DOCUMENTACION DE ARQUITECTURA – VISTA DE MODULOS

ELSA ESTEVEZ
UNIVERSIDAD NACIONAL DEL SUR
DEPARTAMENTO DE CIENCIAS E INGENIERIA DE LA COMPUTACION

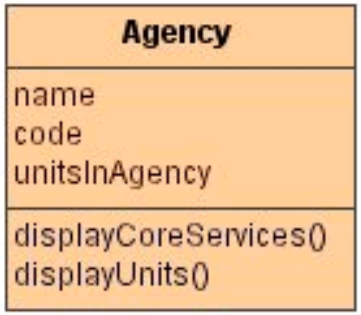
CONTENIDO

ELEMENTOS

- Un elemento de una vista es la implementación de una unidad de software que provee un conjunto de responsabilidades coherentes.
- Se entiende por unidad de software:
  - ✓ desde unidades de un lenguaje de programación - clases de Java o C#, stored procedures de BD, programas C, etc.
  - ✓ hasta agrupaciones de dichas unidades - packages Java, namespaces C#
- Los elementos pueden ser agregados – e.g. integrando funcionalidad o descompuestos – refinando su definición
- Cada elemento tiene una responsabilidad.
- Una responsabilidad es la contribución al sistema que se espera de cada elemento – e.g. las acciones que realiza, el conocimiento que mantiene

ELEMENTOS – EJEMPLO

Clase

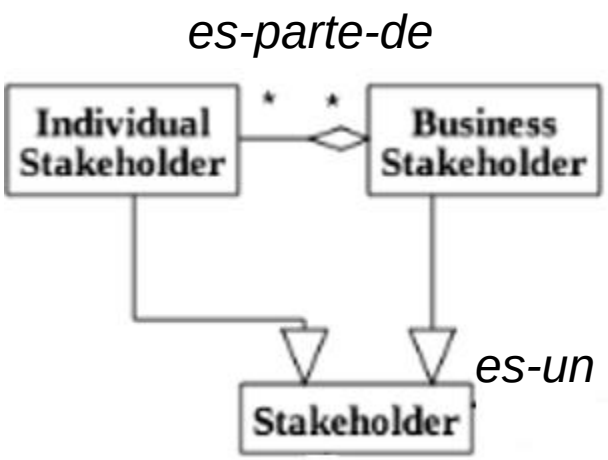


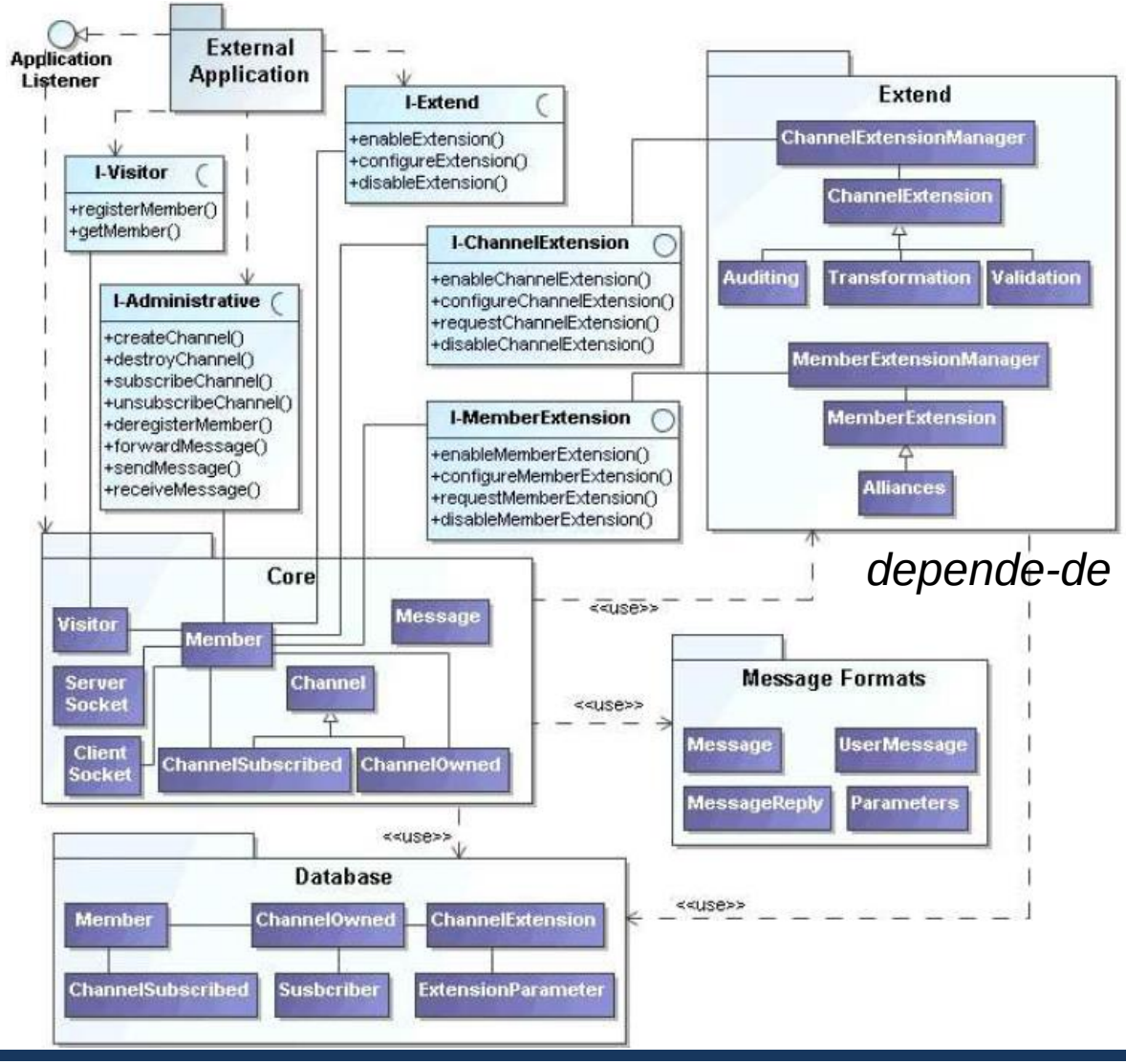
Componente

RELACIONES

es-parte-de	define una relación parte/todo entre el sub-módulo (la parte) y el módulo agregado (el todo)
depende-de	define una relación de dependencia entre módulos. Existen distintos tipos. Cada "estilo de módulo" realiza una interpretación diferente
es-un	define una relación de generalización/especificación entre un módulo más especifico (hijo) y uno más general (padre)

RELACIONES – EJEMPLO



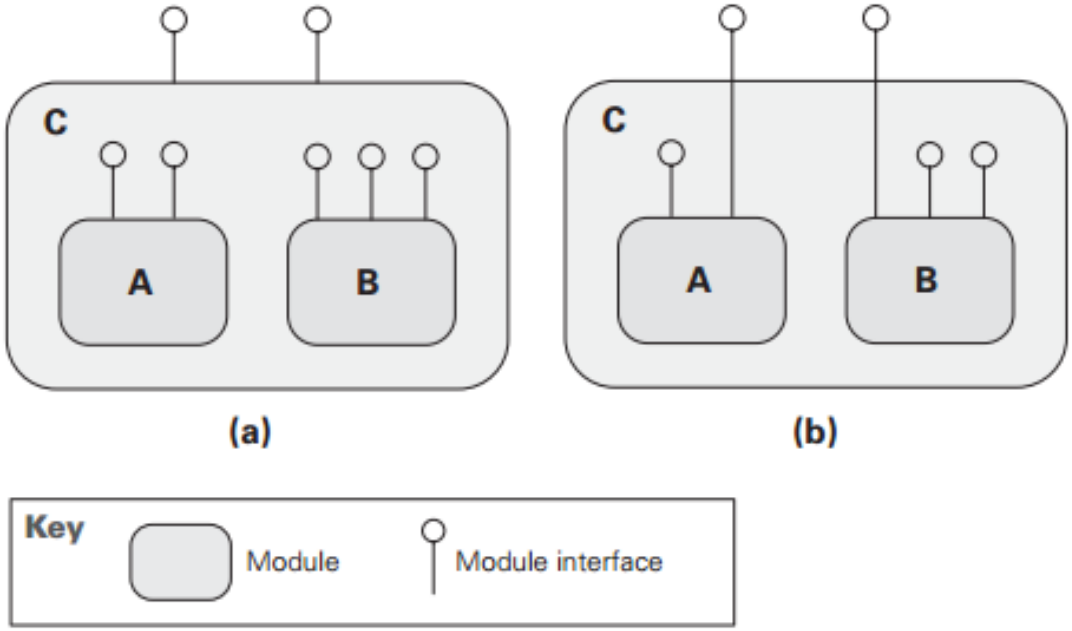


PROPIEDADES

Las propiedades de los módulos que ayudan a guiar la implementación o son información relevante para el análisis deben ser parte de la documentación

Nombre	○ sugiere algo acerca del rol del módulo en el sistema ○ puede reflejar su posición dentro de la jerarquía de descomposición – e.g. Cliente.ValidarDatos
Responsabilidad	○ es la forma de identificar su rol en el sistema y establecer su identidad más allá del nombre ○ la responsabilidad de un módulo se debe especificar claramente
Visibilidad de Interfaces	○ determinar la forma en que las interfaces del módulo son expuestas
Información de implementación	○ se provee para dirigir el desarrollo y la construcción Ejemplos: mapeo a las unidades de código, información para el testing, información de gestión, restricciones de implementación

PROPIEDADES – EJEMPLO

Nombre	C
Responsabilidad	Proveer los servicios A y B
Visibilidad de Interface	
Información de Implementación	Los módulos A y B deben ser implementados en Java

USOS

- proveer un plano para la construcción del código
- explicar la funcionalidad del sistema y la estructura del código base
- facilitar el análisis de impacto
- dar soporte al análisis de trazabilidad de requerimientos
- ayudar en la planificación del desarrollo incremental
- dar soporte a la definición de asignaciones de trabajo, planificación de implementaciones e información de presupuesto
- mostrar la estructura de la información a ser persistida

NOTA: como brinda una representación del sistema estática más que en ejecución, NO es útil para analizar performance, confiabilidad, etc.

NOTACIONES

1) Informal

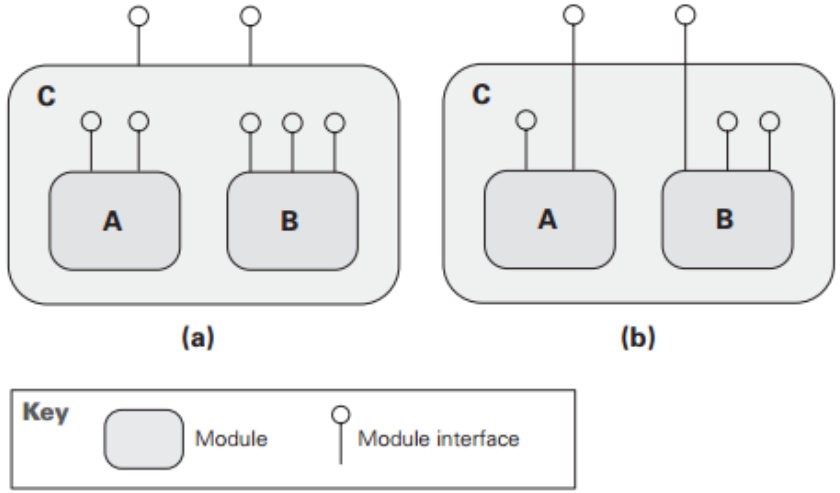
2) UML

3) Matriz de Dependencia

4) Diagramas de Entidad Relación

NOTACION – INFORMAL

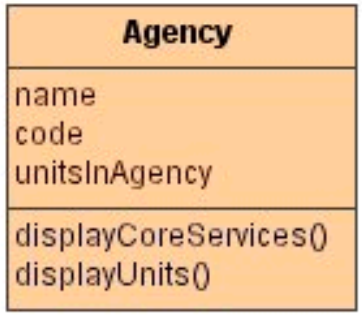
- Cajas que representan módulos y líneas que representan relaciones
- Listado textual de módulos con la descripción de sus responsabilidades.

