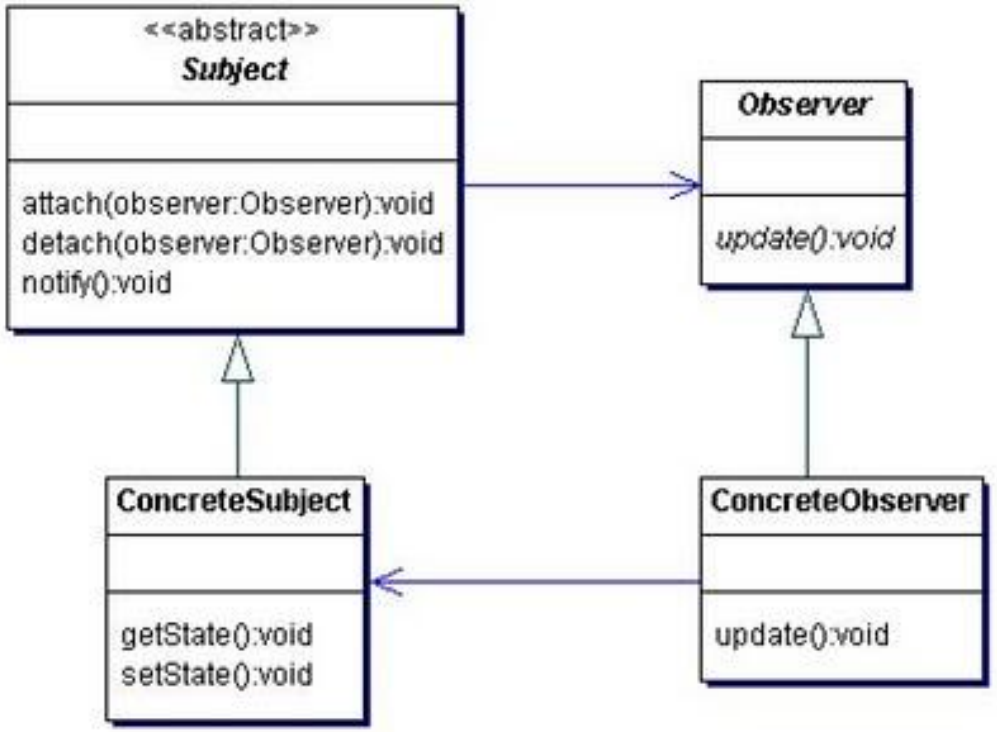
Patrón memento/snapshot

- Sin sacrificar encapsulamiento, captura y externaliza el estado interno de un objeto para que este pueda ser restaurado más adelante



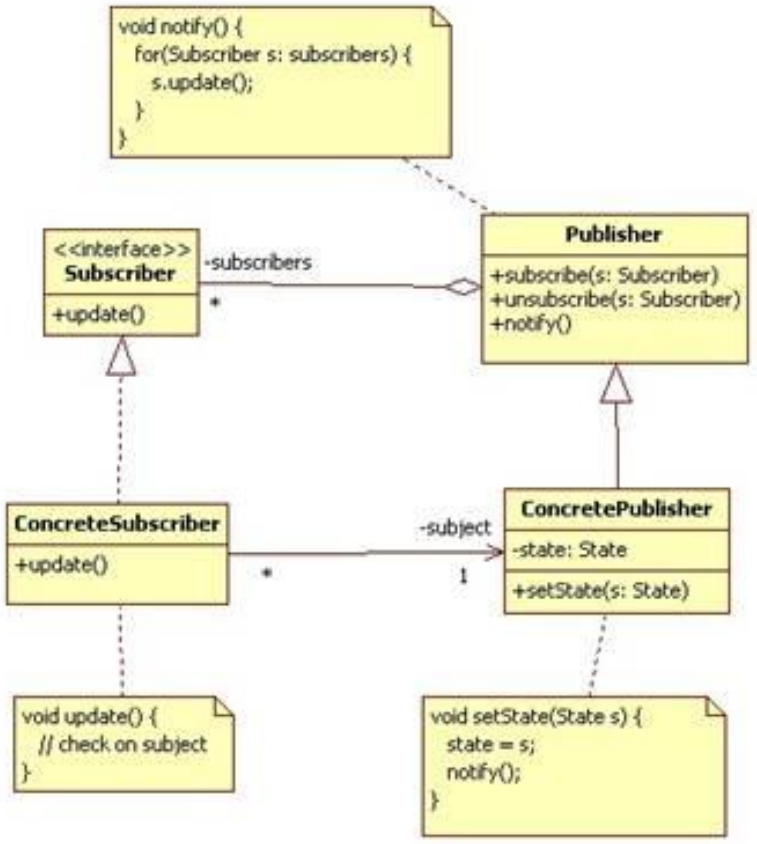
Patrón observador (Observer)

- Permite que los objetos registren dependencias entre ellos dinámicamente y la notificación de cambio de estado entre ellos.



Patrón publicador/subscriptor (Publisher/Subscriber)

- Desacopla los elementos que envían mensajes de los que reciben mensajes.



Universidad de los Andes

¿Preguntas?

Referencias

- Len Bass, Paul Clements, Rick Kazman. **Software Architecture in Practice**, Second Edition

- Len Bass, Paul Clements, Rick Kazman. **Aspectos Avanzados en Arquitectura de Software.** Universidad de los Andes, Curso de Verano 2010, Bogotá, Colombia

- Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. **Pattern-Oriented Software Architecture (POSA), A Pattern Language for Distributed Computing**. Volume 4. 2007