Ingeniería de Software I

Carlos Monsalve
monsalve@espol.edu.ec

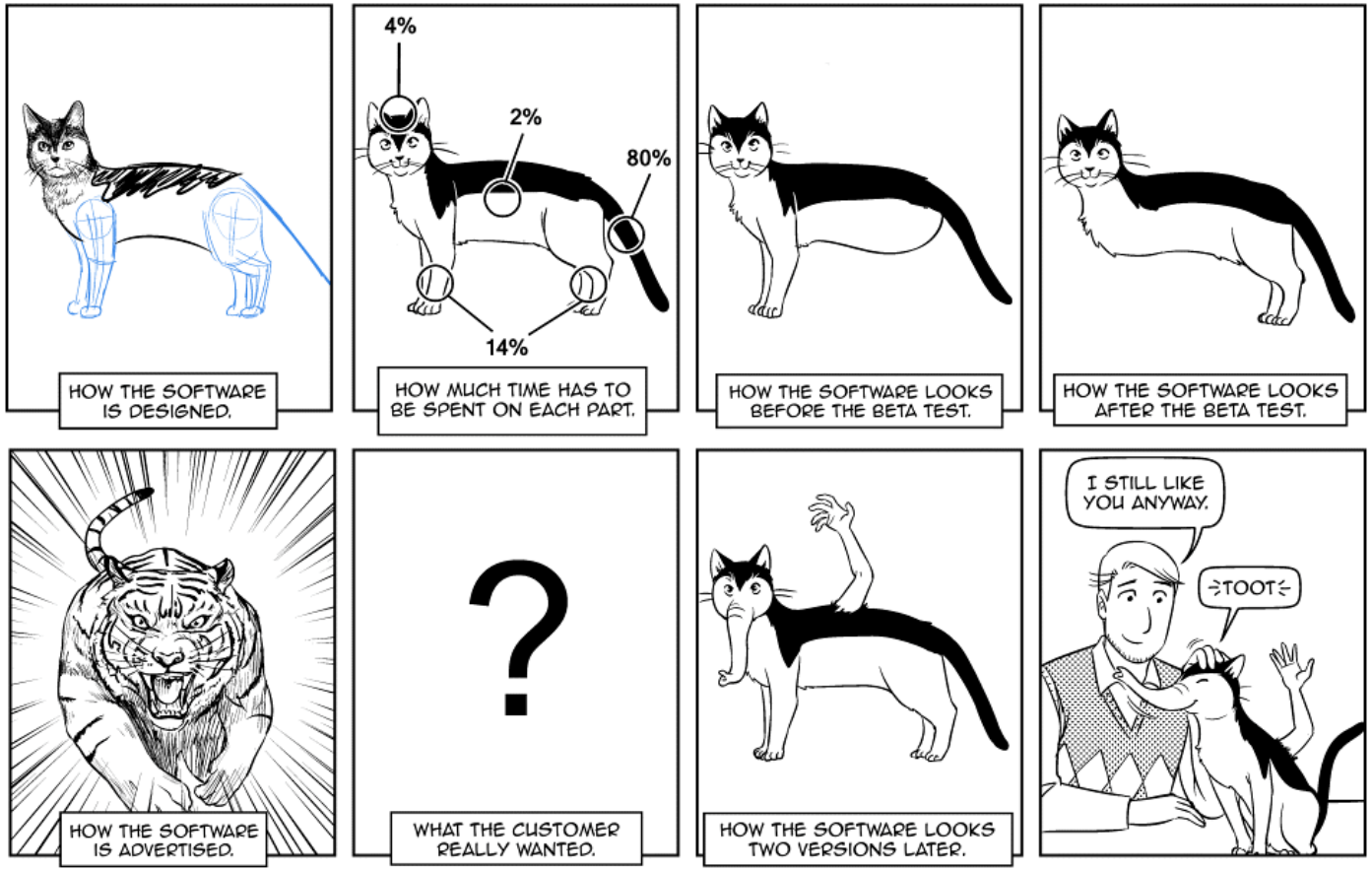
Sección 3: Diseño de Software

Ciclo de Vida | Etapas clásicas

Adaptado de “Introduction to Software Life Cycles”, Element K, 2012

Desarrollando Software

¿Cómo va nuestro proyecto?



Tomado de: Sandra and Woo, http://www.sandraandwoo.com/2012/11/19/0430-software-engineering-now-with-cats/, 2012

Diseño | Características

- IEEE 24765:2010
  - El proceso de definir la arquitectura, componentes, interfaces, y otras características de un sistema o componente; o
  - El resultado del proceso anterior
- Analizar requerimientos → modelo de estructura interna del software
- Permitir adaptarse a nuevos requerimientos

Diseño | Usos y beneficios

- Son los planos de nuestro software
- Podemos analizarlos para:
  - Verificar si cumpliremos con requerimientos
  - Determinar y evaluar soluciones alternativas
  - Planificar actividades de desarrollo
  - Comenzar a codificar nuestro software
  - Construir pruebas para nuestro software

Diseño | A considerar

- Elementos claves del software
- Tecnologías disponibles o necesarias
- **Estándares**
- Protocolos
- **Patrones**
- **Estilos**

Diseño | Niveles

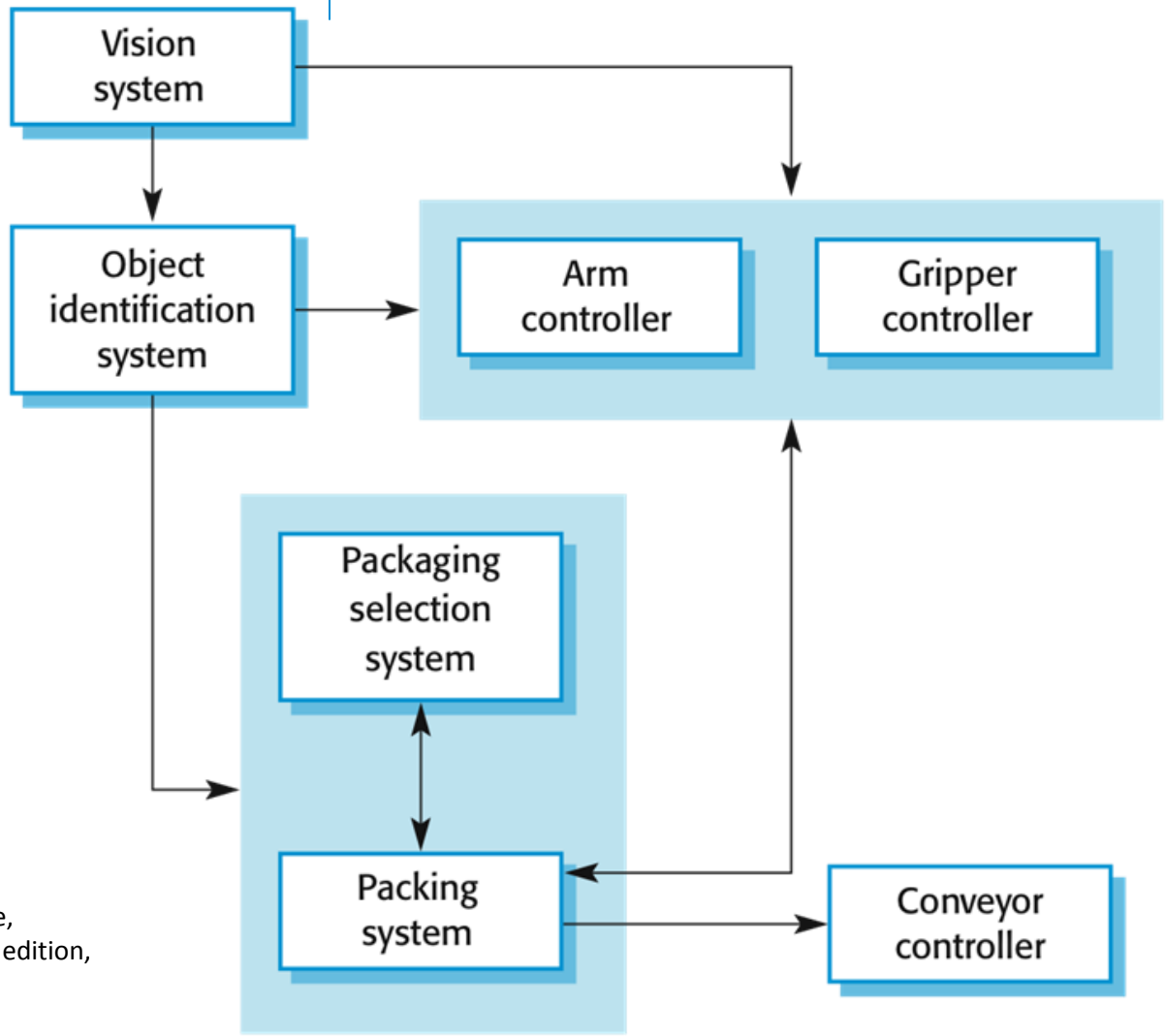
- Diseño arquitectónico
  - Diseño de alto nivel
  - Describe la estructura interna del software (arquitectura)
- Diseño detallado
  - Describe comportamiento de cada componente
  - Nivel de detalle: suficiente para iniciar codificación

Arquitectura | Conceptos

- Descripción de como el software:
  - Se descompone en componentes (subsistemas)
  - Cómo se organizan esos componentes
  - Cómo interactúan esos componentes (interfaces)
- Entendible (vistas)

Diseño arquitectónico

Ejemplo



Tomado de: I. Sommerville, Software Engineering, 9th edition, 2010

Arquitectura | Vistas típicas

Vista	Descripción	Destinatario
Lógica	Funcionalidades	Usuarios finales
Desarrollo	Implementación	Programadores
Proceso	Comportamiento, NFR	Ingenieros, integradores
Física	Topologías, interacciones	Ingenieros
Escenarios	Interacciones actores-sistema, componentes	Clientes

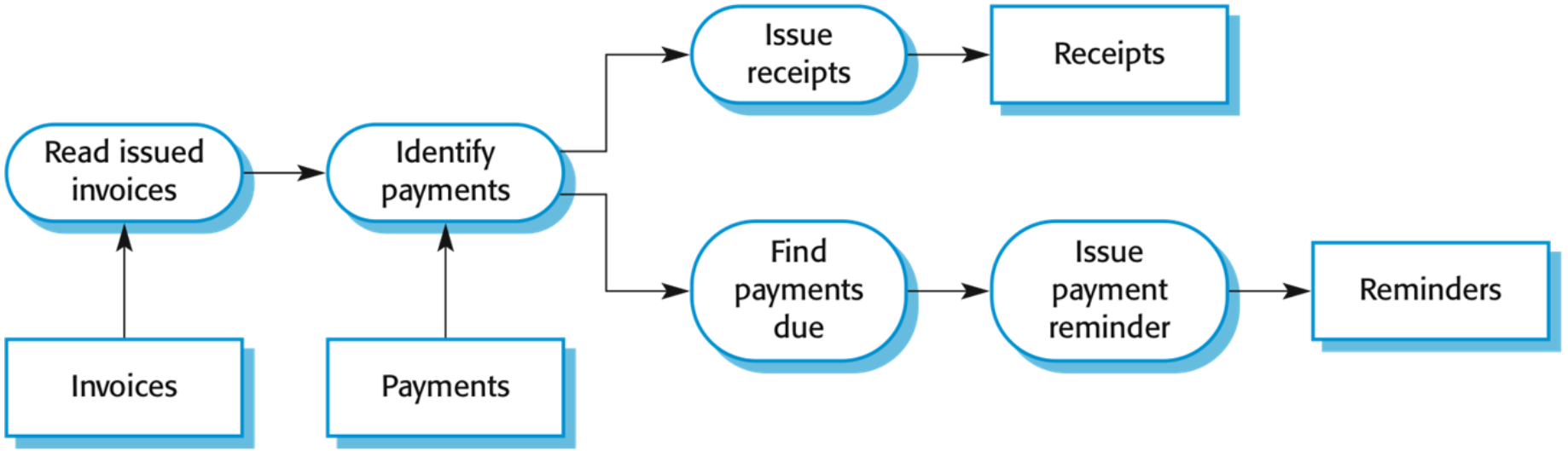
Diseño Arquitectónico

Estilos

- ISO/IEEE 24765:
  - Definición de una **familia de sistemas** en términos de un **patrón** de **organización estructural**
  - Caracterización de una **familia de sistemas** que se relacionan porque **comparten propiedades estructurales y semánticas**
- ¿Patrones o estilos?
  - Reservamos “patrones” para diseño detallado

Estilos | Tuberías y filtros

- Basada en transformaciones funcionales
  - Inputs son transformadas en outputs

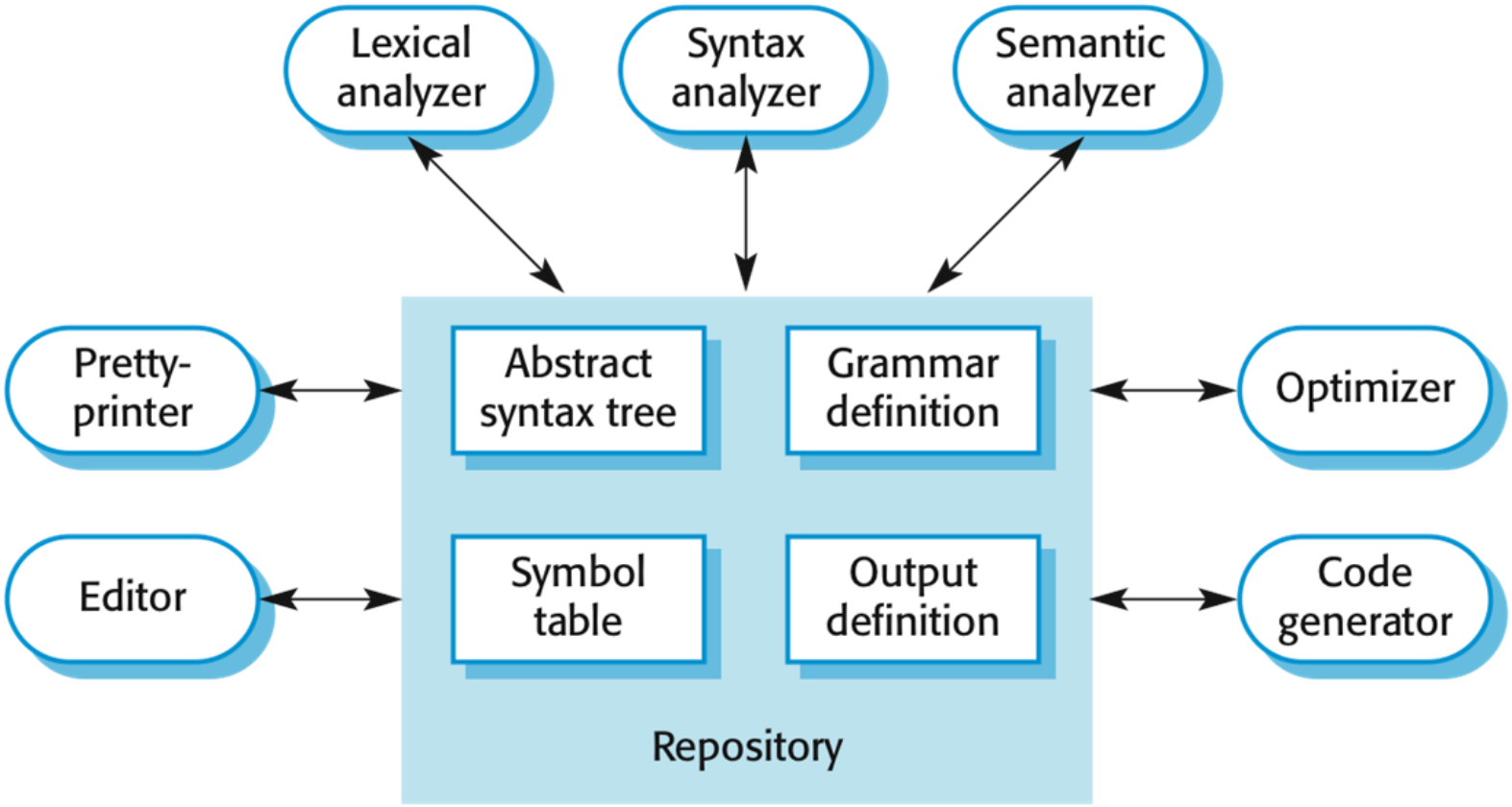


- Filtro: realiza transformación
- Tubería: permite flujo de datos

Tomado de: I. Sommerville, Software Engineering, 9th edition, 2010

Estilos | Repositorios

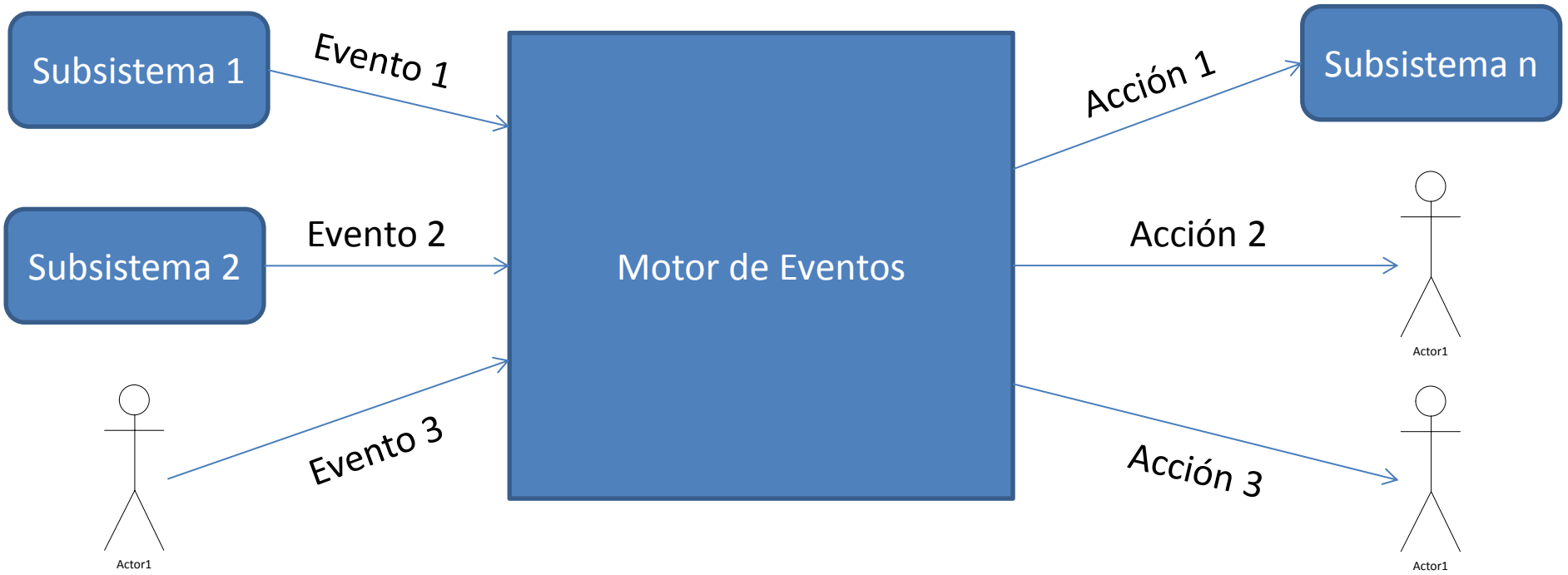
- Subsistemas comparten repositorio central



Tomado de: I. Sommerville, Software Engineering, 9th edition, 2010

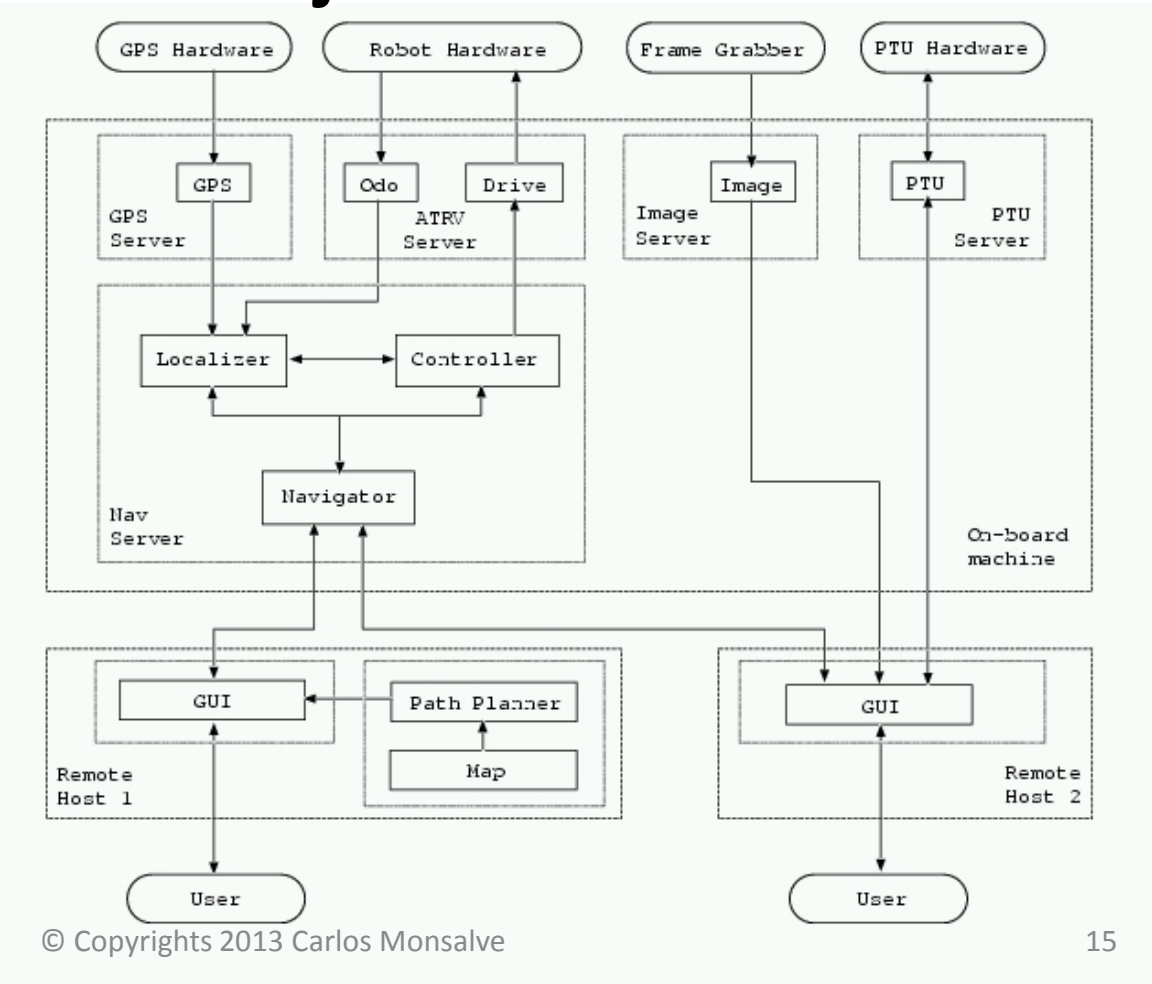
Estilos | Orientada a eventos (EDA)