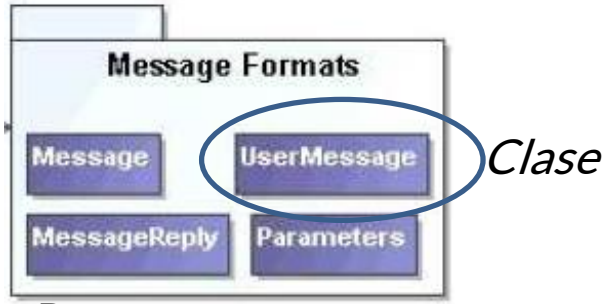
Nombre	C
Responsabilidad	Proveer los servicios A y B
Visibilidad de Interface	

NOTACION – UML

- UML provee una variedad de constructores que pueden ser usados para representar módulos

Clase con atributos y operaciones



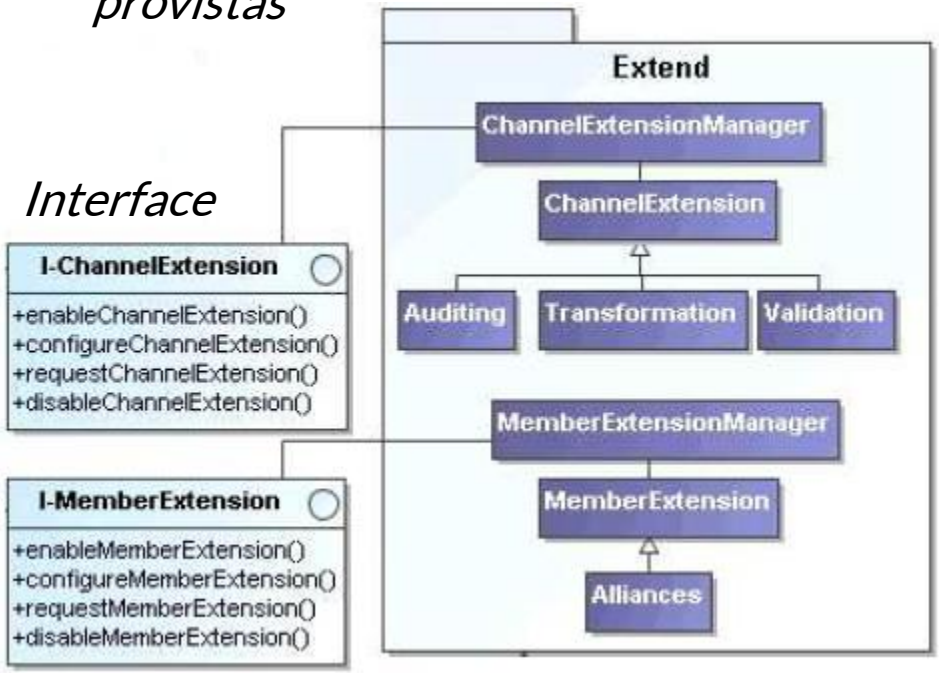


Clase

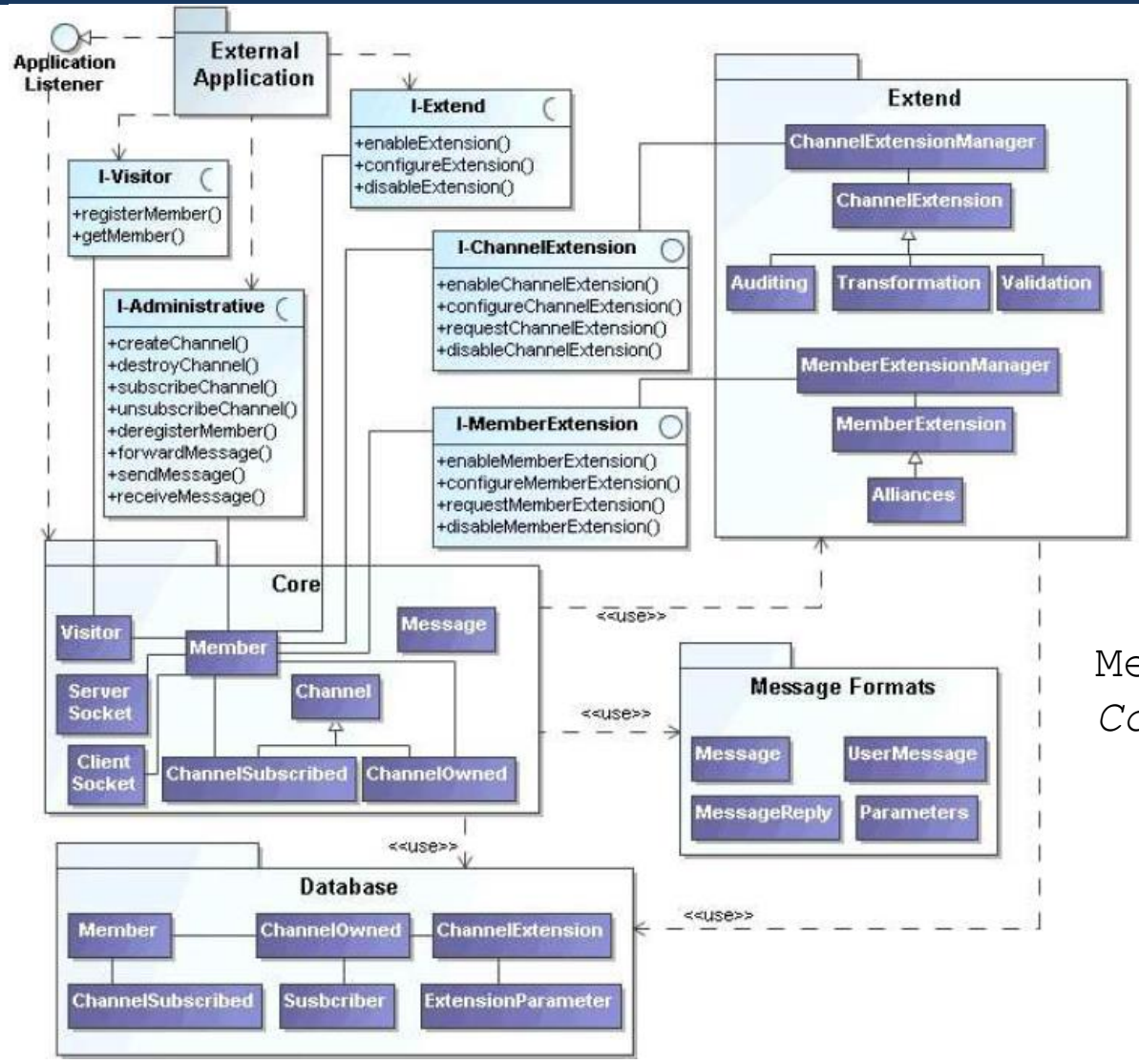
Paquete

Clase con interfaces provistas

Interface



NOTACION UML – EJEMPLO DE RELACIONES



`Core` *depends-of* `Extend`

`Message` *is-part-of* `Core`

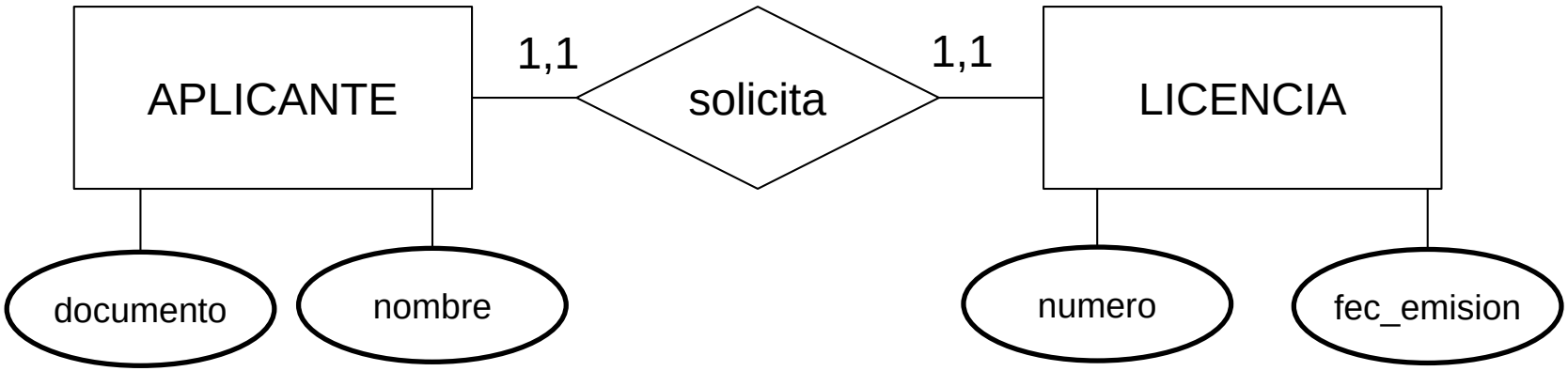
NOTACION – MATRIZ DE DEPENDENCIA

- Matriz cuadrada donde las filas y las columnas son los módulos
- El contenido de la celda (i,j) es distinto de cero si existe una dependencia entre el módulo i y el módulo j
- Generada automáticamente por herramientas de modelado o por entornos de desarrollo

used module \ using module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	**1**	0	0	**1**	0	0	0
cc	0	**1**	0	0	0	0	0
god	**1**	0	0	0	0	0	0
restart	0	0	0	**1**	0	0	0
common	**1**	**1**	0	0	0	0	0
vo	**1**	**1**	**1**	0	0	**1**	0

NOTACION – DIAGRAMAS DE ENTIDAD RELACION

- Vista especifica para modelado de datos



RELACION CON OTRAS VISTAS

Las vistas de módulos son comúnmente mapeadas con vistas de componentes y conectores.

Las unidades de implementación que se muestran en las vistas de módulos se mapean con componentes ejecutables.

Las vistas de módulos proveen elementos de software que se mapean a distintos elementos no-software del entorno del sistema en las vistas de despliegue.

RESUMEN – 1

Los módulos reflejan la forma en que un sistema de software es descompuesto en unidades administrables de responsabilidad, que es uno de los aspectos mas importantes de la estructura de un sistema

Los módulos se relacionan entre sí por las siguientes relaciones: *es-parte-de* , *depende-de*, *es-un*.

Una vista de módulos presenta un esquema (plan) para el código fuente y el modelo de datos.

Debería haber al menos una vista de módulos en el paquete de documentación.

No se debería depender del nombre del modulo para definir las obligaciones funcionales de un modulo – usar la propiedad de *responsabilidad*.

RESUMEN – 2

Documentar la(s) interface(s) del módulo para establecer el rol del módulo en el sistema.

Las vistas de módulos son comúnmente mapeadas a vistas de componentes y conectores.

En general, un módulo puede participar en muchos componentes de ejecución.

CONTENIDO

ESTILO DE DESCOMPOSICION

DESCRIPCION	○ es usado para descomponer un sistema en unidades de implementación ○ describe la organización del código como módulos y sub-módulos y muestra cómo las responsabilidades del sistema son particionadas
ELEMENTOS	Módulo
RELACIONES	La relación de descomposición es una forma de la relación "*es-parte-de*". Se debe especificar el criterio de descomposición.
RESTRICCIONES	○ no se permiten ciclos en el gráfico de descomposición ○ un módulo sólo puede tener un padre

ESTILO DE DESCOMPOSICION – USOS

- Para razonar acerca de y comunicar la estructura del software en porciones manejables
- Para facilitar la asignación de tareas
- Para razonar acerca de la localización de cambios

ESTILO DE DESCOMPOSICION – CRITERIOS

- Implementación de ciertos atributos de calidad - por ejemplo, para satisfacer la modificabilidad del sistema, usar el principio de ocultamiento de información para encapsular aspectos del sistema que pueden cambiar en un módulo separado
- Decisiones construir versus comprar – el uso de módulos pre-existentes hace que la funcionalidad restante sea descompuesta alrededor de la existente
- Implementación de líneas de producto – para soportar eficientemente la implementación de productos de una misma familia distinguiendo módulos comunes a la familia de los específicos de un producto
- Asignación de equipo – para permitir la implementación de diferentes responsabilidades en paralelo, se puede asignar la implementación de módulos separados a distintos equipos

ESTILO DE DESCOMPOSICION – NOTACION

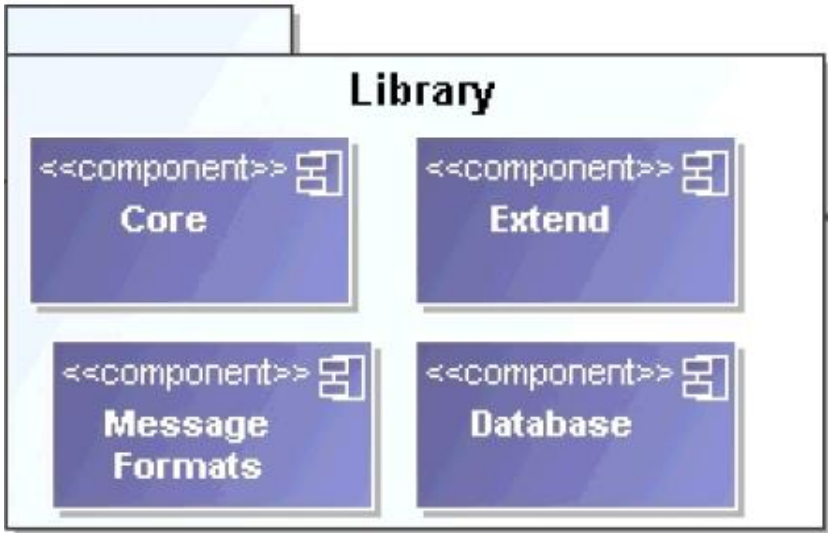
Notación Informal

- Módulos representados como cajas con nombres que pueden contener otras cajas con nombres
- Un listado de nombres de módulos usando indentación para indicar la relación es-parte-de

```
Library
    Core
    Extend
    Message_Formats
    Database
```

