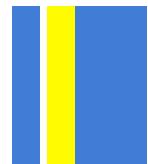




Universidad de  
los Andes

DEPARTAMENTO DE SISTEMAS



# Arquitectura de Software

Punto de Vista Funcional

# Punto de Vista Funcional



Su principal propósito es describir los elementos funcionales del sistema, así como sus principales responsabilidades, interfaces e interacciones.

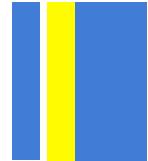
Este punto de vista es normalmente las más utilizada y referenciada por los stakeholders dentro del Documento de Arquitectura.

# Punto de Vista Funcional

Este punto de vista afecta algunos modelos de otros puntos de vista tales como: información, concurrencia y despliegue.

Este punto de vista también tiene un alto impacto en los atributos de calidad del sistema, tal como la habilidad de ser modificado, de ser seguro y su desempeño en ejecución.

- Usos asociados a este punto de vista
  - Describir la capacidad funcional del sistema
  - Definir los elementos importantes del sistema
    - Responsabilidades
    - Interfaces Exuestas
    - Interacciones entre ellos
  - Definir las interfaces externas del sistema
  - Definir la Estructura Interna del sistema
  - Hacer explícita la filosofía de diseño



# Punto de Vista Funcional

- Stakeholders a los que esta dirigido
  - Todos
- Modelos Utilizados
  - Modelo de estructura funcional
  - Diagramas de Componentes

- **Capacidades Funcionales**
  - Definen lo que el sistema requiere hacer
  - Requerimientos Funcionales
    - Explícitos
    - Implícitos
- **Interfaces Externas**
  - Flujos de datos y control entre el sistema a desarrollar y otros sistemas
  - La definición de interfaces debe considerar la sintaxis de las interfaces (Estructura y Datos) y la semántica

- **Estructura Interna**
  - Componentes o elementos estructuradores del sistema a desarrollar
  - Organización interna del sistema
    - Sistemas monolíticos
    - Componentes con bajo acoplamiento
  - La estructura interna del sistema es definida por sus elementos internos
    - Comportamiento
    - Correspondencia con los requerimientos
    - Interacciones

- 
- Estructura Interna
    - Impacto en los atributos de calidad
      - Desempeño
      - Escalabilidad
      - Seguridad

- Filosofía de Diseño
  - Presenta cómo la arquitectura se adhiere a buenos principios de diseño
    - Separación de preocupaciones
    - Cohesión
    - Acoplamiento
    - Volumen de interacción entre elementos
    - Flexibilidad
    - Coherencia



# Punto de Vista Funcional

Cualidad	Descripción	Significado
Separación de Preocupaciones	Cada elemento es responsable de una parte de la operación del sistema	Alta separación facilita el mantenimiento pero va en contra del desempeño
Cohesión	Que tanto están relacionadas las funciones de un componente con las ofrecidas por otros componentes	Es deseable una alta cohesión
Acoplamiento	Qué tan fuerte es la relación entre componentes	Un bajo acoplamiento facilita el mantenimiento pero va en contra de la escalabilidad
Volumen de interacción entre los elementos	Qué porcentaje de los pasos de procesamiento envuelven interacciones entre los elementos	Hay un costo en la comunicación entre elementos
Flexibilidad Funcional	Qué tan flexible es el sistema para responder a cambios funcionales	Entre más flexible más difícil de construir
Coherencia	La arquitectura “se ve bien ??”	Difícil de entender para los stakeholders

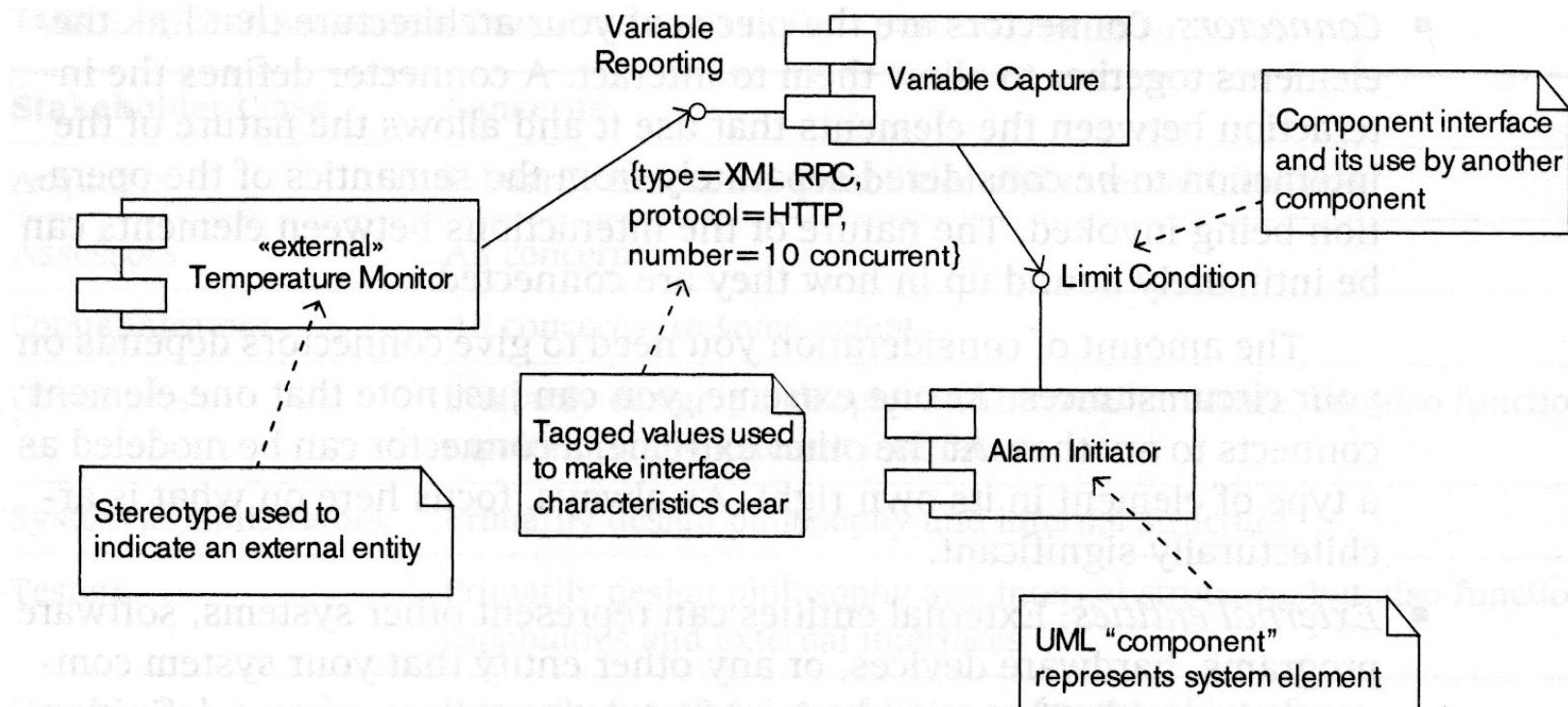
- Modelos de Estructura Funcional
  - Elementos Funcionales
    - Una parte bien definida del sistema en ejecución, con responsabilidades particulares e interfaces bien definidas
  - Interfaces
    - Mecanismos bien definidos mediante los cuales los elementos del sistema pueden ser accedidos por otros elementos
      - Entradas
      - Salidas
      - Semántica de cada operación

- Conectores
  - Piezas dentro de la arquitectura que enlazan elementos de la arquitectura y les permiten interactuar
- Entidades Externas
  - Representan otros sistemas, aplicaciones, hardware, o cualquier otra entidad con la cual se comunica el sistema
- Notación Sugerida
  - Diagramas de componentes UML

- Diagramas de componentes UML
  - Cada elementos del sistema es representado con un ícono de componente
  - Es posible utilizar estereotipos para hacer más clara la estructura funcional
  - Se utilizan íconos de Interface (“lollipops”)
    - Tipo de interface
    - Protocolo utilizado para su acceso
    - Número de usuarios concurrentes