- Acciones basadas en interpretación de eventos



Estilos | Orientada a objetos (OOA)

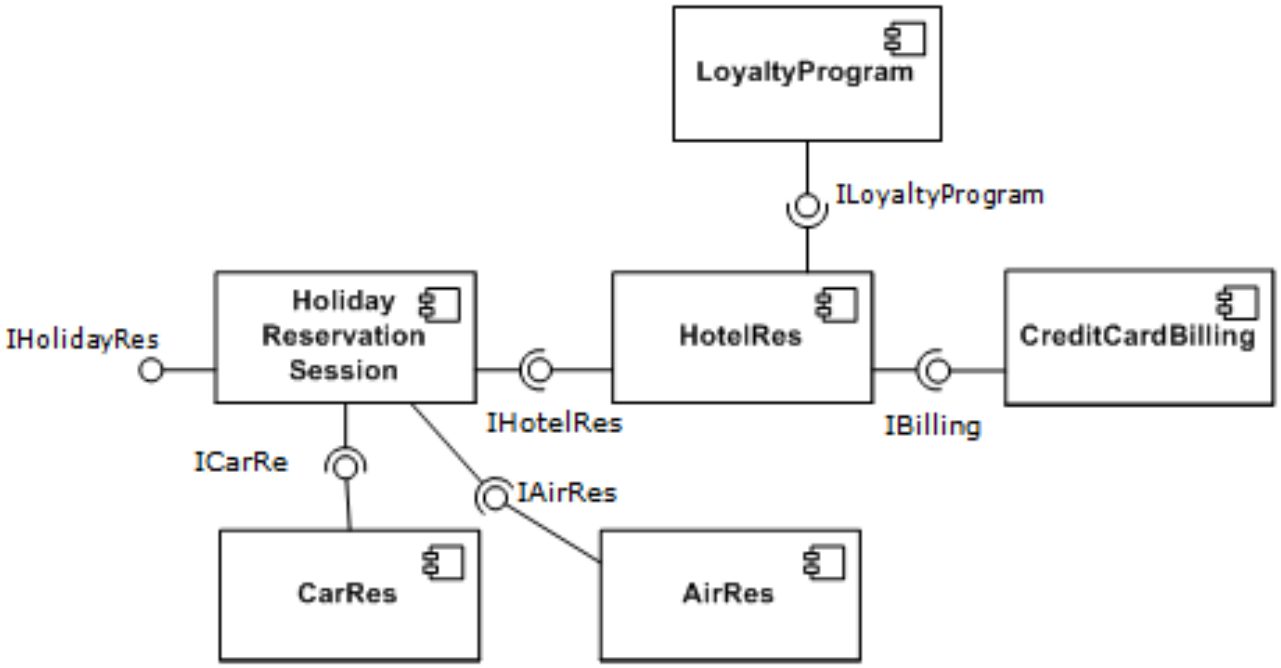
- Sistema se divide en **objetos** auto-suficientes y reutilizables



Tomado de: A. Georgiev y P. K. Allen, The AVENUE Project, University of Columbia, http://www.cs.columbia.edu/robotics/projects/avenue/

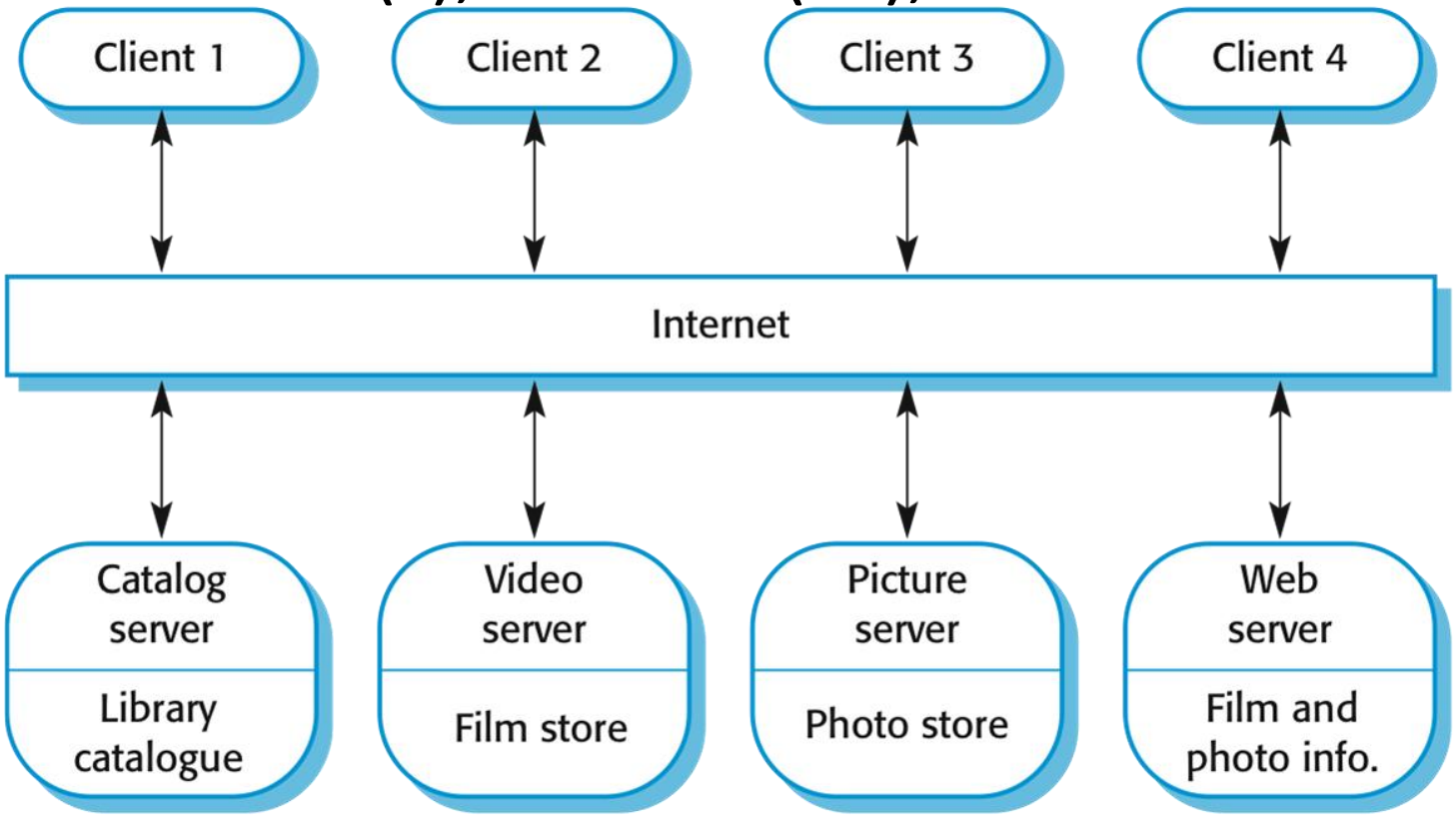
Estilos | Basada en componentes

- Sistema se descompone en componentes **funcionales** que se comunican por interfaces
- Nivel de abstracción mayor al de OOA



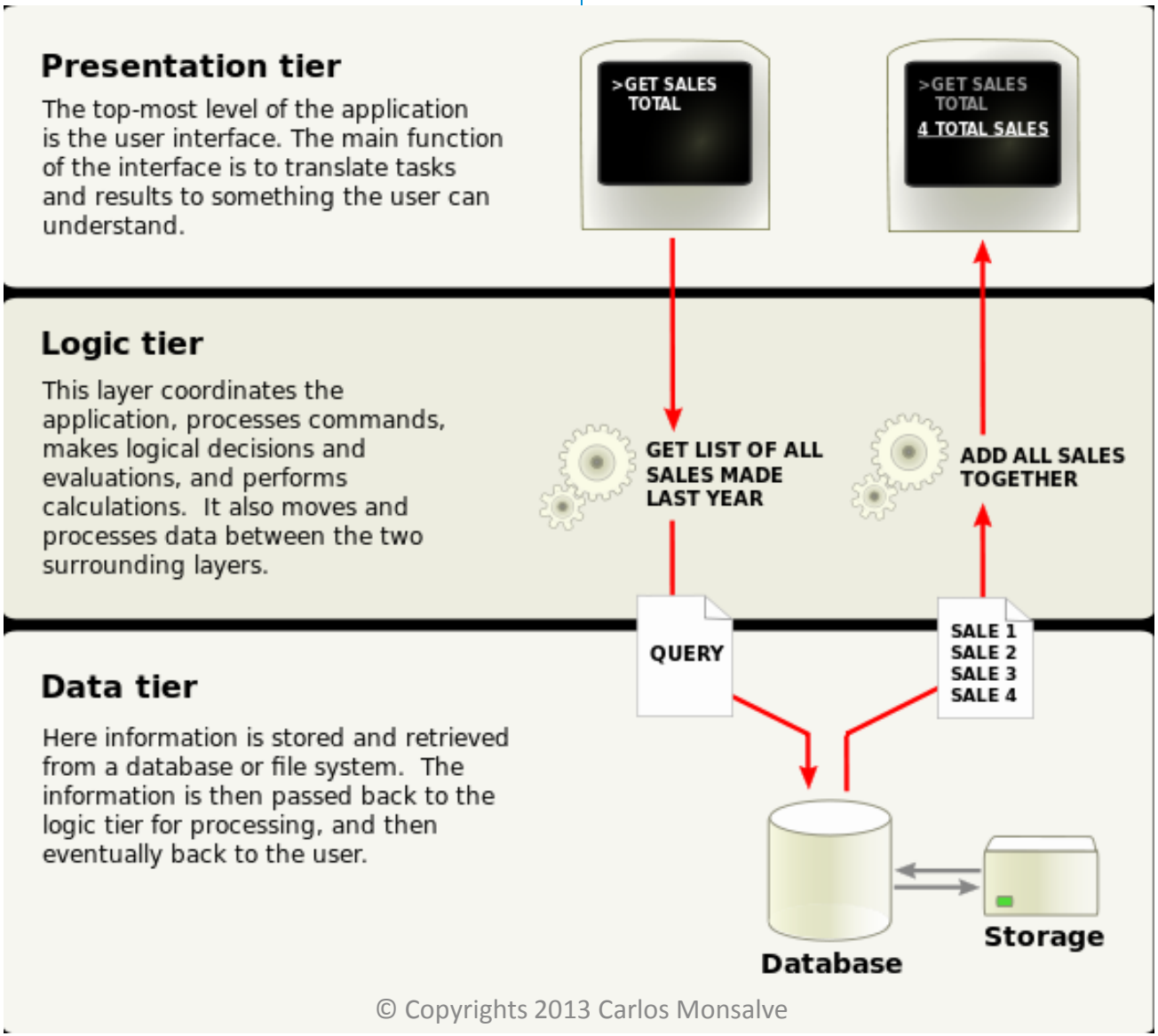
Estilos | Cliente-servidor

- Sistema distribuido con 3 componentes: cliente(s), servidor(es), red



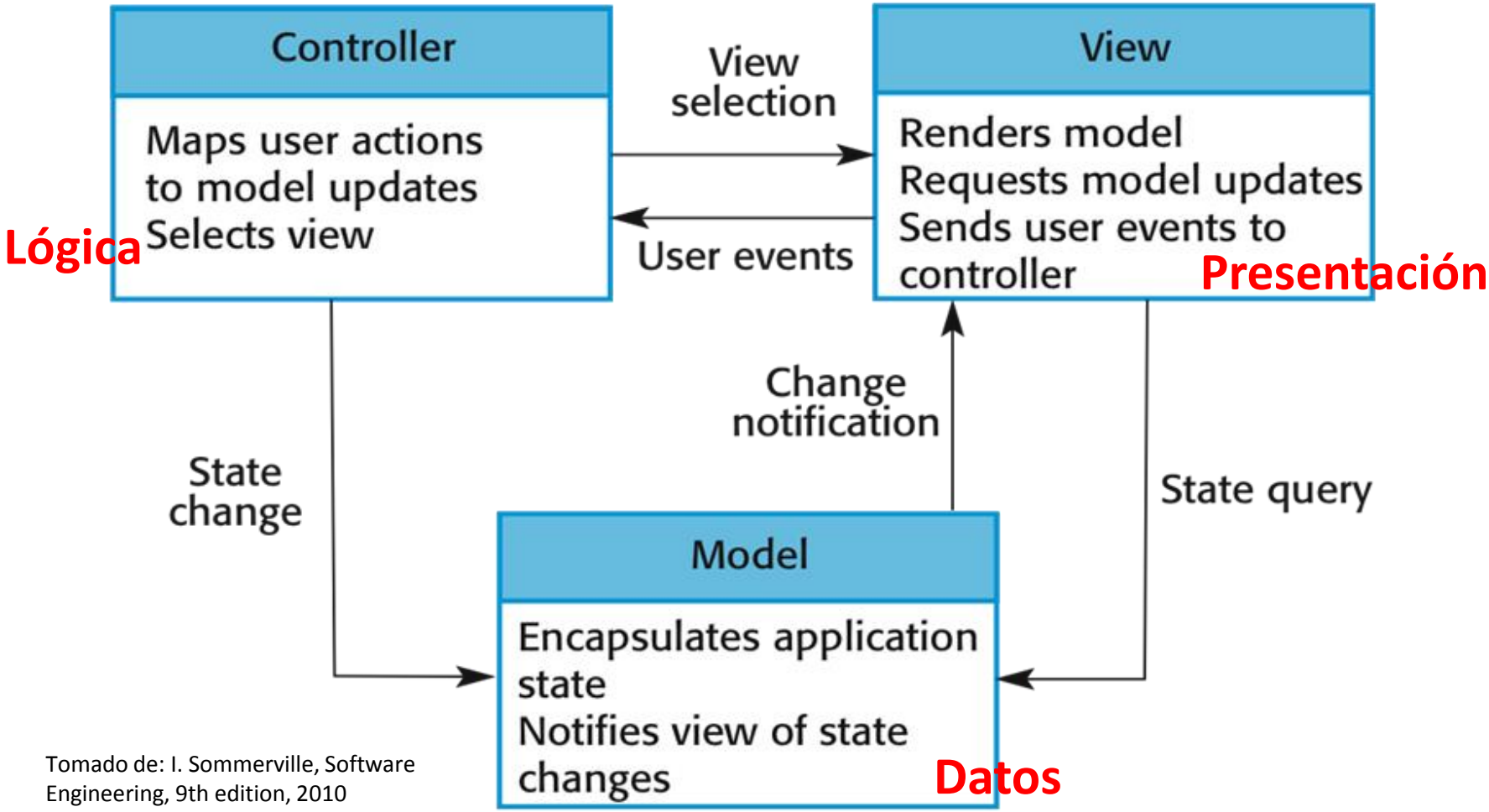
Tomado de: I. Sommerville, Software Engineering, 9th edition, 2010

Estilos | De 3 capas (3-tier)



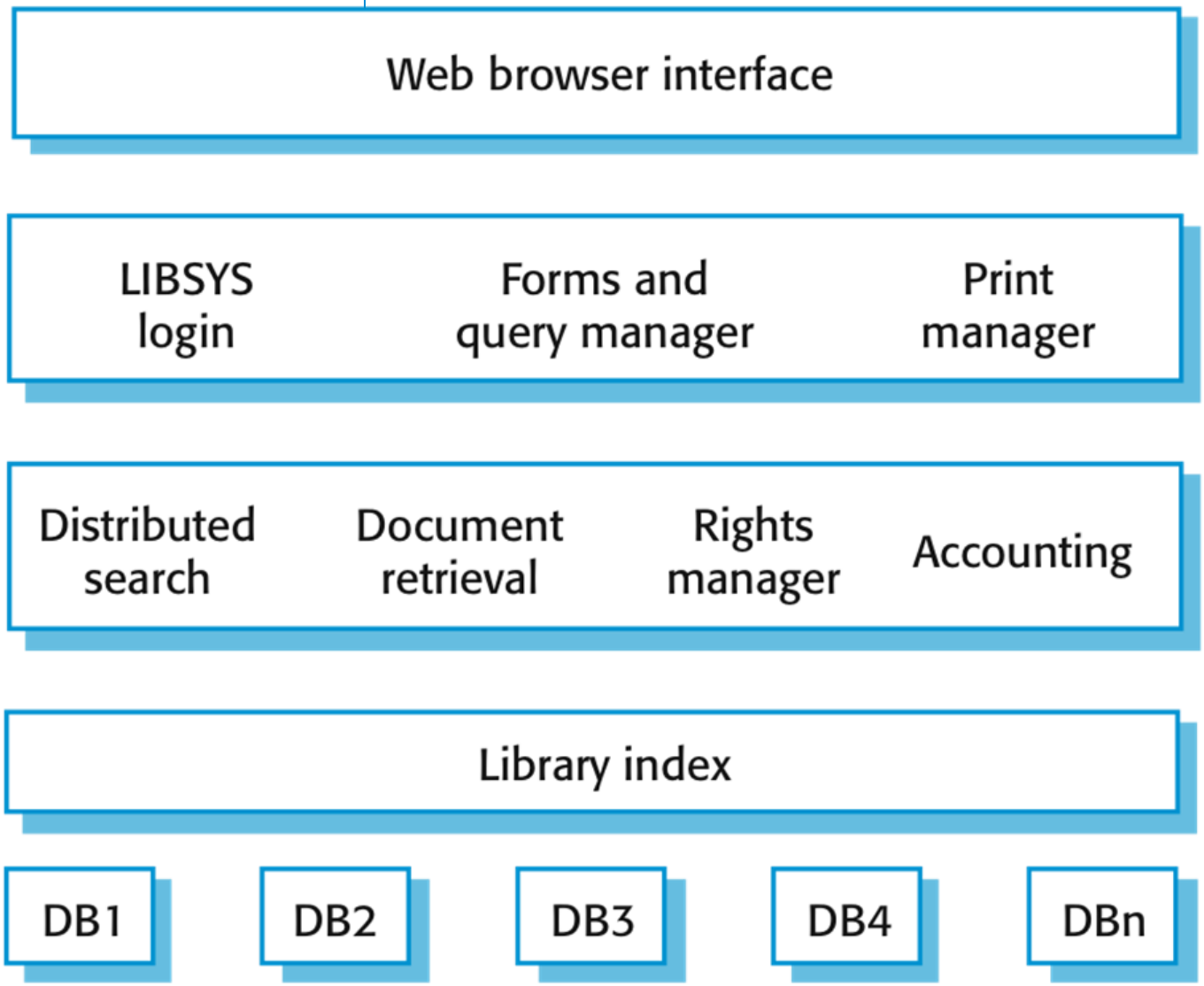
Estilos

Modelo-Vista-Controlador (MVC)



Tomado de: I. Sommerville, Software Engineering, 9th edition, 2010

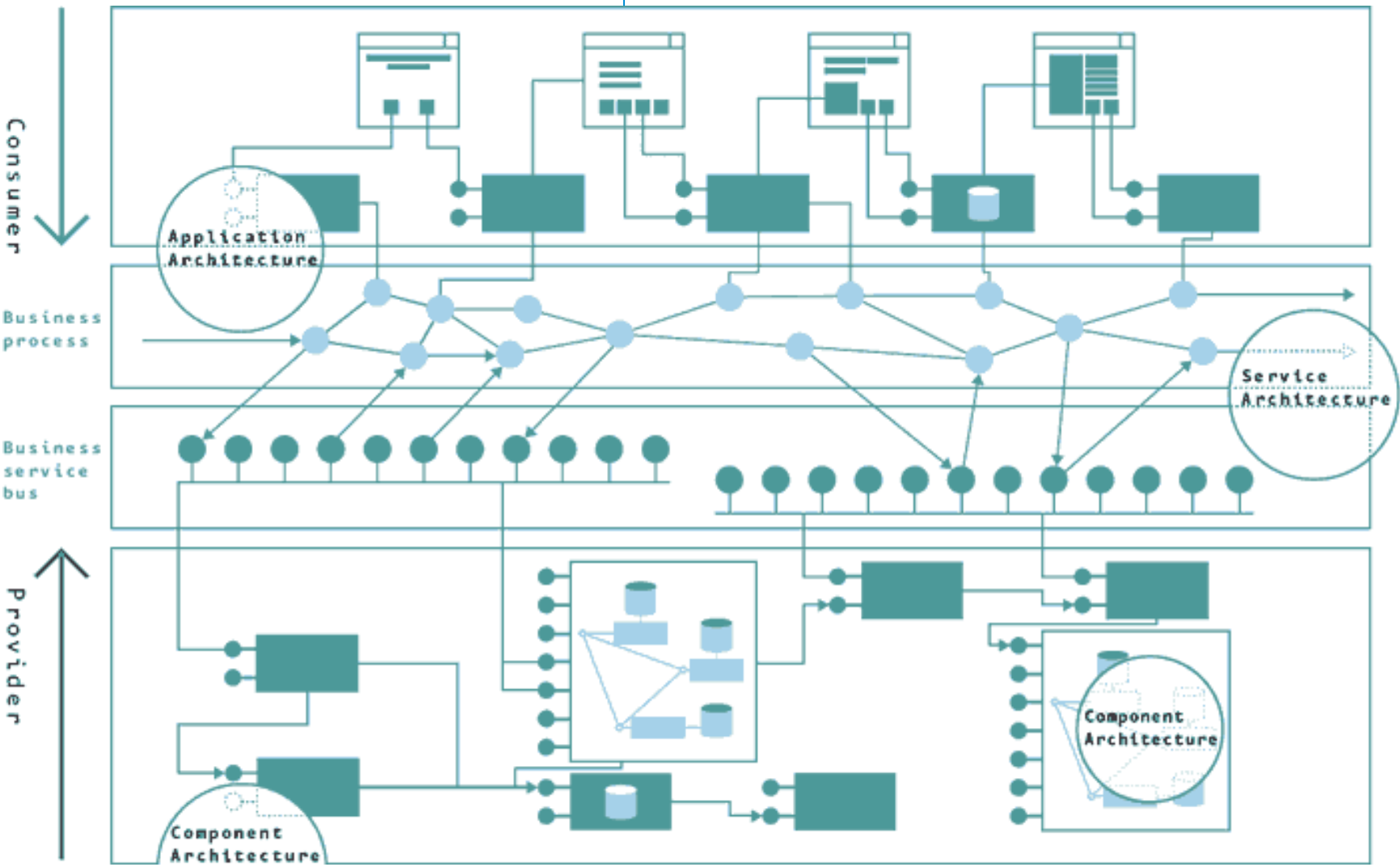
Estilos | Multicapas



Tomado de: I. Sommerville, Software Engineering, 9th edition, 2010

Estilos

Orientada a servicios (SOA)