Notación Semiformal – UML

- Se puede usar el paquete para mostrar módulos que contienen otros módulos
- La descomposición se muestra como:
  - ✓ módulos anidados
  - ✓ una sucesión de dos diagramas, donde el segundo muestra detalles de un modulo contenido en el primero



ESTILO DE DESCOMPOSICION – EJEMPLO 1

SISTEMA:
Army Training Information Architecture-Migrated (ATIA-M)

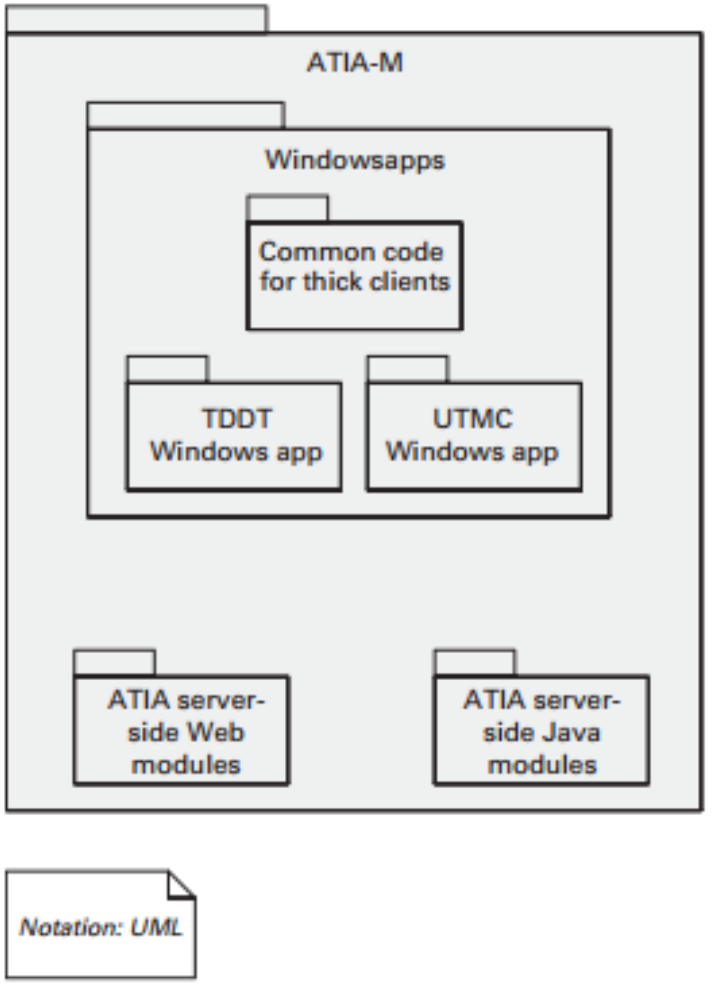
DESCRIPCION:

- es una gran aplicación JEE que da soporte al entrenamiento en el ejército de EEUU
- tiene clientes pesados - aplicaciones de escritorio Windows desarrolladas en C#
- se comunican con componentes JEE del lado del servidor usando web services

ESTILO DE DESCOMPOSICION – EJEMPLO 2

DESCOMPOSICION:

- `Windowsapps` - contiene el código de los clientes pesados. Tiene 3 submódulos, Training and Doctrine Developement Tool (`TDDT`), Unit Training Management Configuration (`UTMC`) y un módulo separado con código común usado en apps Windows. `TDTT` y `UTMC` son 2 apps Windows originalmente planificadas.
- `ATIA server-side Web modules` - módulos no-Java en el servidor(JSP, Javascript, HTML y applets)
- `ATIA server-side Java modules` - todo el código Java que corre del lado del servidor.

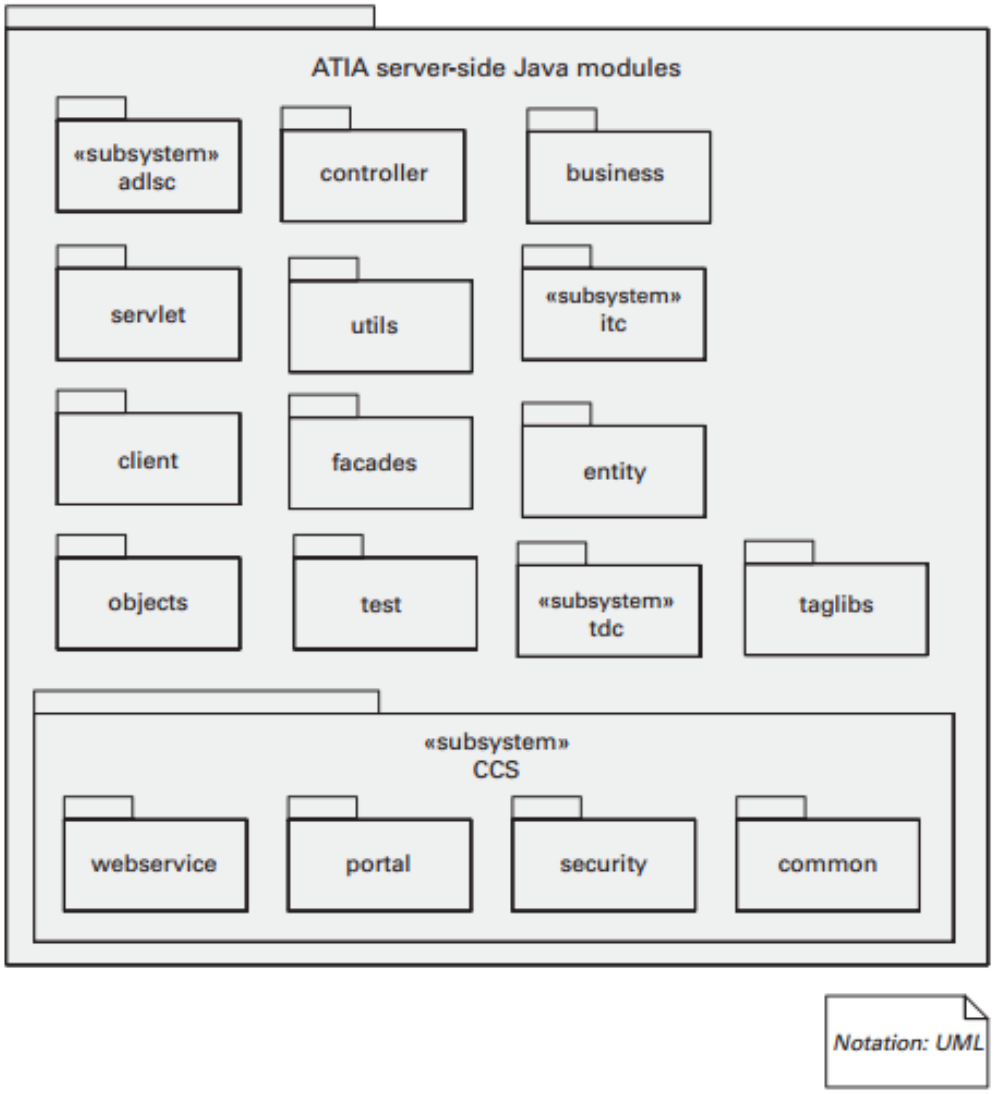


ESTILO DE DESCOMPOSICION – EJEMPLO 3

DESCOMPOSICION

ATIA server-side Java modules

Refinamiento del módulo Java del lado del servidor para mostrar cómo se descompone en sub-módulos



CONTENIDO

ESTILO DE USOS

DESCRIPCION	○ muestra cómo los módulos dependen unos de otros ○ un módulo dependen de otro si su correctitud depende de la correctitud del otro
ELEMENTOS	Módulo
RELACIONES	La vista documenta la relación usa, la cual es una forma de la relación depende-de. Un módulo A usa un módulo B si A depende de la presencia de B correctamente funcionando para satisfacer sus propios requerimientos.
RESTRICCIONES	No tiene restricciones topológicas. Sin embargo, si las relaciones de uso presentan ciclos, amplios abanicos, o largas cadenas de dependencia, la capacidad de que la arquitectura sea entregada en subconjuntos incrementales se verá dañada.

ESTILO DE USOS – USOS

- planificar el desarrollo incremental
- debugging y testing
- medir los efectos de los cambios

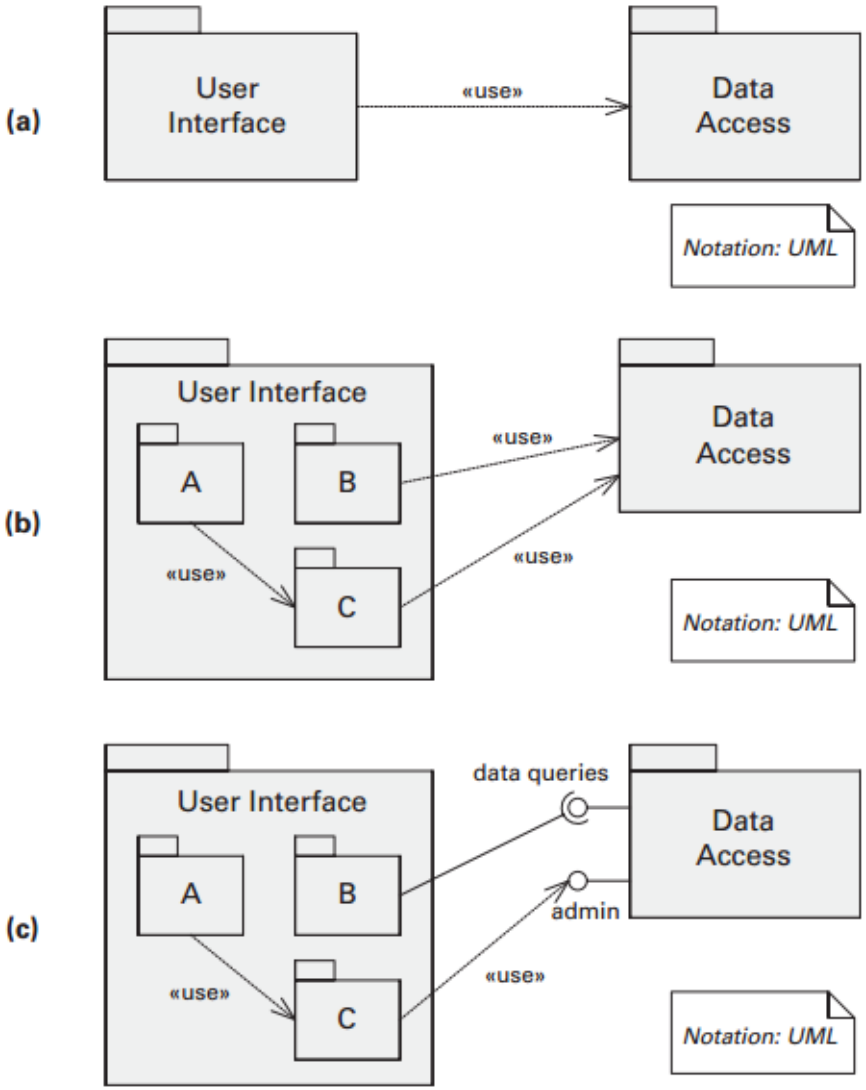
ESTILO DE USOS – NOTACION 1

Notación Informal

- Tablas de dos columnas - Elementos y Elementos usados

Notación Semiformal – UML

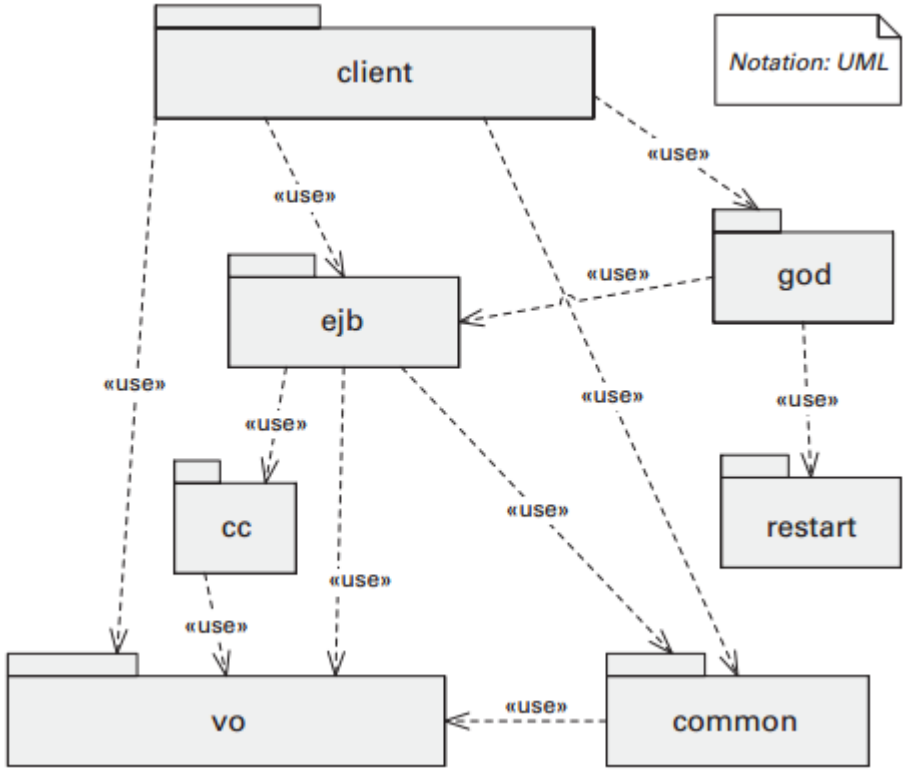
- Módulos → Paquetes
- *usa* → Relación de dependencia con estereotipo <<*use*>>