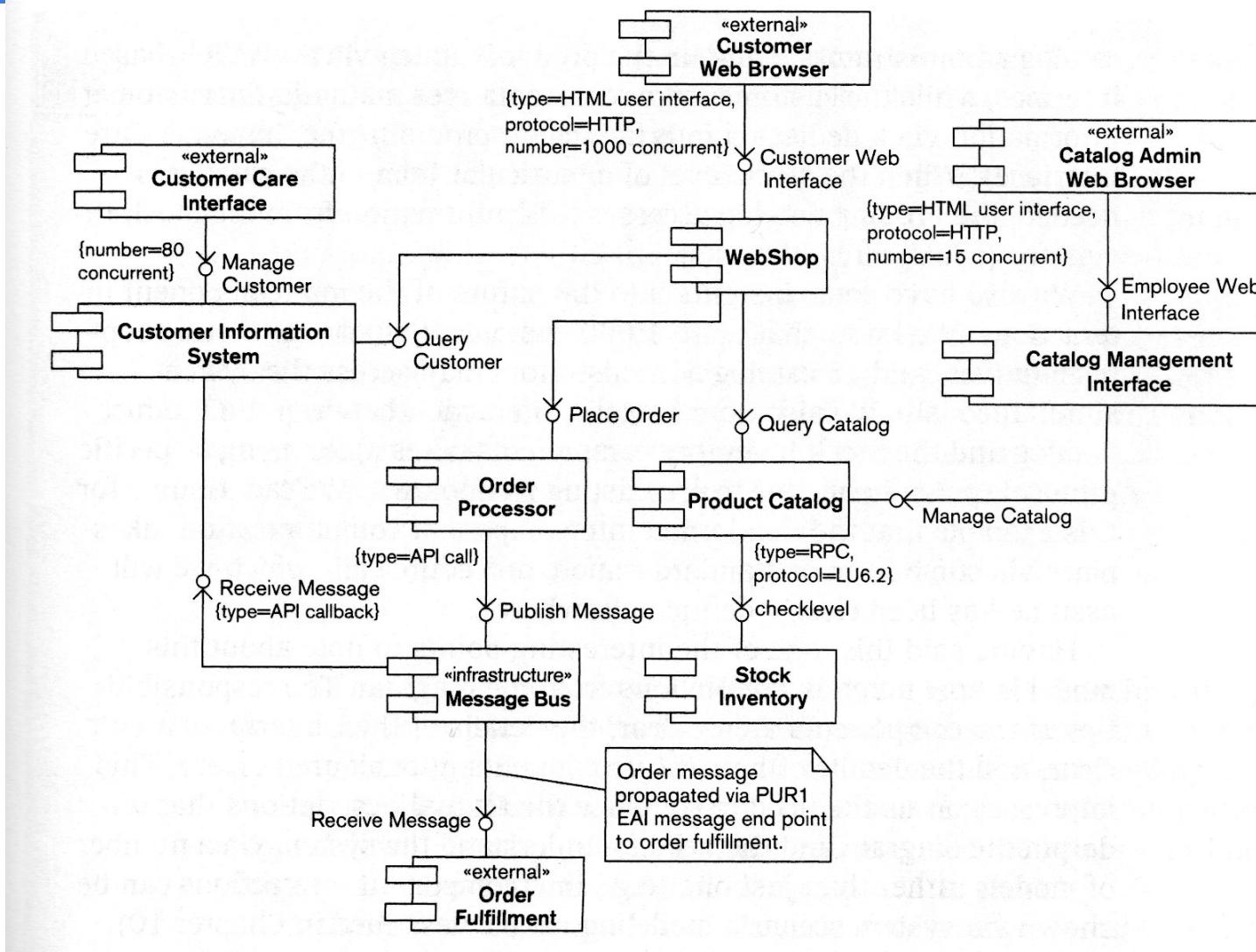


# Punto de Vista Funcional



Tomado de [1] página 220

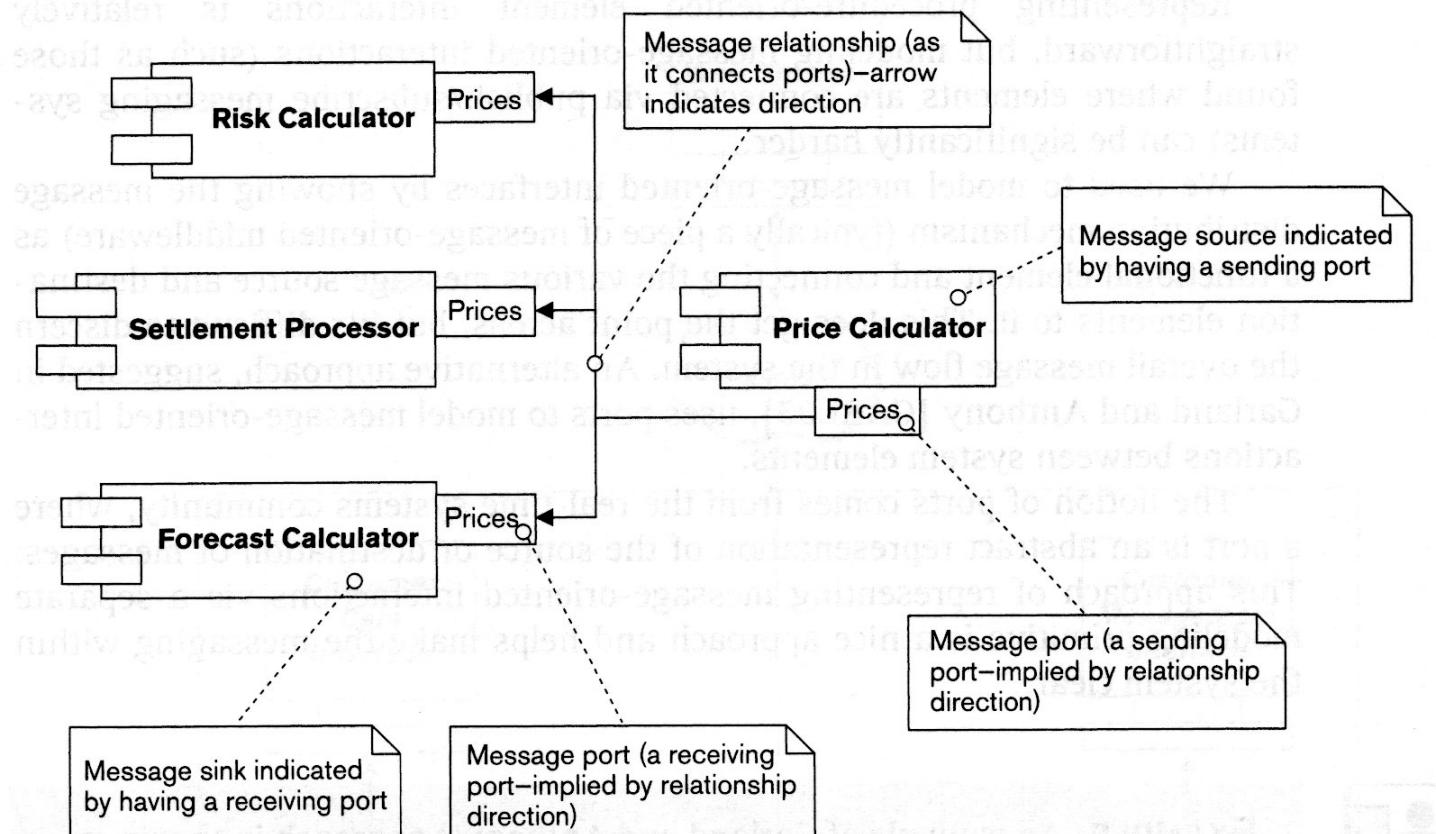
# Punto de Vista Funcional



Tomado de [1] página 221

- 
- Otras notaciones
    - Diagramas de cajas y líneas
    - Bocetos

# Punto de Vista Funcional



Tomado de [1] página 226

- Actividades a Desarrollar
  - Identificar los Elementos
  - Asignar responsabilidades a los elementos
  - Diseñar las Interfaces
  - Verificar los requerimientos funcionales
  - Comparar contra escenarios
  - Analizar interacciones
  - Analizar la Flexibilidad del sistema

- 1. Identificar los Elementos
  - Utilizar los requerimientos funcionales para identificar responsabilidades
  - Identificar elementos funcionales que cumplirán esas responsabilidades
  - Evaluar los elementos identificados contra los criterios de diseño deseables
  - Iterar sobre los pasos anteriores hasta obtener un conjunto de elementos razonable

- 2. Asignar Responsabilidades a los Elementos
  - Después de identificar elementos candidatos, se les asignan responsabilidades claras
    - Información manejada
    - Servicios Ofrecidos
    - Actividades iniciadas por este elemento

- 3. Diseñar las Interfaces
  - Para cada interface expuesta por el elemento definir claramente
    - Operaciones ofrecidas por la interfaz
    - Naturaleza de la Interfaz (mensaje, RPC, WebService)
    - Entradas
    - Salidas
    - Precondiciones
    - Efectos de cada operación
  - Para su definición se puede utilizar
    - Lenguajes de programación
    - IDLs
    - UML
    - DataFlow

- 4. Verificar los requerimientos funcionales
  - Hacer el seguimiento de cada requerimiento, utilizando la estructura funcional
  - Es aconsejable utilizar una tabla de requerimientos funcionales contra elementos del modelo estructural

