Ingeniería de Software I (CI3715)

Patrones de Diseño de Software

(Clase XIV)

Profesor:
Jean Carlos Guzmán
Mail:
soportepregradousb2014@gmail.com

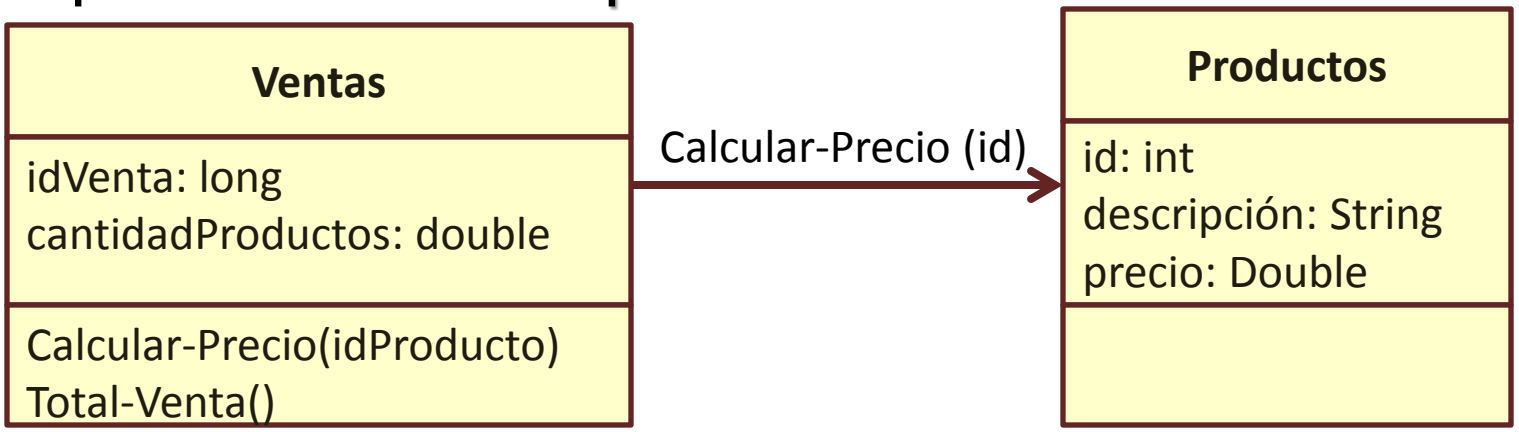
Sartenejas, Marzo 2017

Agenda I

- Estilos basados en eventos e invocación implícita:
  - Mediator
  - Publisher/Subscriber
- Estilos basados en Capas (layers):
  - Patrón Adapter
  - Patrón Bridge
  - Patrón Façade
- Patrones de Diseños Estructurales:
  - Patrón Decorator
  - Patrón Proxy

Estilos basados en eventos e invocación implícita

- En un sistema en el cual las interfaces de las componentes proporcionan un conjunto de procedimientos y funciones, tales como:
  - En sistemas O-O, las componentes interactúan por invocación explícita de estas rutinas.



Ejemplo de Invocación Explícita

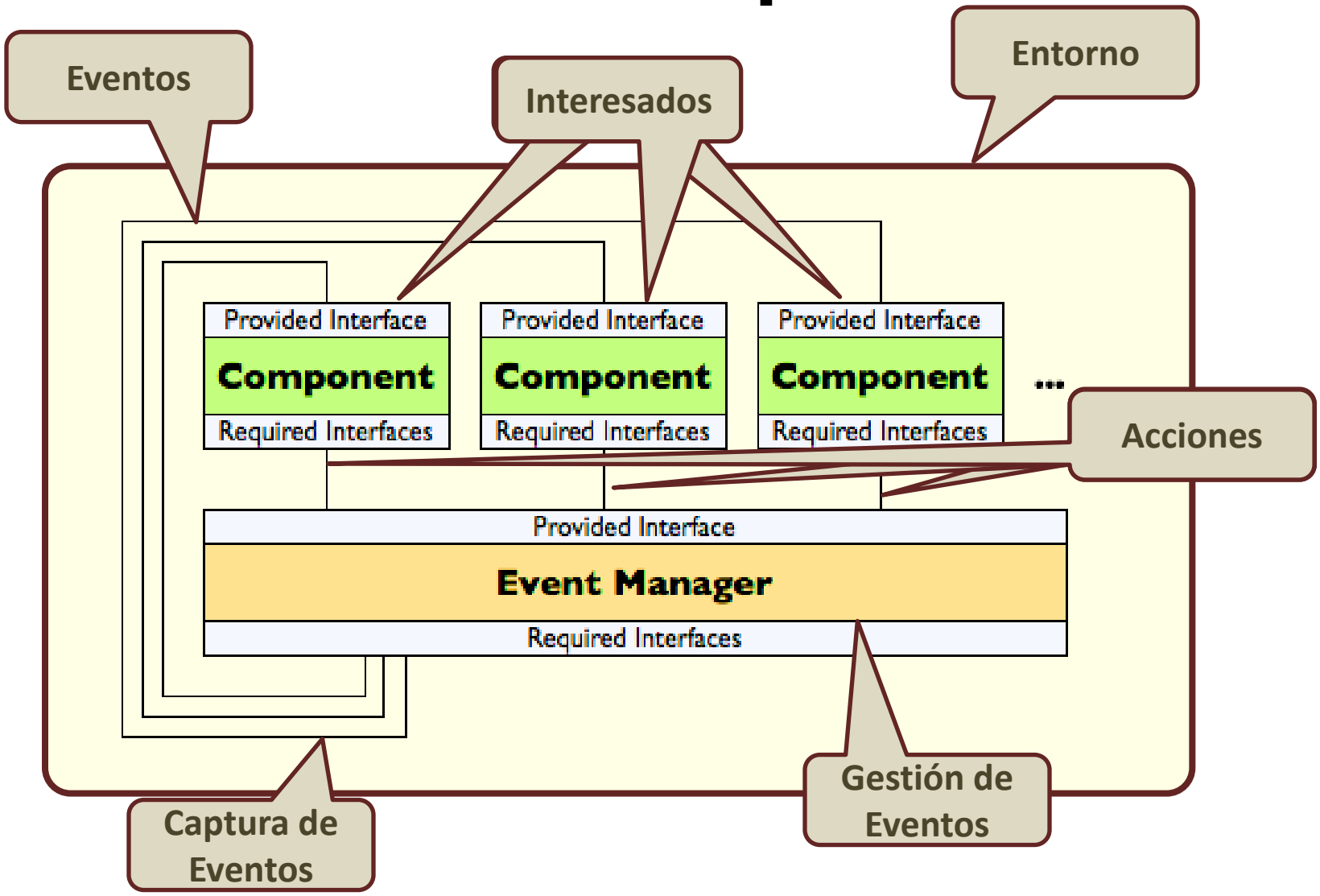
Estilos basados en eventos e invocación implícita

…

- Otra técnica de interacción se denomina invocación implícita, integración reactiva o broadcast selectivo:
  - Al producirse un evento el sistema invoca a todos los componentes registrados
  - El componente que anuncia los eventos no sabe a quien afectará

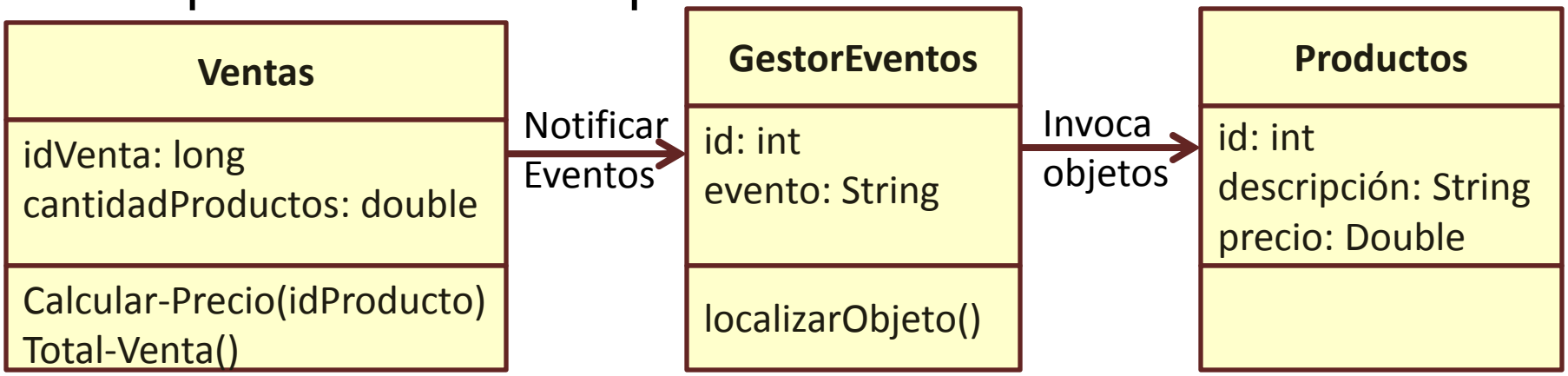
Este estilo es utilizado en los sistemas basados en mecanismos como actores y demonios

Estilos basados en eventos e invocación implícita



Estilos basados en eventos e invocación implícita

- La idea de la invocación implícita es la siguiente:
  - Un componente puede anunciar uno o más eventos, en lugar de invocar directamente un componente.
  - Otros componentes pueden registrar o manifestar un interés en alguno de esos eventos asociados con un procedimiento en particular.



Ejemplo de Invocación Implícita

Estilos basados en eventos e invocación implícita

- La idea de la invocación implícita es la siguiente:

  …

  – Cuando un evento es anunciado, el sistema mismo invoca todos los procedimientos que se han registrados para ese evento.
  – El anuncio de un evento implícitamente causa la invocación de los procedimientos presentes en otros módulos.