Tomado de: D. Sprott y L. Wilkes, Understanding Service-Oriented Architecture,MSDN, http://msdn.microsoft.com/en-us/library/aa480021.aspx, 2004

Diseño | Calidad

- Cumplimiento de criterios
- Cumplimiento de requerimientos
- ¿Cómo evaluarlo?
  - Revisiones en grupo
  - Análisis estático (inspecciones)
  - Simulación y prototipos

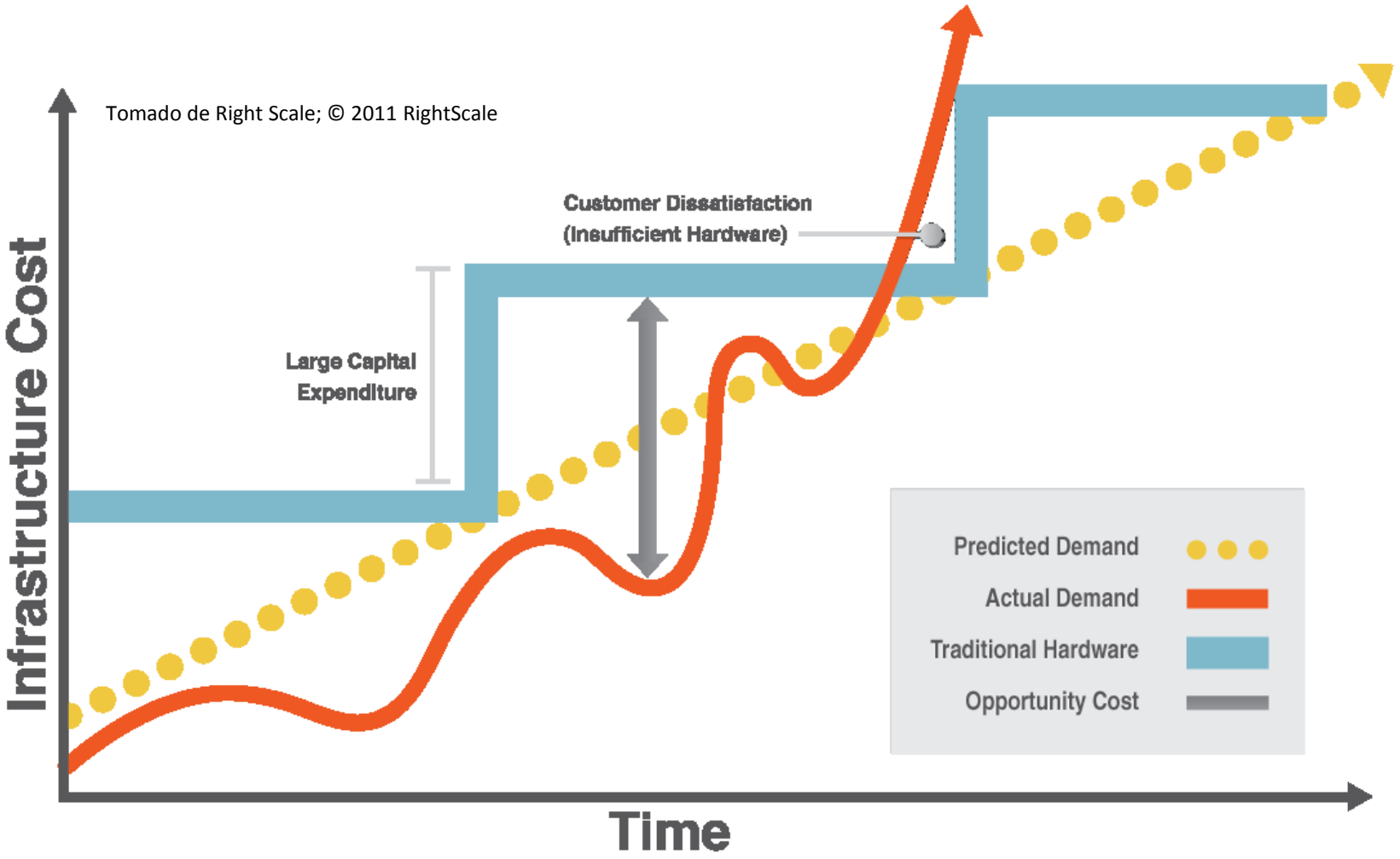
Diseño | Criterios

- Modularidad
- Compatibilidad (consistencia)
- Mantenibilidad (cambiar, actualizar, probar)
- Escalabilidad (requerimientos, transacciones)
- Extensibilidad (crecer sin afectar al resto)
- Reusabilidad (solución genérica)

Diseño | Criterios (2)

- Confiabilidad
- Usabilidad (comprensión)
- A prueba de fallos
- Aislamiento de errores
- Robustez
- Seguridad

Escalabilidad | Modelo



Diseño | Técnicas

1. Abstracción
2. Acoplamiento: relación entre componentes
3. Cohesión: relaciones internas del componente
4. Descomposición y modularización
5. Encapsulación (ocultar información)
6. Separar interfaces de implementación
7. Suficiencia, completo

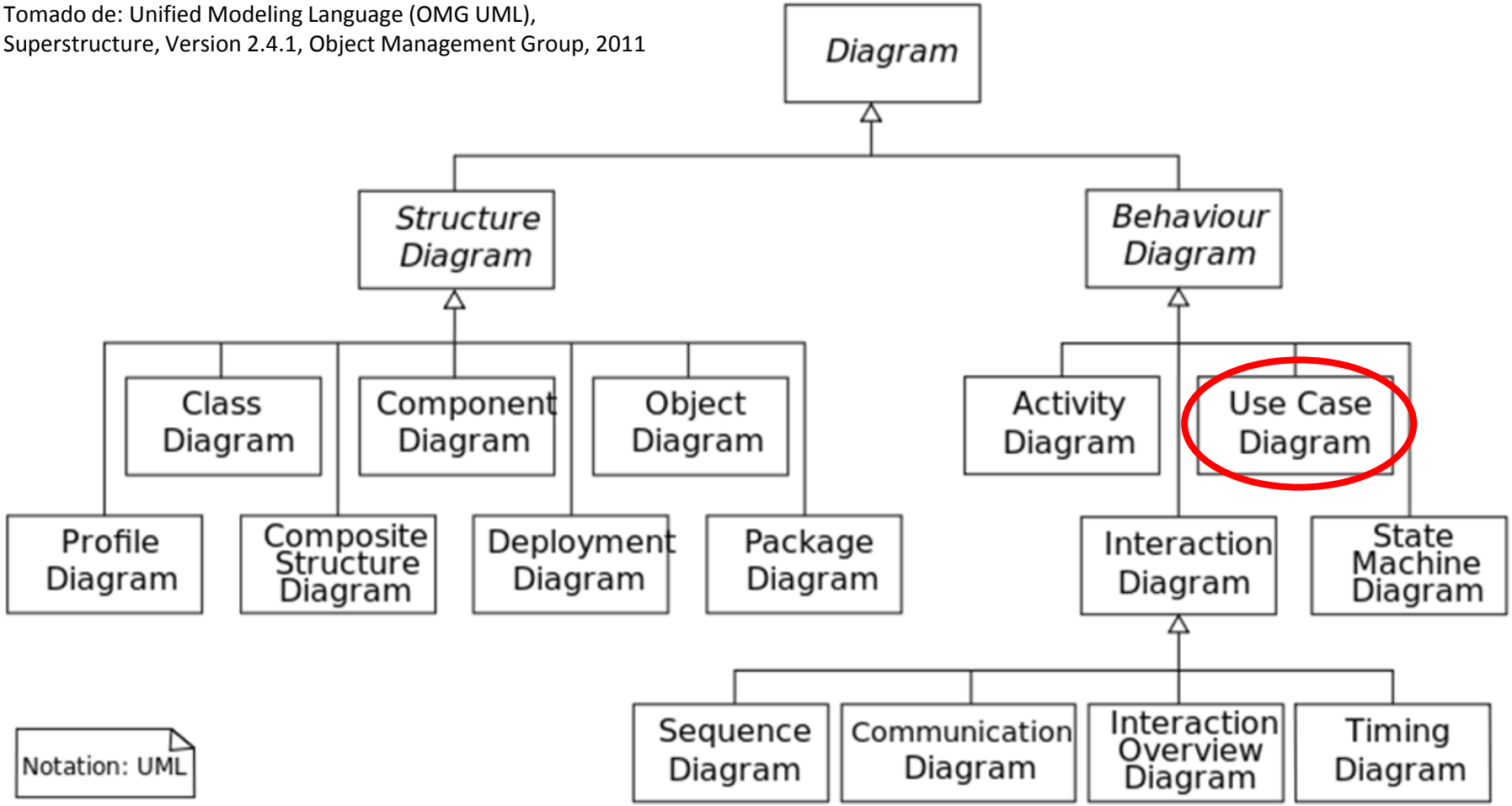
P.O.O | Conceptos a recordar

Lenguaje | UML

- Estándar de la OMG y de la ISO
- Varios tipos de diagramas
  - Diferentes aspectos de la información
  - Estructurales: estructura estática
  - Comportamiento: dinamia
    - Interacción: aspectos
- Facilitan comunicación

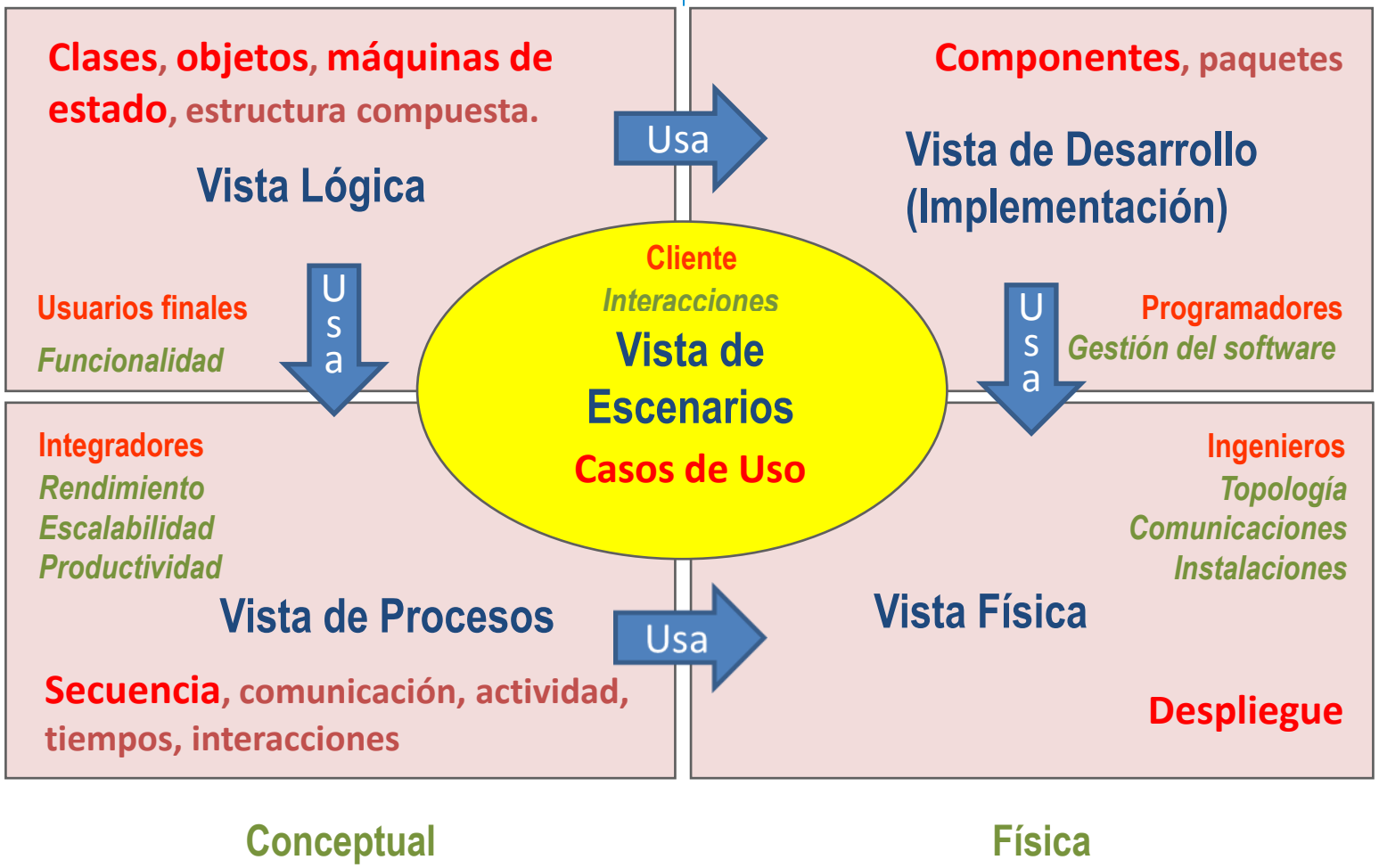
UML | Diagramas

Tomado de: Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1, Object Management Group, 2011



Diseño Arquitectónico

Vistas (Modelo 4+1)



Adaptado de: P. Kruchten, Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software, Vol. 12, No. 6, 1995

Escenarios | UML: casos de uso

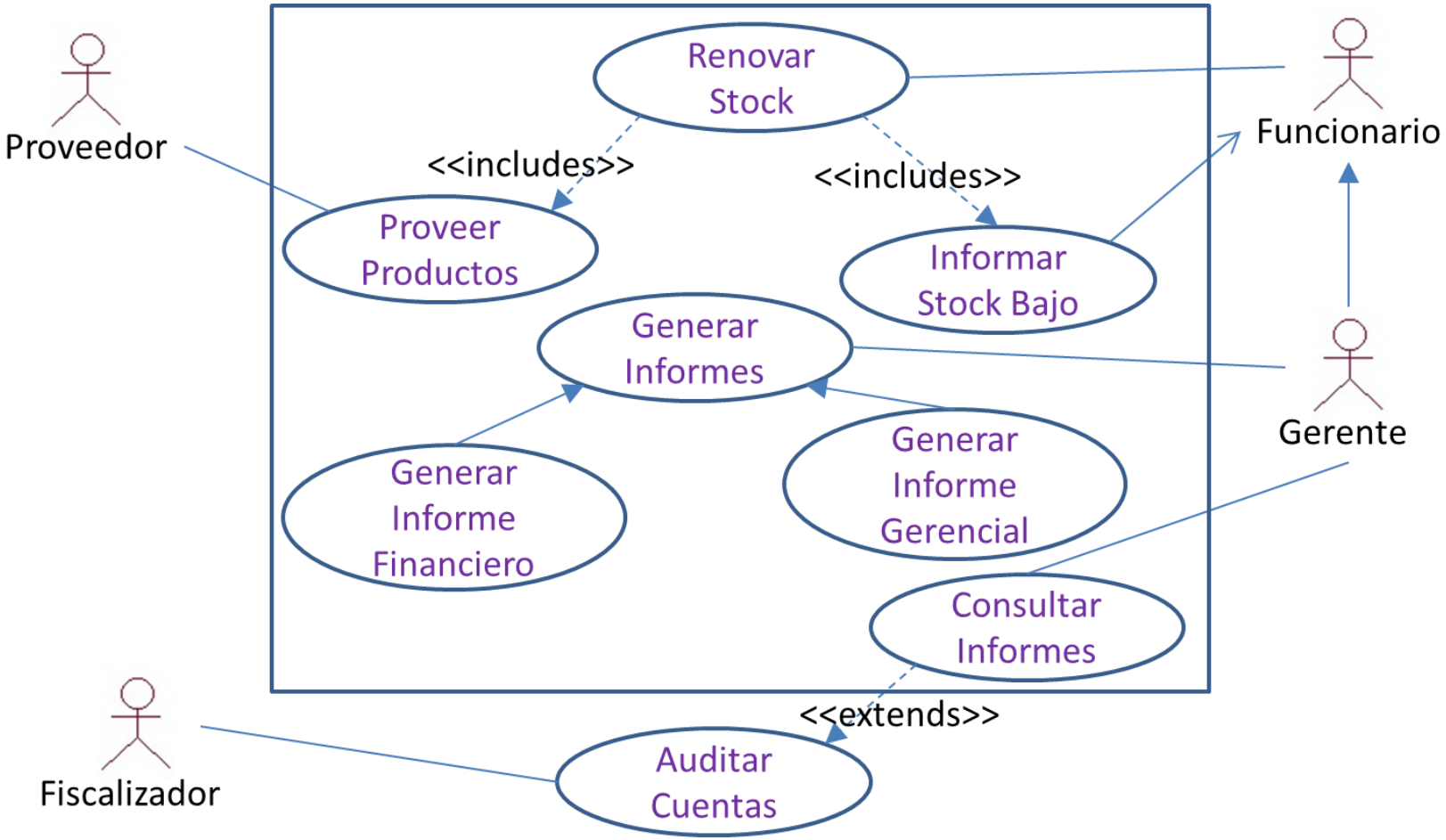
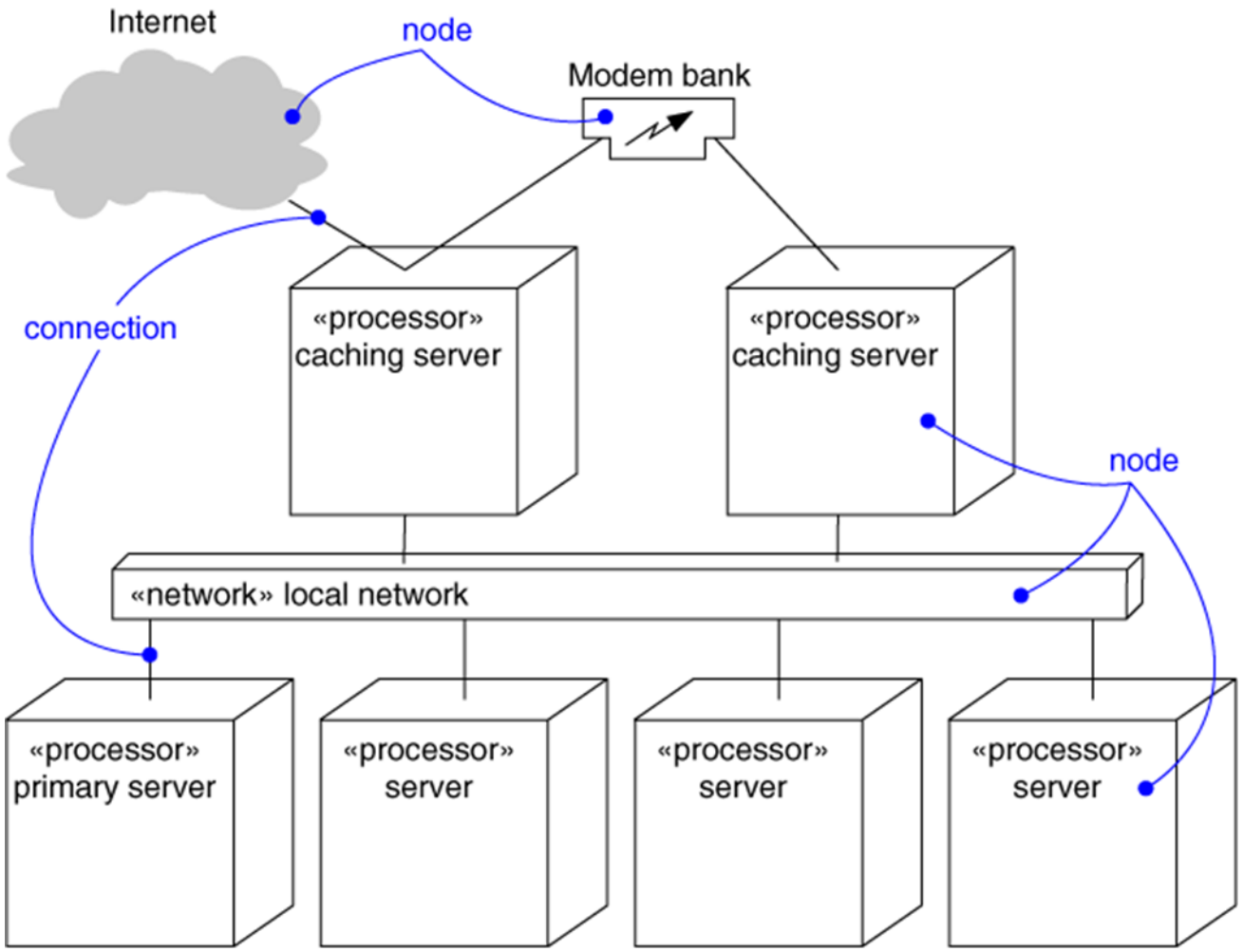


Diagrama de Despliegue

Versión UML



Tomado de: G. Booch, Software Architecture and the UML, Rational Software, 1999

Diagrama de Despliegue

Versión más gráfica

Tomado de: L. M. Cáceres, M. A. Pinto, Modelo de programación asíncrona para Web transaccionales en un ambiente distribuido, Ingeniería: Revista chilena de ingeniería, Vol. 19, No. 1, 2011