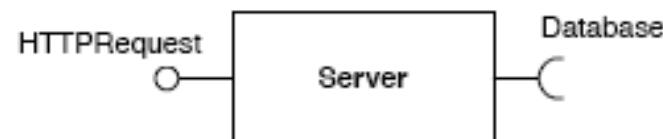
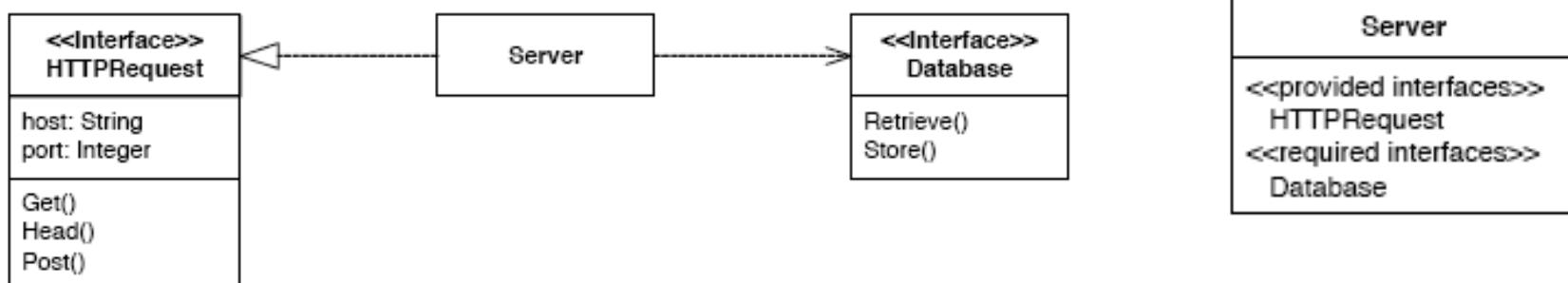
- 5. Comparar contra Escenarios
  - Analizar la estructura funcional propuesta, en conjunto con los stakeholders, a través de escenarios de uso.
- 6. Análisis de Interacciones
  - Analizar la estructura propuesta en busca de interacciones excesivas
- 7. Análisis de Flexibilidad
  - “what if ” escenarios

- Problemas durante su utilización
  - Mala definición de interfaces
  - Mala definición de responsabilidades
  - Elementos de Infraestructura modelados como elementos funcionales
  - Nivel inapropiado de detalle
  - Número elevado de dependencias
  - “God Element” / “Manager”
    - 50% de las responsabilidades en menos del 25% de los elementos

- Lista de Chequeo
  - Su modelo tiene entre 15 y 20 elementos?
  - Todos los elementos tienen nombre, responsabilidades claras e interfaces claramente definidas?
  - Todas las interacciones entre los elementos ocurren a través de interfaces y conectores entre ellas
  - Los elementos tienen una alta cohesión?
  - Los elementos muestran un bajo acoplamiento?
  - Se ha validado la estructura propuesta contra los requerimientos funcionales?
  - Ha considerado como se porta la arquitectura en escenarios hipotéticos de cambio?
  - El punto de vista tiene en cuenta los intereses de los stakeholders?

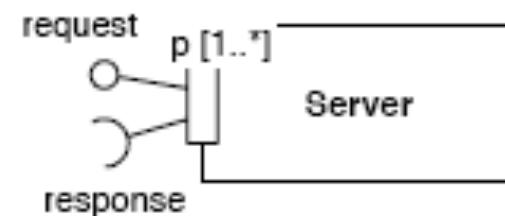
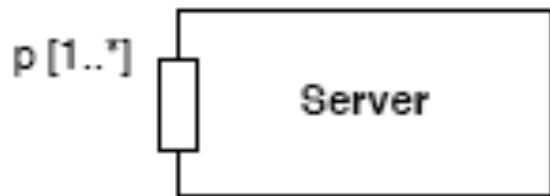
- Documentación de la Vista Funcional con UML-2.0
  - Interfaces
  - Puertos
  - Clasificadores Estructurados
  - Componentes
  - Conectores

- Interfaces
  - Requeridas
  - Provistas

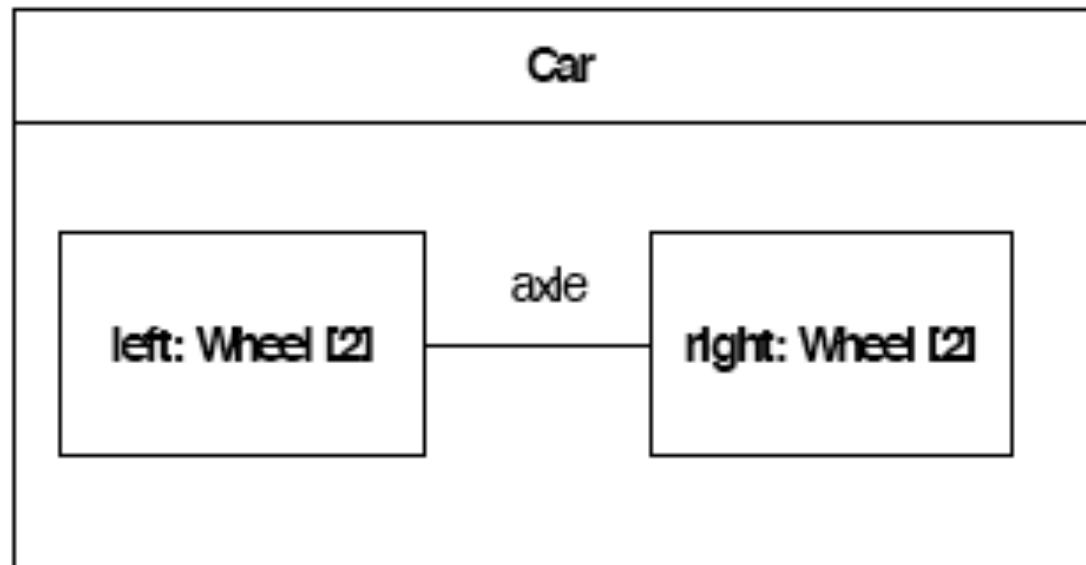


Tomado de [2] página 12

- Puertos
  - Similar a las interfaces, describe como un clasificador interactúa con el ambiente
  - A diferencia de las interfaces cada puerto es un punto de interacción diferente
  - Pueden tener tipos y Multiplicidad
  - Puede tener interfaces



- Clasificadores Estructurados
  - Una nueva manera de representar descomposición de clasificadores



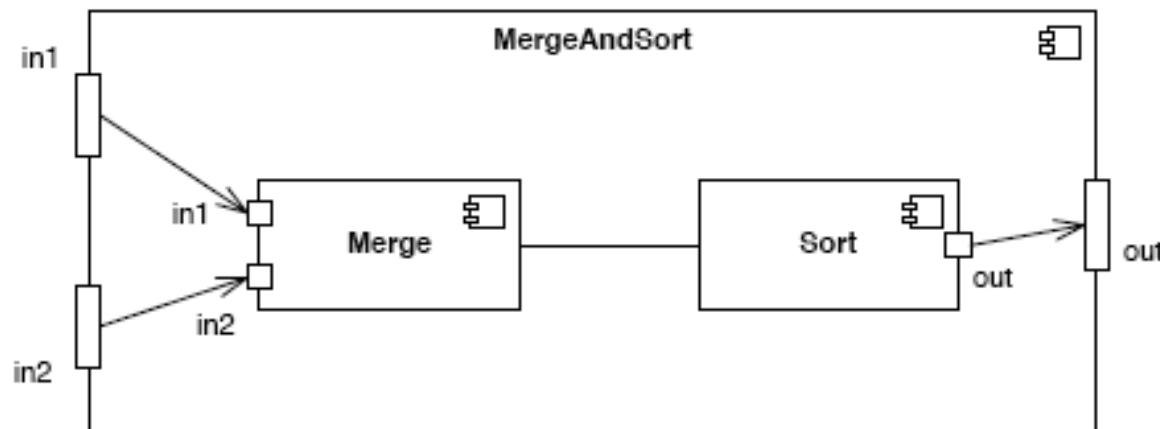
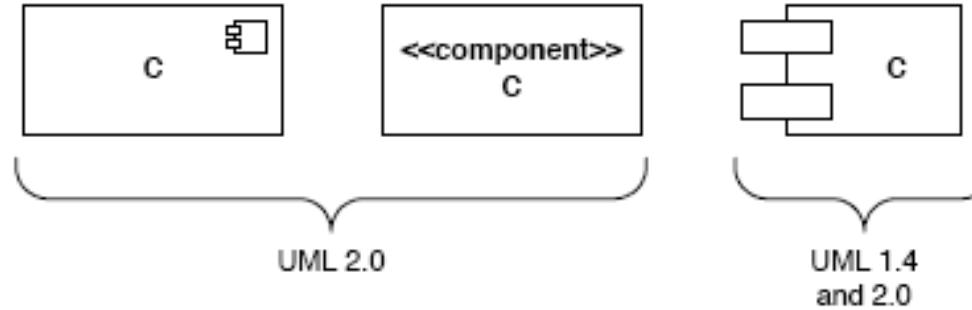
Tomado de [2] página 14



- Componentes

- UML 1.4 “a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces” [OMG 01, p. 2-31]
- UML 2.0 “a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment” [OMG 03, p. 136].
- Extienden el comportamiento de las clases (a nivel de metamodelo)

# Punto de Vista Funcional



Tomado de [2] página 16

- Principales Modelos
  - Componente Conector
    - Representan la ejecución del sistema
  - Module
    - Representan la construcción del sistema