ESTILO DE USOS – NOTACION 2

Notación Semiformal – Matriz de Dependencia

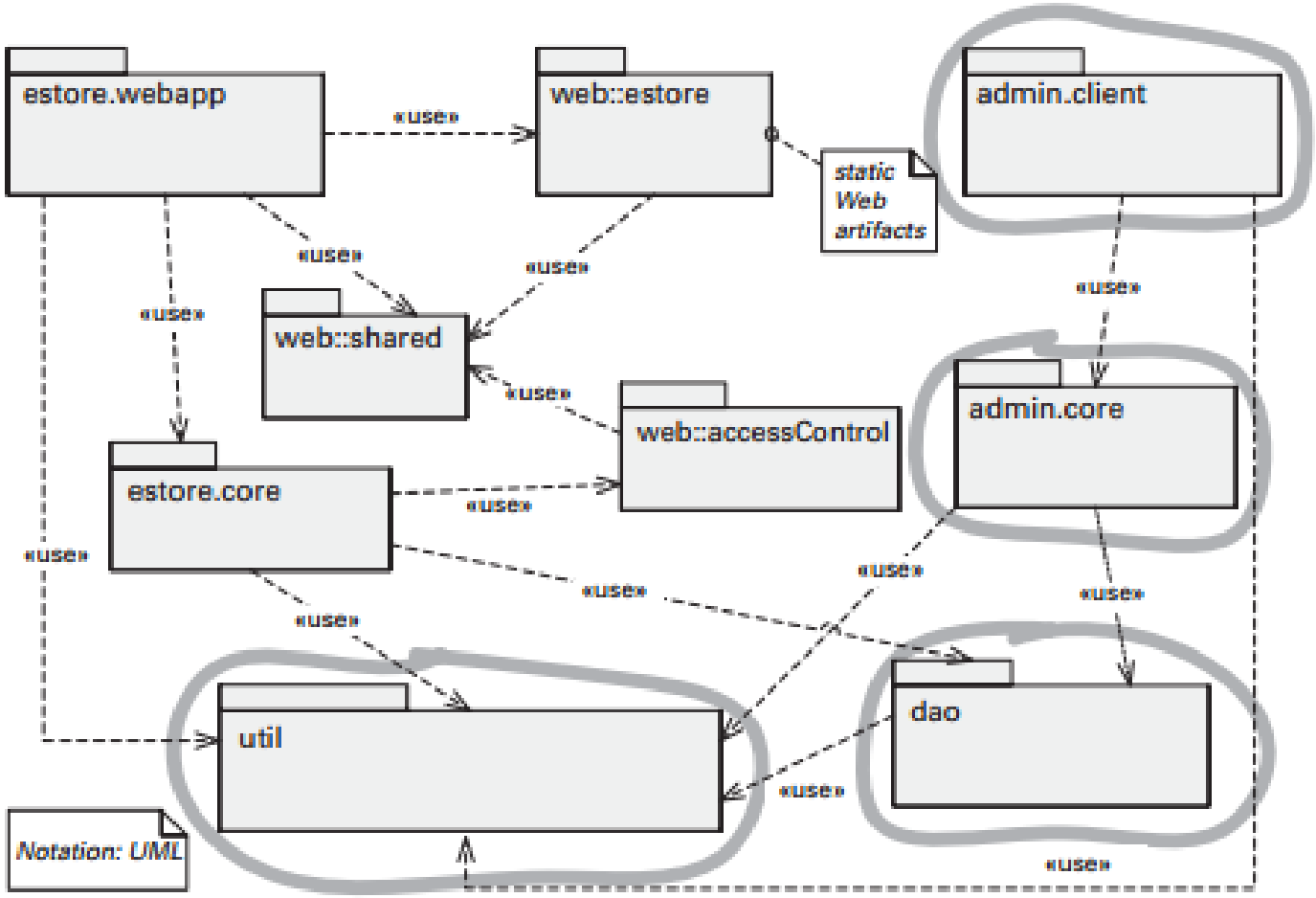
Los siguiente diagramas son intercambiables y expresan lo mismo.



using module / used module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	**1**	0	0	**1**	0	0	0
cc	0	**1**	0	0	0	0	0
god	**1**	0	0	0	0	0	0
restart	0	0	0	**1**	0	0	0
common	**1**	**1**	0	0	0	0	0
vo	**1**	**1**	**1**	0	0	**1**	0

Key: "1" means module in column uses module in row

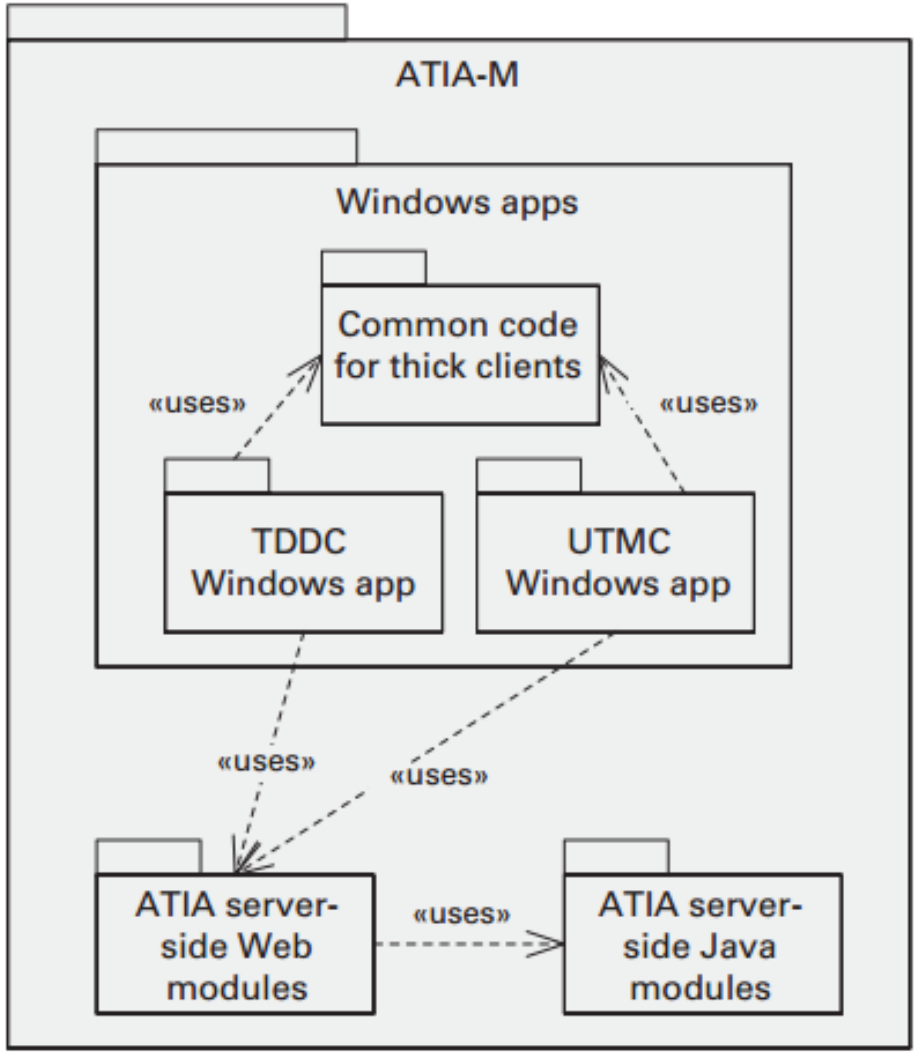
ESTILO DE USOS – EJEMPLO 1



Suponiendo que el plan de desarrollo incremental requiere la implementación del módulo `admin.client`, la vista de usos permite determinar qué módulos necesitan resolverse previamente: `admin.core`, `dao` y `util`

ESTILO DE USOS – EJEMPLO 2

Sistema `ATIA-M`



CONTENIDO

ESTILO DE GENERALIZACION

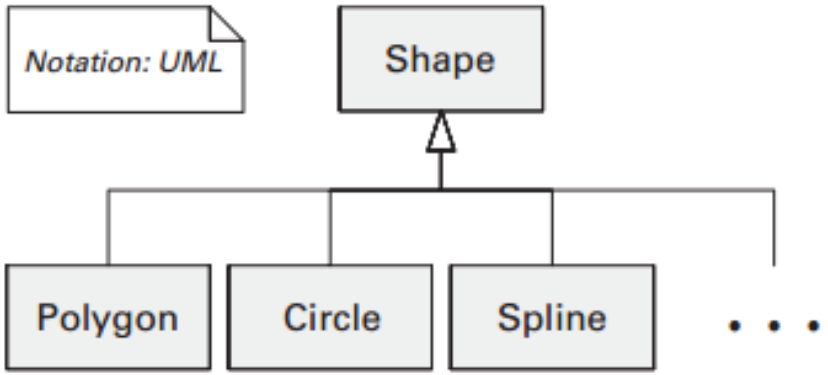
DESCRIPCION	○ emplea una relación *es-un* para soportar la extensión y evolución de arquitecturas y elementos individuales ○ los módulos en este estilo son definidos de forma tal que capturen las cosas comunes y las variaciones
ELEMENTOS	Módulo Un módulo puede tener una propiedad *abstracto* para indicar que no contiene una implementación completa
RELACIONES	La *generalización* es una especialización de la relación *es-un*. Puede ser una herencia de clase, una herencia de interface o una implementación de una interface.
RESTRICCIONES	○ un módulo puede tener múltiples padres, aunque la herencia múltiples es frecuentemente considerada una aproximación de diseño peligrosa ○ no se permiten los ciclos en la relación de generalización

ESTILO DE GENERALIZACION – USOS

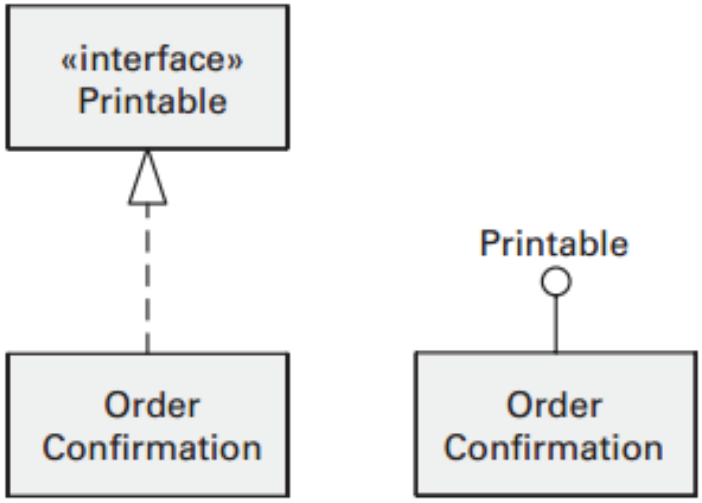
- expresar herencia en diseños orientados a objetos
- describir incrementalmente la evolución y extensión
- capturar comportamiento común en los padres y variaciones en los hijos
- facilitar el reuso