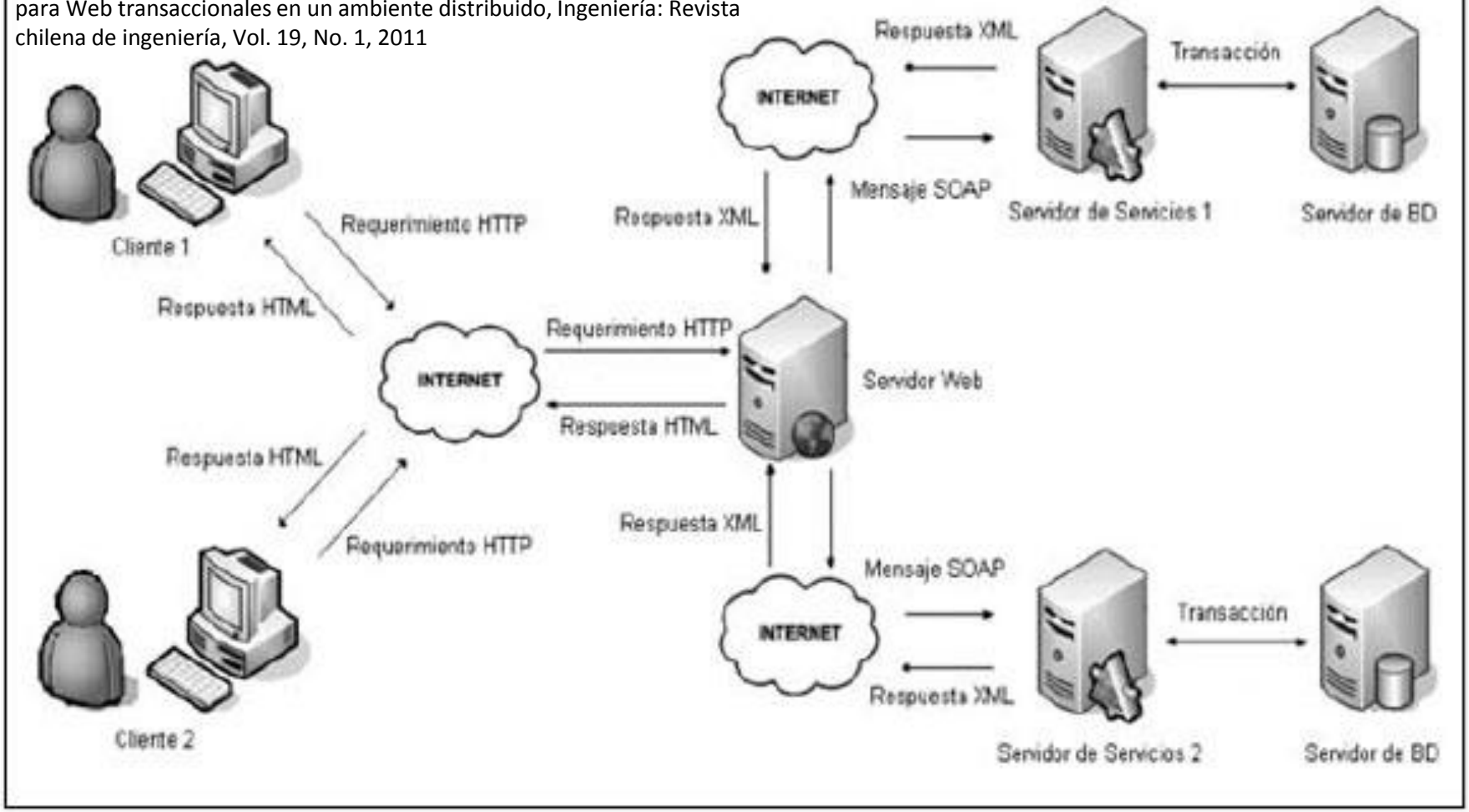
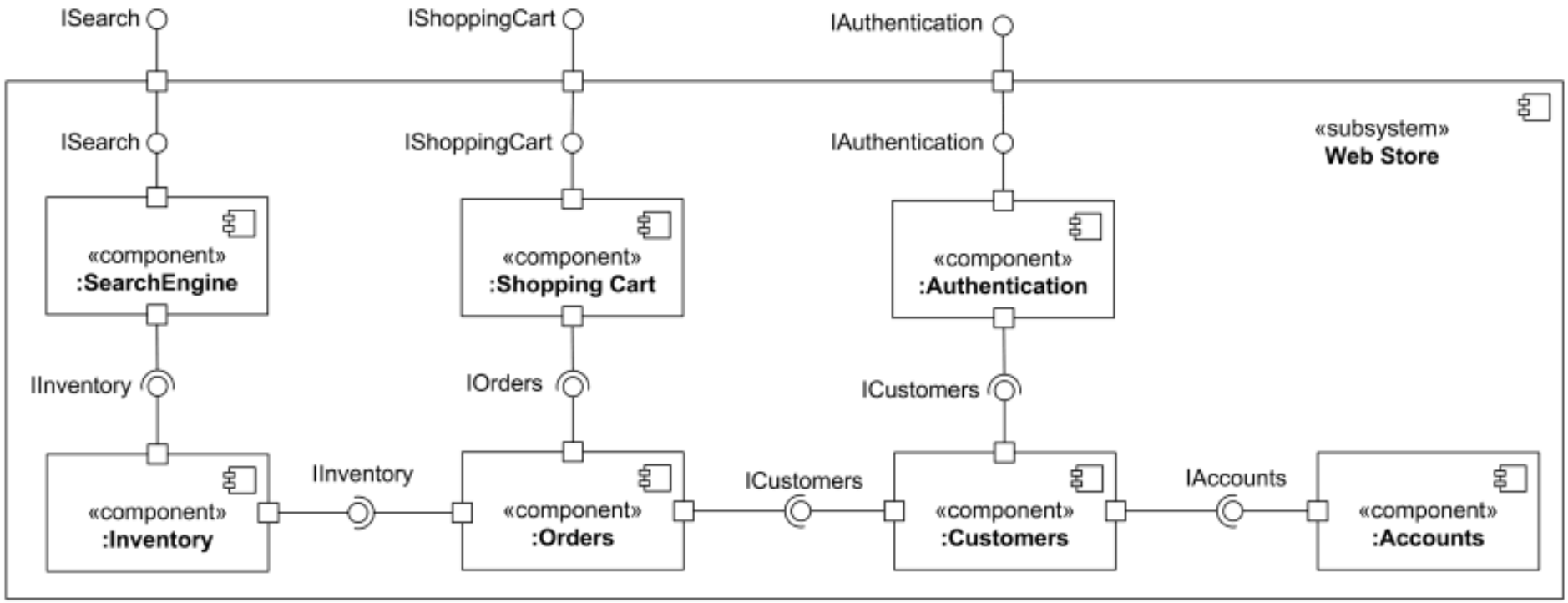


Diagrama de Componentes

UML



Tomado de: K. Fakhroutdinov, UML Component Diagrams Exampleshttp://www.uml-diagrams.org/component-diagrams-examples.html

Diagrama de Componentes

Versión más simplificada

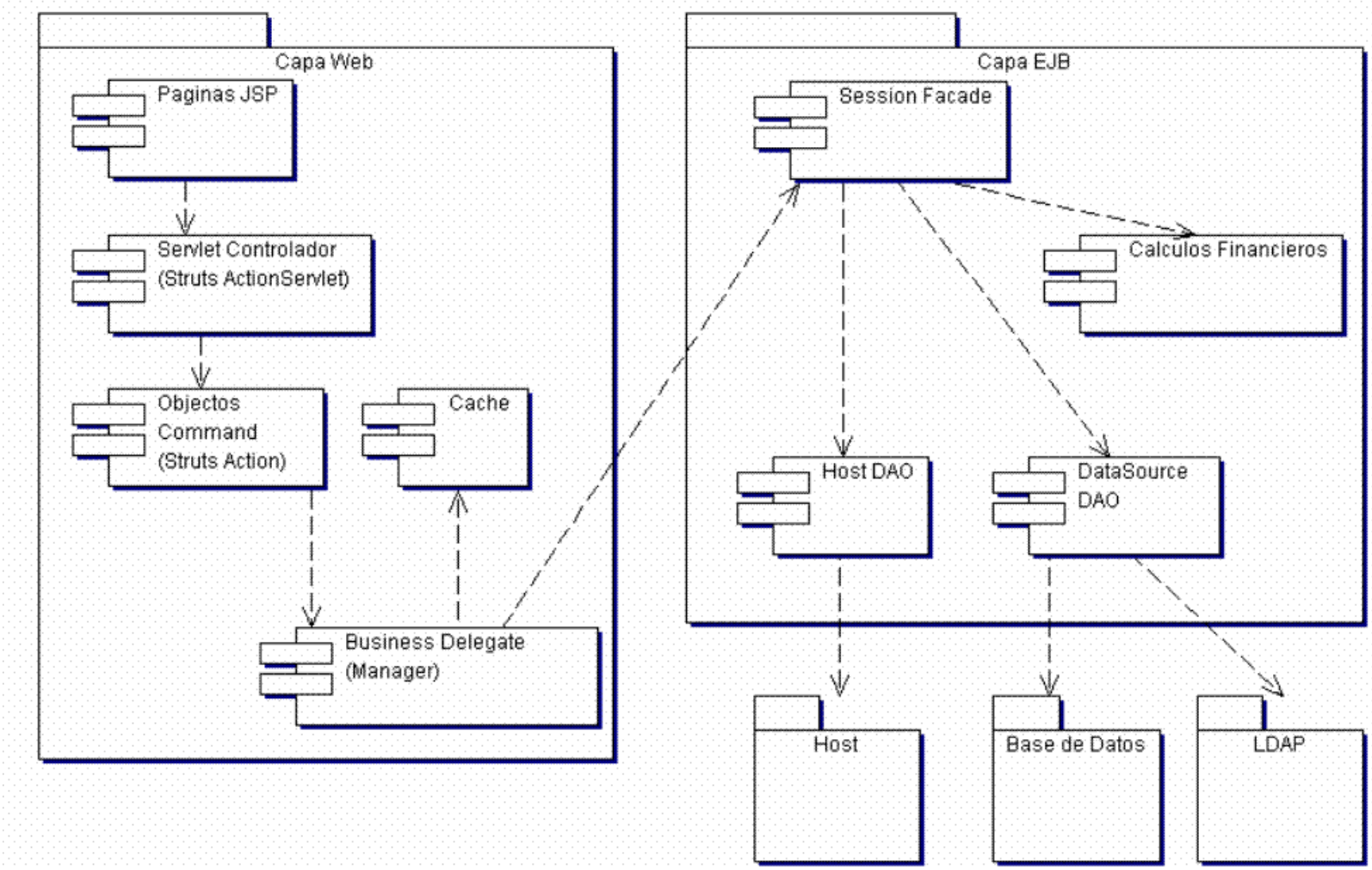
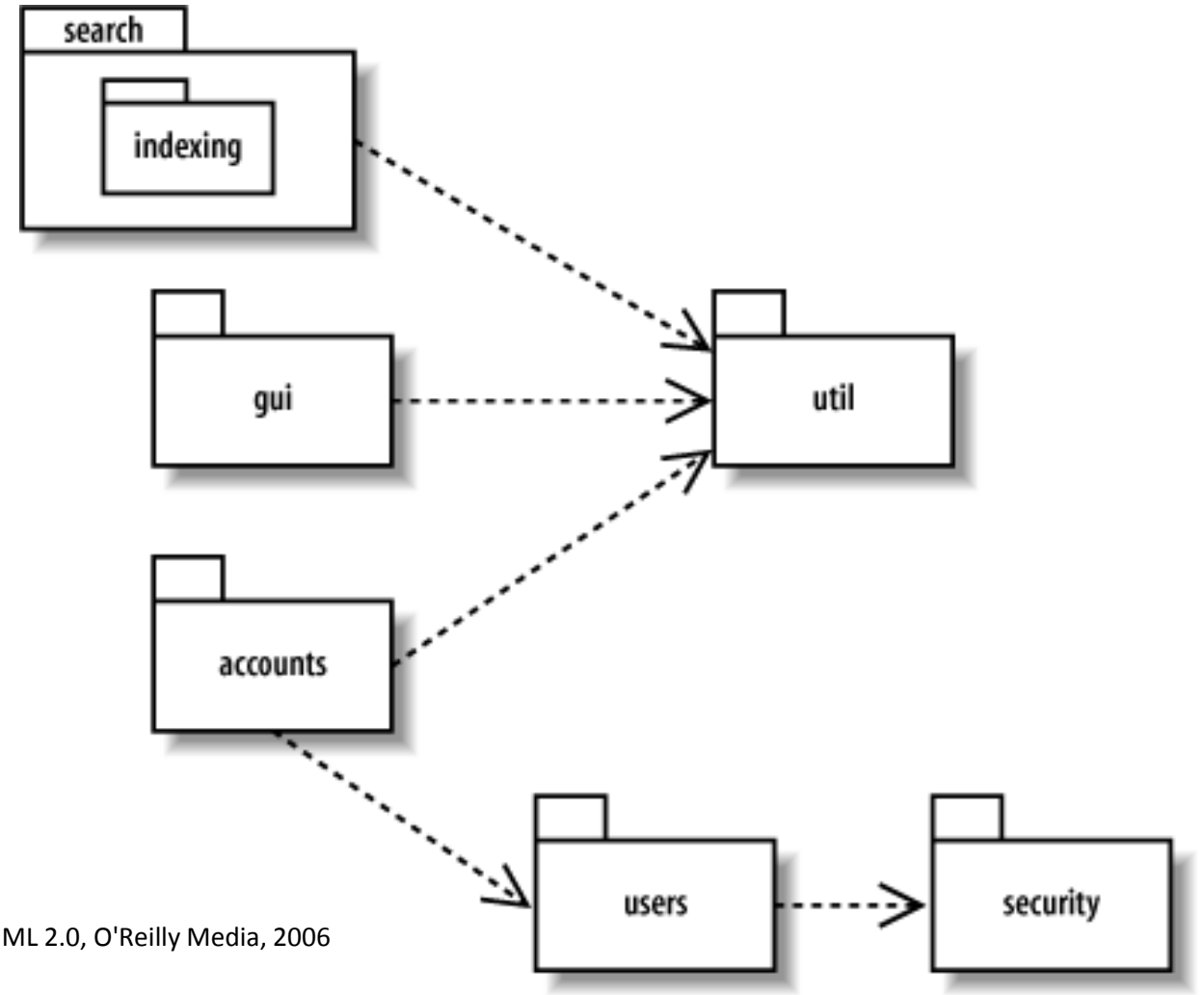


Diagrama de Paquetes

UML

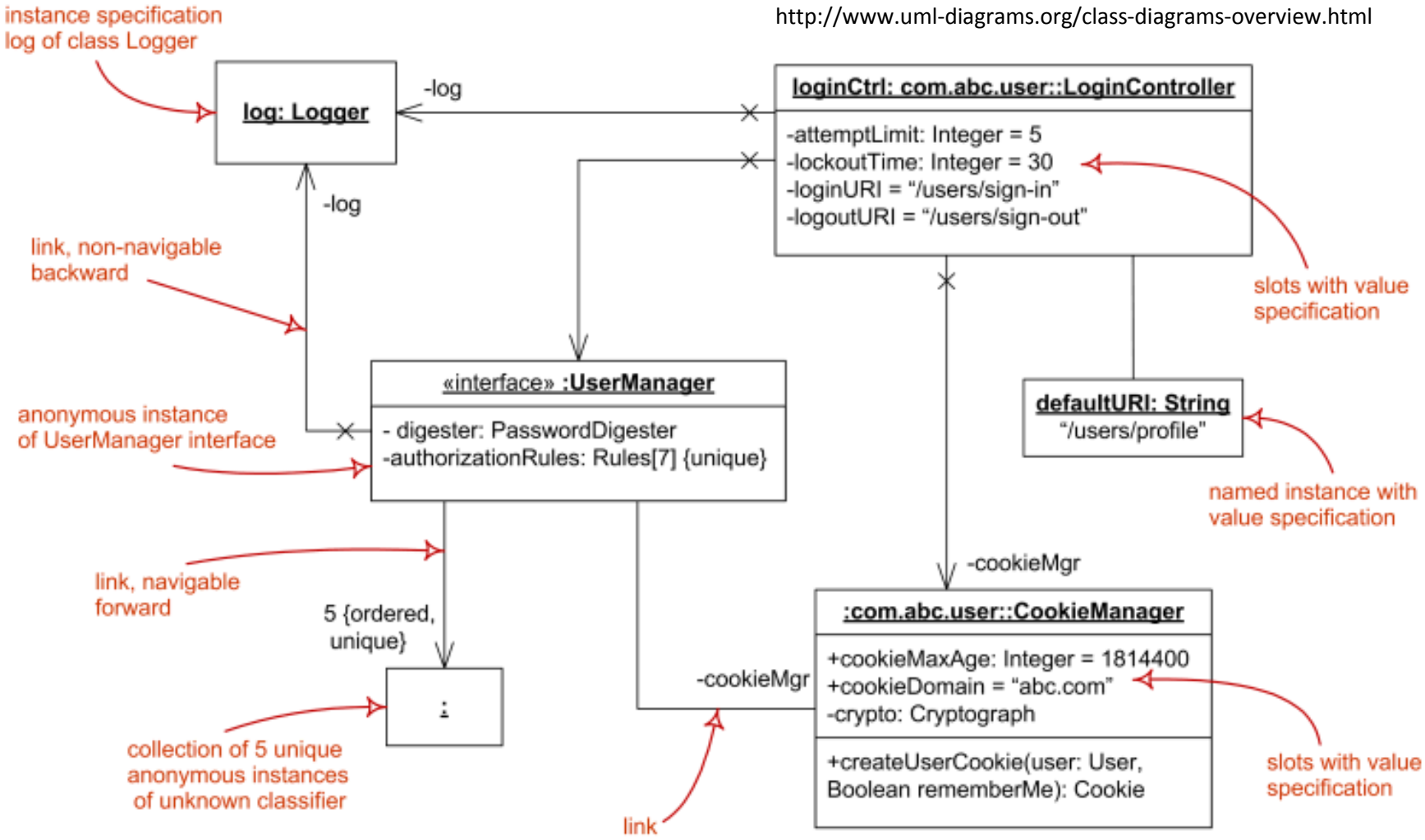


Tomado de: R. Miles and K. Hamilton, Learning UML 2.0, O'Reilly Media, 2006

Diagrama de Objetos

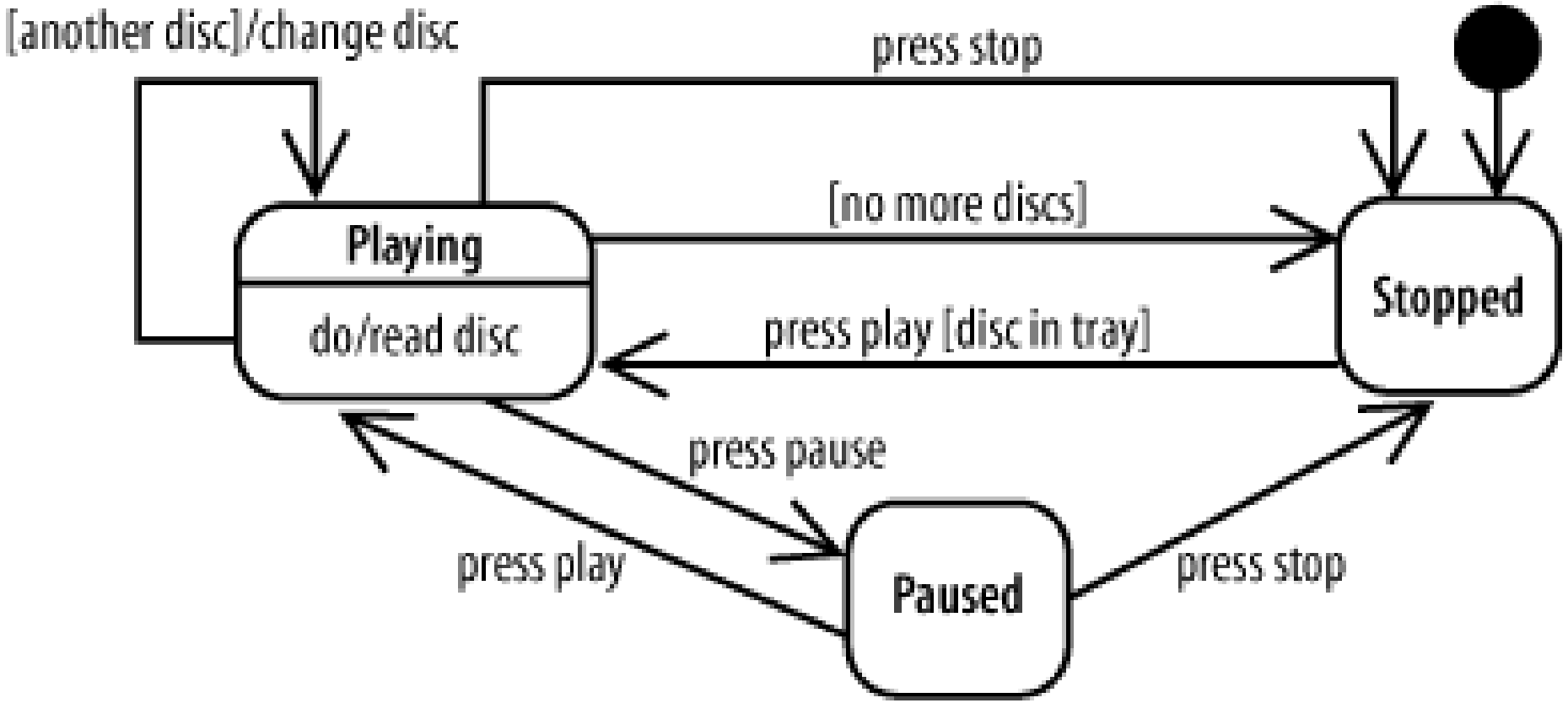
UML

Tomado de: K. Fakhroutdinov, Class and Object Diagrams Overview, http://www.uml-diagrams.org/class-diagrams-overview.html