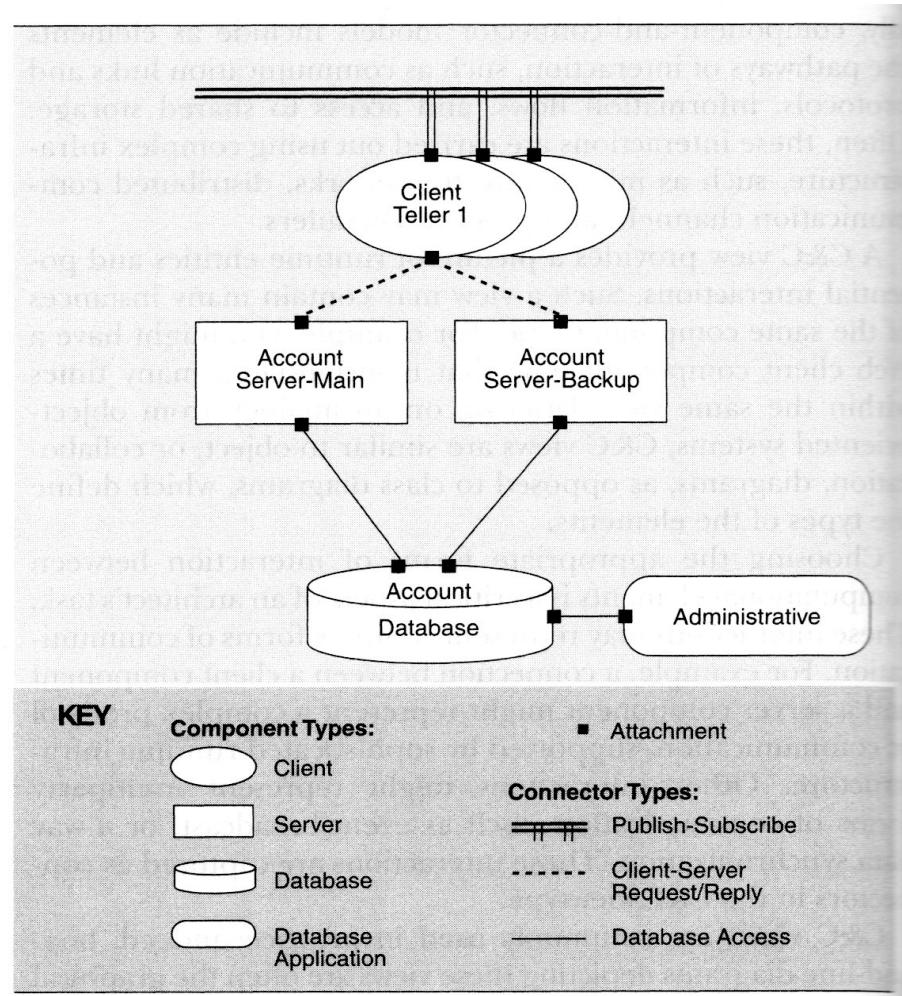
# Conceptos Básicos

## Component-and-Connector Viewtype

Elementos	Components Connectors
Relaciones	Attachment (puertos y conectores)
Propiedades	Nombre Tipo
Topología	No tiene restricciones
Usos	<ul style="list-style-type: none"><li>•Razonar sobre atributos de calidad del sistema durante su ejecución</li><li>•Identificar los principales componentes en ejecución</li><li>•Identificar controlflow y dataflow</li><li>•Identificar paralelismo</li></ul>
Notaciones	<ul style="list-style-type: none"><li>•Informales</li><li>•ADLs</li><li>•UML</li></ul>

# Conceptos Básicos

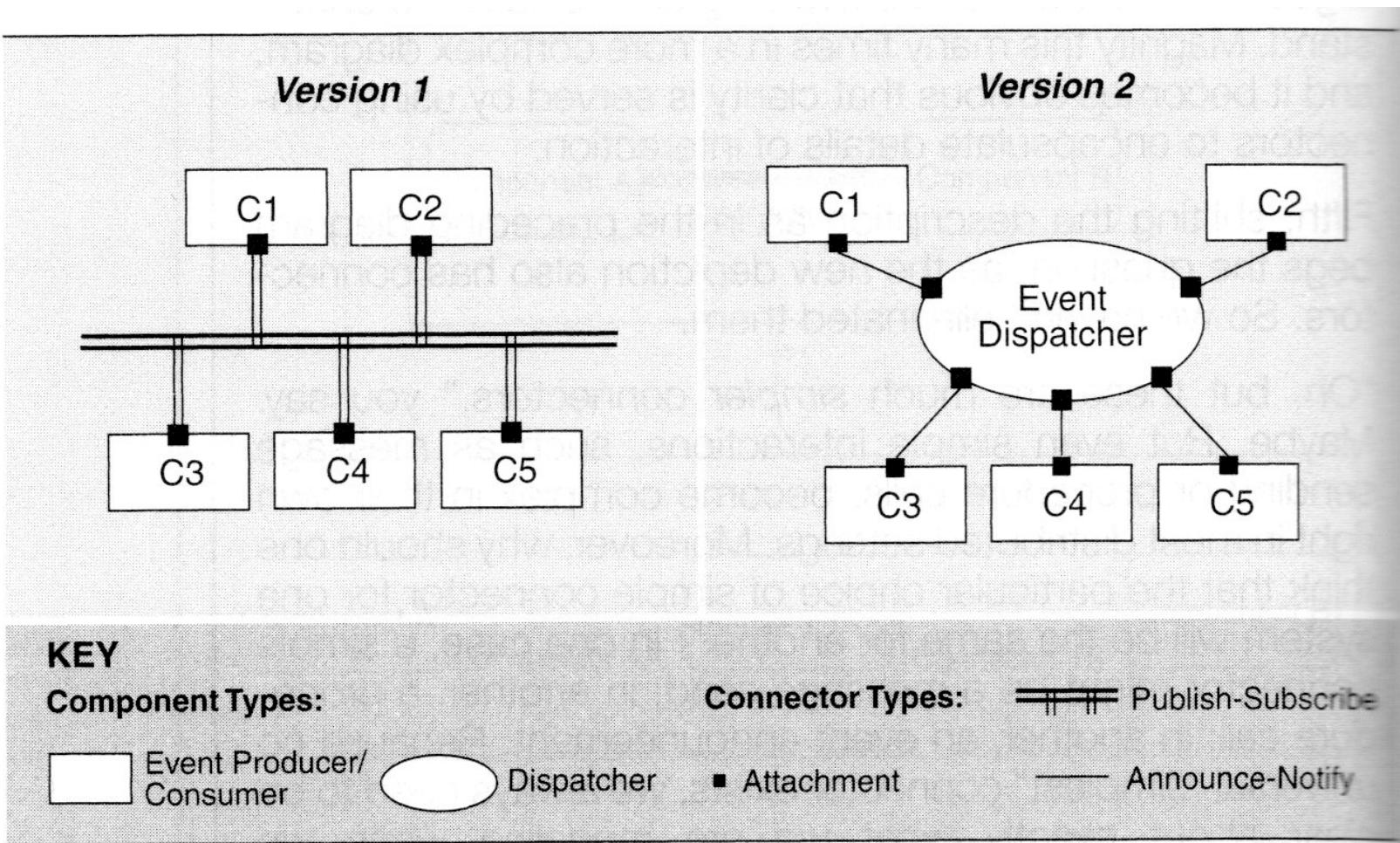
Ejemplo de notación informal para representar un estilo C&C



Tomado de “Documenting Software Architectures” pag 104

# Conceptos Básicos

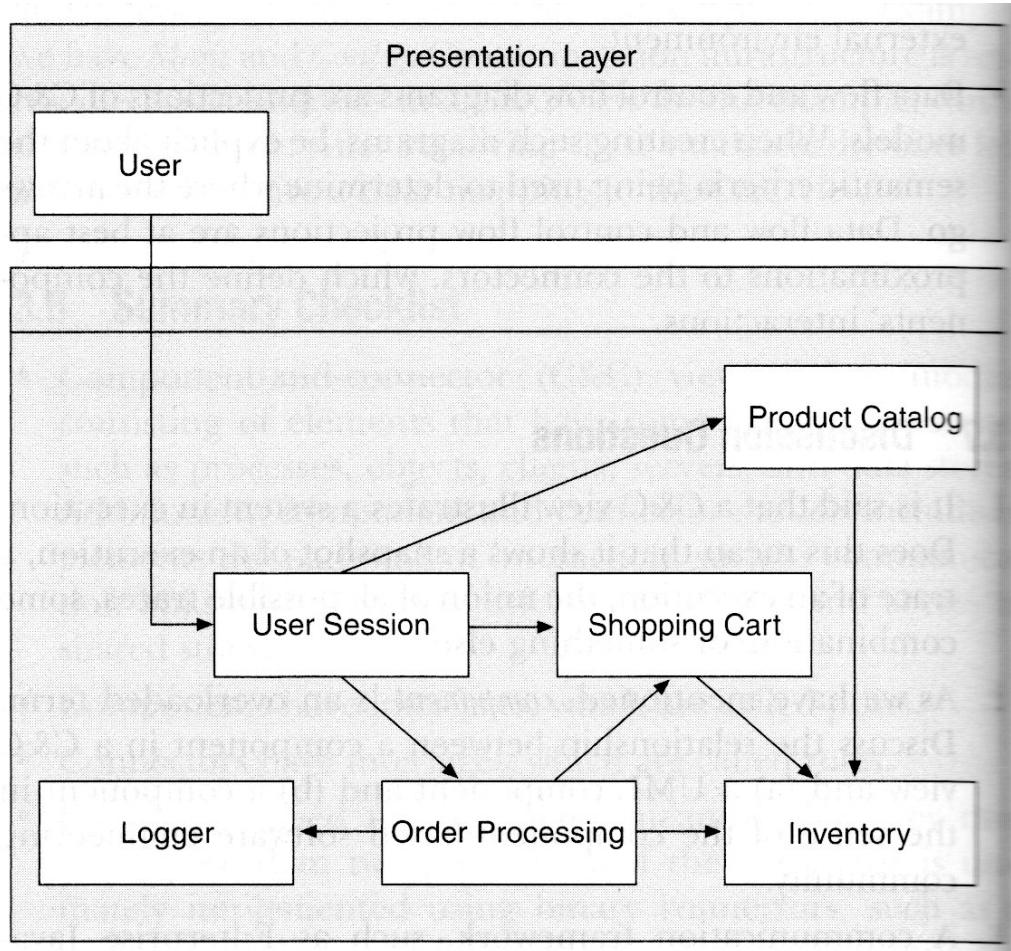
Ejemplo de notación informal para representar un estilo C&C