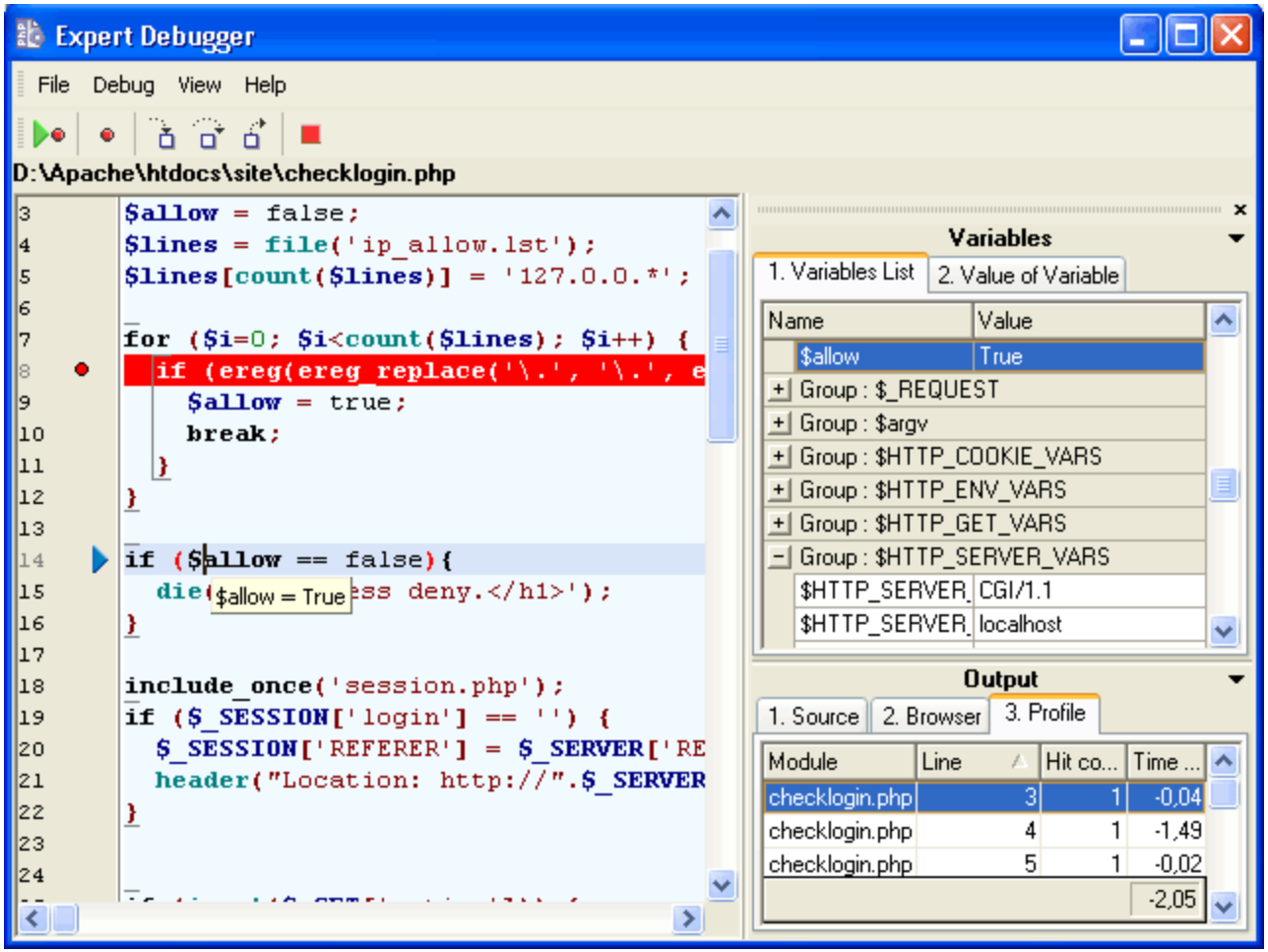
Estilos basados en eventos e invocación implícita

- Ejemplo:
  - Los editores programas registran puntos de parada (breakpoint) de un debugger.
  - Cuando un debugger se interrumpe en uno de esos puntos, anuncia un evento que hace que el sistema automáticamente invoque los procedimientos registrados por las herramientas.

Estilos basados en eventos e invocación implícita

- Ejemplo:
  - Estos procedimientos pueden hacer scrolling en el editor hasta ubicar la línea particular de código fuente donde ocurrió la interrupción, o hacer redisplay de las variables monitoreadas.
  - En este esquema, el debugger simplemente anuncia un evento, pero no conoce las acciones (si las hay) que las otras herramientas van a emprender para "manejar" ese evento.

Estilos basados en eventos e invocación implícita



Estilos basados en eventos e invocación implícita

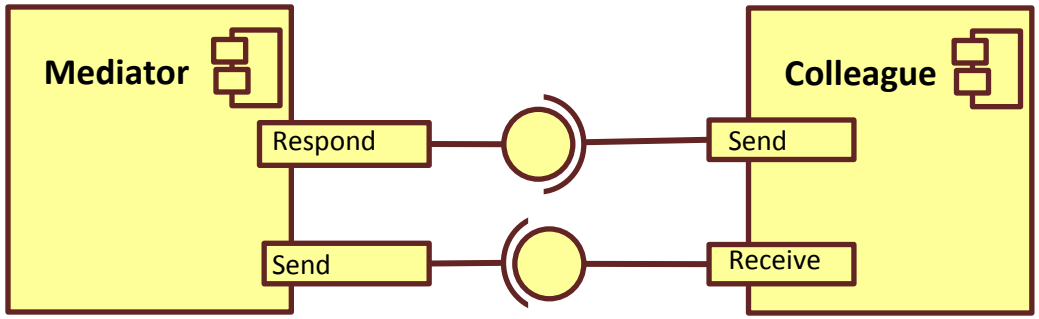
- Ejemplos de sistemas con estos mecanismos se utilizan en:
  - Integración de herramientas en ambientes de programación: Buildbot.
  - Sistemas manejadores de bases de datos para asegurar consistencia: SQL Server.
  - Interfaces usuario para separar la presentación de los datos de la aplicación que los maneja
  - Editores dirigidos por la sintaxis para soportar chequeo semántico: write, work, etc.

Patrones de Diseños representantes de este estilo

- Patrón Mediator:
  - Define un objeto que encapsula como un conjunto de objetos interactúa
  - Es un intermediario que proporciona un bajo acoplamiento, haciendo que los objetos dejen de referirse uno a otro explícitamente, dejando variar su interacción independientemente.
  - Es responsable de controlar y coordinar las interacciones de un grupo de objetos.
  - Los objetos solo conocen a su mediator, reduciendo el numero de interconexiones (complejidad).

Patrones de Diseños representantes de este estilo

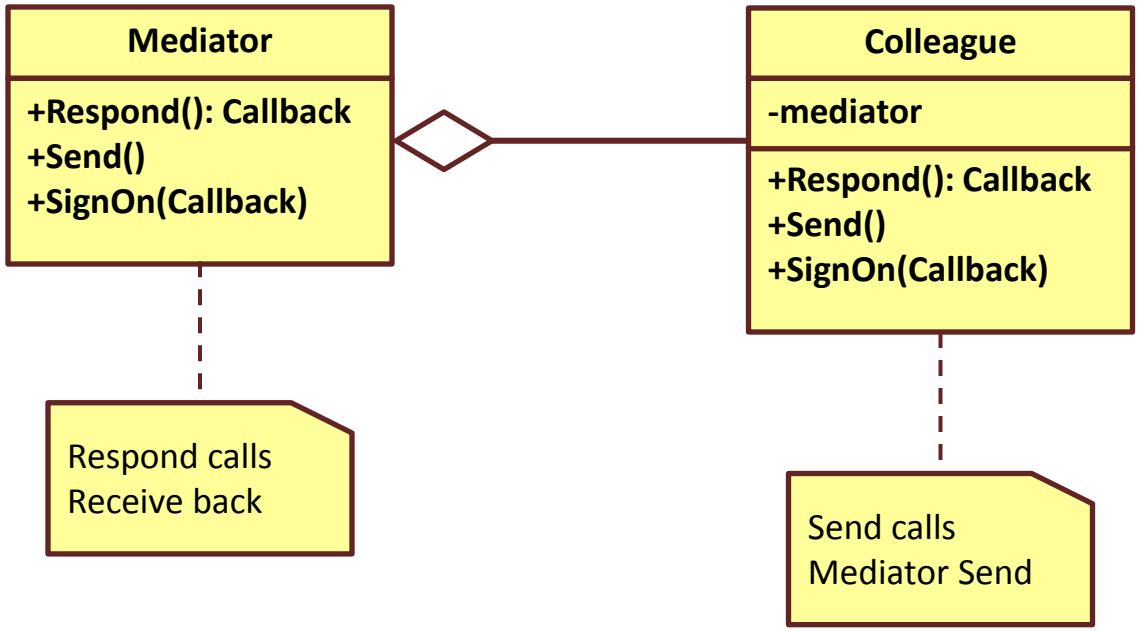
- Patrón Mediator:



Patrón Mediator expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños representantes de este estilo

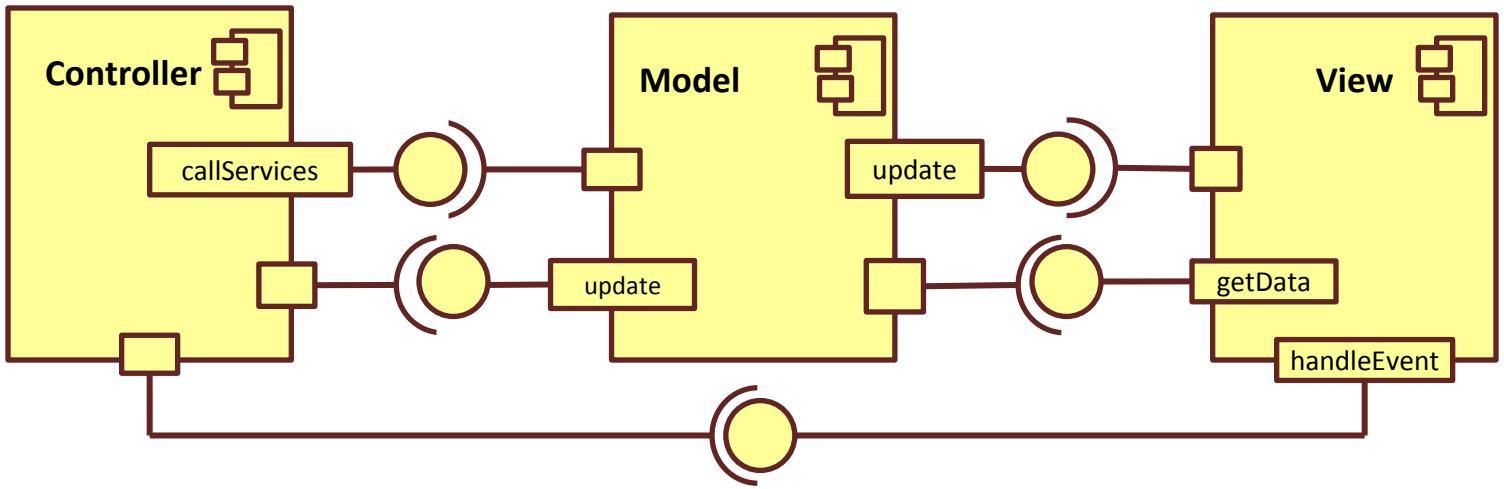
- Patrón Mediator:



Patrón Mediator expresado en el Diagrama de Clases UML 2.0

Patrones de Diseños representantes de este estilo

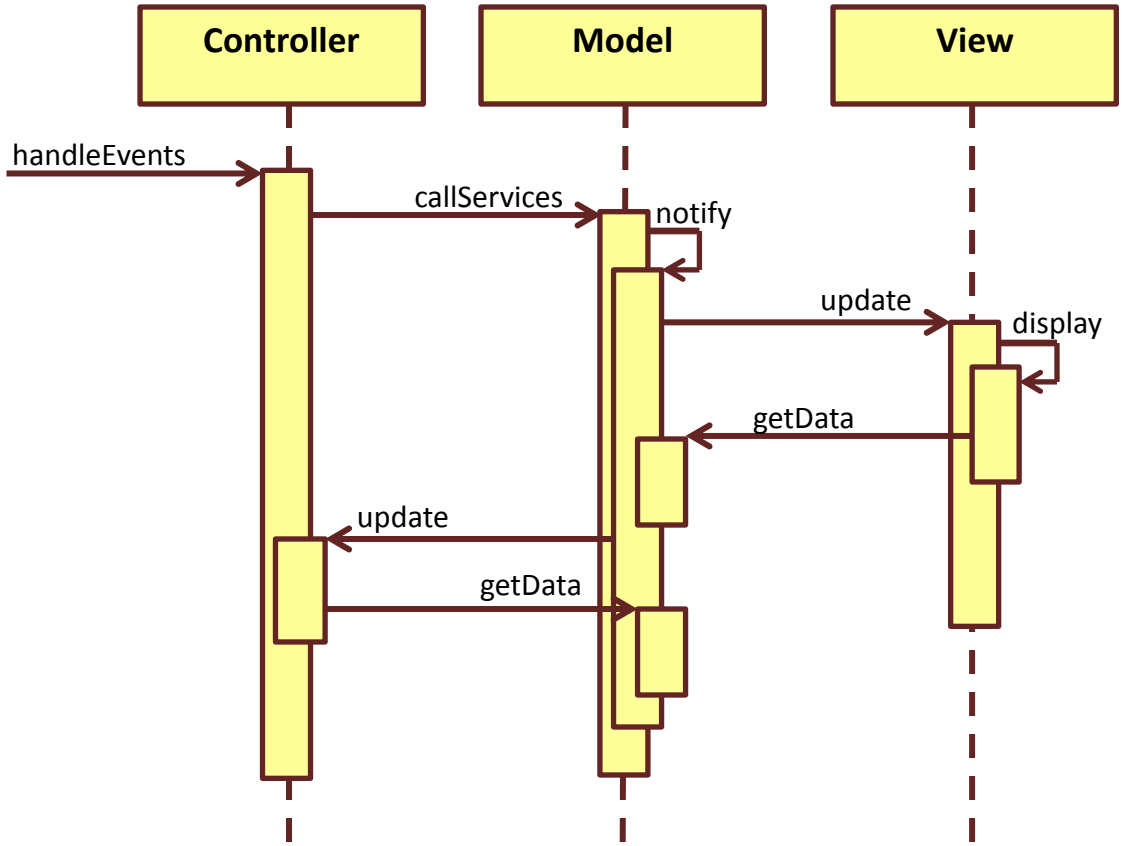
- Patrón MVC: variante de Mediator



Patrón MVC expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños representantes de este estilo