ESTILO DE GENERALIZACION – NOTACION

Herencia de clases o interfaces



Realización de interfaces

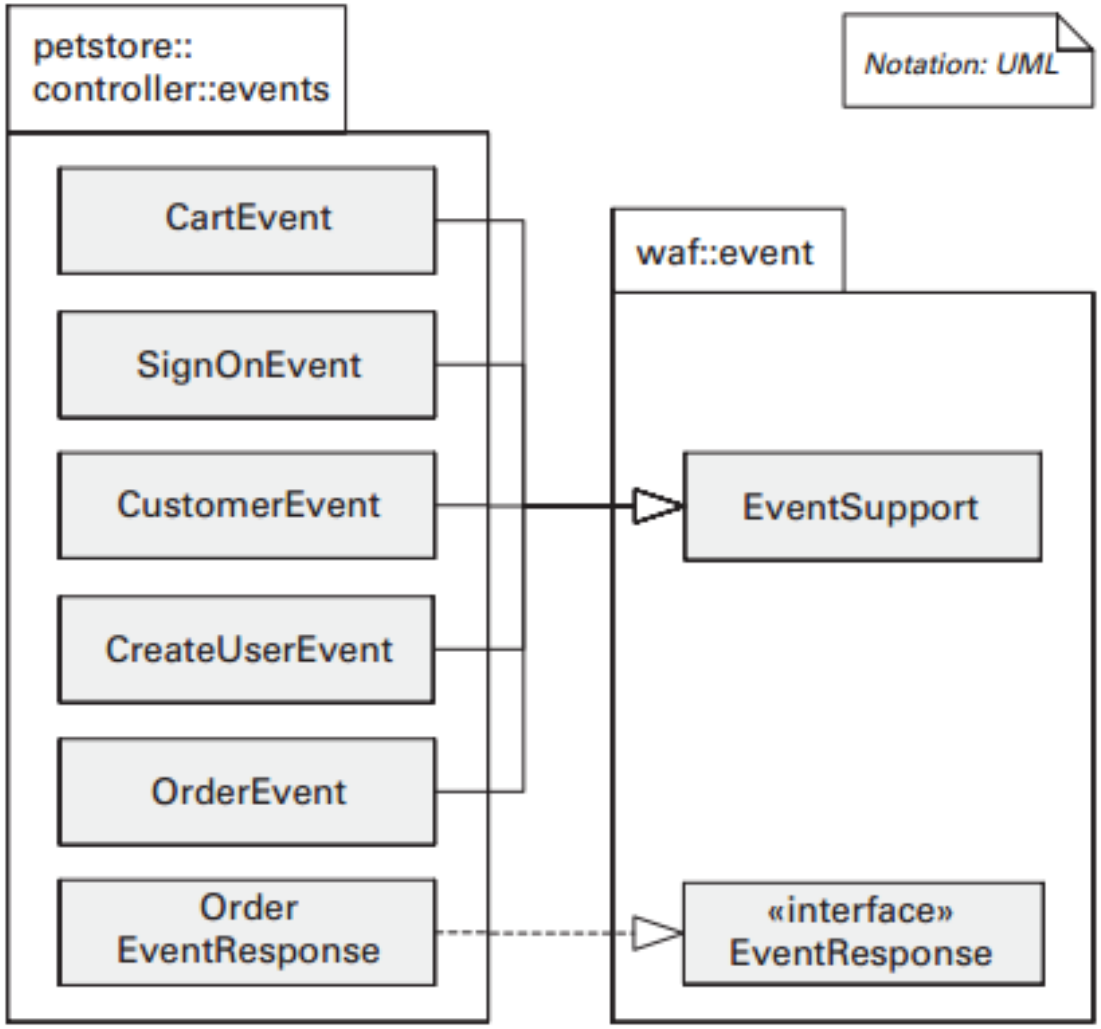


ESTILO DE GENERALIZACION – EJEMPLO

SISTEMA: PET STORE

DESCRIPCION:

- Aplicación web multi-capas que implementa un almacén de mascotas
- el diagrama muestra una parte del sistema, indicando una jerarquía de clases para representar eventos en el sistema
- el paquete de la derecha es parte de un framework (WAF) que ofrece servicios manejados por eventos



CONTENIDO

ESTILO DE CAPAS

DESCRIPCION	○ pone las capas juntas en una relación unidireccional "permitido usar"
ELEMENTOS	Capa Una capa es una agrupación de módulos que ofrece un conjunto de servicios cohesivos. La descripción de una capa define qué módulos contiene.
RELACIONES	"*permitido-usar*" es una especialización de la relación genérica *depende-de*. El diseño define las reglas de uso de las capas y cualquier excepción permitida.
RESTRICCIONES	○ cada pieza de software está alocada exactamente en una capa ○ al menos habrá 2 capas (típicamente 3 o más) ○ la relación *permitido-usar* no debería ser circular (una capa más baja no debería usar a una capa más alta).

ESTILO DE CAPAS – USOS

- promover la modificabilidad y portabilidad
- manejar la complejidad y facilitar la comunicación de la estructura del código a los desarrolladores
- promover el reúso
- facilitar la separación de intereses y la asignación de tareas

ESTILO DE CAPAS – PROPIEDADES DE UNA CAPA

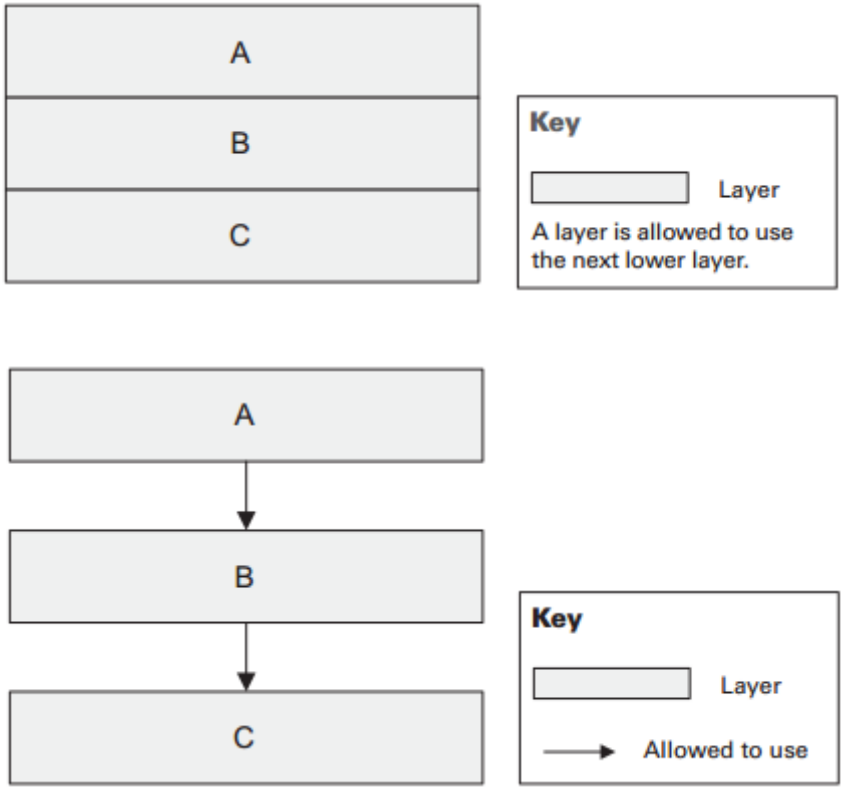
- El contenido de una capa - la descripción de una capa debería proveer lineamientos sobre los módulos que contendrá y cómo implementarlos.
- El uso de otras capas –
  - ¿una capa sólo puede usar la capa por debajo, o cualquiera de las inferiores?
  - si una capa está segmentada horizontalmente, ¿los módulos de un segmento pueden usar a los de otro segmento?

  Se deben explicar todas las excepciones.

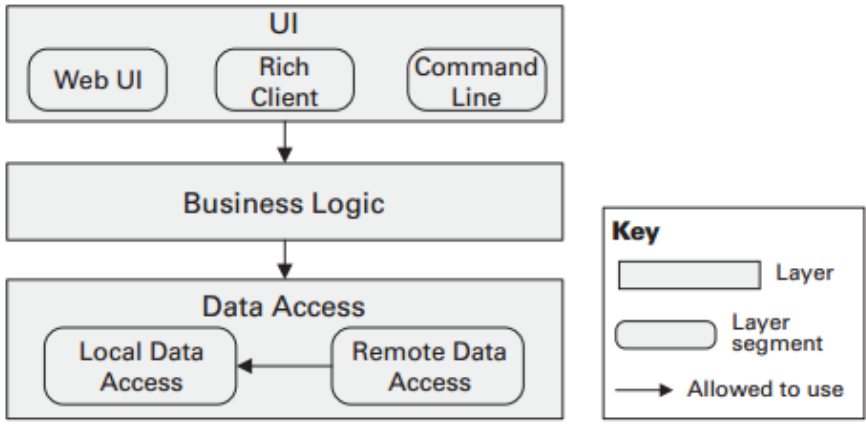
ESTILO DE CAPAS – NOTACION 1

Notación Informal

- Pilas



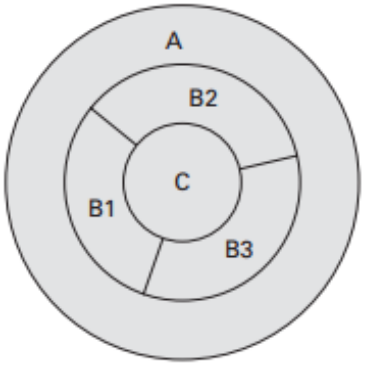
- Capas segmentadas

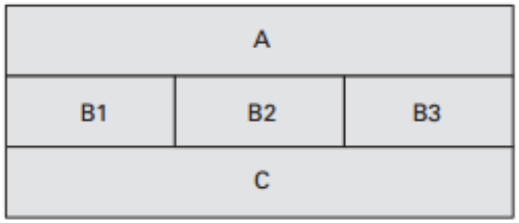


ESTILO DE CAPAS – NOTACION 2

Notación Informal

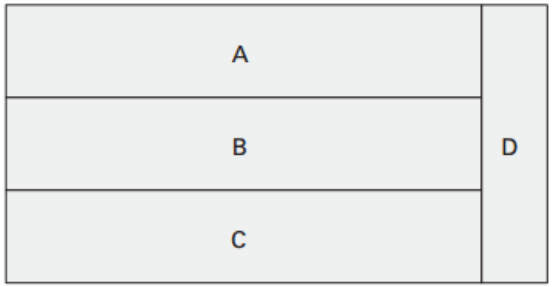
- Anillos

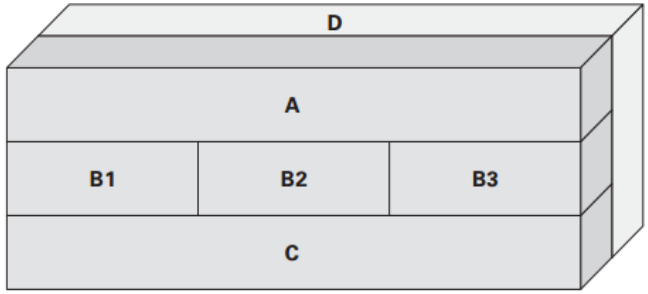




- Capas con lateral

  Las capas laterales generalmente representan librerías utilitarias o servicios de la plataforma (sistema operativo o ambiente de ejecución)





Tamaño y Color

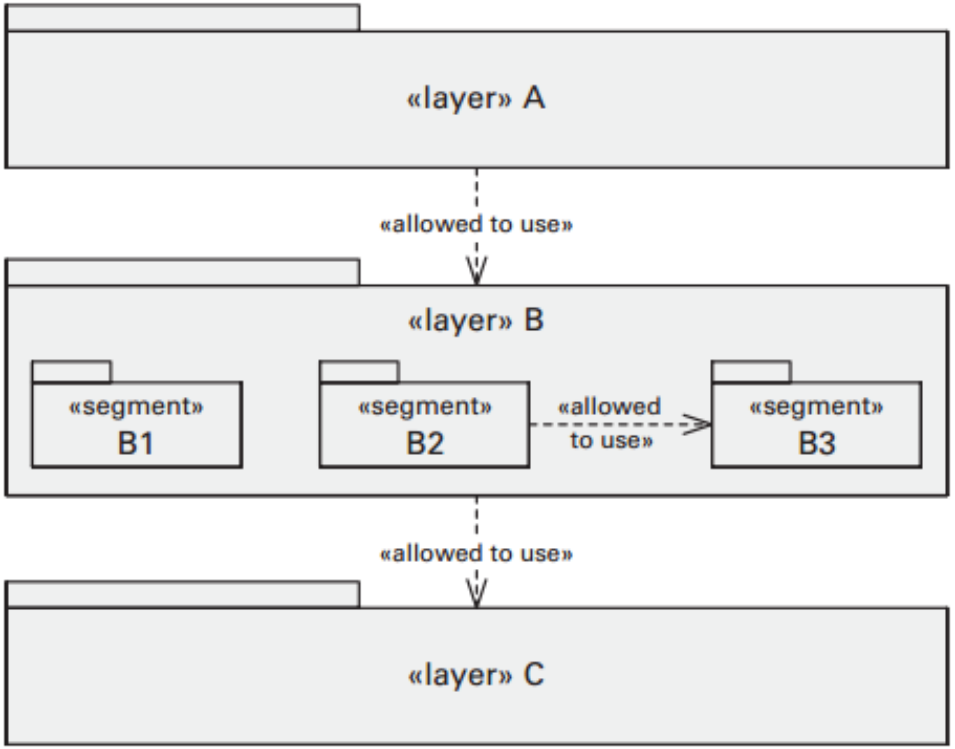
- A veces se usan colores para denotar, por ejemplo, qué equipo es responsable de cada capa.