Tomado de “Documenting Software Architectures” pag 114

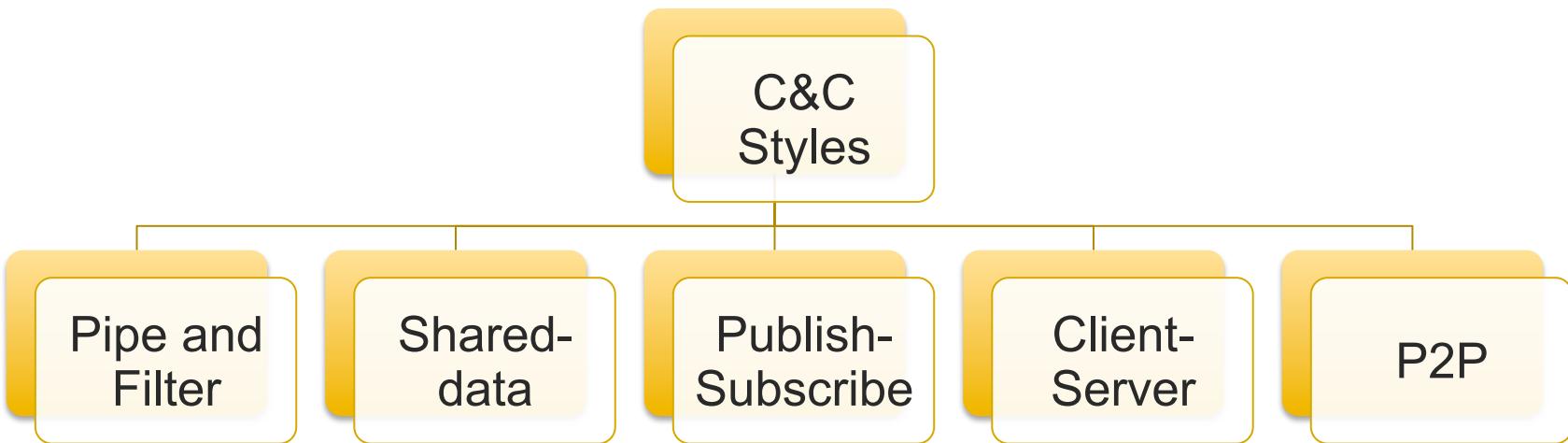
# Conceptos Básicos

Ejemplo de notación informal para representar un estilo C&C. Correcto?



Tomado de "Documenting Software Architectures" pag 122

# Conceptos Básicos

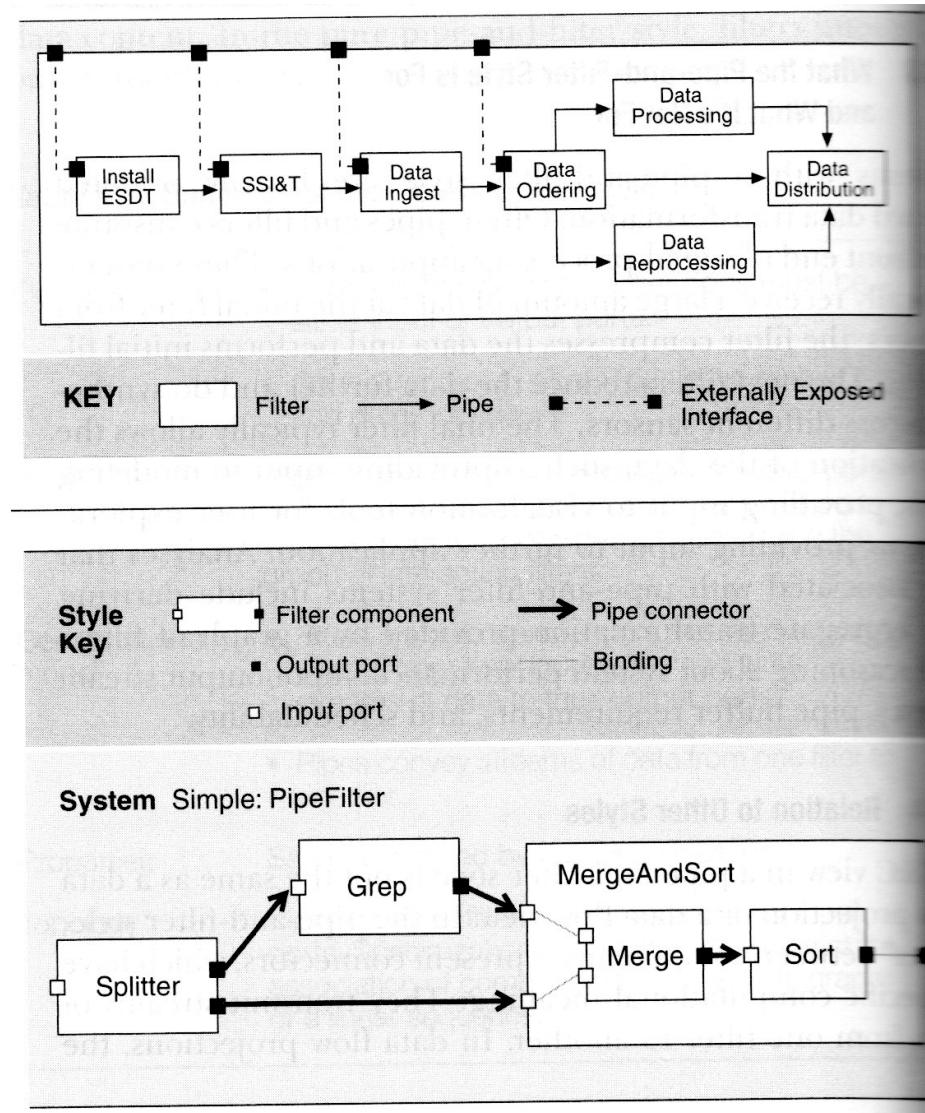


# Conceptos Básicos

## Pipe-and-Filter Style

Elementos	pipes, filtros
Relaciones	Attachment
Propiedades	
Modelo	Filtros transformar datos Pipes transportar datos
Topología	Salida de un pipe se conecta con la entrada de otro pipe
Notación	Informales UML