Máquina de Estados

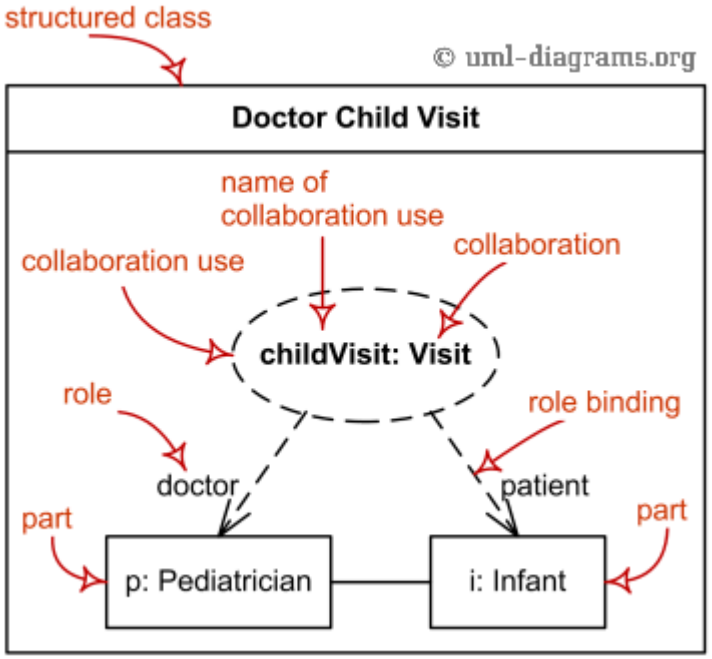
UML

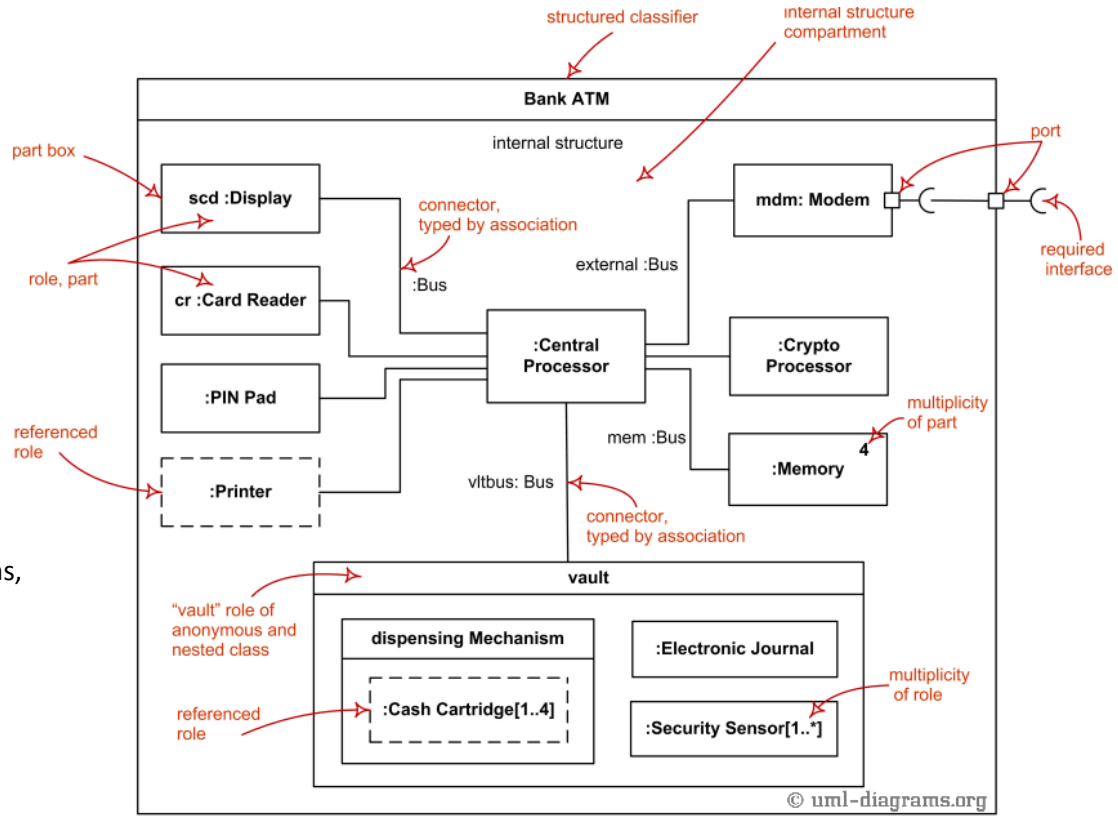


Tomado de: R. Miles and K. Hamilton, Learning UML 2.0, O'Reilly Media, 2006

Estructura Compuesta

UML

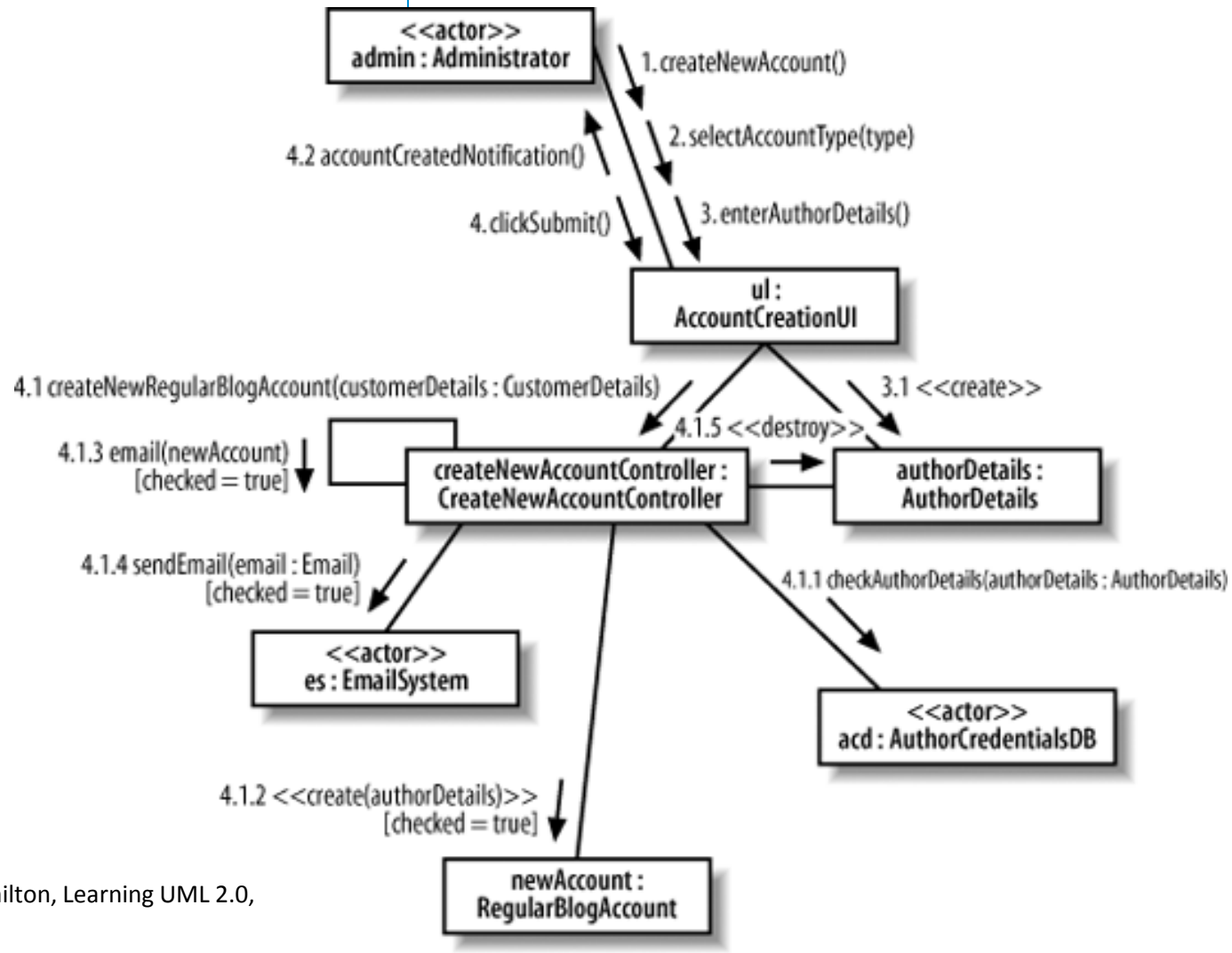




Tomado de: K. Fakhroutdinov, Composite Structure Diagrams, http://www.uml-diagrams.org/composite-structure-diagrams.html#port

Diagrama de Comunicación

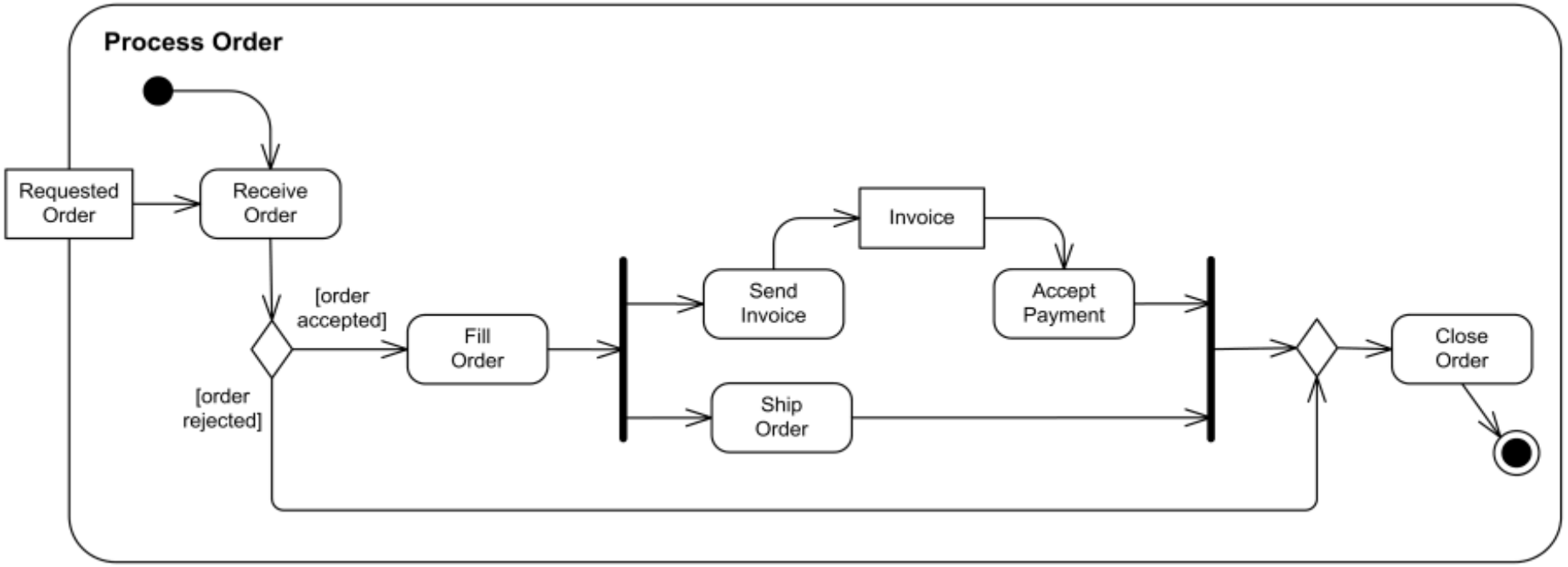
UML



Tomado de: R. Miles and K. Hamilton, Learning UML 2.0, O'Reilly Media, 2006

Diagrama de Actividad

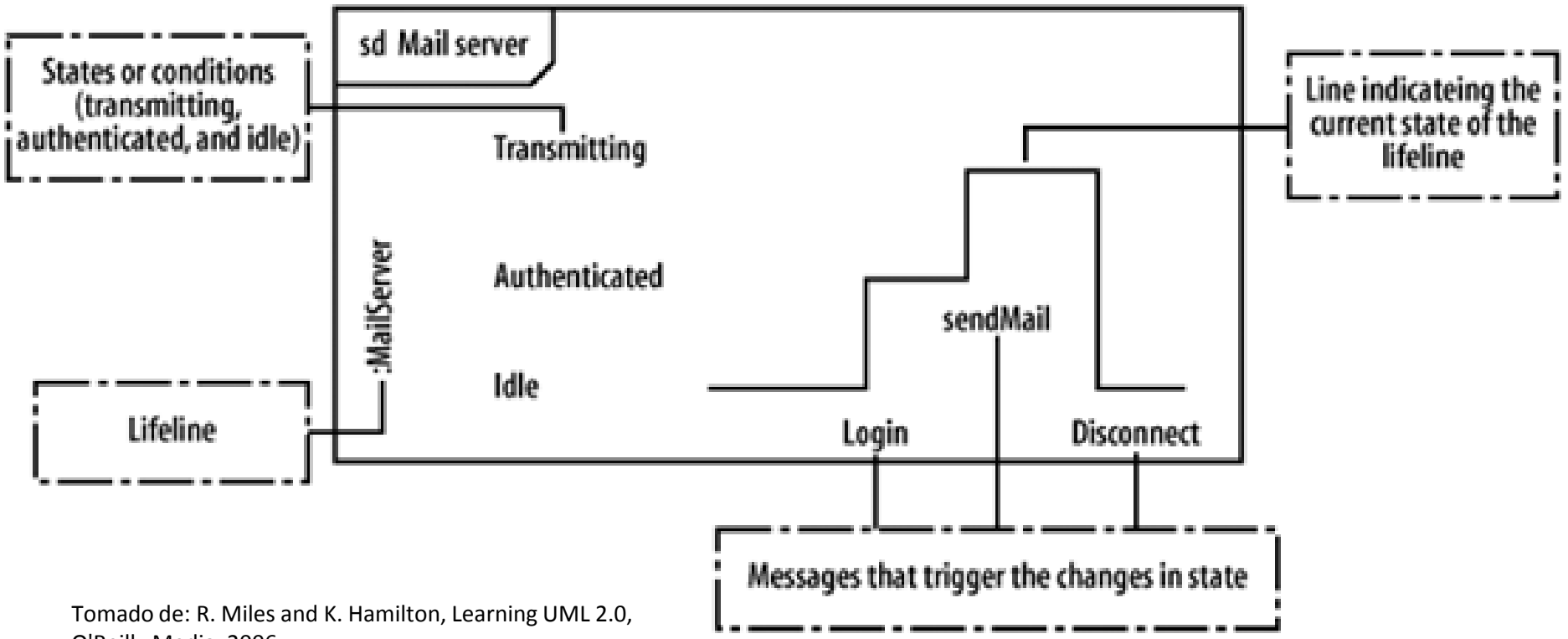
UML



Tomado de: K. Fakhroutdinov, Activity Diagram Examples, http://www.uml-diagrams.org/activity-diagrams-examples.html

Diagrama de Tiempos

UML

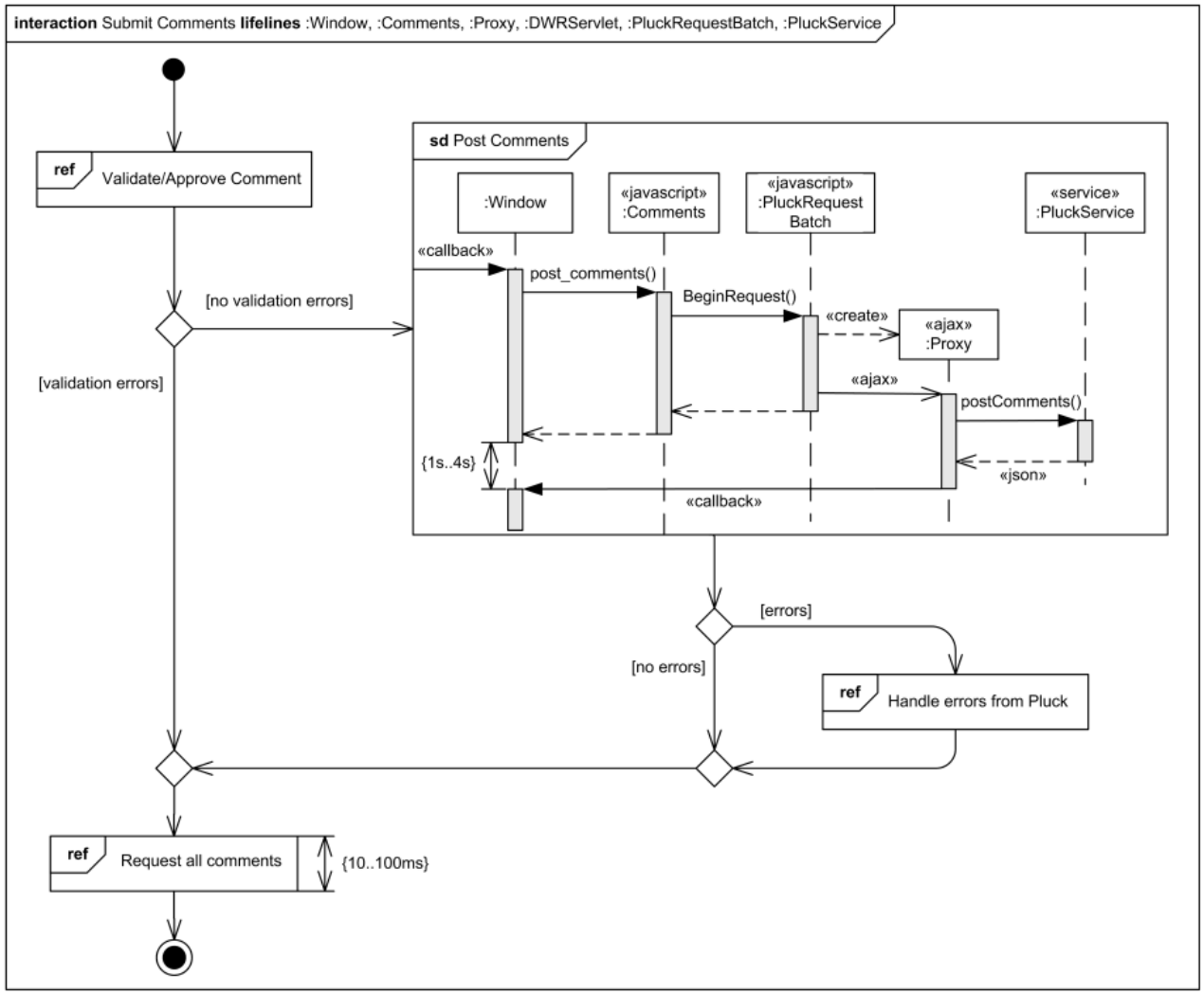


Tomado de: R. Miles and K. Hamilton, Learning UML 2.0, O'Reilly Media, 2006

Diagrama Global de Interacciones

UML

Tomado de: K. Fakhroutdinov, Interaction Overview Diagrams Examples, http://www.uml-diagrams.org/interaction-overview-diagrams-examples.html

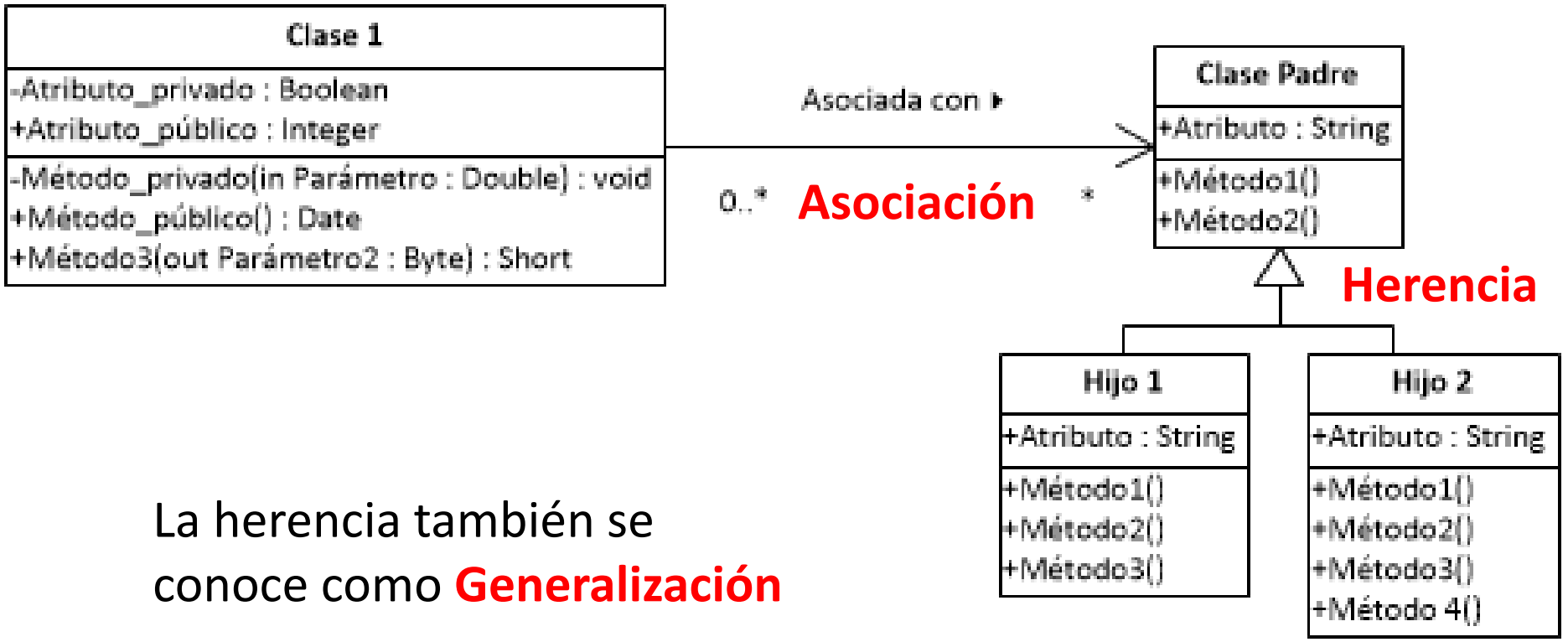


UML | Diagrama de clases

- Describe la estructura estática de nuestras clases

Nombre de la Clase
-Atributo_privado : Boolean +Atributo_público : Integer
-Método_privado(in Parámetro : Double) : void +Método_público() : Date +Método3(out Parámetro2 : Byte) : Short