- El tamaño es usado par dar una vaga idea del tamaño relativo de los módulos que constituyen las distintas capas.

ESTILO DE CAPAS – NOTACION 3

Notación Semiformal – UML

- capas → paquetes estereotipados
- relacion *permitido-usar* → dependencia estereotipada



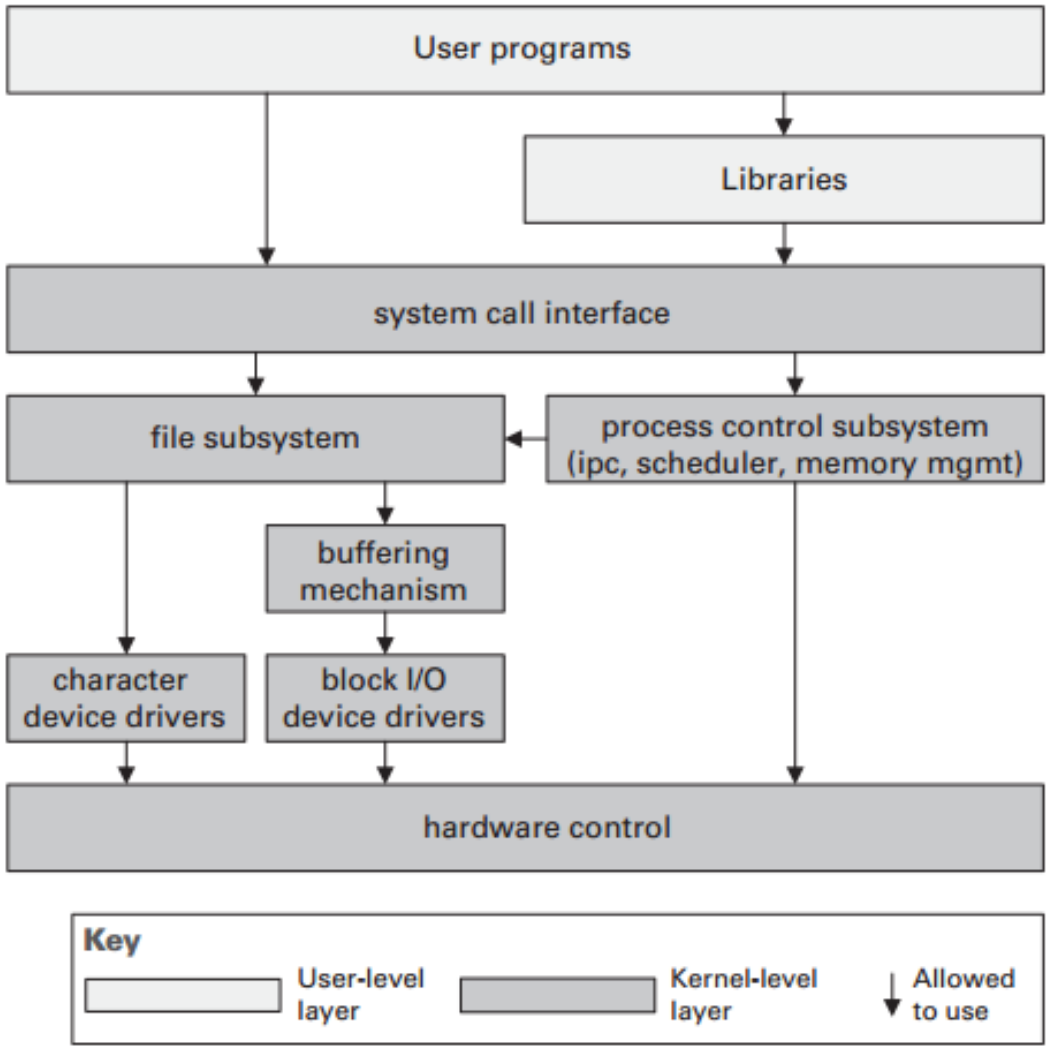
ESTILO DE CAPAS – EJEMPLO 1

SYSTEM V (Sys V) de UNIX

Desarrollado originalmente por AT&T y lanzado en 1983.

System V Release 4, o SVR4, fue la versión más popular, y la fuente de varias características comunes de Unix, tales como "SysV init scripts" (/etc/init.d), usadas para el control de inicio y apagado del sistema.

Ref:Wikipedia

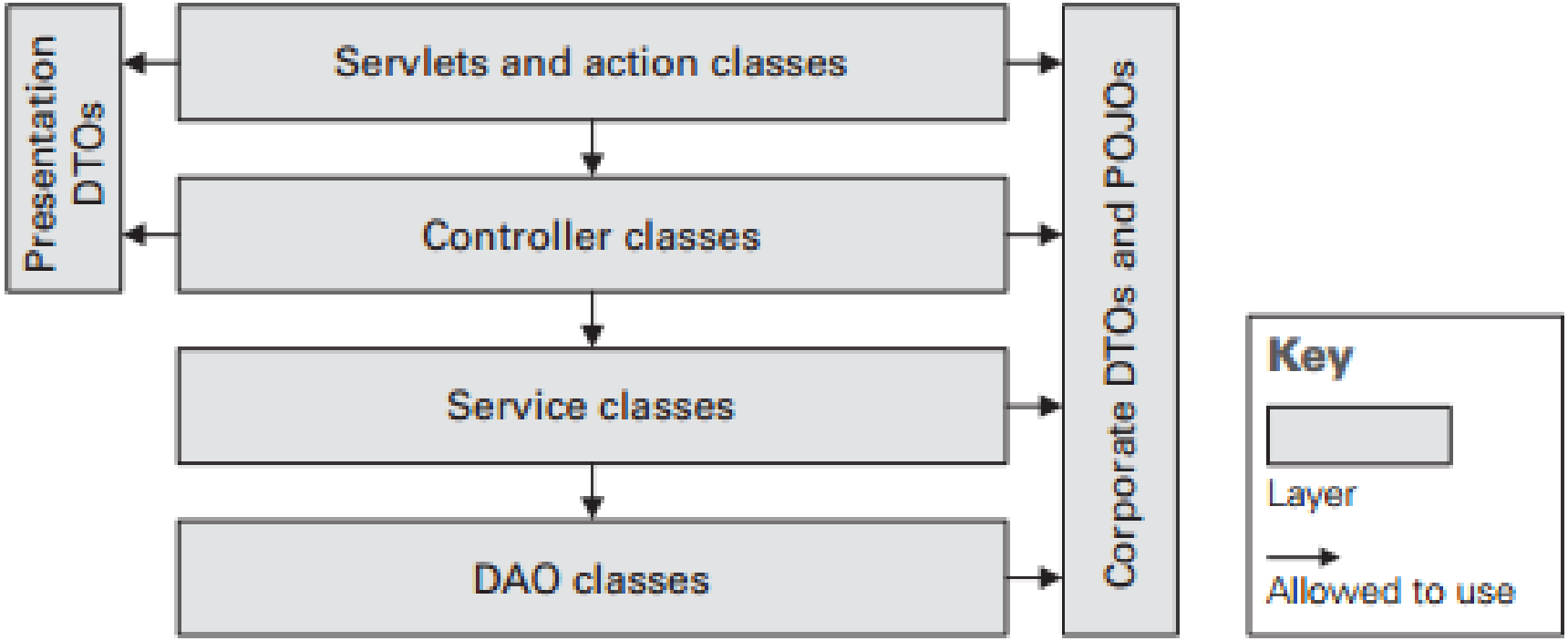


ESTILO DE CAPAS – EJEMPLO 2

APLICACION JAVA EE

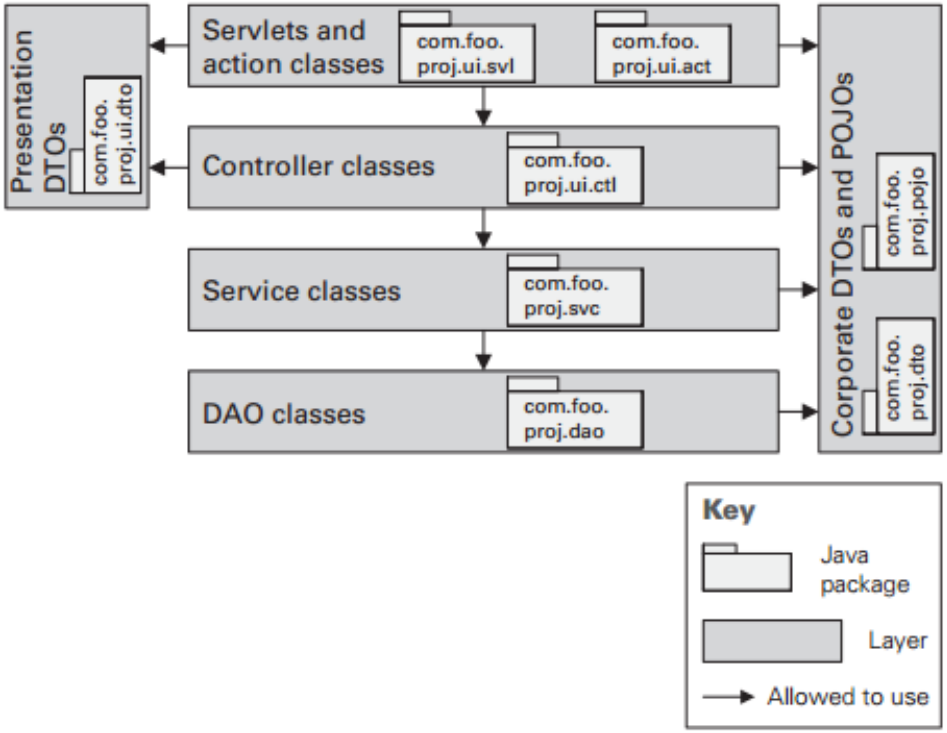
DTOs – Data Transfer Objects

POJOs – Plain Old Java Objects



ESTILO DE CAPAS – EJEMPLO 3

APLICACION JAVA EE con Matriz de Dependencia



using module / used module	com.foo.proj.ui.svl	com.foo.proj.ui.act	com.foo.proj.ui.ctl	com.foo.proj.ui.dto	com.foo.proj.svc	com.foo.proj.dao	com.foo.proj.dto	com.foo.proj.pojo
com.foo.proj.ui.svl	0	0	0	0	0	0	0	0
com.foo.proj.ui.act	0	0	0	0	0	0	0	0
com.foo.proj.ui.ctl	1	1	0	0	0	0	0	0
com.foo.proj.ui.dto	1	1	1	0	0	0	0	0
com.foo.proj.svc	0	0	1	0	0	0	0	0
com.foo.proj.dao	0	0	0	0	1	0	0	0
com.foo.proj.dto	1	1	1	0	1	1	0	0
com.foo.proj.pojo	1	1	1	0	1	1	0	0

CONTENIDO

ESTILO DE ASPECTOS

DESCRIPCION	○ muestra los módulos de aspectos que implementan cuestiones transversales (crosscutting concerns) y cómo están ligados a otros módulos
ELEMENTOS	Aspecto, un módulo especializado que contiene la implementación de una cuestión transversal (aspecto)
RELACIONES	*atraviesa (crosscuts)*, la cual asocia un módulo de aspecto a un módulo que será afectado por la lógica del aspecto
RESTRICCIONES	○ un aspecto puede atravesar uno o más módulos regulares, así como también módulos de aspectos ○ un aspecto que se atraviesa a sí mismo, puede causar recursión infinita, dependiendo de la implementación