# Conceptos Básicos



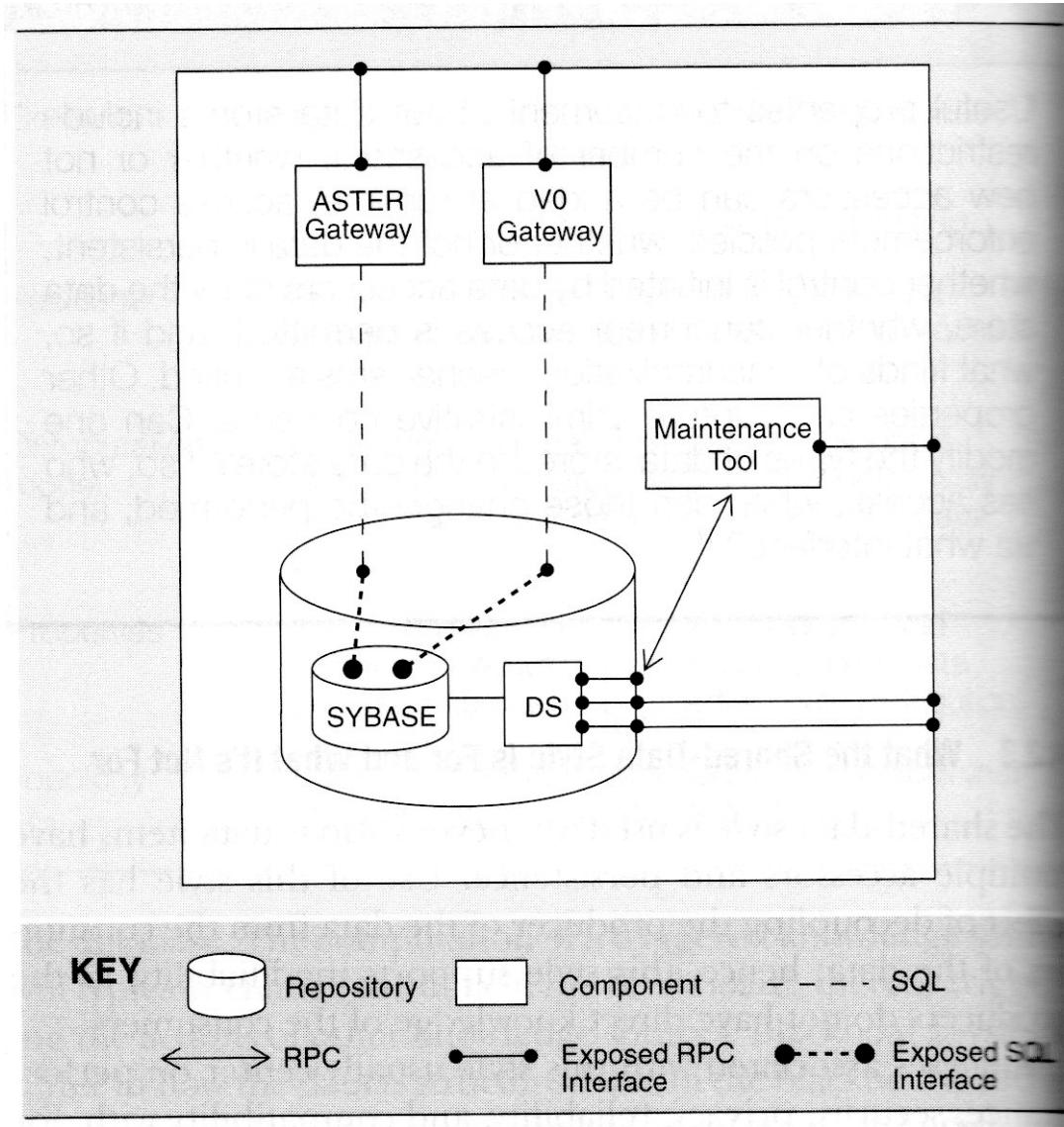
Tomado de “Documenting Software Architectures” pag 128

# Conceptos Básicos

## Shared-Data Style

Elementos	Componentes Conectores
Relaciones	Attachment
Propiedades	
Modelo	Comunicación entre los componentes es mediada por un depósito compartido de datos
Topología	
Notación	Informales UML

# Conceptos Básicos



Tomado de "Documenting Software Architectures" pag 132



# Conceptos Básicos

## Publish-Subscribe Style

Elementos	Componentes Conectores
Relaciones	Attachment
Propiedades	
Modelo	Sistema de componentes independientes que anuncian eventos y reaccionan a otros eventos anunciados
Topología	
Notación	Informales UML

# Conceptos Básicos

Ejemplo de notación informal para representar un estilo publish-subscribe

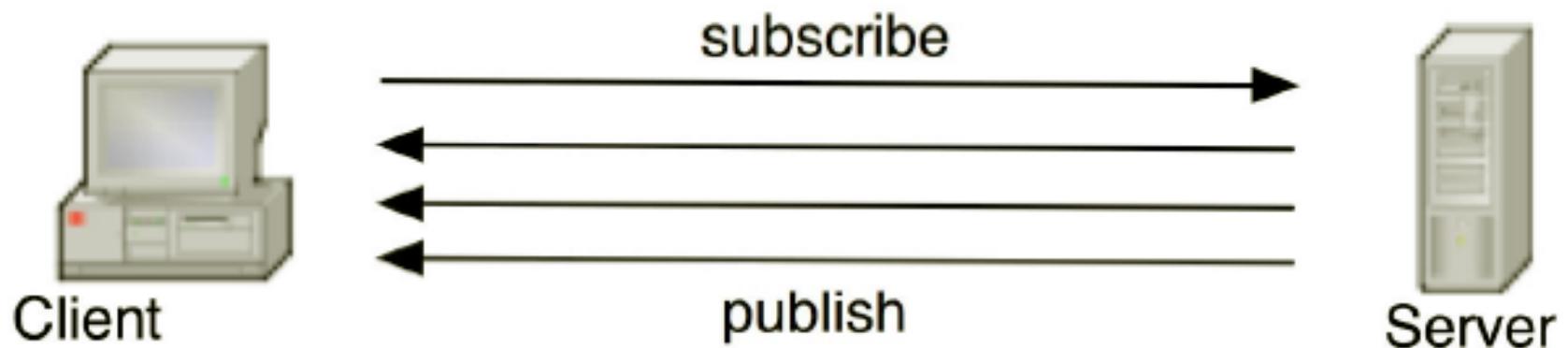


Imagen tomada de: "Service Oriented Architectures" - Phillip J. Windley

# Conceptos Básicos

## Client-Server Style

Elementos	Componentes (cliente, servidores) Conectores (request/reply)
Relaciones	Attachment
Propiedades	
Modelo	Los clientes inician actividades, solicitan servicios de los servidores y esperan las respuestas
Topología	
Notación	Informales UML



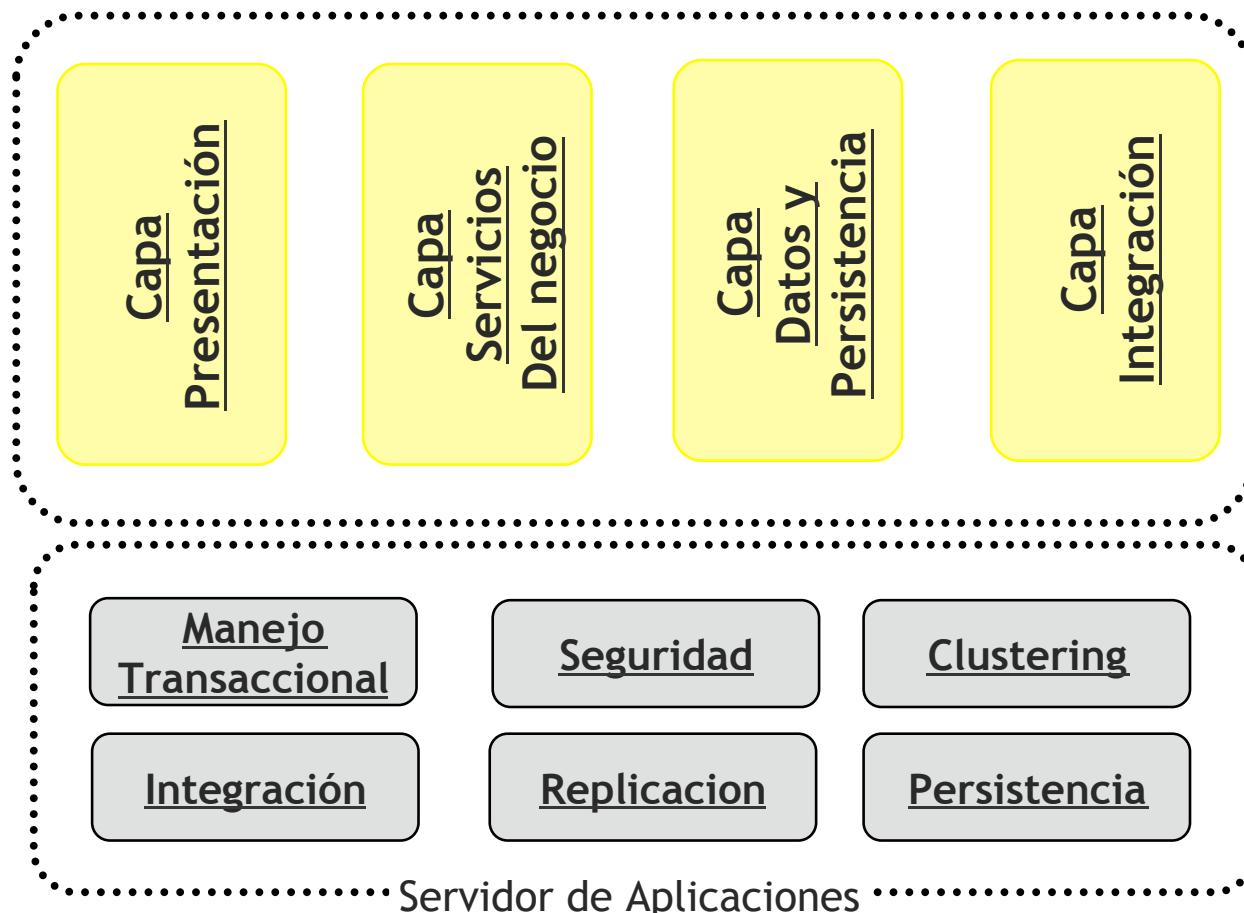
# Conceptos Básicos

## N-Tier Style

Elementos	Componentes (cliente, servidores, presentación) Conectores (request/reply, publish/subscribe)
Relaciones	Attachment
Propiedades	
Modelo	Los clientes se conectan a componentes de negocio y solicitan servicios. La persistencia está separada de la presentacion
Topología	
Notación	Informales UML