Diagrama de Clases | Relaciones

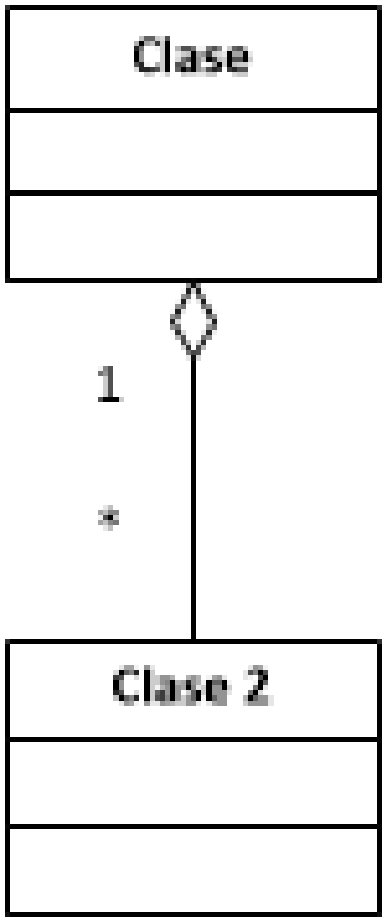


La herencia también se conoce como **Generalización**

Relaciones

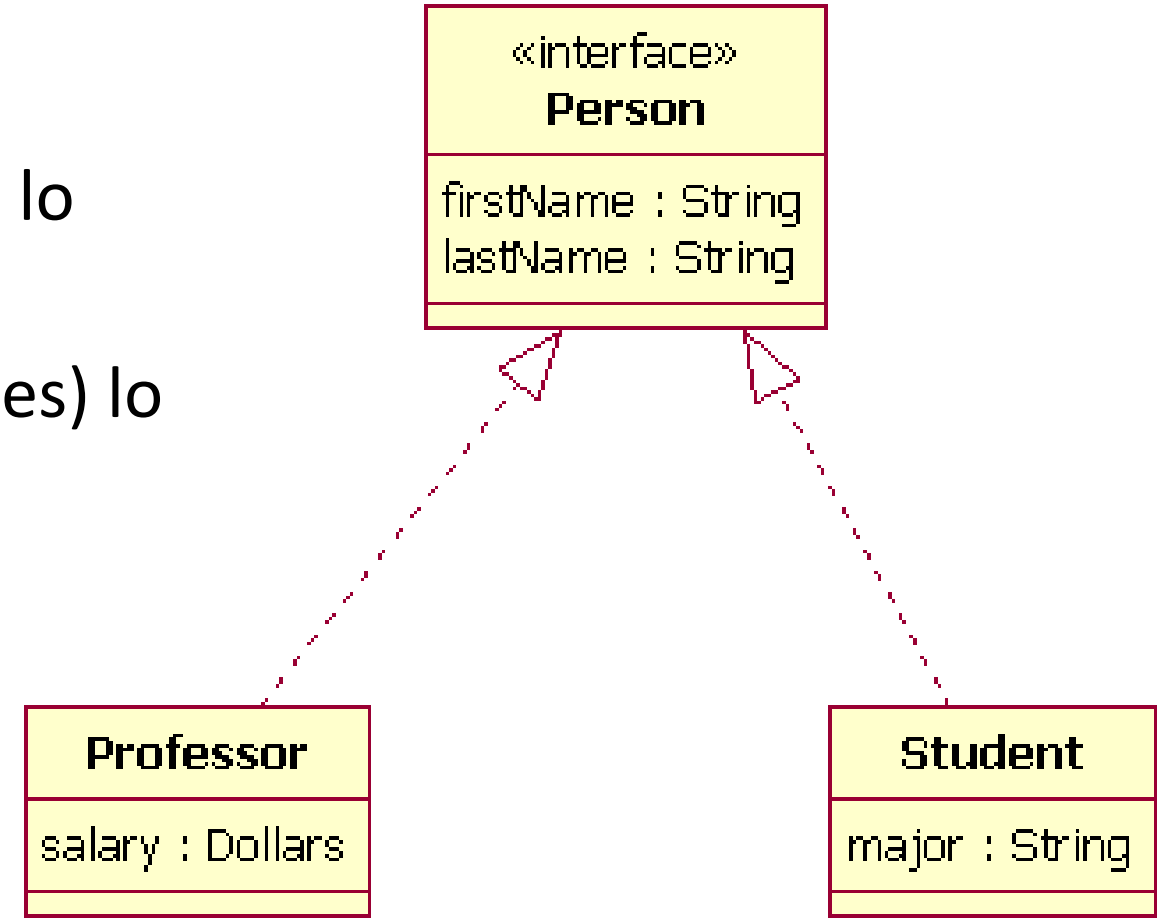
Agregación por (referencia)

- Variante: **Composición**
  - Por valor: rombo negro
- Una clase pertenece a una colección
- Ejemplo:
  - Clase tiene una colección de Clase2



Relaciones | Interfaces

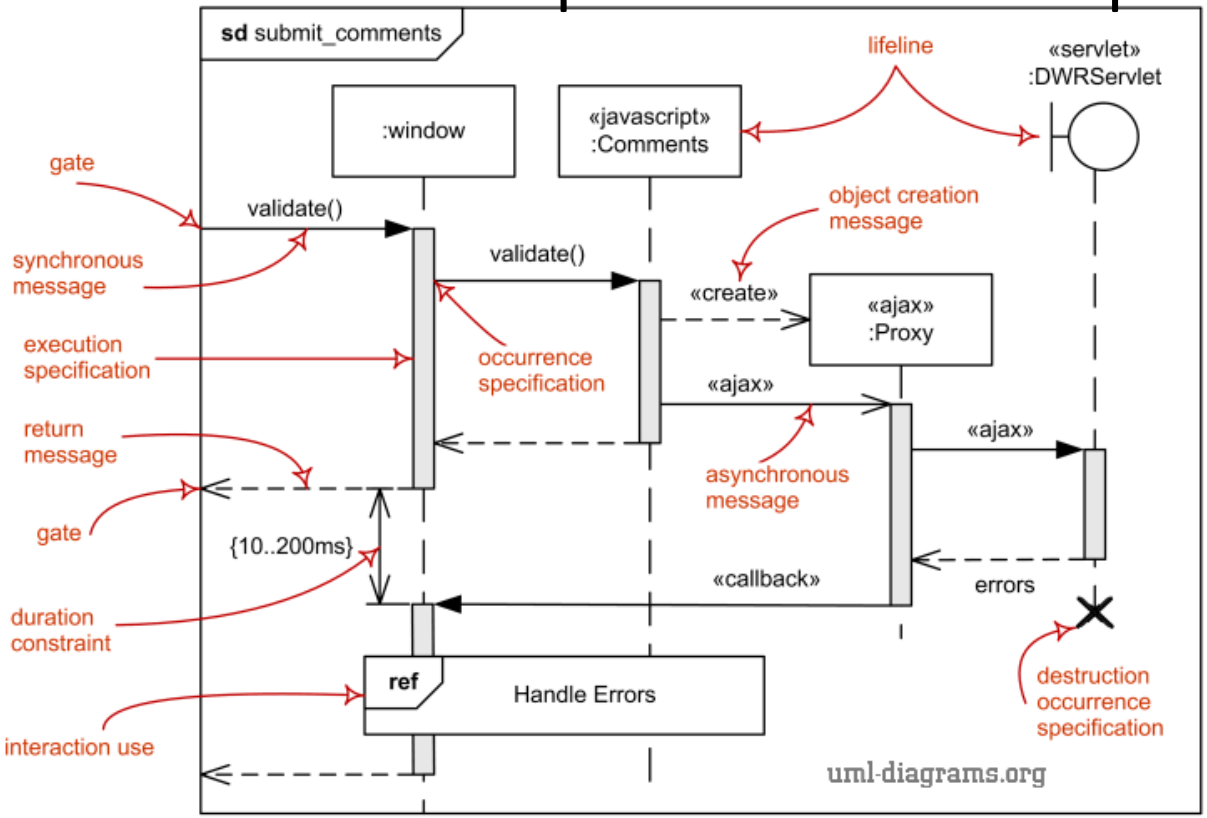
- Interfaz define comportamiento (métodos) pero no lo implementa
- Otras clases (clientes) lo implementan



Tomado de: D. Bell, UML basics: The class diagram.- An introduction to structure diagrams in UML 2, 2004, http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

UML | Diagrama de secuencia

- Muestra como las clases interactúan
  - El orden en que los eventos se producen



Tomado de: K. Fakhroutdinov, Sequence Diagrams, http://www.uml-diagrams.org/sequence-diagrams.html

UML | Ejemplo

- Ejemplo de Citas - Eventos

Mal Diseño | Problemas

- Inflexible a los cambios
- No amigable para el mantenimiento
- Requiere más esfuerzo en la programación
- Reduce la productividad del equipo

Ver Ejemplo de Añadir Nuevas Clases a Citas - Eventos

Mejorando Diseño | Refactoring

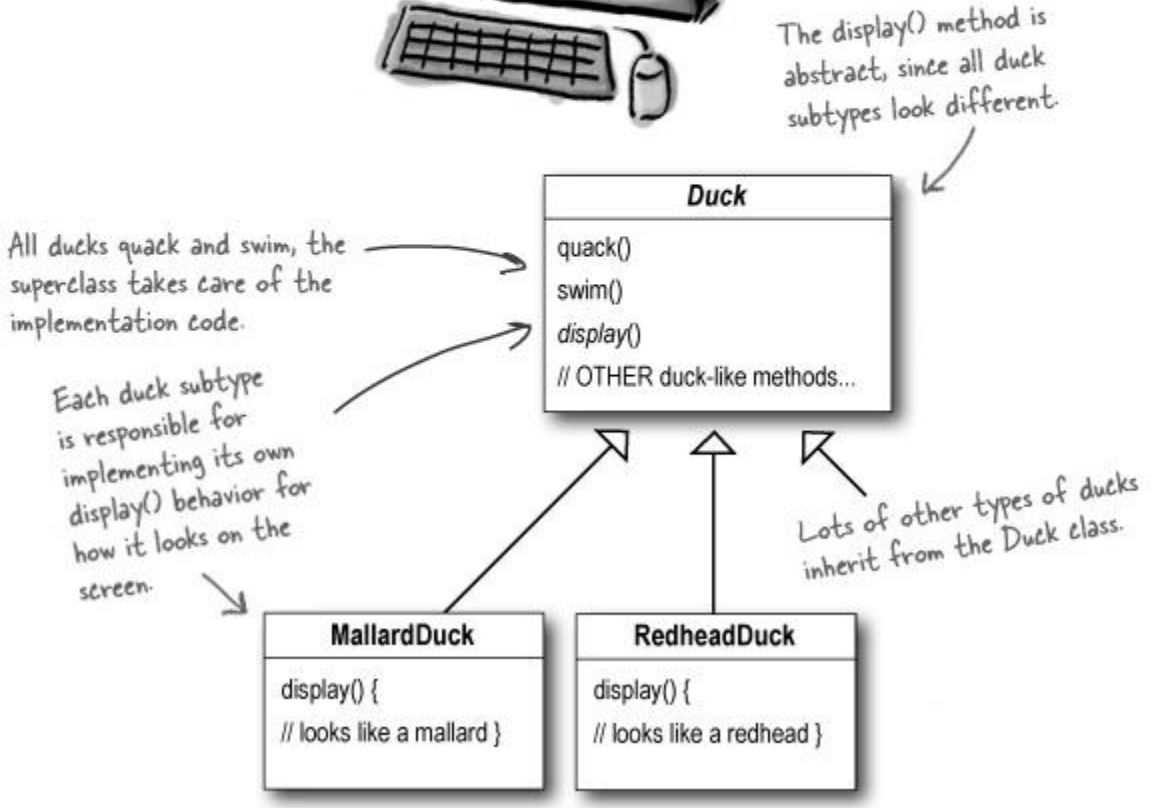
- Si encontramos problemas en diseño
  - Introducir mejoras: **refactoring**
  - Creando y eliminando clases
  - Moviendo métodos de una clase a otra
- Afecta el código
- Se lo debe realizar continuamente

Buen Diseño | ¿Cómo lograrlo?

- Asegurar criterios y **principios de buen diseño**
  - SPR, DRY, reusabilidad, mantenibilidad, otros…
- Felizmente hay guías de diseño
  - **Patrones de diseño**
- Patrón de diseño:
  - Diseño probado y reutilizable en problemas similares
  - Se basan en: aislar lo que cambia de lo fijo

Patrones | Ejemplo

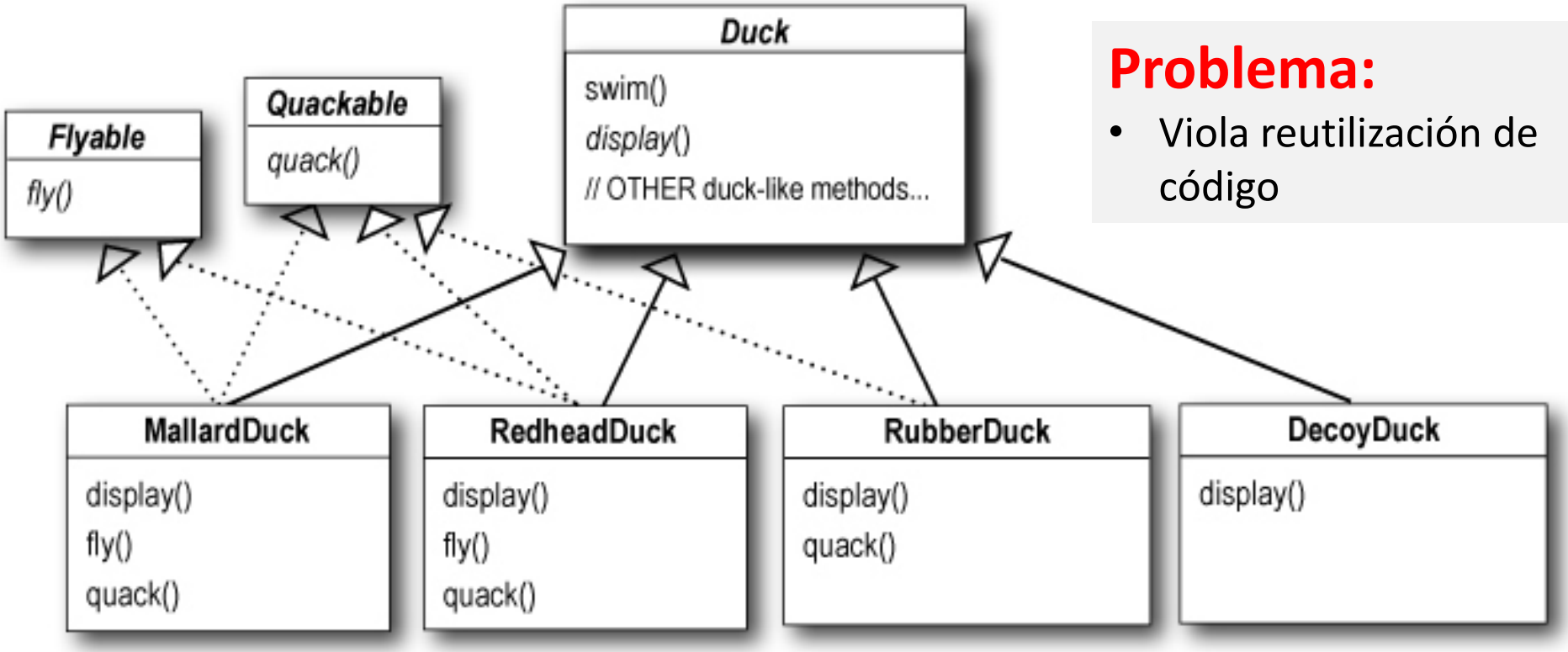
Tomado de: E. Freeman, E. Robson, B. Bates, K. Sierra, Head First Patterns Design, O'Reilly Media, 2004



Queremos:

- Añadir patos de goma
- Añadir método volar()

Solución | ¿Interfaces?



Problema:

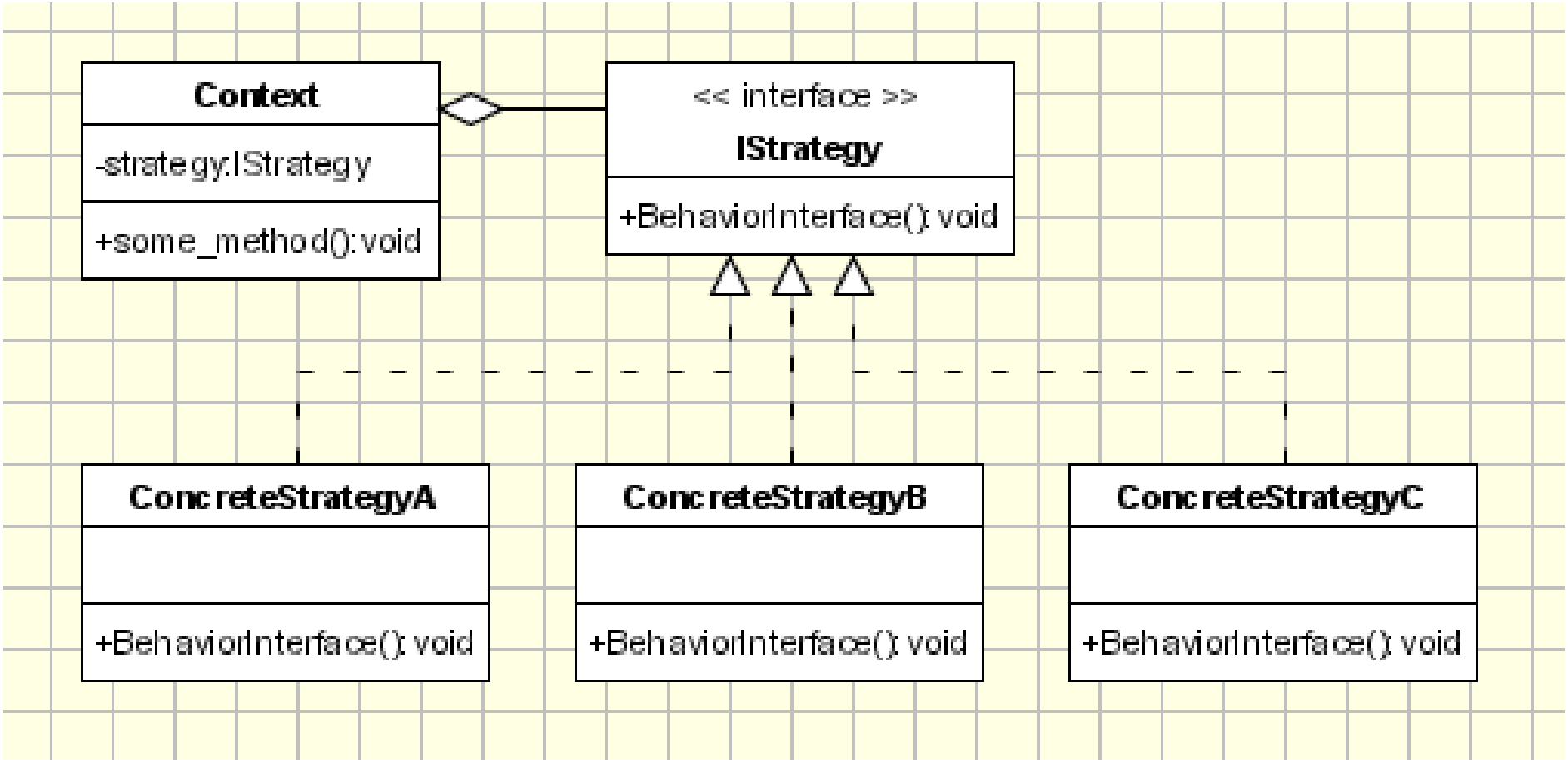
- Viola reutilización de código

Tomado de: E. Freeman, E. Robson, B. Bates, K. Sierra, Head First Patterns Design, O'Reilly Media, 2004

Patrón | Estrategia

- Aplica **encapsulación** y **polimorfismo**
- Saca y encapsula todo comportamiento que varía
- Para cada comportamiento variable:
  - Se crea interfaz
  - Se crean posibles implementaciones
- Clase principal:
  - Variable polimórfica para comportamiento
  - Método que llama a comportamiento para que se ejecute
  - Método que establece tipo de comportamiento

Patrón Estrategia | UML



Tomado de: OODesign.com, Design Patterns, http://www.oodesign.com/

Solución

Usando patrón estrategia

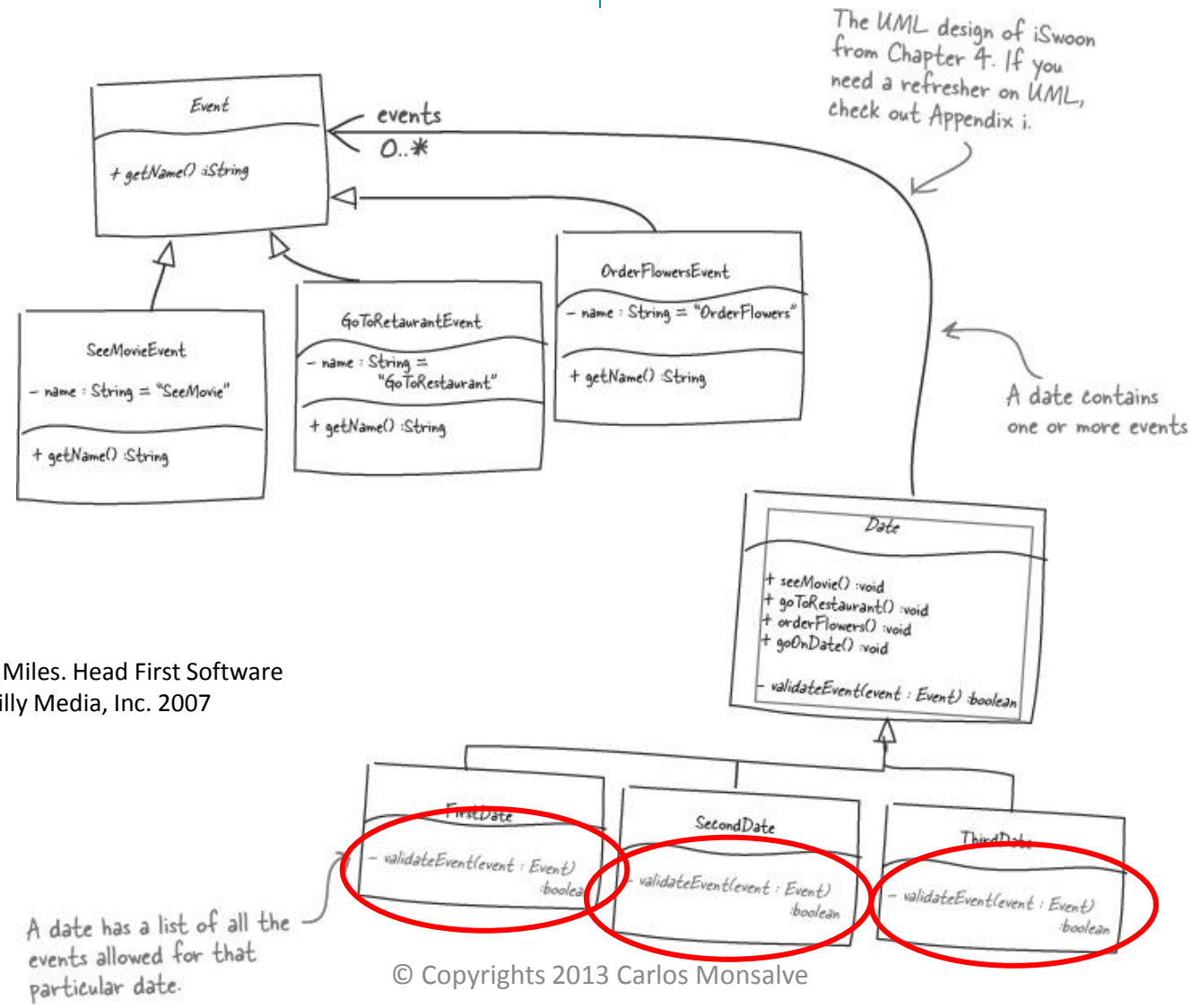
- Refactoring de simulador de patos

Principio de Diseño | DRY

- **Principio de No Te Repitas**
  - Don´t Repeat Yourself (DRY)
  - Evitar duplicación de código
  - Realizar una buena abstracción
  - Poner cosas similares en una misma localidad

¿Es nuestro diseño Citas – Eventos DRY?