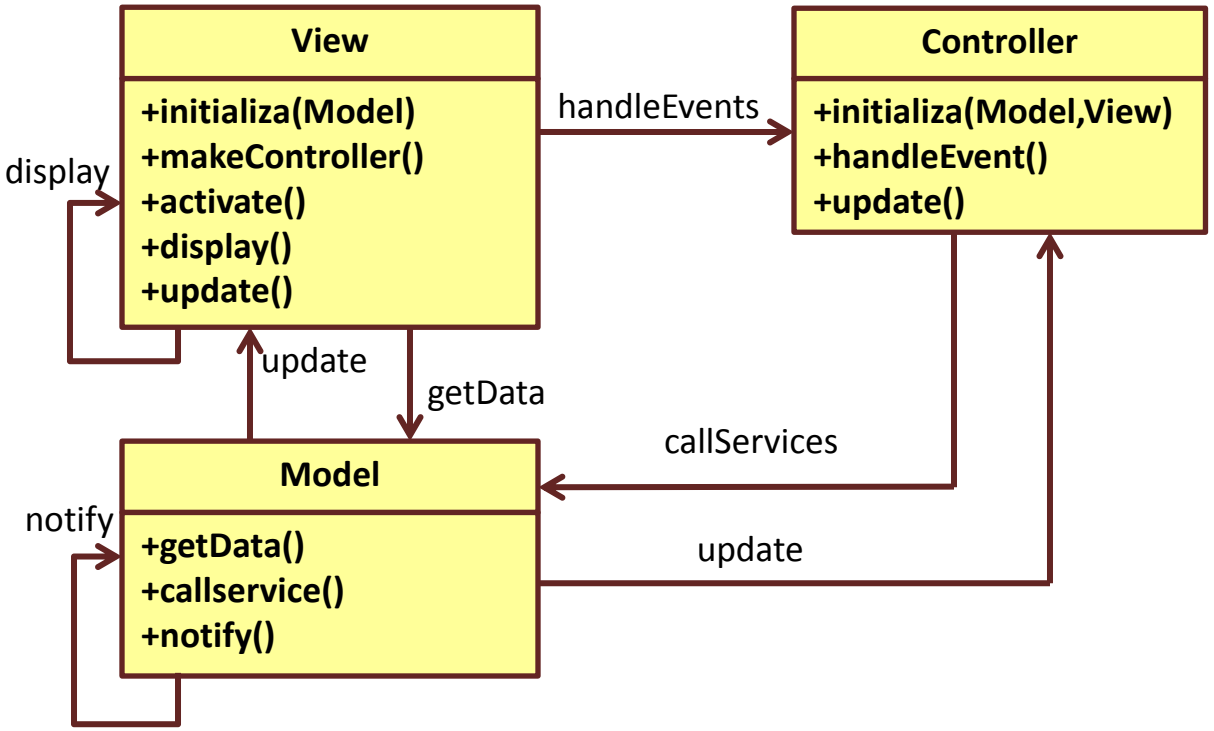
- Patrón MVC:



Patrón MVC expresado en el Diagrama de Secuencias UML 2.0

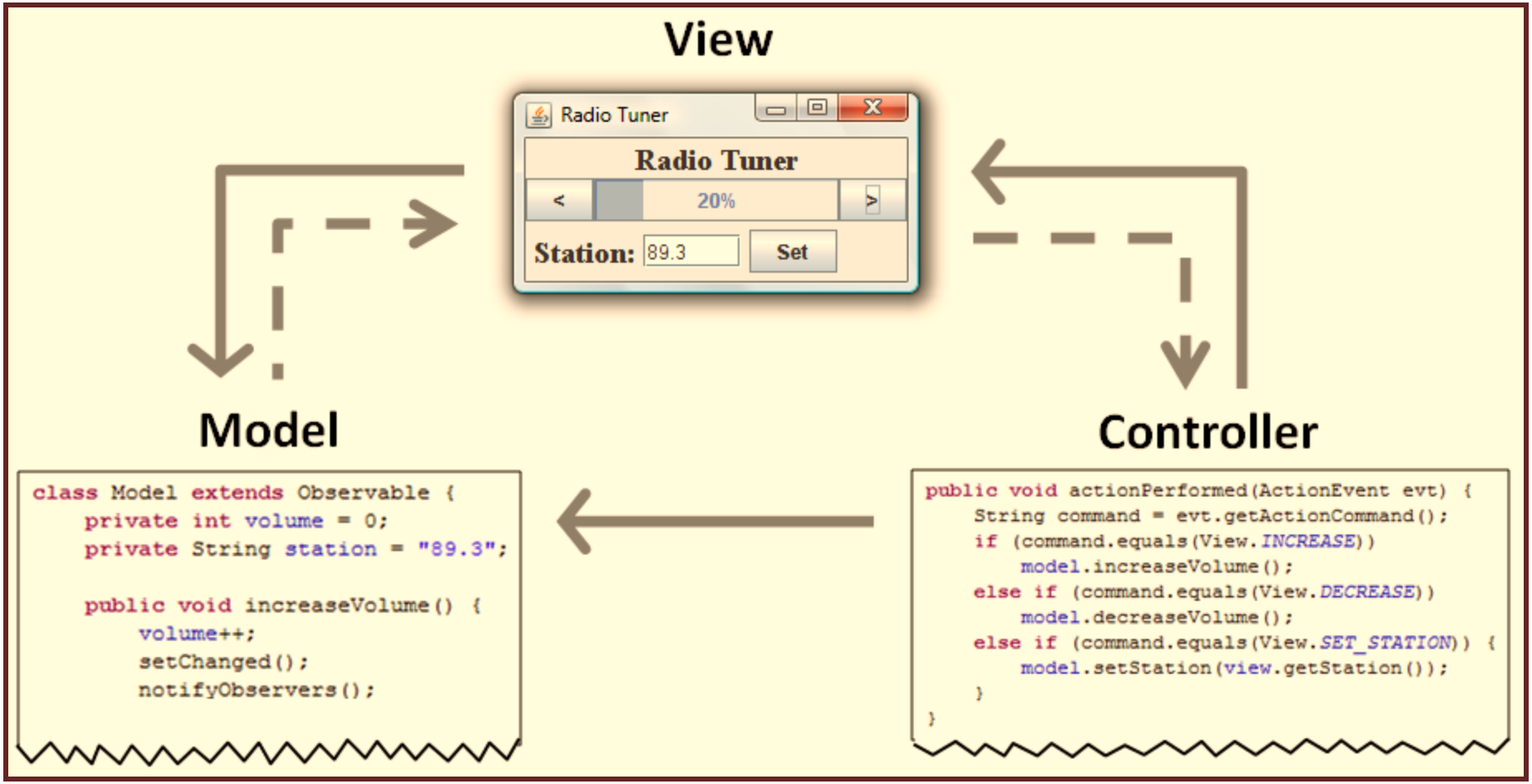
Patrones de Diseños representantes de este estilo

- Patrón MVC:



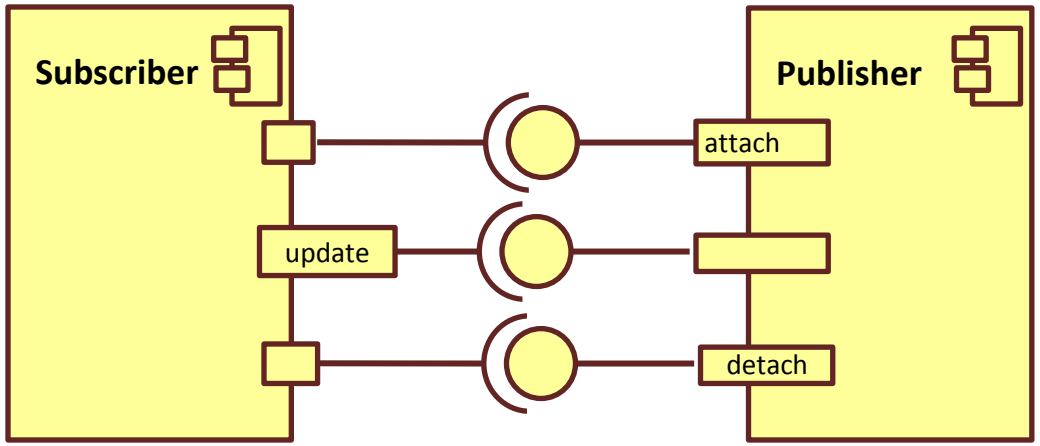
Patrón MVC expresado Clases UML 2.0

Patrones de Diseños representantes de este estilo



Patrones de Diseños representantes de este estilo

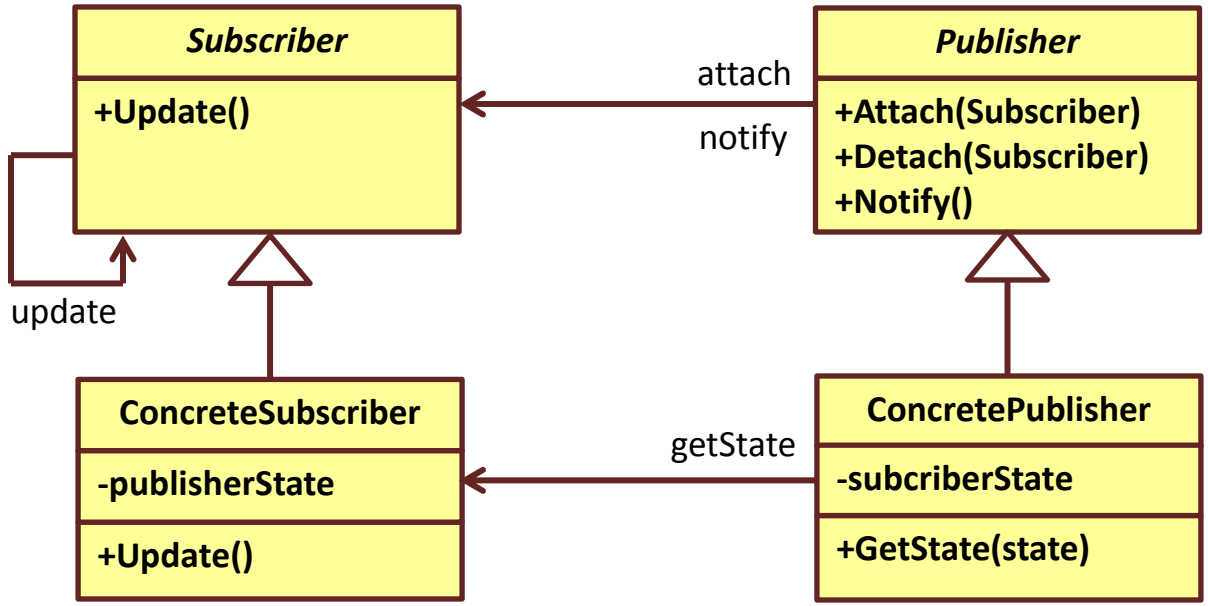
- Patrón Publisher/Subscriber (Subject/observer [GOF]):