# Conceptos Básicos

Ejemplo de notación informal para representar un estilo N-Tier



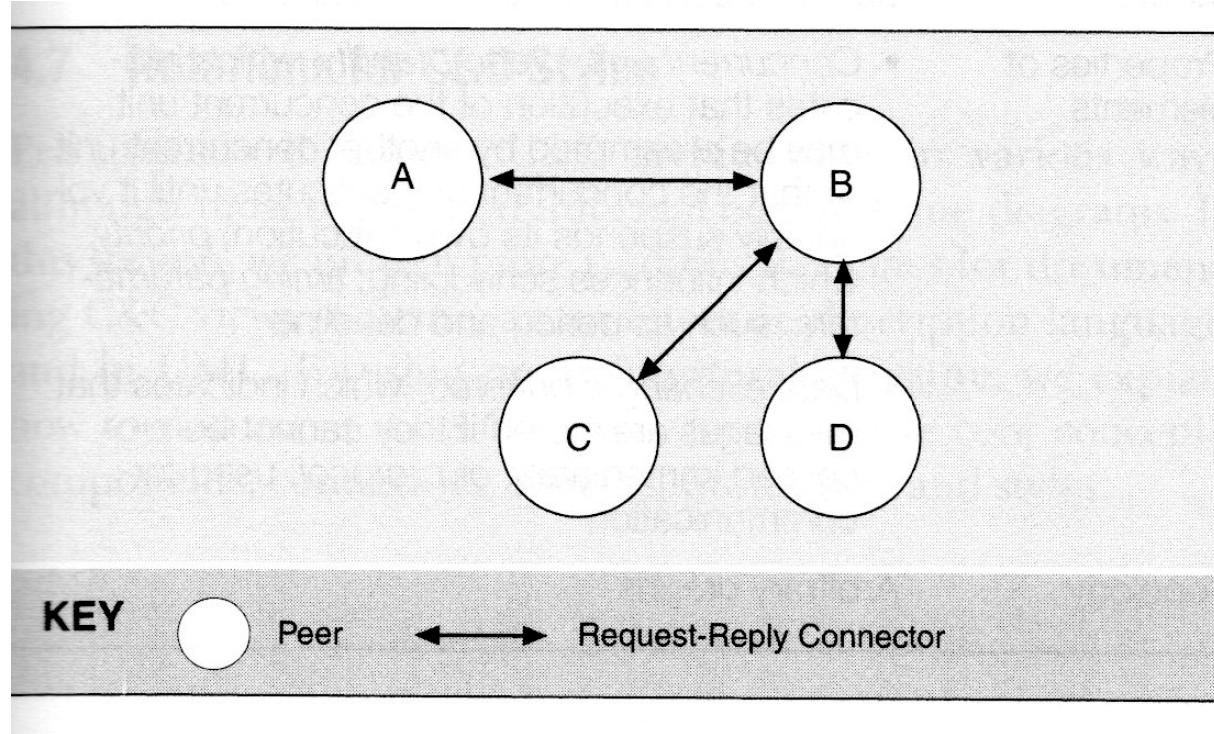
# Conceptos Básicos

## Peer-toPeer Style

Elementos	Componentes (peers) Conectores
Relaciones	Attachment
Propiedades	
Modelo	El sistema se basa en peers que cooperan entre sí, requeriendo servicios entre ellos
Topología	
Notación	Informales

# Conceptos Básicos

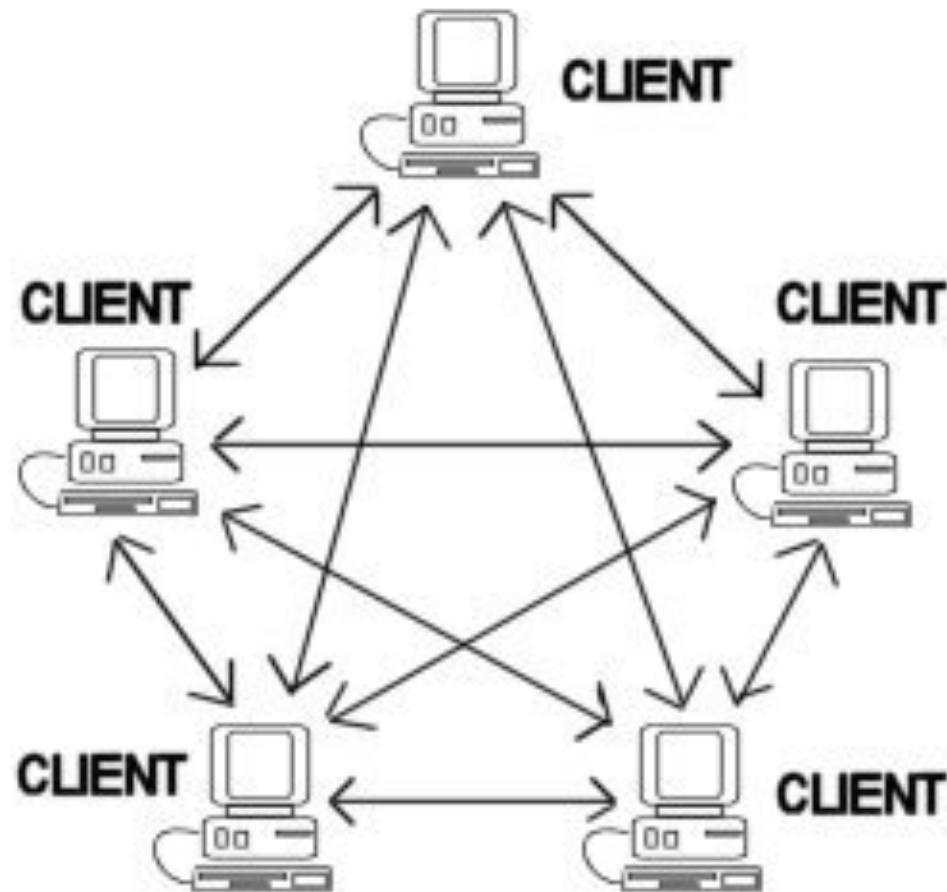
Ejemplo de notación informal para representar un estilo peer-to-peer



Tomado de “Documenting Software Architectures” pag 141

# Conceptos Básicos

Ejemplo de notación informal para representar un estilo peer-to-peer





# Conceptos Básicos

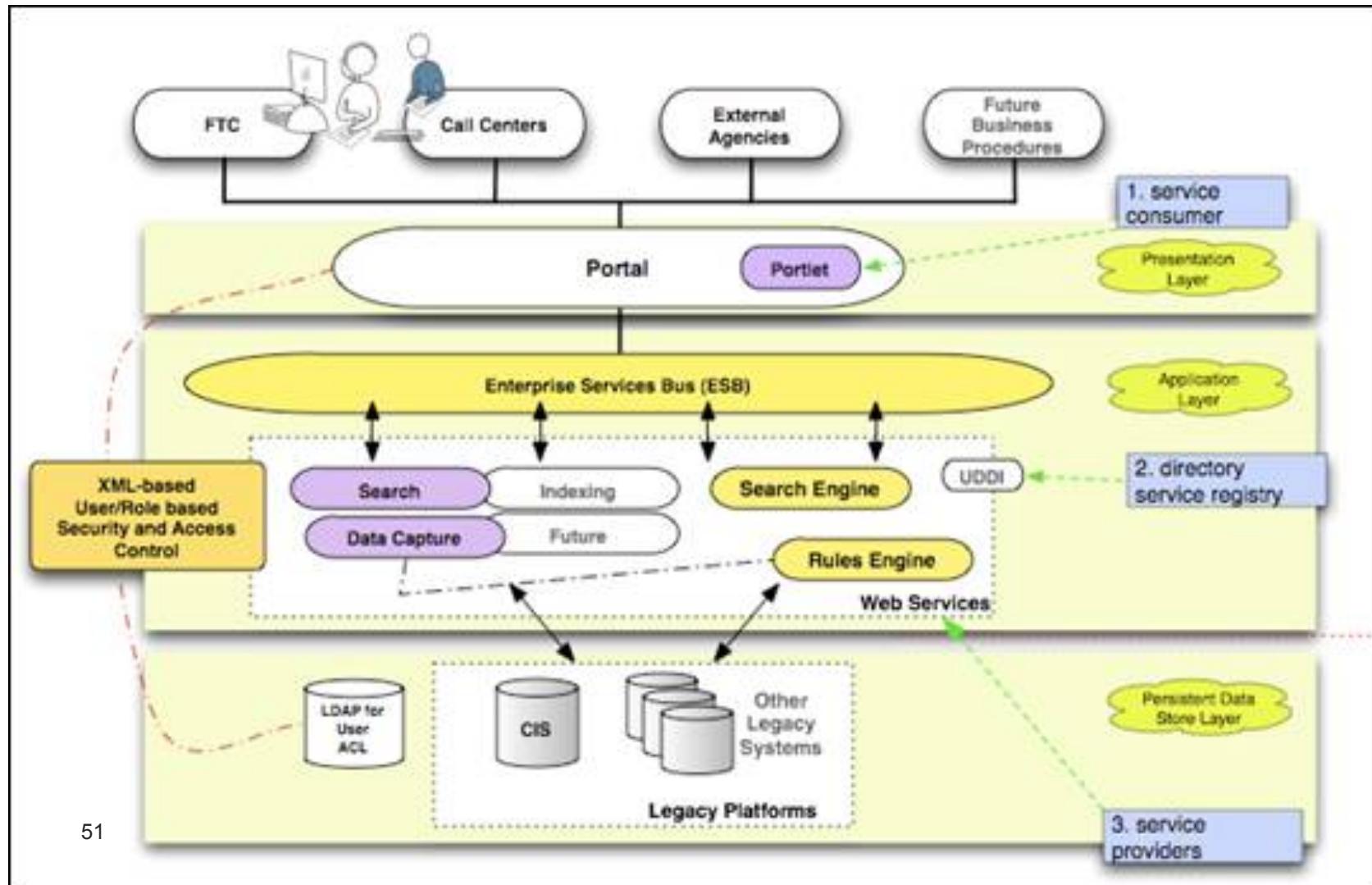
## Service-Oriented Architecture (SOA) Style