Citas-Eventos | ¿DRY?



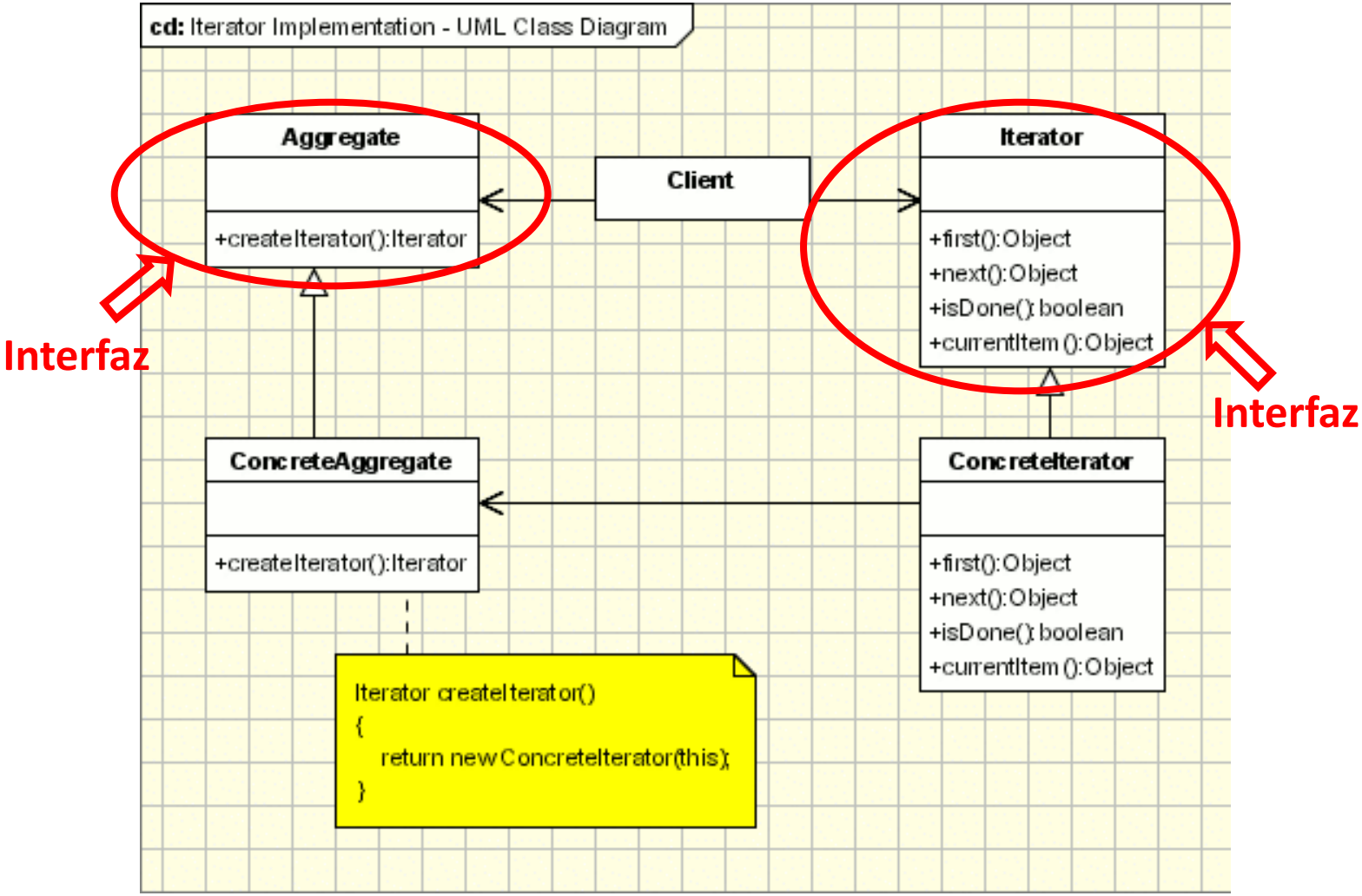
Tomado de: Pilone, Miles. Head First Software Development. O'Reilly Media, Inc. 2007

DRY | Patrón: Repetidor (iterator)

- Maneja colecciones de objetos
- Varias formas de implementar colecciones
  - Arreglos, listas, etc.
- Permite acceder a colecciones sin exponer su implementación
  - **Encapsula** iteraciones para acceder a objetos de la colección
- El repetidor (iterator) es una interfaz

Patrón Iterator | UML

Tomado de: OODesign.com, Design Patterns, http://www.oodesign.com/



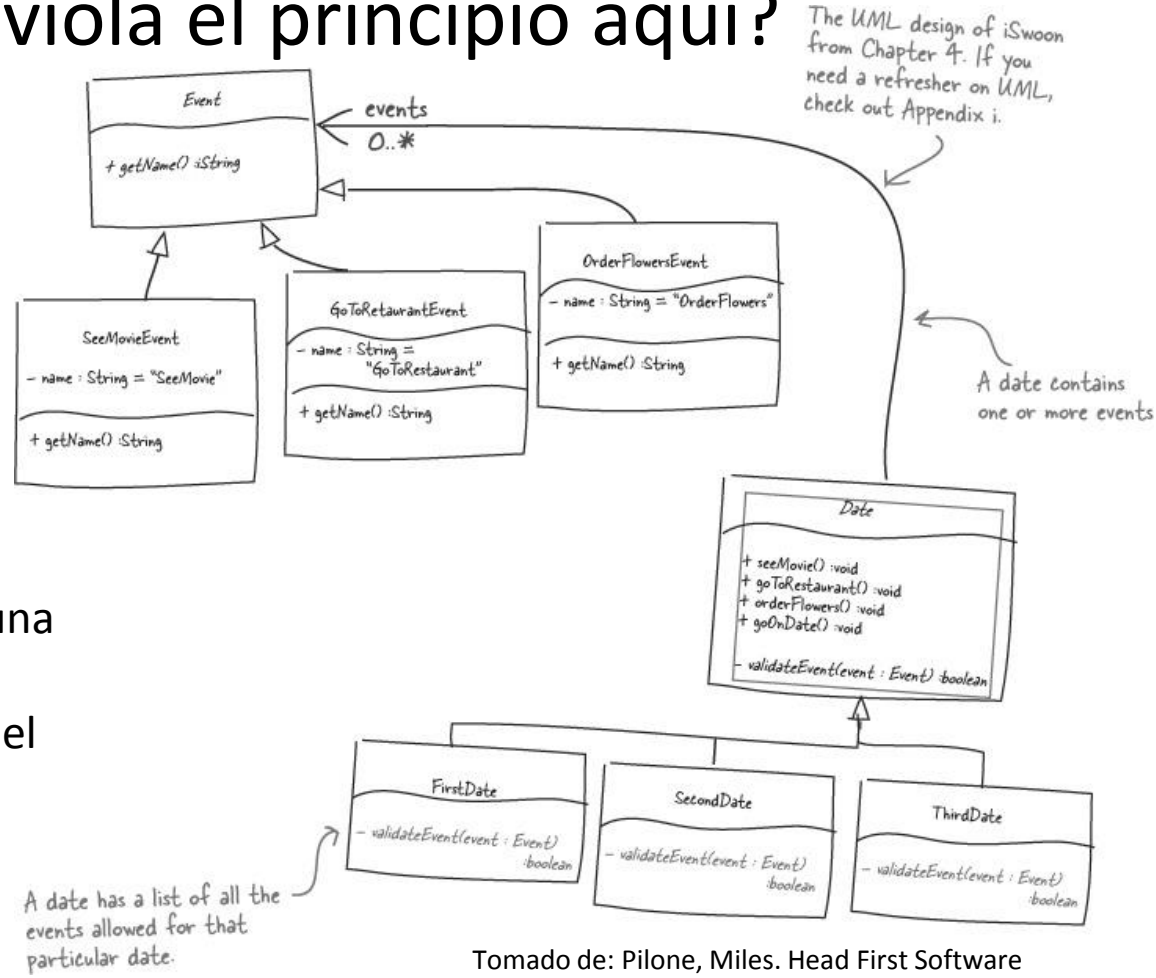
Diseño O.O. | Principios SOLID

Principio de Diseño | SRP

- **Principio de la Responsabilidad Única**
  - Single Responsibility Principle (SRP)
  - Lo que atañe a una clase se aísla en esa clase
  - Cambios en una clase son transparentes para otras
  - Entonces: Cada objeto de mi software tiene una única responsabilidad

SRP | Ejemplo

- ¿Se respeta o se viola el principio aquí? ¿Porqué?



Ayudas:

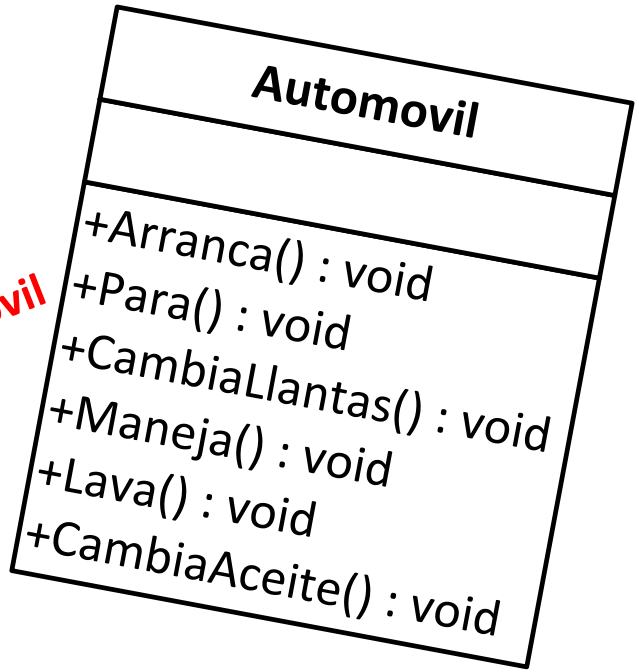
1. Considerar que sucede si añado una nueva clase evento
2. Considerar que sucede si cambio el nombre a una clase evento

Tomado de: Pilone, Miles. Head First Software Development. O'Reilly Media, Inc. 2007

SRP | Una simple prueba

- Construir esta oración para cada método:
  - El objeto NombreDeLaClase se Método a sí mismo
- Si la frase tiene sentido
  - Cumplimos con SRP
- Si no hace sentido
  - Violamos SRP

Ejercicio:
Analicemos Automóvil



Basado en: Pilone, Miles. Head First Software Development. O'Reilly Media, Inc. 2007

SRP | Otros malos síntomas

- Clase con varios métodos y varios atributos
  - Cada método utiliza 1 solo atributo
  - Cada variable es usada por 1 solo de los métodos
- Métodos que requieren grupos grandes de parámetros
  - Siempre hay que enviar y recibir grupos grandes de parámetros
- Clase que inicialmente era simple se ha vuelto compleja

Buen Diseño | Otra clave (2)

- **Principio de Interfaces Segregadas**
  - Problema: interfaz de una clase es usada por varios clientes de características diferentes
  - Interfaz debe segregarse en subconjuntos útiles para cada tipo de cliente
  - Modificar una interfaz podría requerir modificar la clase completa

Figura tomada de: Universal Multi 10 In 1 Cell Phone Game USB Charger Cable Car Charger, #00545368, http://www.miniinthebox.com

Principio ISP | Violación

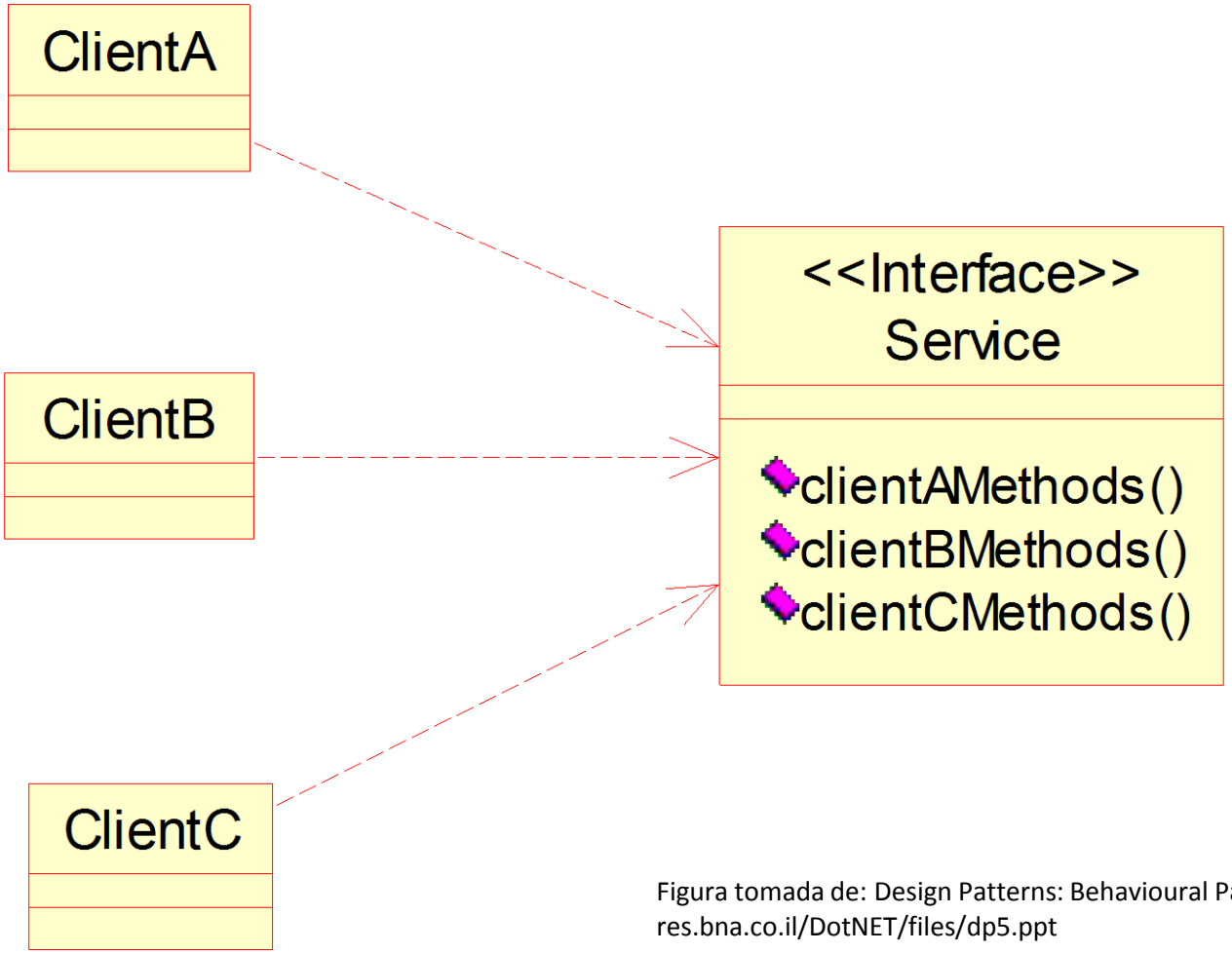


Figura tomada de: Design Patterns: Behavioural Patterns, res.bna.co.il/DotNET/files/dp5.ppt

Principio ISP | ¿Cómo mejorarlo?

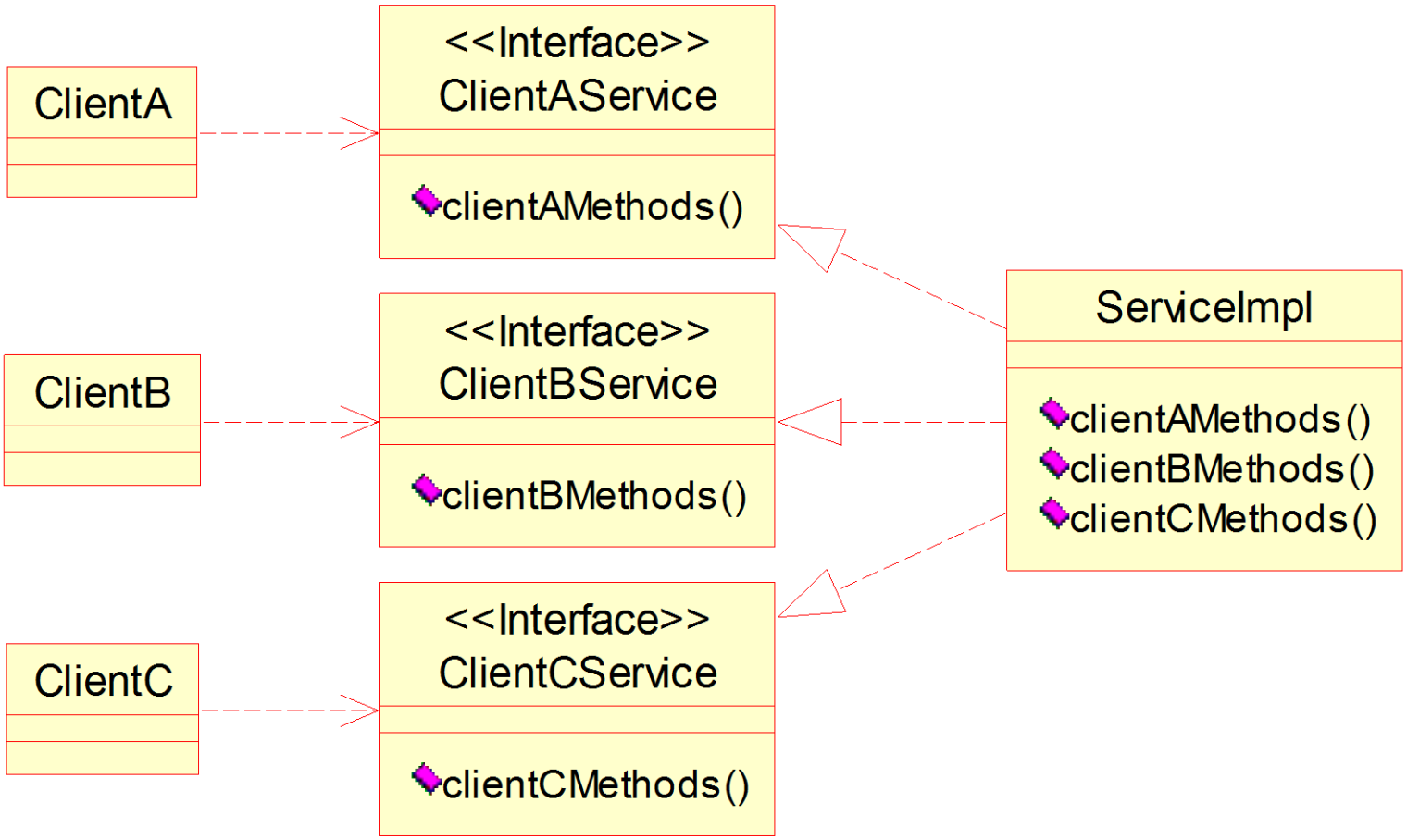


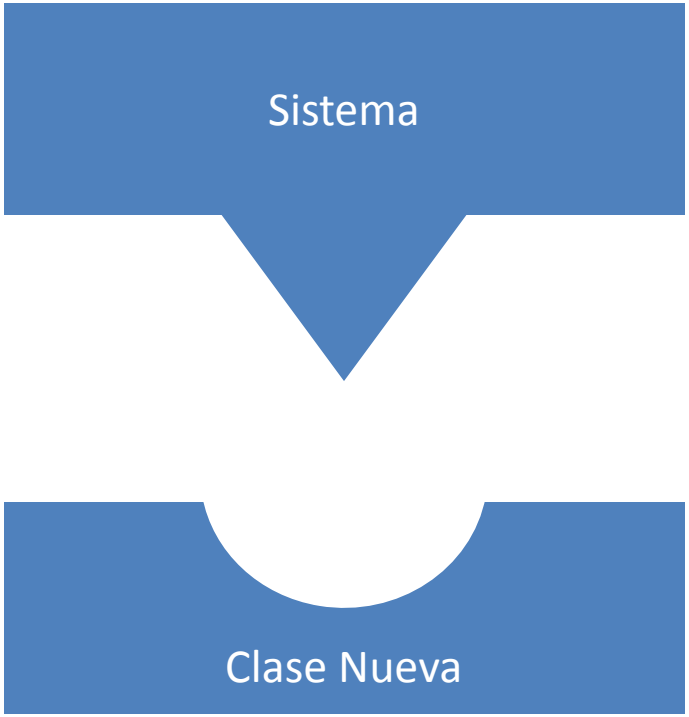
Figura tomada de: Design Patterns: Behavioural Patterns, res.bna.co.il/DotNET/files/dp5.ppt