Patrón Publisher/Subscriber expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños representantes de este estilo

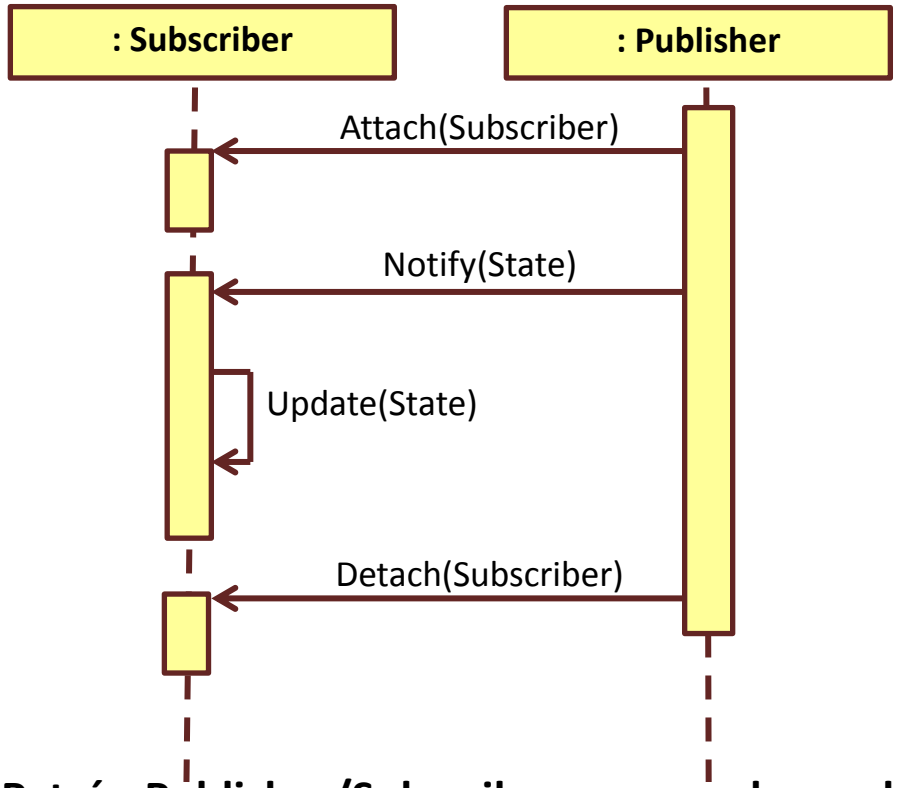
- Patrón Publisher/Subscriber (Subject/observer [GOF]):



Patrón Publisher/Subscriber expresado en el Diagrama de Clases UML 2.0

Patrones de Diseños representantes de este estilo

- Patrón Publisher/Subscriber (Subject/observer [GOF]):



Patrón Publisher/Subscriber expresado en el Diagrama de Secuencias UML 2.0

Patrones de Diseños representantes de este estilo

- Patrón Publisher/Subscriber :
  - Un publisher puede tener cualquier numero de subscribers
  - Todos los subscribers son notificados cada vez que el publisher cambia su estado.
  - El publisher puede solicitar a los subscribers que sincronicen sus estados, de acuerdo con el estado del publisher

Ventajas y Desventajas

Ventajas

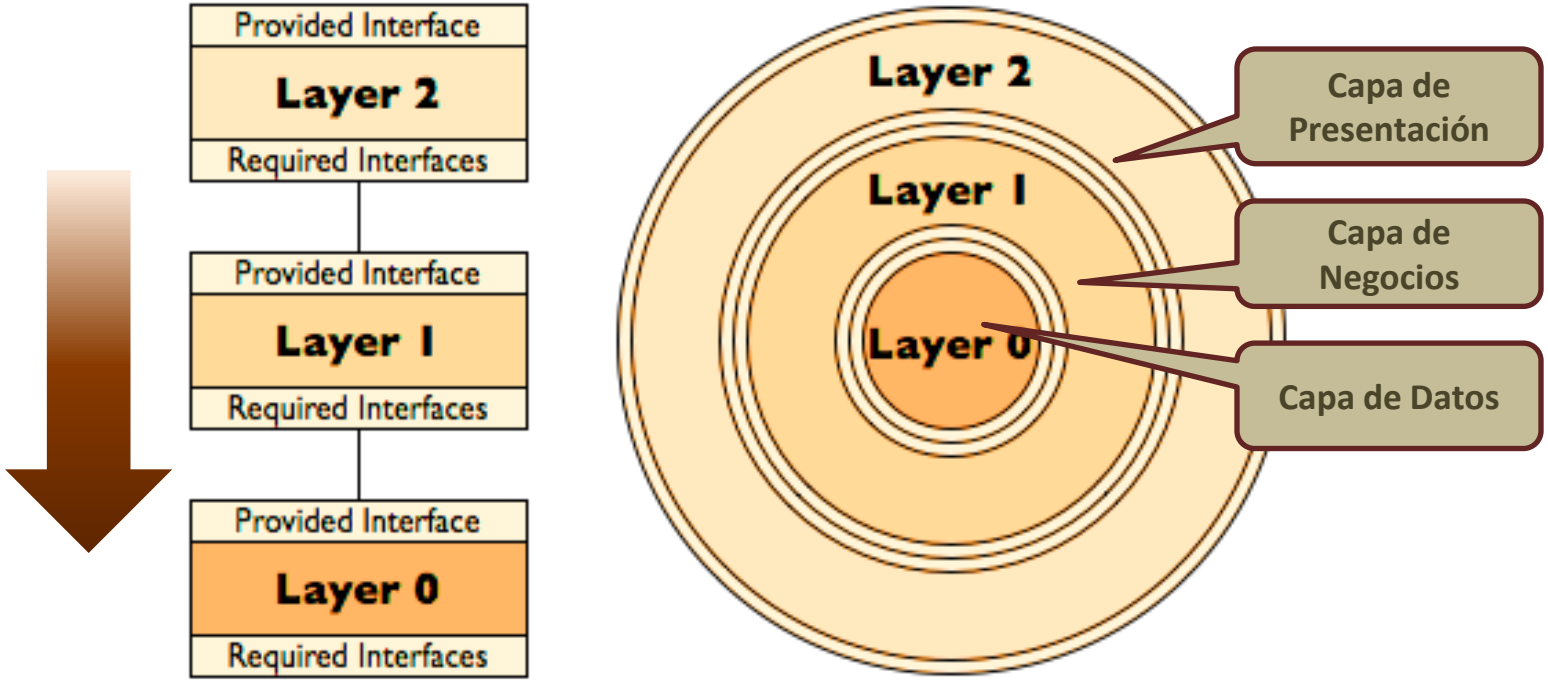
- Soporte para la reutilización: cualquier componente puede ser introducida, simplemente con registrarla para los eventos del sistema
- Ayuda a la extensión y el mantenimiento: componentes pueden ser reemplazadas sin afectar las interfaces de otros componentes en el sistema

Desventajas

- Problema con el control: Pérdida de control sobre las acciones realizadas por el sistema (respuesta).
- Intercambio de datos: En algunos casos se pueden pasar datos junto con un evento, pero en otros casos los eventos comparten un repositorio para la interacción (rendimiento)

Estilos basados en Capas (layers)

- Un sistema en capas está organizado jerárquicamente.
- Cada componente (capa) proporciona un servicio a la capa que está por encima de ella y sirve como cliente a la capa situada debajo de ella.



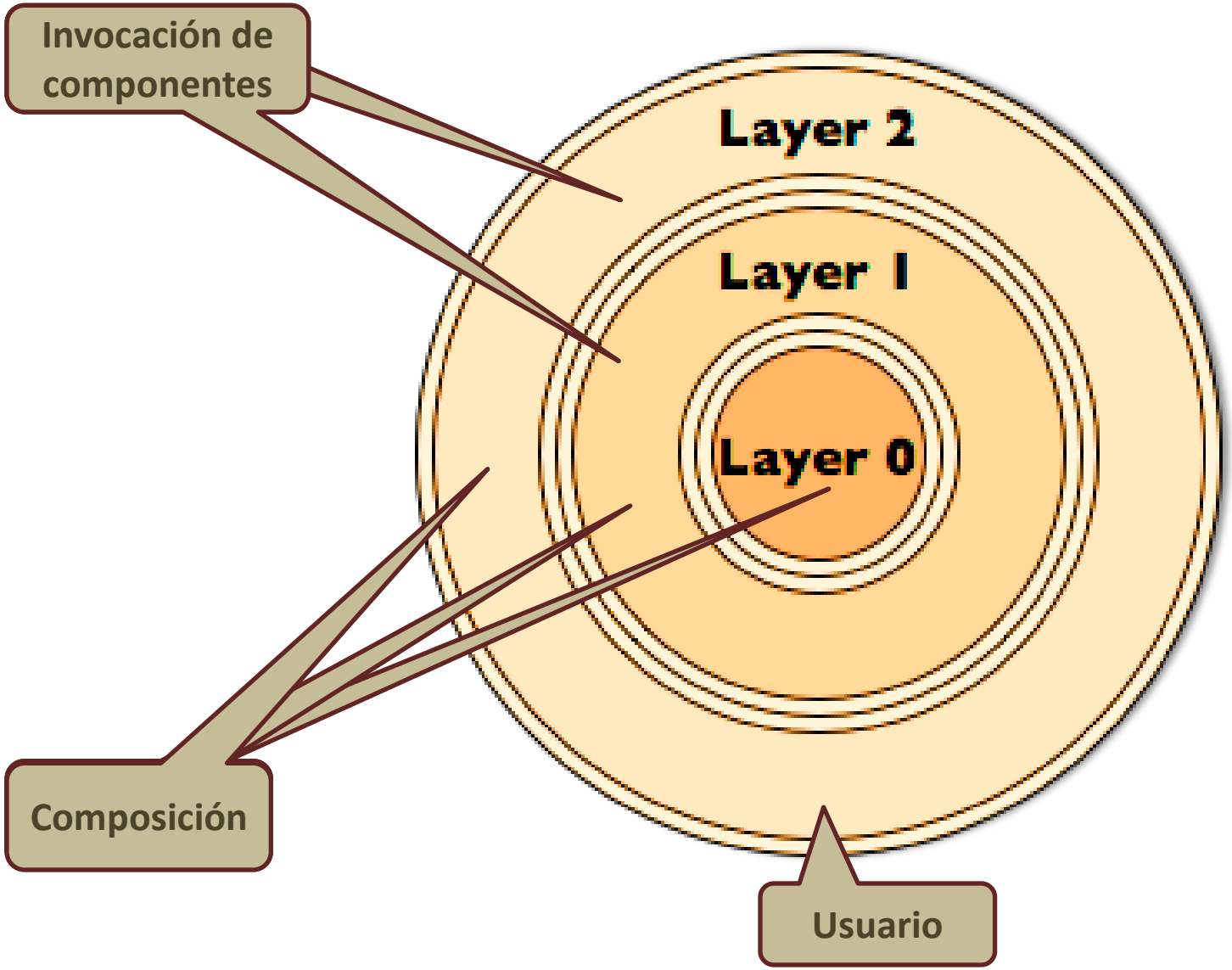
Estilos basados en Capas (layers)

- En algunos sistemas, las capas más internas están ocultas para las demás capas, salvo por la adyacente superior, exceptuando algunas funciones especiales de exportación.