ESTILO DE ASPECTOS – USOS

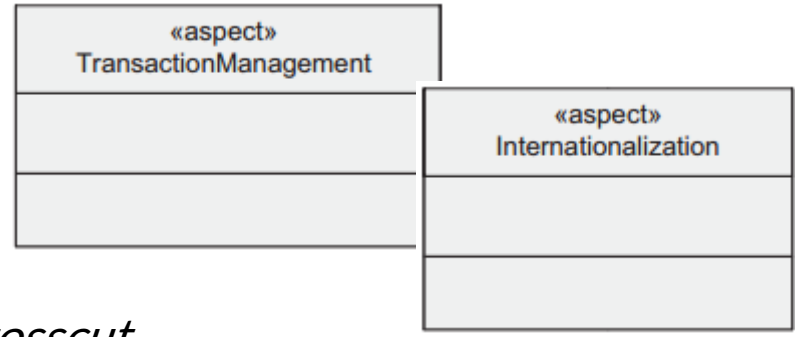
- modelar cuestiones transversales en diseños orientados a objetos
- mejorar la modificabilidad

ESTILO DE CAPAS – PROPIEDADES MODULO DE ASPECTO

- las mismas propiedades que un módulo regular
- una propiedad que indica qué módulos serán afectados por el módulo de aspecto (pointcut specification)

ESTILO DE CAPAS – NOTACION

Notación Semiformal – UML

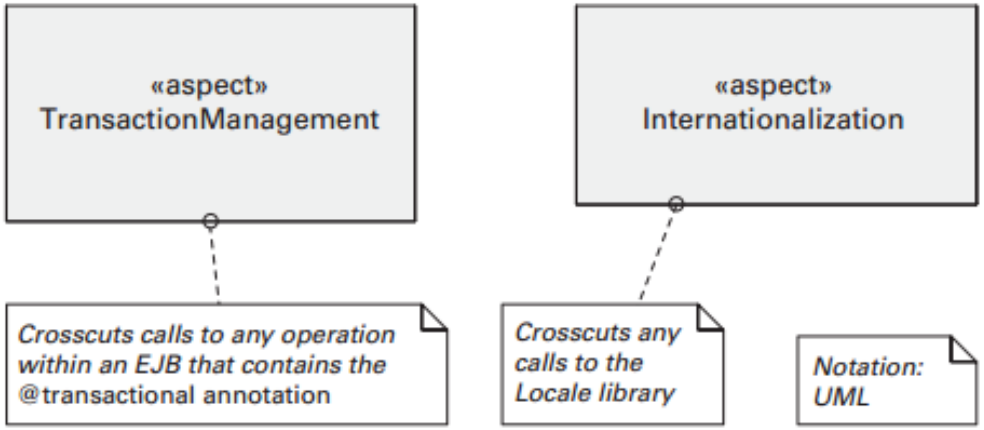


- Módulo de Aspecto → Clases estereotipadas

Dos alternativas para representar la relacion de *crosscut*

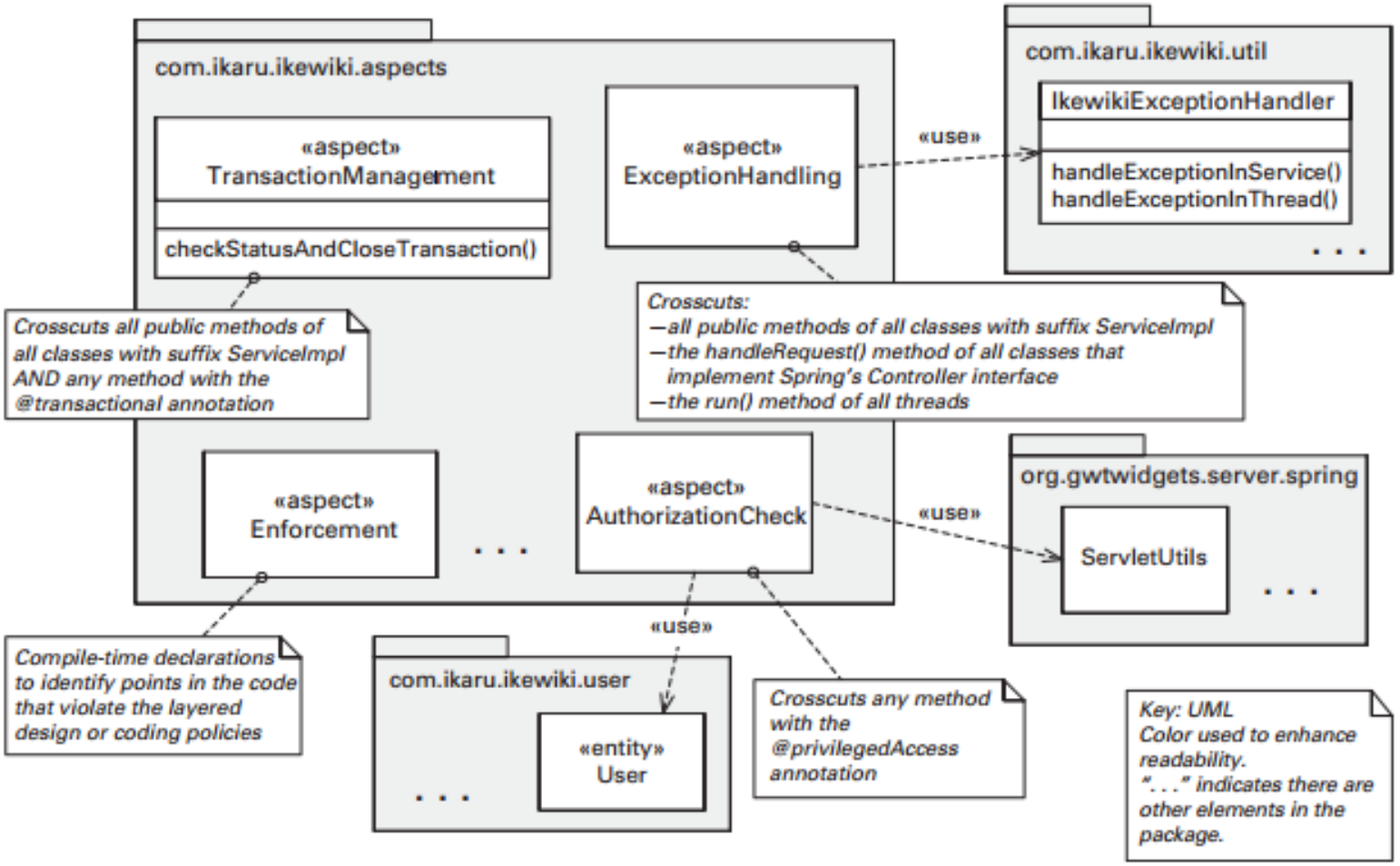
1) dependencia estereotipada desde el aspecto a los módulos regulares (no escala)

2) omitirlos en el diagrama y agregar un comentario al módulo del aspecto para caracterizar qué módulos son atravesados por el aspecto



ESTILO DE ASPECTOS – EJEMPLO

IkeWiki – Un motor de wiki semántico



CONTENIDO

ESTILO DE MODELO DE DATOS

DESCRIPCION	describe la estructura de las entidades de datos y sus relaciones
ELEMENTOS	Entidad de datos Una entidad de datos es un objeto que mantiene información que necesita ser almacenada
RELACIONES	○ Relaciones *uno-a-uno*, *uno-a-muchos*, *muchos a muchos* - que representan asociaciones lógicas entre entidades de datos. ○ *generalización/especialización* que indican una relación es un entre entidades ○ *a*gregación - que convierte una entidad en una entidad agregada
RESTRICCIONES	Dependencias funcionales serían evitadas

ESTILO DE MODELO DE DATOS – USOS

- describir la estructura de los datos utilizados en el sistema
- realizar análisis de impacto de cambios en el modelo de datos
- forzar la calidad de los datos evitando redundancia e inconsistencia
- guiar la implementación de los módulos que acceden a los datos

ESTILO DE CAPAS – PROPIEDADES DE ENTIDAD DE DATOS

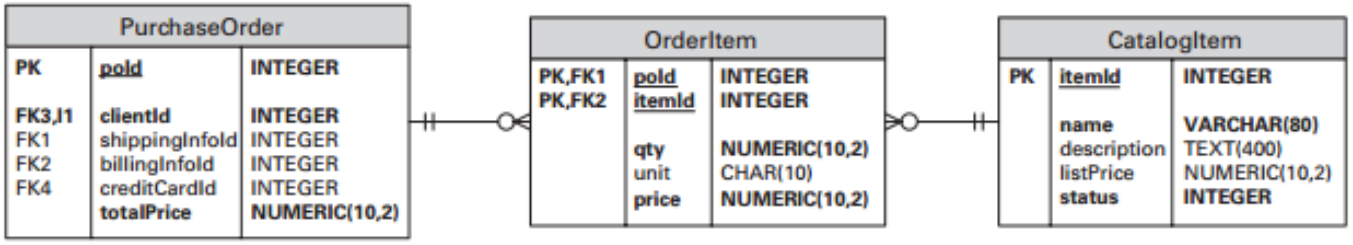
- nombre
- atributos de los datos
- clave primaria
- reglas para garantizar los permisos de usuario para acceder a la entidad.