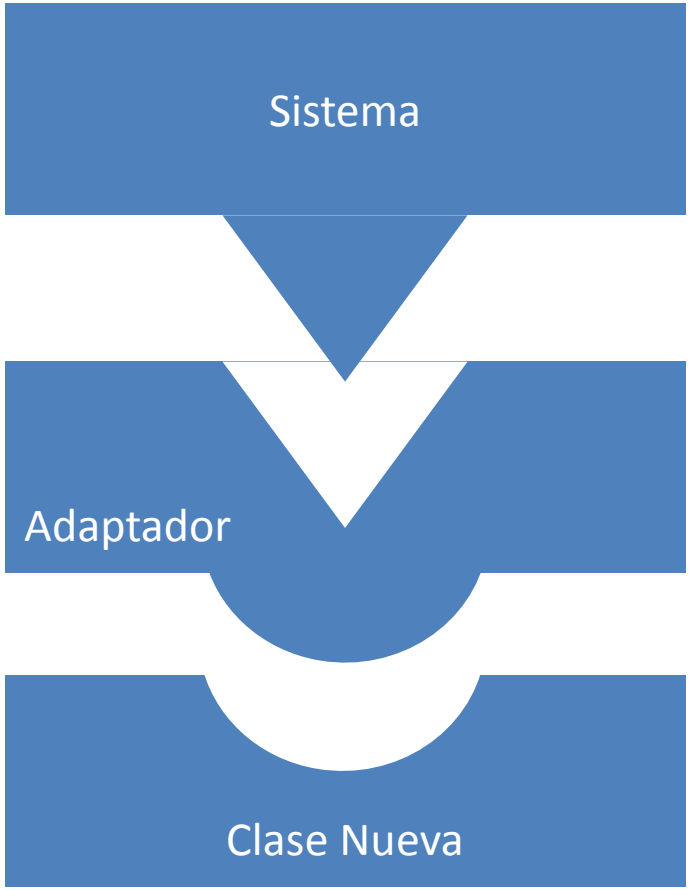
Patrón Adaptador | Características

- Envuelven a un objeto para que interfaces aparenten algo que no son
- Adaptan una interfaz según necesidades de cliente

Figura tomada de: Direct Industry, http://www.directindustry.es/prod/phihong/alimentaciones-electricas-ac-dc-adaptadores-lineales-34351-785893.html

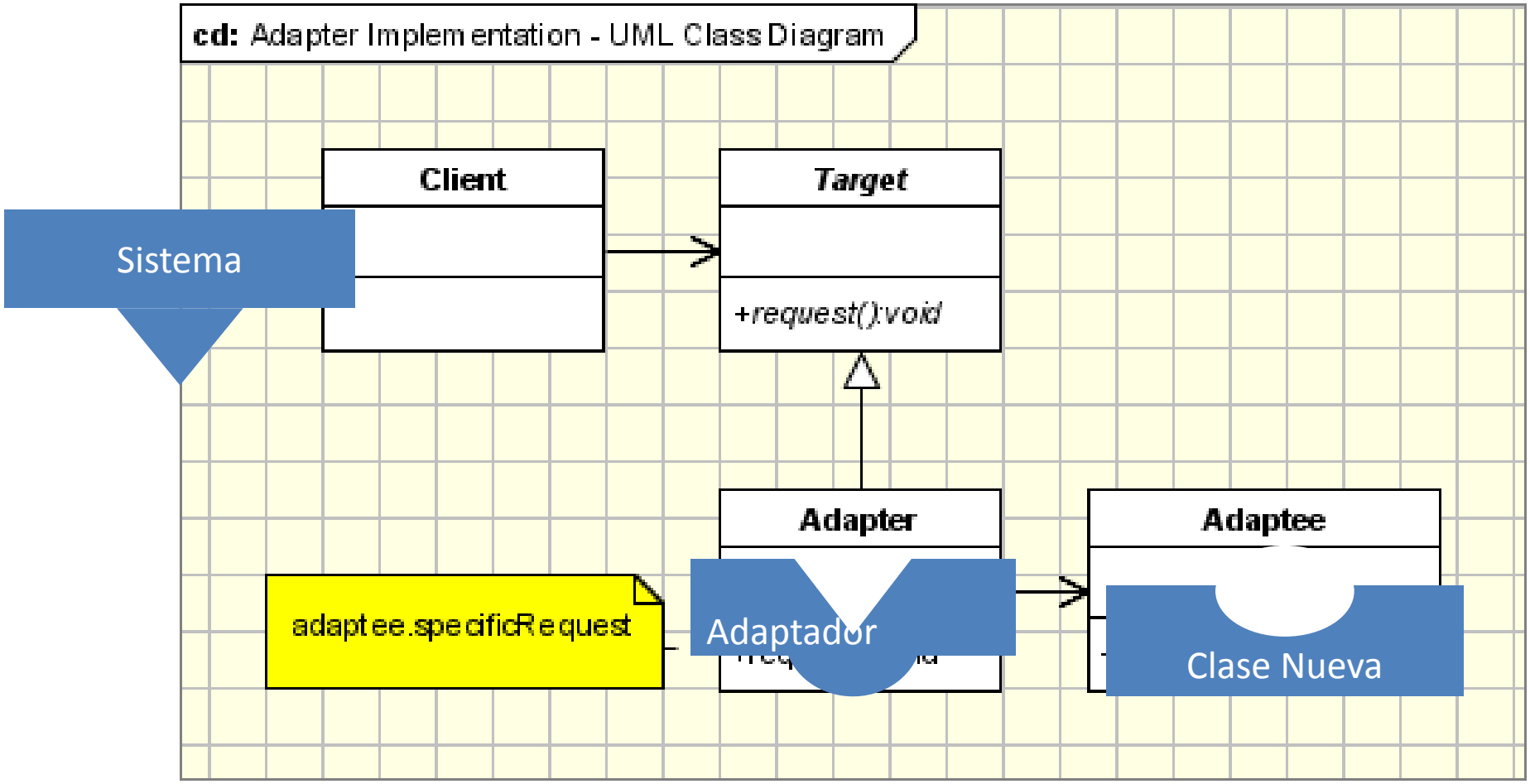
Patrón Adaptador | Idea conceptual





Basado en: Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, Head First Design Patterns, 2004, O'Reilly Media

Patrón Adaptador | UML



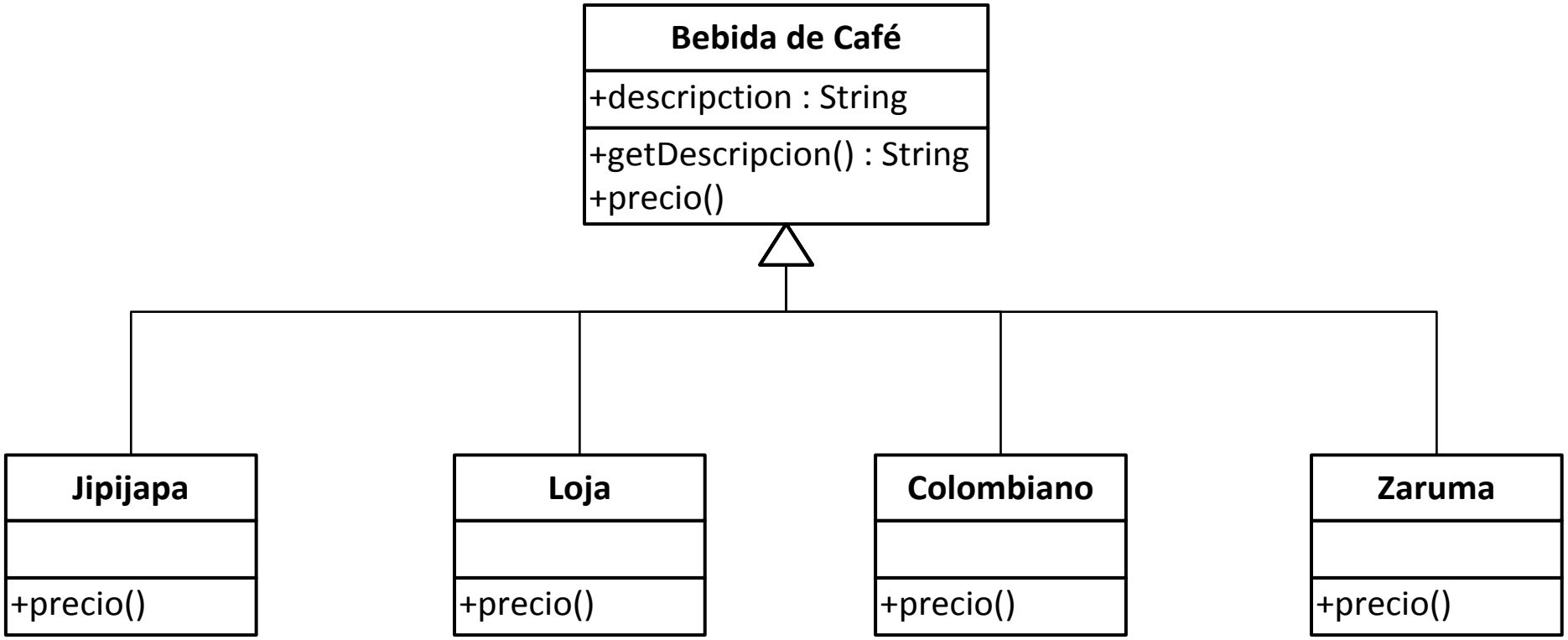
Tomado de: OODesign.com, Design Patterns, http://www.oodesign.com/

Ejemplo | Patos y Gallos

Buen Diseño | Otra clave (3)

- **Principio de Abierto/Cerrado**
  - Open Closed Principle (OCP)
  - Clase está abierta para extensiones
  - Clase está cerrada para modificaciones
  - Puedo modificar comportamiento sin alterar el código

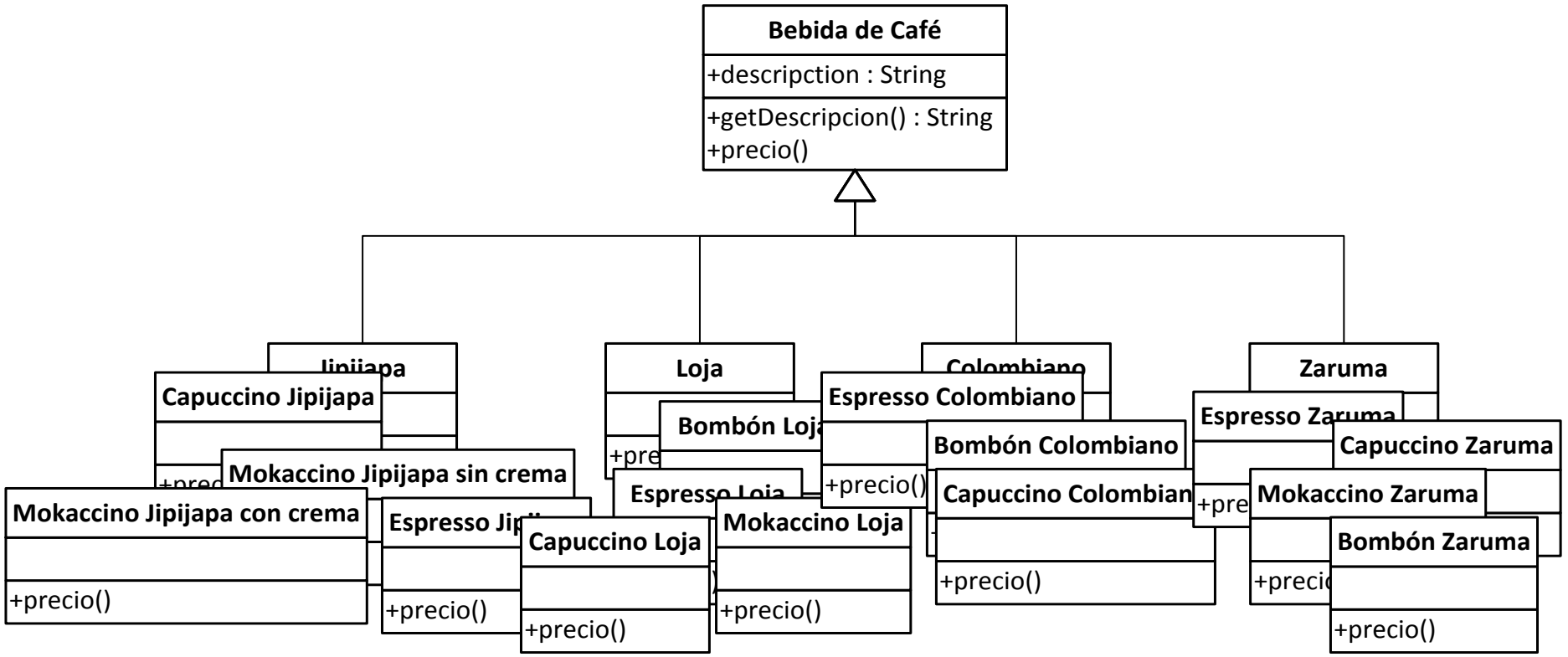
Ejemplo | Cafetería



¿Qué pasa con las variantes de bebidas?

Basado en: Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, Head First Design Patterns, 2004, O'Reilly Media

Cafetería | El problema



Basado en: Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, Head First Design Patterns, 2004, O'Reilly Media

Cafetería | ¿Mejora?

Bebida de Café
-descripcion -leche -leche_condensada -chocolate -crema
+getDescripcion() +costo() +tieneLeche() +setLeche() +tieneChocolate() +setChocolate() +tieneCondensada() +setCondensada() +tieneCrema() +setCrema()

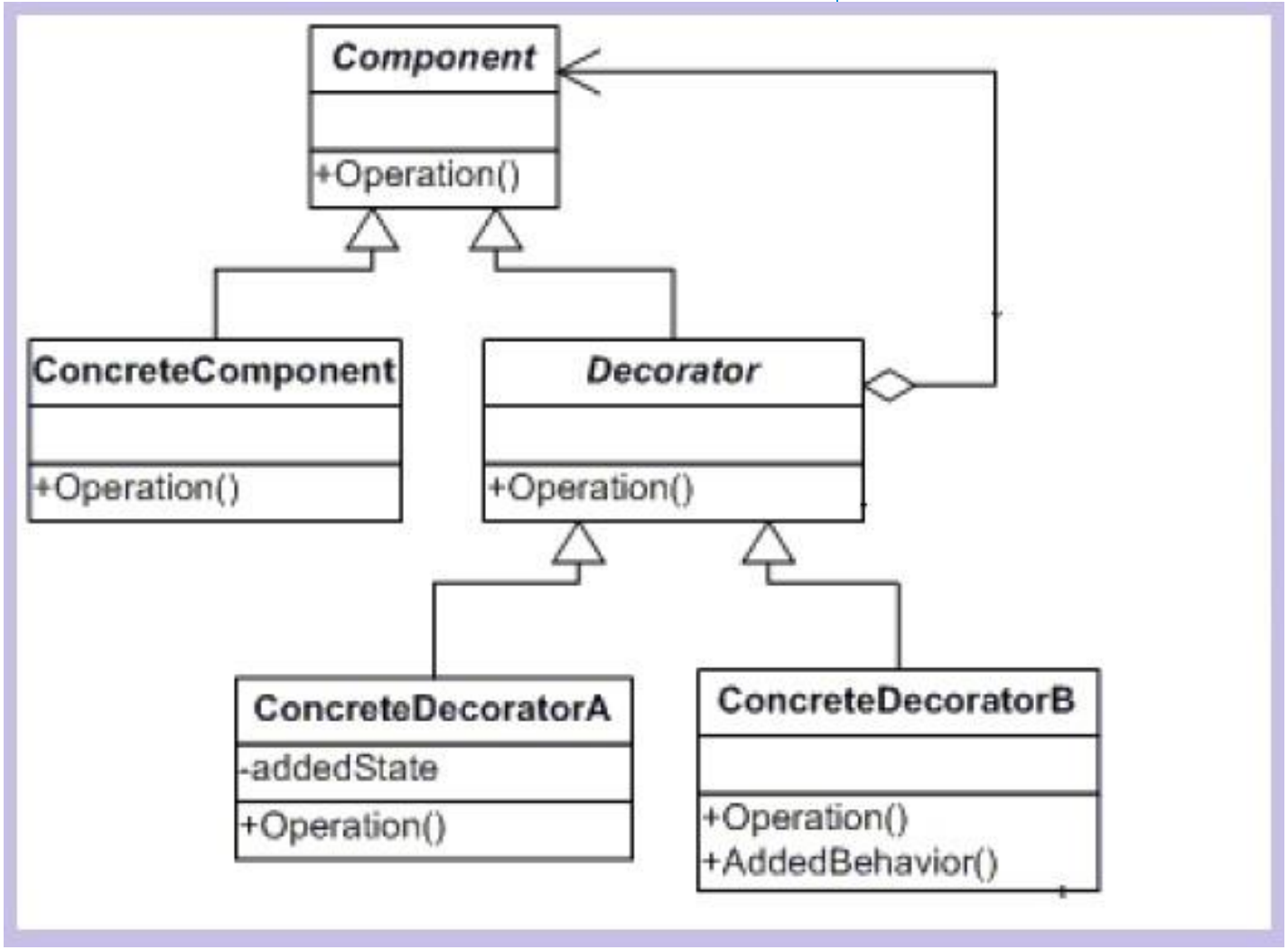
- ¿Y si los productos (no café) cambian de precio?
- ¿Y si queremos añadir nuevos productos (ejemplo: licores, tipos de leche?
- ¿Y si la base no es café sino te?
- ¿Y si quiero hacer cafés fríos?
- ¿Y si quiero un espresso doble?

Basado en: Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, Head First Design Patterns, 2004, O'Reilly Media

Patrón | Decorador

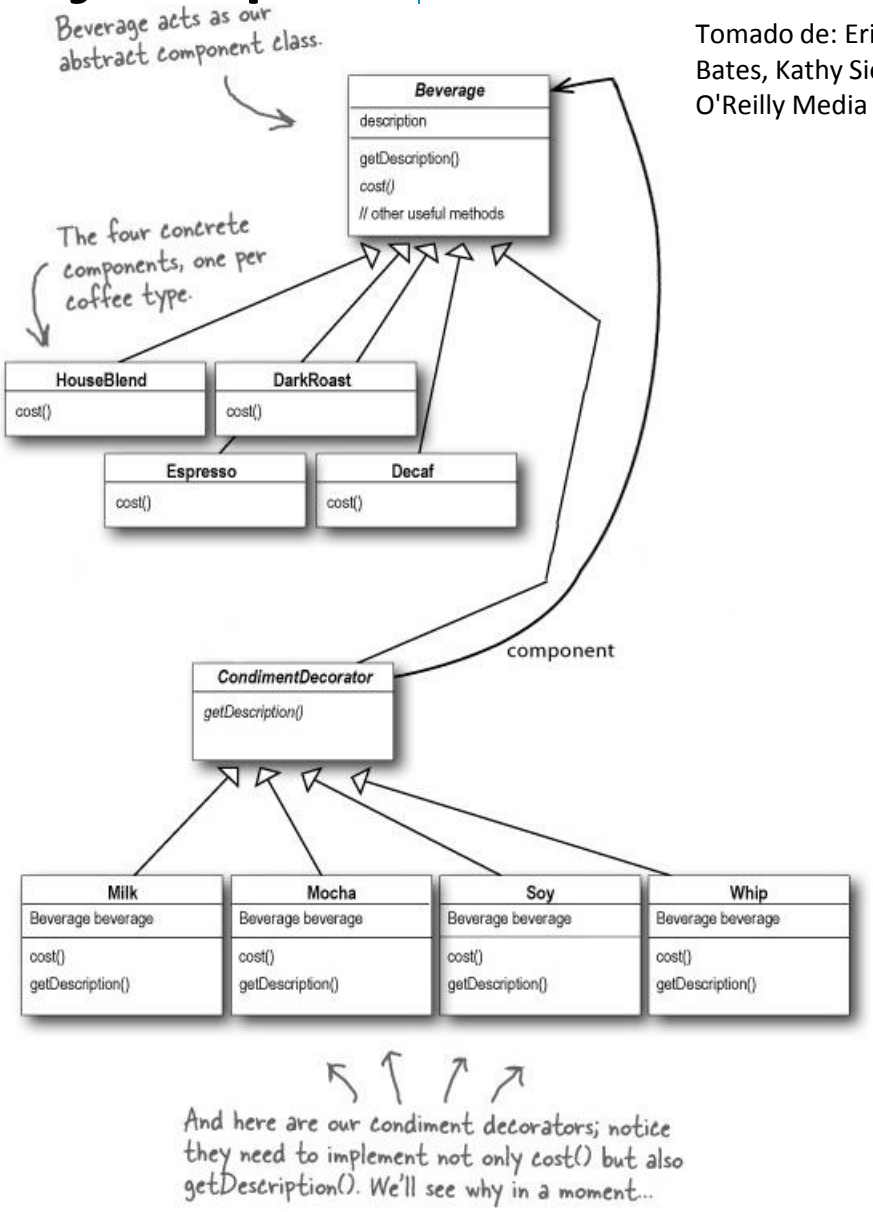
- La clase no se modifica en su codificación
  - La “decoramos” en tiempo de ejecución
- Los decorados envuelven a la clase modificada
- Los decoradores adoptan el tipo de dato de la clase que envuelven
- Un mismo objeto puede ser envuelto por varios decoradores

Decorador | UML



Tomado de: Rahul Rajat Singh, Understanding and Implementing Decorator Pattern in C#, Code Project For those who code, 2012, http://www.codeproject.com/Articles/479635/UnderstandingplusandplusImplementingplusDecoratorp

Ejemplo | Cafetería



Tomado de: Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, Head First Design Patterns, 2004, O'Reilly Media