Estilos basados en Capas (layers)

- En estos sistemas, las componentes implementan una máquina virtual en la jerarquía de capas:
  - Los conectores son definidos por protocolos, que determinan la interacción entre las capas.
- Las restricciones para alguna capa son limitaciones para la interacción solo de capas adyacentes.

Estilos basados en Capas (layers)



Estilos basados en Capas (layers)

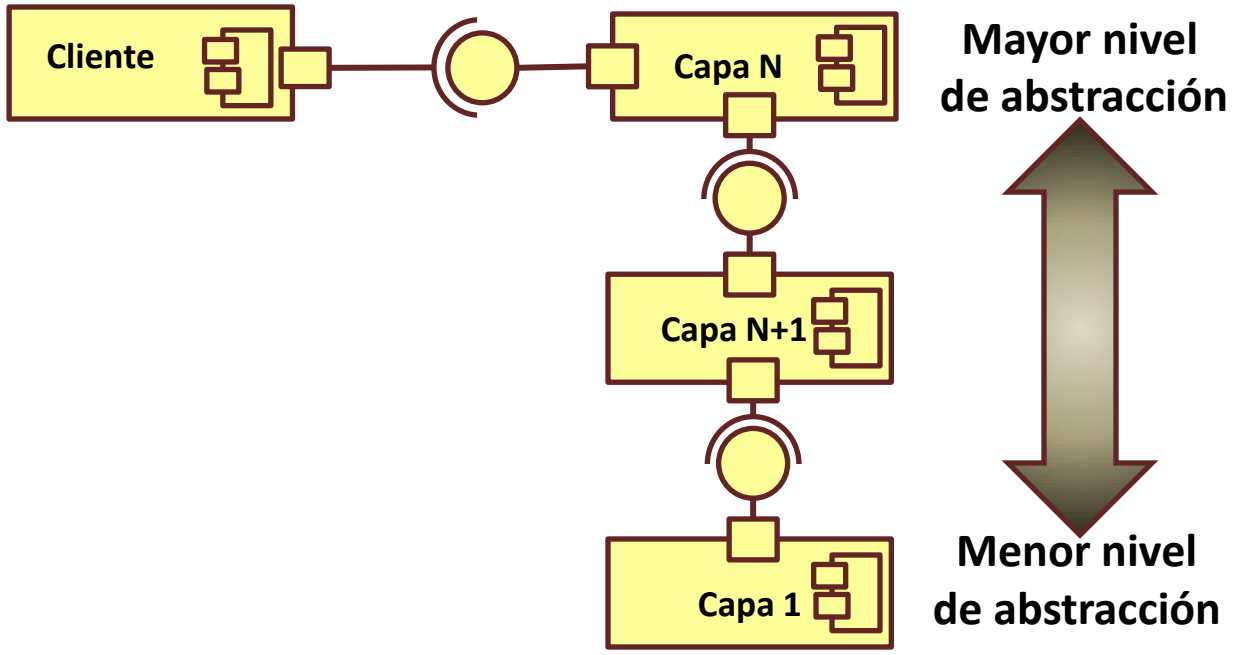
1. Protocolos de comunicación en capas:
   - Cada capa proporciona facilidades de comunicación en algún nivel de abstracción.
   - Los niveles inferiores definen los niveles de interacción inferiores.
   - El último nivel está definido por las conexiones con el hardware.

Estilos basados en Capas (layers)

2. Sistemas de bases de datos
3. Sistemas operativos
4. Ambientes integrados (herramientas CASE)

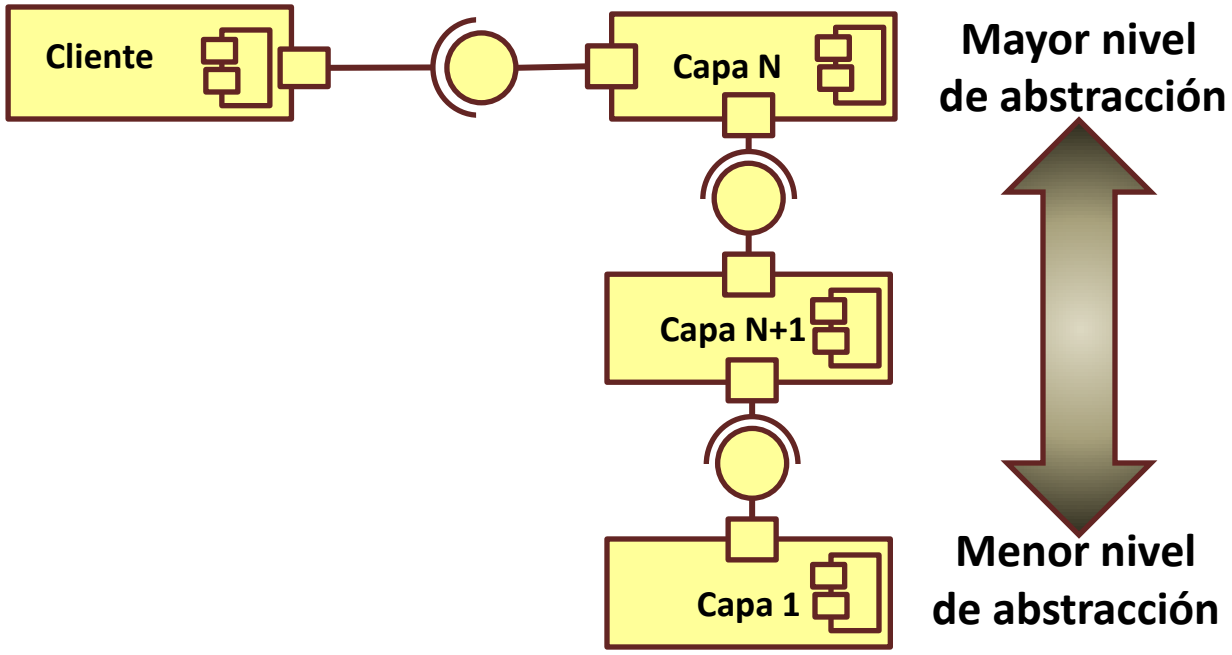
Estilos basados en Capas (layers)

- La característica estructural más importante de este estilo es que los servicios de la Capa N solo deben ser utilizados por la Capa N+1.
- No deben haber más dependencias directas entre capas.



Estilos basados en Capas (layers)

- Esta estructura puede ser comparada con la de una pila (stack).
- Cada capa individual protege a todas las capas inferiores de un acceso directo por capas superiores.



Estilos basados en Capas (layers)

- La componentes en cada capa pueden ser muy complejas.
  - Uno de los protocolos de comunicación más utilizados es el TCP/IP o “Internet Protocol suite”.
  - La International Standardization Organization (ISO) ha definido el siguiente modelo de arquitectura en capas (OSI-7)

Estilos basados en Capas (layers)

- Ejemplo: Modelo OSI-7 Capas

Estilos basados en Capas (layers)

- Ejemplo: Modelo TCP/IP

Estilos basados en Capas (layers)

- Aplicaciones de esta arquitectura:
  - Máquinas virtuales: los niveles o capas inferiores se conocen como máquinas virtuales, que aíslan los detalles de alto nivel de los detalles de bajo nivel del hardware.

Estilos basados en Capas (layers)

...

- Ejemplo:
  - La Máquina Virtual Java (JVM) define un formato de código binario.
  - El código escrito en lenguaje Java es traducido a código binario: es independiente de la plataforma (byte-code).
  - Es entregado a la JVM para ser interpretado.

Estilos basados en Capas (layers)

...

- Ejemplo:
  - La JVM es específica de la plataforma, hay implementaciones para diferentes sistemas operativos y procesadores.
    - Este proceso de traducción en dos pasos permite la generación de un código independiente de la palataforma

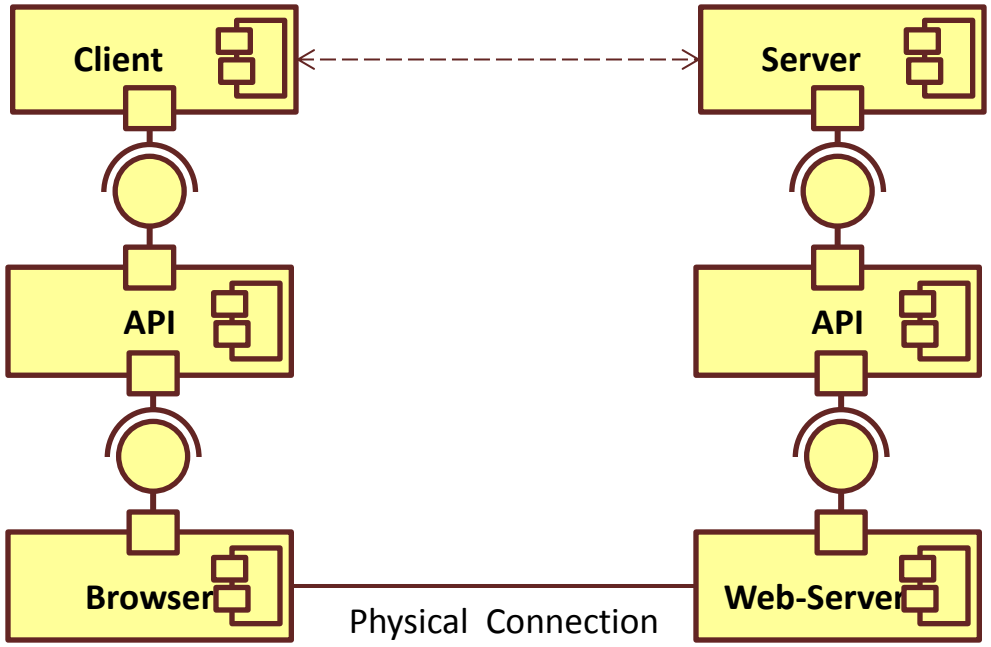
Estilos basados en Capas (layers)

...

- Ejemplo:
  - La producción de código no comprensible para los humanos (los bytecodes pueden ser transformados a representación ASCII, casi un código ensamblador OO.
  - Este código puede ser leído con cierta dificultad por el ser humano)

Estilos basados en Capas (layers)

- Aplicaciones de esta arquitectura:
  - APIs (Application Programming Interface): Es una capa que encapsula capas de bajo nivel de funcionalidades muy utilizadas.



Estilos basados en Capas (layers)

- Aplicaciones de esta arquitectura:

  …

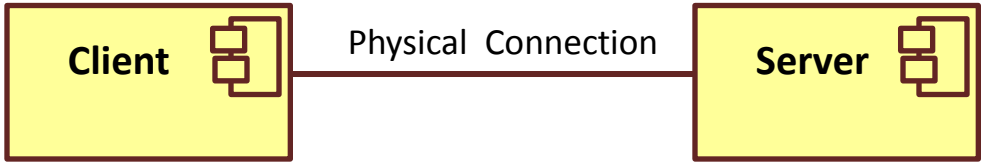
  – Es una colección "plana" de especificaciones de funciones: como los "systems calls" de Unix (archivos planos)
    - Plano significa que las llamadas, para tener acceso al sistema de archivos de Unix, no están separadas de las llamadas para la locación de memoria.
    - Sólo la documentación indica a que grupo pertenecen estas funciones.