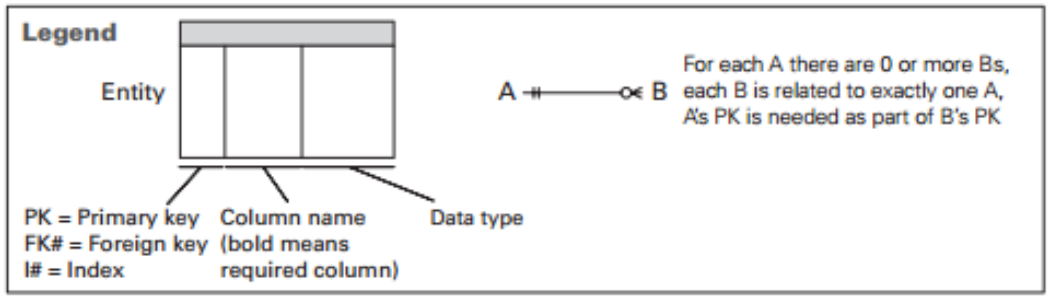
ESTILO DE MODELO DE DATOS – NOTACION

Notacion Informal

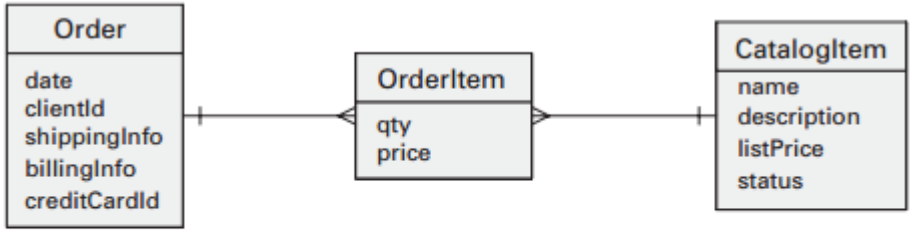
- Conceptual

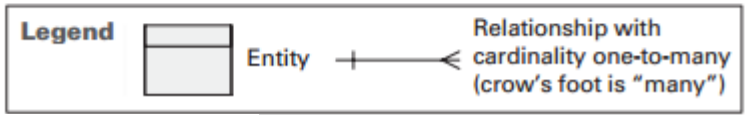
- Física





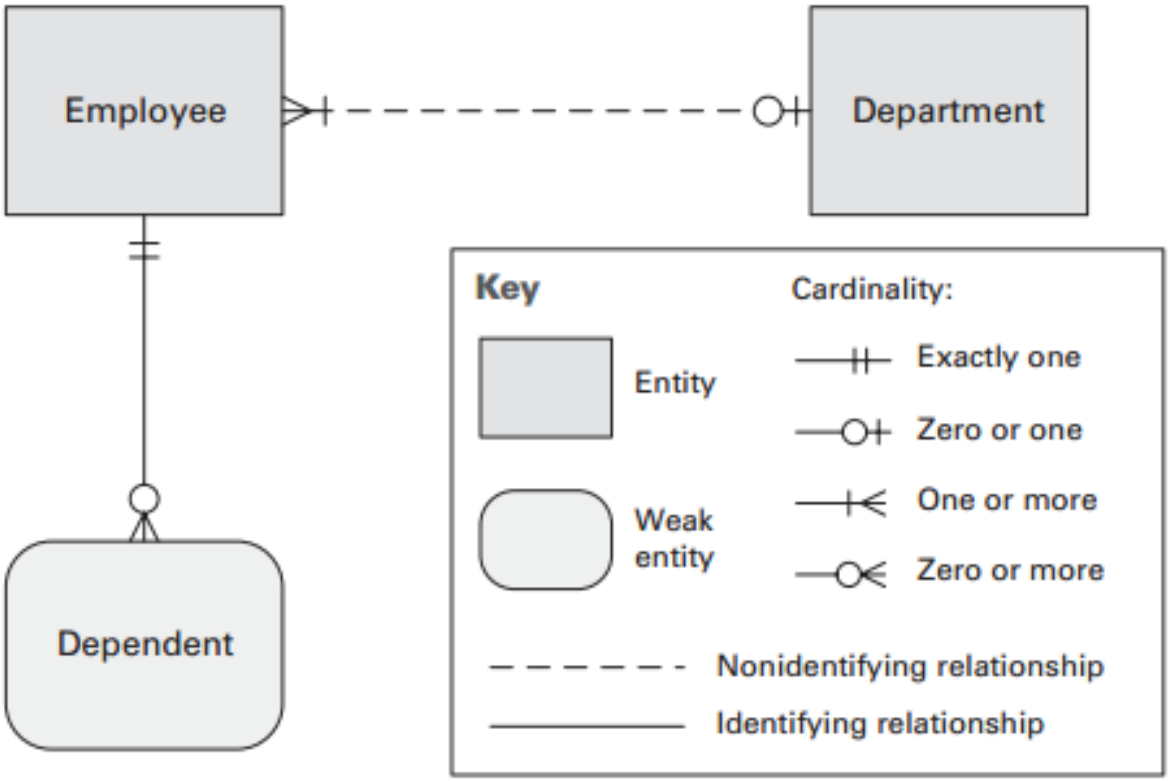
- Lógica





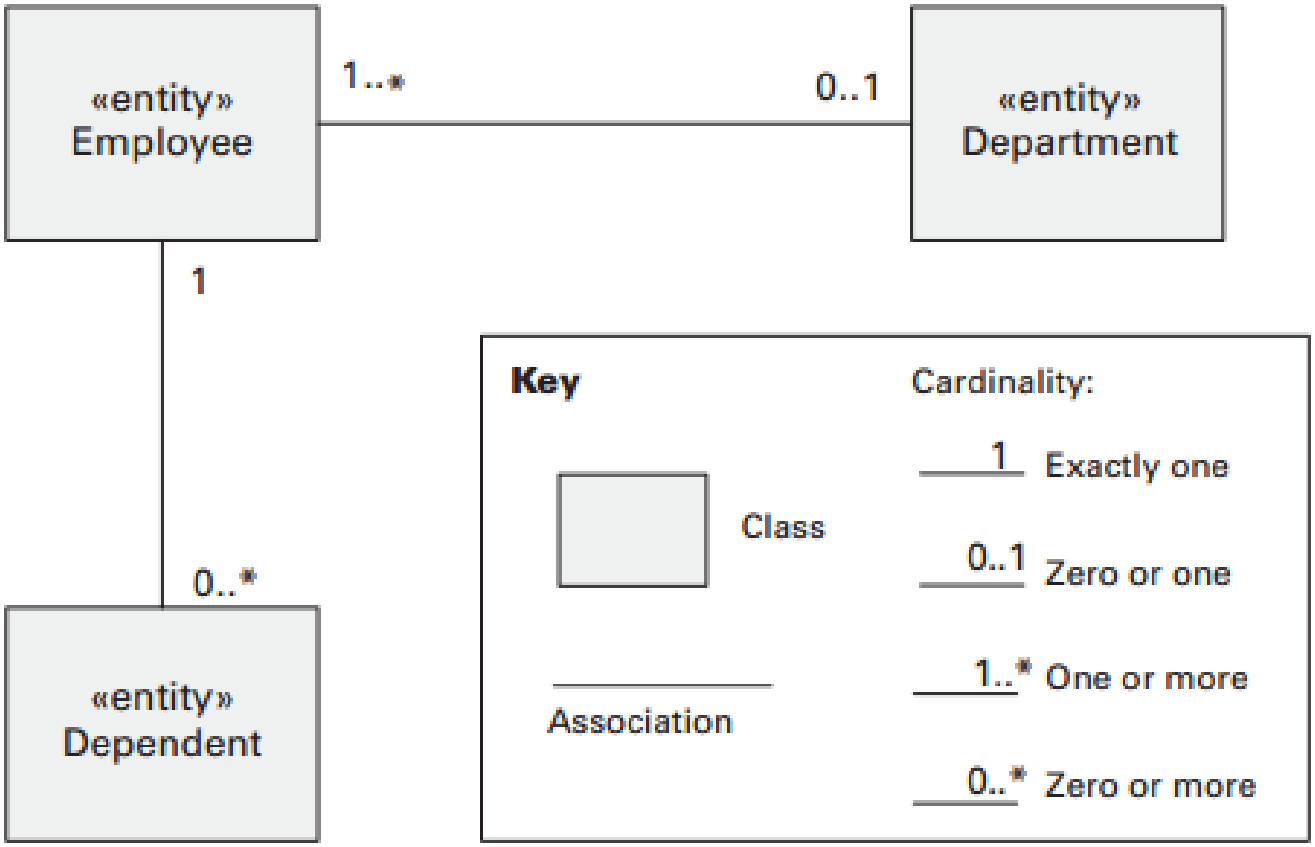
ESTILO DE MODELO DE DATOS – NOTACION

Notacion Semiformal – Modelo Entidad-Relación



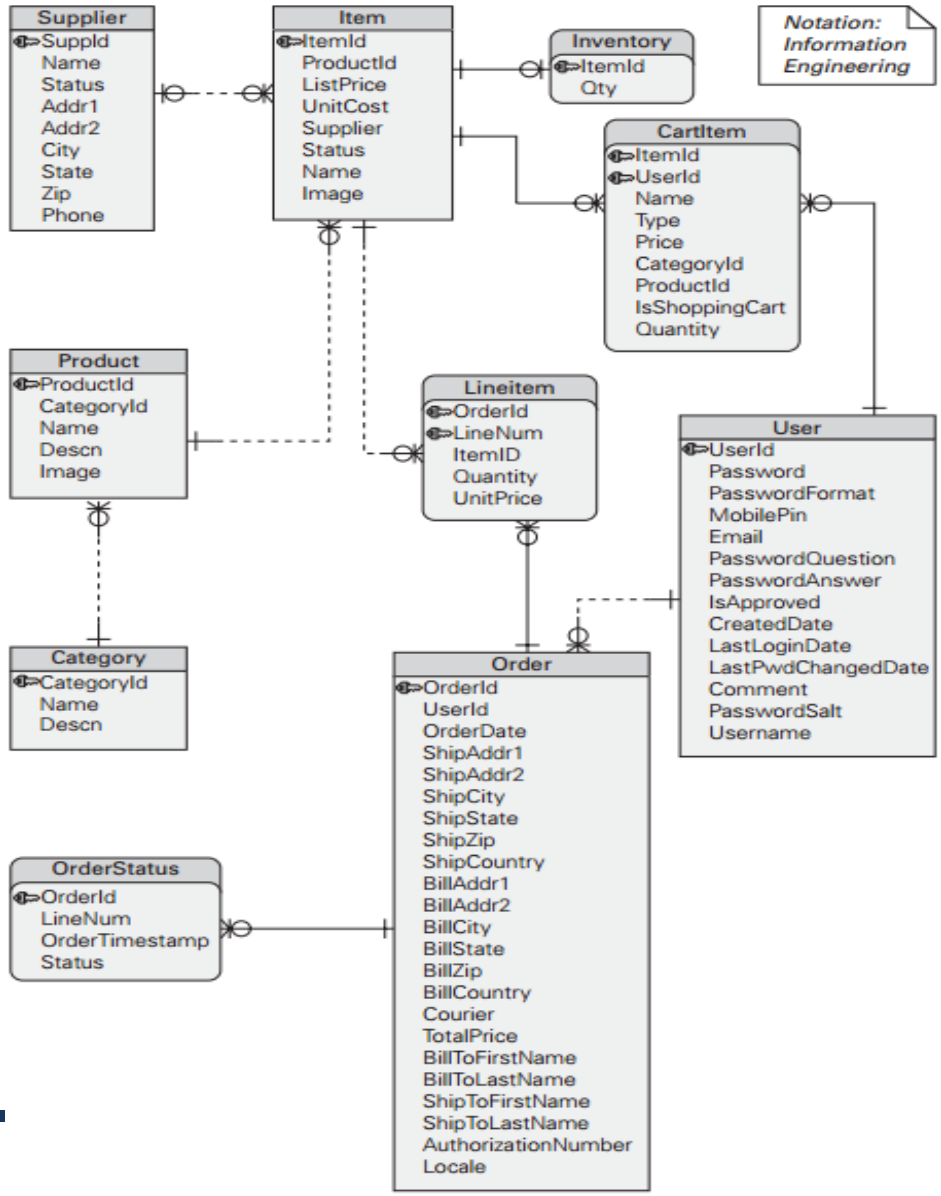
ESTILO DE MODELO DE DATOS – NOTACION

Notacion Semiformal – UML



ESTILO DE MODELO DE DATOS – EJEMPLO

SISTEMA: PET STORE



PARA DISCUTIR

Cual seria la vista de módulos mas adecuada si se necesita...

- Especificar funcionalidad transversal?
- Diseñar la base de datos?
- Coordinar las interfaces con otros sistemas?
- Planificar el tiempo de desarrollo?
- Explicar funcionalidad del sistema a gerentes usuarios?
- Mostrar detalles especificos del producto X parte de una familia de productos?

ESTILOS:

- Descomposición
- Usos
- Generalización
- Capas
- Aspectos
- Modelo de Datos

RESUMEN – 1

- Una vista de descomposición muestra como las responsabilidades son asignadas a módulos y sub-módulos.
- Una vista de usos muestra la dependencia entre módulos. Esta vista sirve para un desarrollo incremental y para hacer análisis de impacto de cambios.
- Una vista de generalización relaciona módulos mostrando como uno es una generalización o especialización de otro. Esta vista es usada ampliamente en sistemas orientados a objetos, donde se usa el principio de herencia para explotar las características comunes entre objetos.

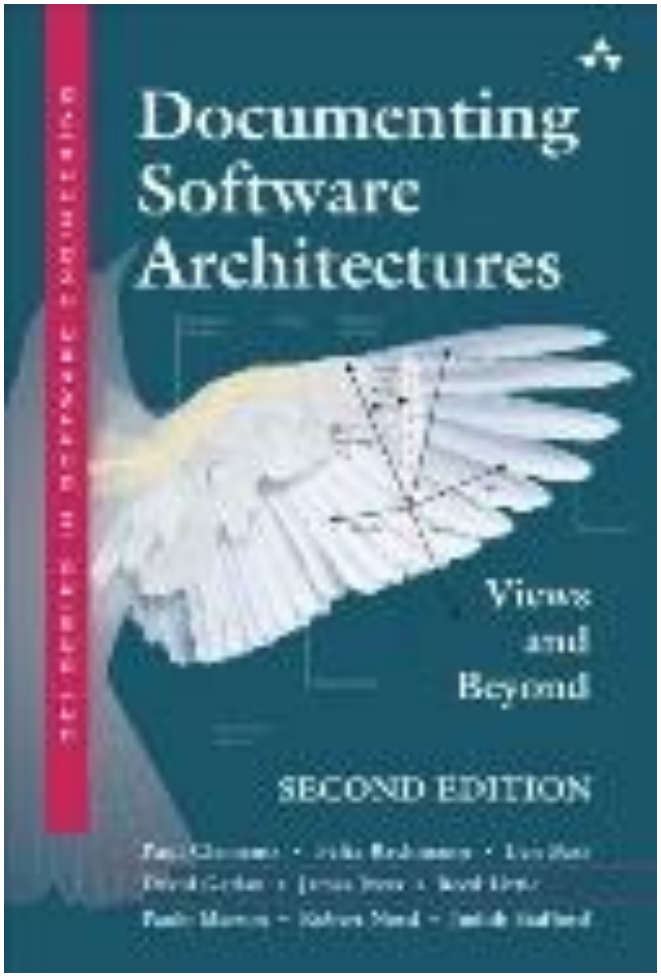
RESUMEN – 2

- Una vista de capas divide al sistema en grupos de módulos que proveen responsabilidades cohesivas. Estos grupos son llamados capas y se relacionan unidireccionalmente con la relación “permite-usar”. Un sistema en capas ayuda a que el sistema logre portabilidad y modificabilidad.

- Una vista de aspectos muestra módulos especiales llamados “aspectos” que son responsables por cuestiones transversales. Esta vista es particularmente útil si la implementación del sistema adopta programación orientada a aspectos (POA).

- Una vista de modelo de datos describe la estructura de datos usada en el sistema en termino de entidades y sus relaciones. La vista guía la implementación y ayuda a mejorar la performance y modificabilidad en los sistemas centrados en datos.

BIBLIOGRAFIA



Documenting Software Architectures
Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford
Addison-Wesley, 2011

Elsa Estevez
ece@cs.uns.edu.ar