Estilos basados en Capas (layers)

- Aplicaciones de esta arquitectura:
  - …
  - Por encima de las llamadas al sistema se encuentra la capa de las librerías standard de C: printf(), fopen().
    - Estas librerías son las que permiten la portabilidad entre diferentes sistemas operativos y proporcionan servicios adicionales de alto nivel, como buffering y formateo de salidas.

Estilos basados en Capas (layers)

- Aplicaciones de esta arquitectura:
  - Cliente-Servidor: posee dos niveles el cliente (App Win o Web) y el servidor (Servidores Web y Base de Datos).



Estilos basados en Capas (layers)

- Aplicaciones de esta arquitectura:

  …

  – En vista que la UI es de alto acoplamiento con las representaciones de datos:
    - Se introduce una tercera capa, la capa de negocio, que modela la lógica del negocio (reglas de negocios).

Estilos basados en Capas (layers)

- Aplicaciones de esta arquitectura:
  - Windows NT: Se estructura de acuerdo a un patrón arquitectónico denominado Microkernel, es la componente NT Executive.

Estilos basados en Capas (layers)

...

- Es un sistema flexible, que se estructura en las siguientes cinco capas (5):
  - Servicios del sistema, manejo de recursos, kernel, abstracción de hardware (oculta diferentes familias de procesadores) y hardware.
- La flexibilidad respecto a los sistemas de capas, consiste en que el kernel y el I/O manager (en la capa de recursos) accede al hardware directamente por razones de eficiencia.

Ventajas y Desventajas

Ventajas

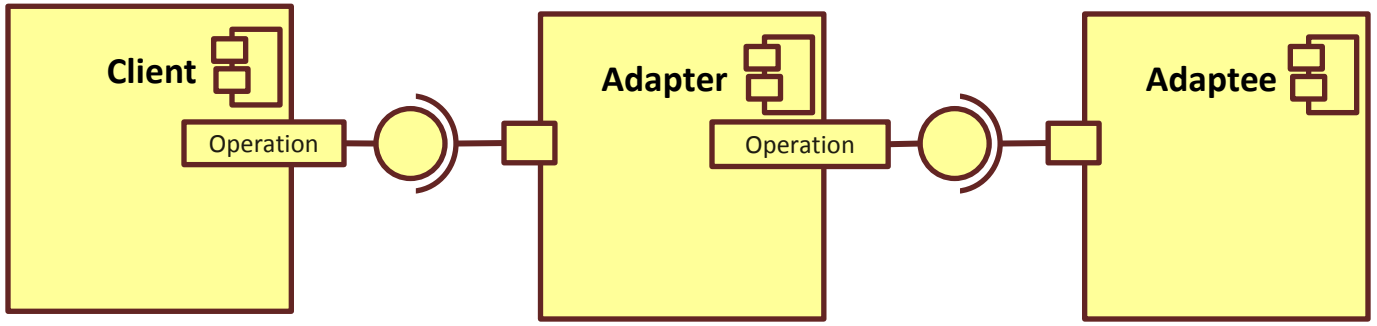
- Soportan diseños basado en niveles de abstracción crecientes.
- Soportan extensión: los cambios solo afectan a las capas adyacentes que interactúan entre si.
- Soportan la reutilización: permiten que diferentes implementaciones de la misma capa puedan ser usada indistintamente (son estándar), siempre que ofrezcan la misma interfaz a las capaz adyacentes.

Desventajas

- Desde el punto de vista lógico, no todos los sistemas se pueden estructurar fácilmente en capas.
- Aunque esto se logre, consideraciones de rendimiento pueden requerir un acoplamiento entre funciones de alto nivel y sus implementaciones del nivel más bajo en la jerarquía.
- Además, puede ser difícil encontrar el nivel de abstracción correcto.

Patrones de Diseños representantes de este estilo

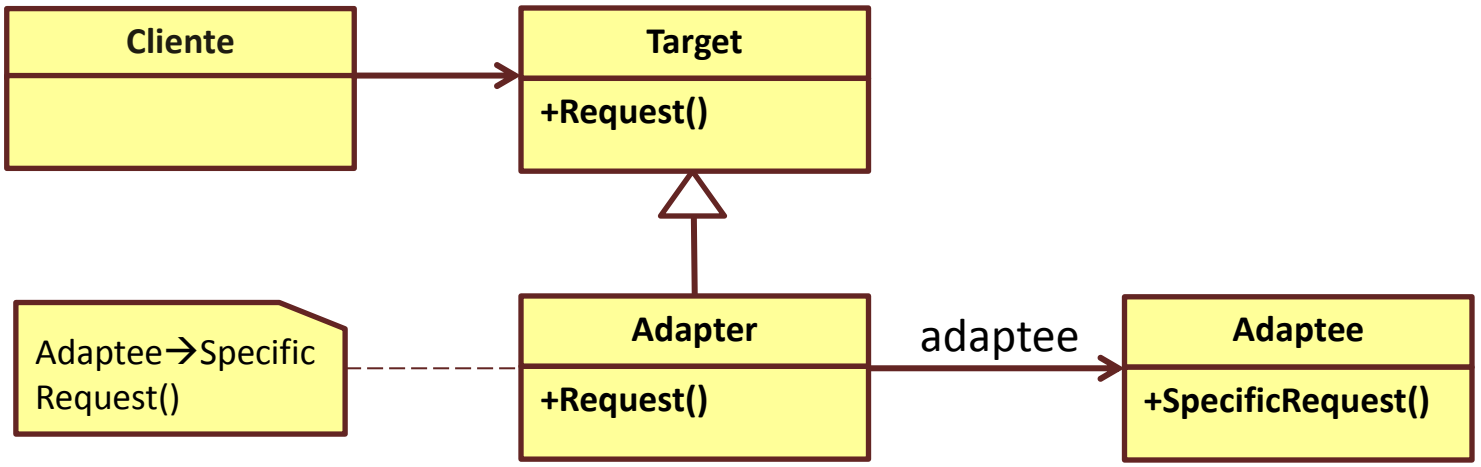
- Patrón Adapter: actúa como intermediario entre dos clases, convirtiendo la interfaz de una clase en otra interfaz que los clientes esperan.



Patrón Adapter expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños representantes de este estilo

- Patrón Adapter:



Patrón Adapter expresado en el Diagrama de Clases UML 2.0

Patrones de Diseños representantes de este estilo

- Participantes:
  - *Cliente*: usa el adaptador.
  - *Target*: interfaz que el cliente utiliza.
  - *Adapter*: clase que implementa la interfaz Target en términos de SpecificRequest. Permite que el *Adaptee* específico se cargue dinámicamente al momento de ejecución.
  - *Adaptee:* implementación que necesita la adaptación:

Patrones de Diseños representantes de este estilo

- Aplicabilidad:
  - Usar un objeto en un ambiente que espera una interfaz diferente a la del objeto.
  - Traducción de interfaces entre múltiples fuentes
  - Un objeto debe actuar como intermediario para un grupo de clases y no es posible anticipar cual es esa clase.

Sistemas de software que utilizan varios tipos de RDBMS:
SQLAlchemy en python: SQL Server, Oracle, Postgre SQL, etc.

Patrones de Diseños representantes de este estilo

- Variantes:
  - Multi-Adaptee-Adapters: puede ser conveniente que el Adapter sea parte del cliente. Un Adapter de este tipo frecuentemente actúa como intermediario entre el sistema y multiples Adaptee.
  - Adapters no interfaces: el uso de una interfaz no siempre es posible (Java), dado que los componentes que no implementan interfaces, son menos flexibles.

Patrones de Diseños representantes de este estilo

- Variantes:

  …

  – Una capa interfaz entre el Invocador-Adapter y otra entre Adapter-Adaptee:
    - Entre Invocador-Adapter permite añadir nuevos Adapter durante ejecución.
    - Entre Adapter-Adaptee permite cargar dinámicamente Adaptee.
    - La combinación permite desarrollar diseños intercambiables de Adapter con Adaptee modificables en un sistema en ejecución.

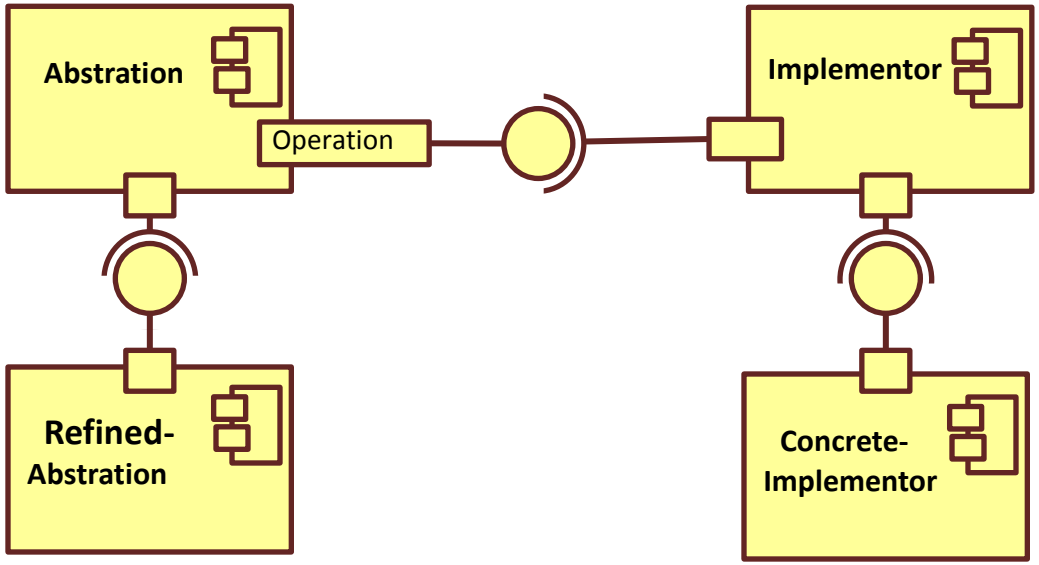
Patrones de Diseños representantes de este estilo

- Usos conocidos:
  - Invocación de librerías externas: python
  - Reutilización de código no O-O
  - Restricción de acceso a clases
  - En middleware:
    - Integración de servicios (persistencia, transacciones)
    - Interoperabilidad con otros middleware

Middleware: software que conecta componentes o aplicaciones para que puedan intercambiar datos entre éstas. Es utilizado a menudo para soportar aplicaciones distribuidas.

Patrones de Diseños representantes de este estilo

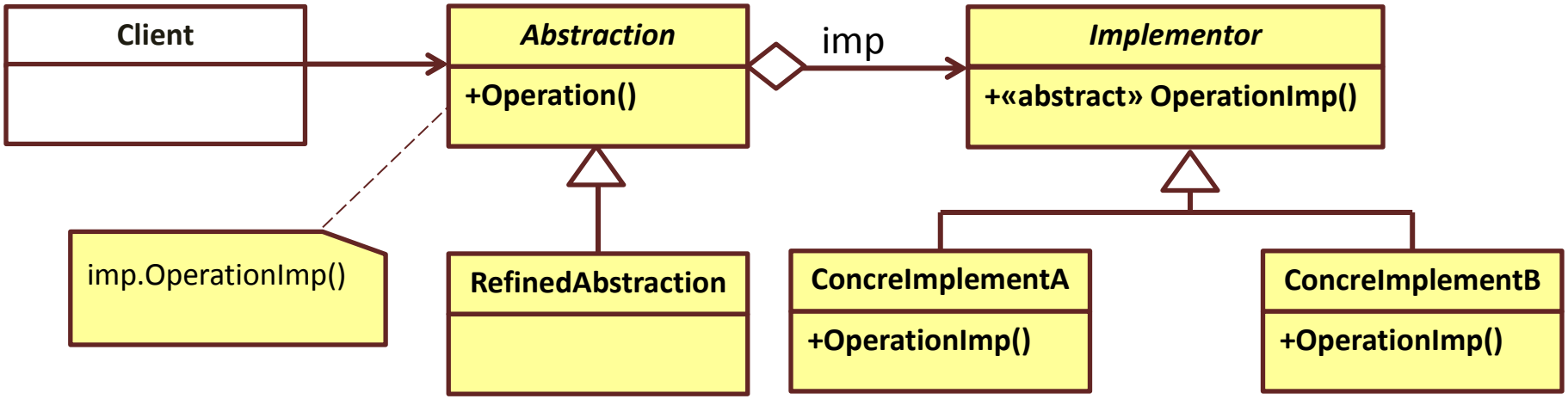
- Patrón Bridge: desacopla una abstracción de su implementación para que ambos puedan variar independientemente.



Patrón Bridge expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños representantes de este estilo

- Patrón Bridge: divide un componente complejo en dos jerarquías separadas pero relacionadas:
  - La abstracción funcional y la implementación.
  - Esto simplifica modificar estos aspectos del componente.



Patrón Bridge expresado en el Diagrama de Clases UML 2.0

Patrones de Diseños representantes de este estilo

- Participantes:
  - Abstraction: define la abstracción funcional con comportamiento y estructura estándar.
    - Tiene referencia a una instancia de la implementación.
  - RefineAbstraction: extiende Abstraction y suministra comportamiento adicional .
  - Implementor: interfaz de las funcionalidade3s usadas por instancias de Abstraction
  - ConcreteImplementor: implementa la interfaz y provee comportamiento y estructura de clases de implementación.

Patrones de Diseños representantes de este estilo

- Aplicabilidad:
  - Se requiere evitar relaciones estáticas entre la abstracción y su implementación.
  - Se identifican varias abstracciones e implementaciones.
  - Un cambio en la implementación de una abstracción no tenga impacto sobre el cliente.
  - Se tiene una proliferación de clases (relación de generalización anidada)

Patrones de Diseños representantes de este estilo

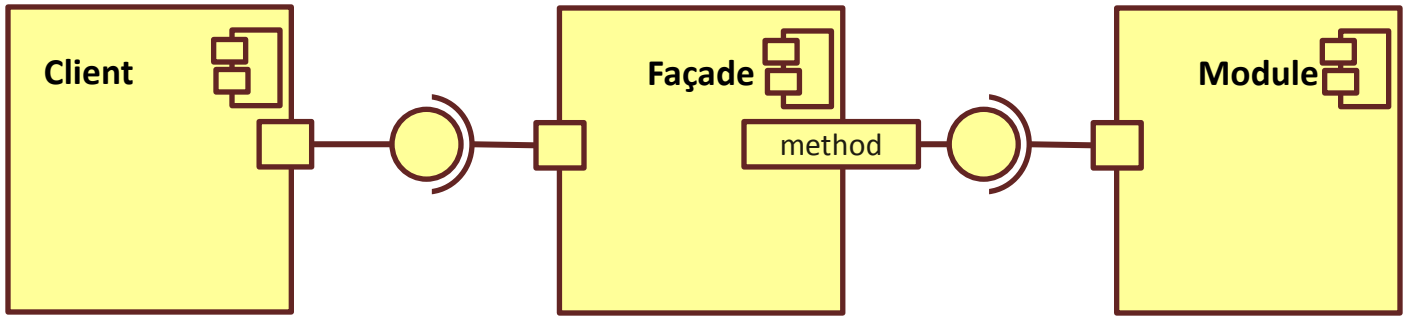
- Variantes:
  - Bridge automatizado – variar la implementación sin que el usuario final interactúe, solo depende de la aplicación o de la plataforma.
  - Implementaciones compartidas – particularmente si las clases son sin estado. Pueden compartirse entre múltiples objetos de la aplicación.
  - Una implementación – que le da servicio a múltiples clases de abstracción.

Patrones de Diseños representantes de este estilo

- Variantes:
  - Escojer Implementor puede hacerse con Abstract Factory
  - Implementors sin estado pueden ser compartidos (Abstract Factory se convierte en un Flyweight Factory)

Patrones de Diseños representantes de este estilo

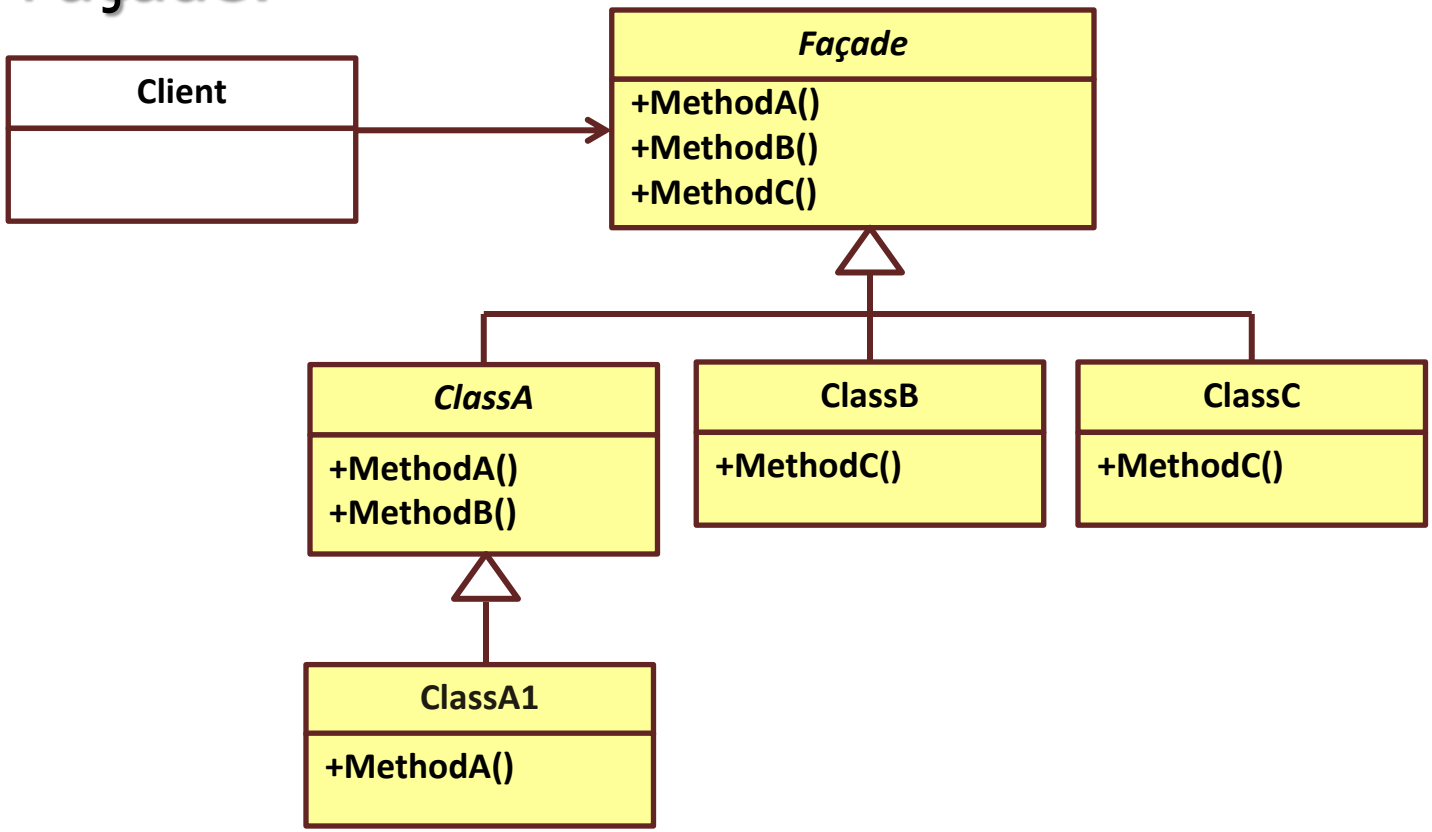
- Patrón Façade: provee una interfaz unificada a un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace al subsistema fácil de usar.



Patrón Facade expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños representantes de este estilo

- Façade:



Patrón Façade expresado en el Diagrama de Clases UML 2.0

Patrones de Diseños representantes de este estilo

- Participantes:
  - Façade: Esta clase contiene el conjunto de funciones simples que se ponen a disposición del cliente y que ocultan las complejidades de los subsistemas difíciles de usar.
  - Class: Estas clases contienen la funcionalidad que se ofrece a través de la fachada.

Patrones de Diseños representantes de este estilo

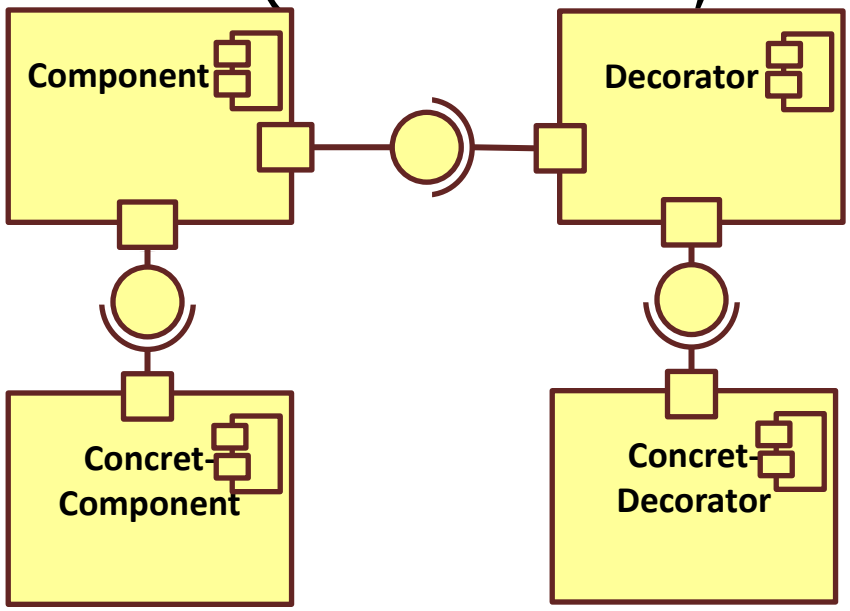
- Aplicabilidad:
  - Se requiere proveer un interfaz simple a un subsistema complejo.
  - Reducir el acoplamiento entre cliente y subsistema.
  - Definir un punto de entrada a cada nivel del subsistema.

Patrones de Diseños representantes de este estilo

- Variantes:
  - Reducir acoplamiento entre clientes y subsistemas con una Facade abstracta o interfaz.
  - Varios Facade que puedan proveer diferentes interfaces sobre el mismo conjunto de subsistemas.

Patrones de Diseños Estructurales

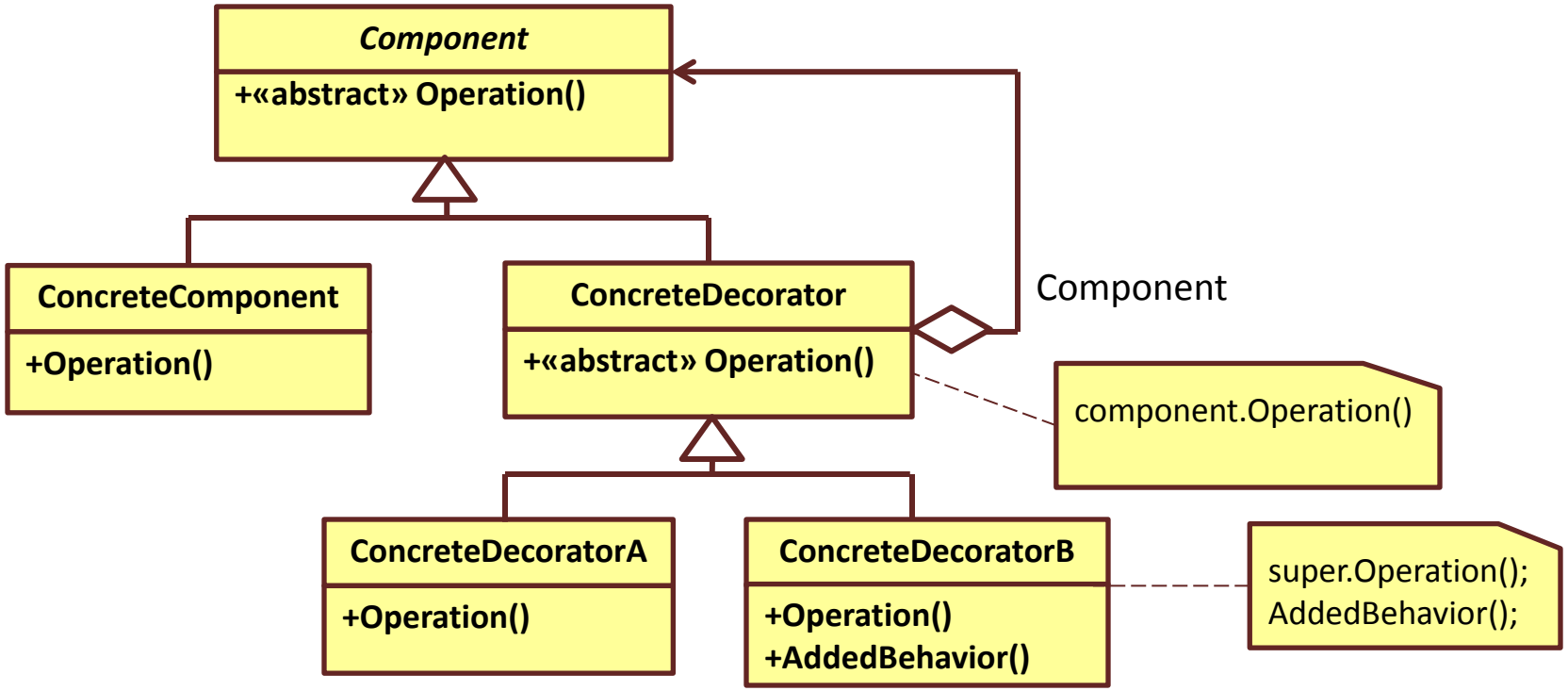
- Patrón Decorator: agrega responsabilidades adicionales a un objeto dinámicamente. Provee una alternativa flexible a subclases para agregar o remover funcionalidades sin alterar su apariencia externa o función (Gamma al et.,1995).



Patrón Decorator expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños Estructurales

- Patrón Decorator:



Patrón Decorator expresado en el Diagrama de Clases UML 2.0

Patrones de Diseños Estructurales

- Participantes:
  - Component: Es una clase base abstracta para todos los componentes y decoradores concretos.
  - ConcreteComponent: Esta clase hereda de Component. Puede haber varias clases de componentes concretas, cada una definiendo un tipo de objeto que puede ser envuelto por los decoradores.
  - Decorator: Representa la clase base abstracta para todos los decoradores. Añade un constructor que acepta un objeto Component como parámetro.
  - ConcreteDecorator: Esta clase representa el decorador concreto para un componente. Puede incluir algunos métodos adicionales que amplían la funcionalidad de los componentes.

Patrones de Diseños Estructurales

- Aplicabilidad:
  - Incorporar responsabilidades a un objeto dinámicamente.
  - Hay varias características independientes y pueden ser usadas en distintas combinaciones sobre un componente.

Patrones de Diseños Estructurales

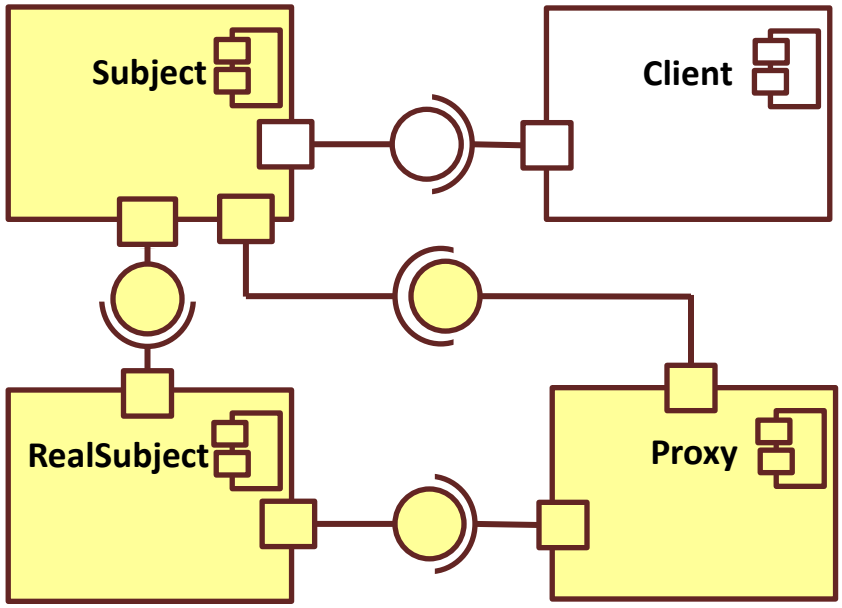
- Variantes:
  - Omitir la clase abstracta Decorator cuando solo exista una posible variación para el componente.
  - Clases Decorator con referencia hacia adelante y atrás para facilitar remover instancias.
  - Redefinición del comportamiento de decoradores.

Patrones de Diseños Estructurales

- Usos conocidos:
  - Embellecer Documentos
    - Background, Bordes, Notas, ...
  - Streams
    - Encrypted, Buffered, Compressed

Patrones de Diseños Estructurales

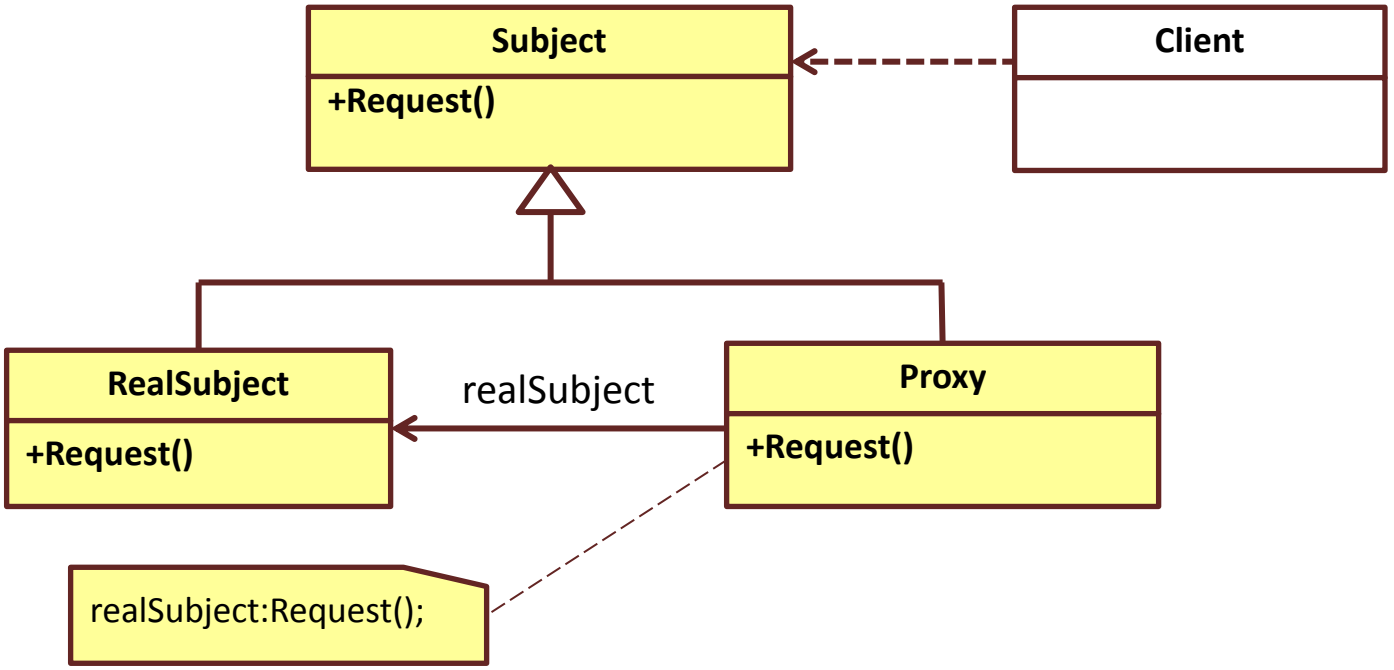
- Patrón Proxy: provee un sustituto de un objeto para controlar el acceso a él. Suministra un representante de un objeto por razones de seguridad, acceso, eficiencia (Gamma al et.,1995).



Patrón Proxy expresado en el Diagrama de Componentes UML 2.0

Patrones de Diseños Estructurales

- Patrón Proxy:



Patrón Proxy expresado en el Diagrama de Clases UML 2.0

Patrones de Diseños Estructurales

- Participantes:
  - Subject: Interfaz (o clase abstracta) implementada por el RealSubject y que representa sus servicios. La interfaz debe ser implementada por el proxy, puede ser utilizado en cualquier lugar donde el RealSubject sea utilizado.
  - RealSubject: Representa una clase compleja o clase de recurso sensible que deseamos utilizar de una manera efictiva.
  - Proxy: Hace referencia al objeto RealSubject. La aplicación ejecuta los métodos de clase proxy que son transferidos a los métodos del RealSubject.

Patrones de Diseños representantes de este estilo

- Aplicabilidad:
  - Proxy remoto: provee un representante local de un objeto en otro espacio de dirección.
  - Proxy virtual: post-poner la creación de objetos expansivos en demanda.
  - Proxy de protección: controlar el acceso de un objeto real.

Patrones de Diseños representantes de este estilo

- Variaciones:
  - El Proxy no conoce el objeto real.

Taller Teórico-Práctico III

(Valor 5 pts)

Objetivo: Estimar tareas de un proyecto de Desarrollo de Software

Proyecto: Gestión de Tramites de Pasaportes

Actividades:

1) Desglosar el proyecto en historias de usuarios, actividades y tareas:
   a. Definir épicas (el equipo) e Historias de Usuarios
   b. Derivar Historias de Usuarios de las Épicas
   c. Precisar Tareas a partir de las Historias de Usuario
2) Asignar las tareas a las parejas del equipo (el equipo)
3) Estimar el esfuerzo requerido para completar cada tarea (las parejas)
4) Validar la estimación (el profesor)

Condiciones de Entrega: los equipos tienen hasta el domingo 12/03 a las 8 pm

Taller Teórico-Práctico III

(Valor 5 pts)

Historias de Usuarios

1. Como **Usuario**, puedo **solicitar mi Pasaporte** para **Gestionar Visas en otros países**
2. Como **Usuario**, puedo **identificarme en el sistema** para **controlar el acceso no autorizad**o

Referencias

- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). **Pattern – Oriented Software Architecture. A System of Patterns.** John Wiley & Sons, England.
- Gamma E., Helm R., Johnson R. and Vlissides J. (1995). **Design Patterns. Elements of Reusable Object-Oriented Software**. Addison Wesley, Reading, Massachussets.