Elementos	Componentes (cliente, servidores, presentación, servicios, bus) Conectores (request/reply, publish/subscribe)
Relaciones	Attachment
Propiedades	
Modelo	Orquestación de servicios
Topología	
Notación	Informales UML

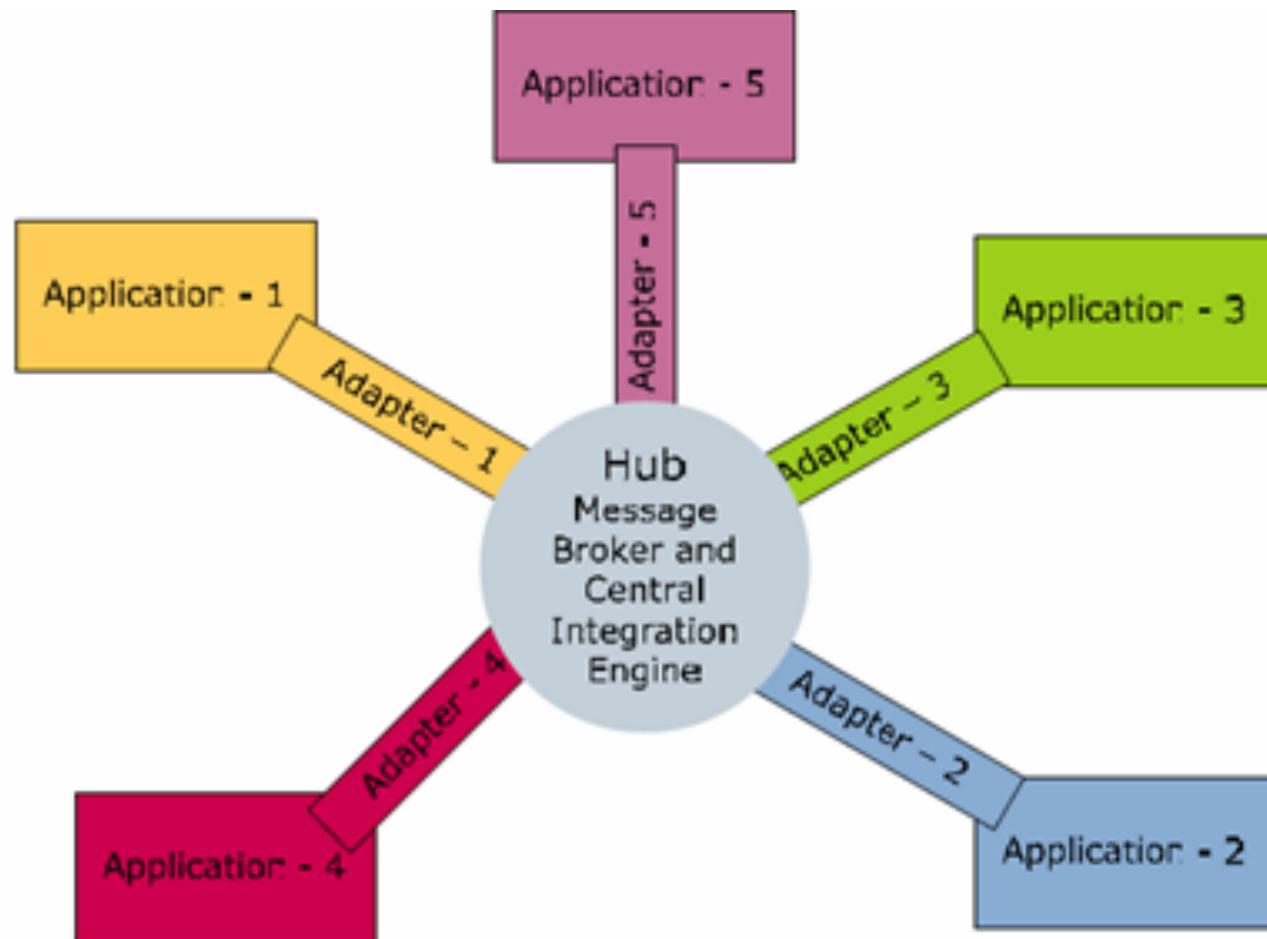


# Conceptos Básicos

Ejemplo de notación informal para representar un estilo SOA

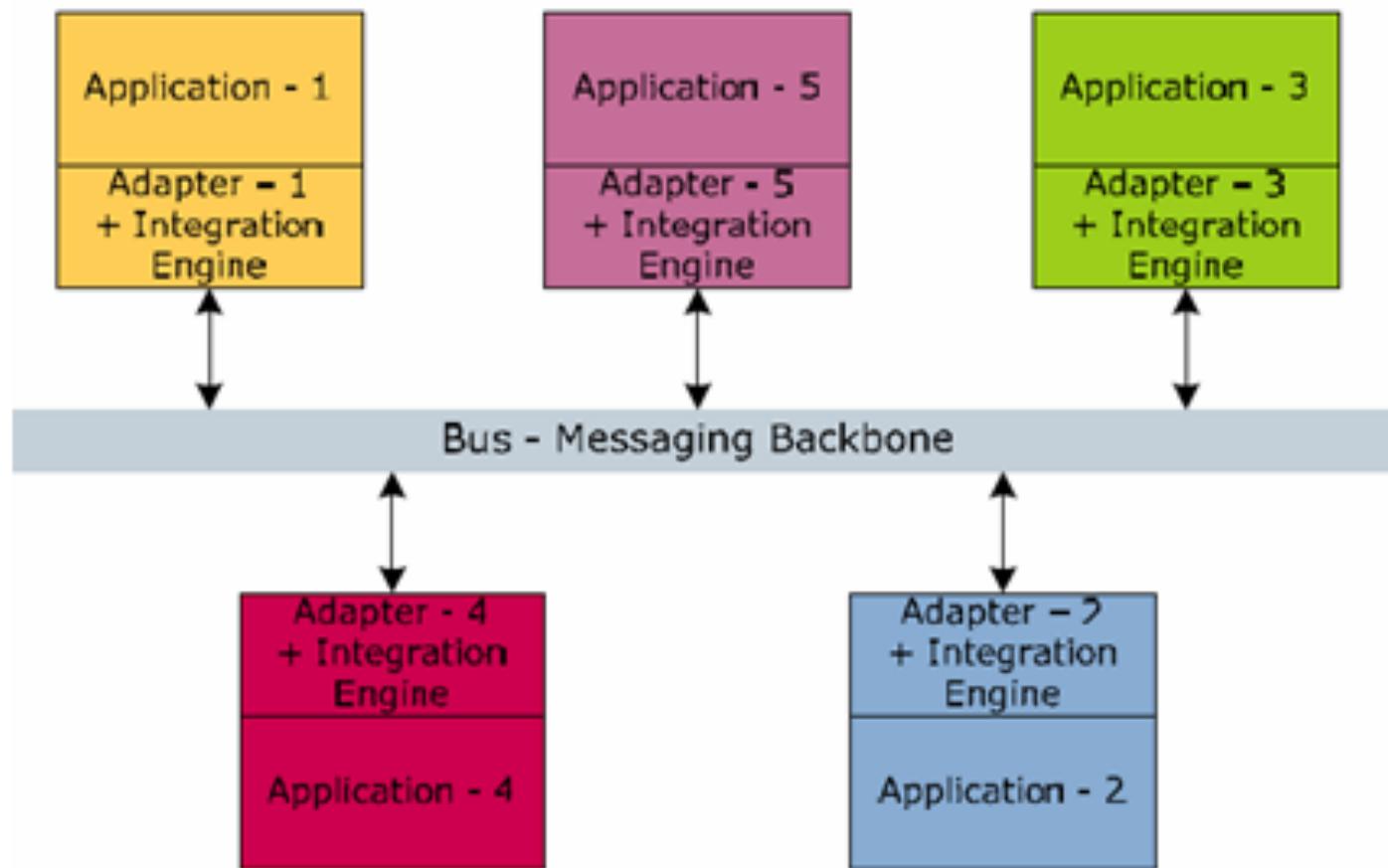


## Estilo Hub and Spoke





## Estilo Message Bus





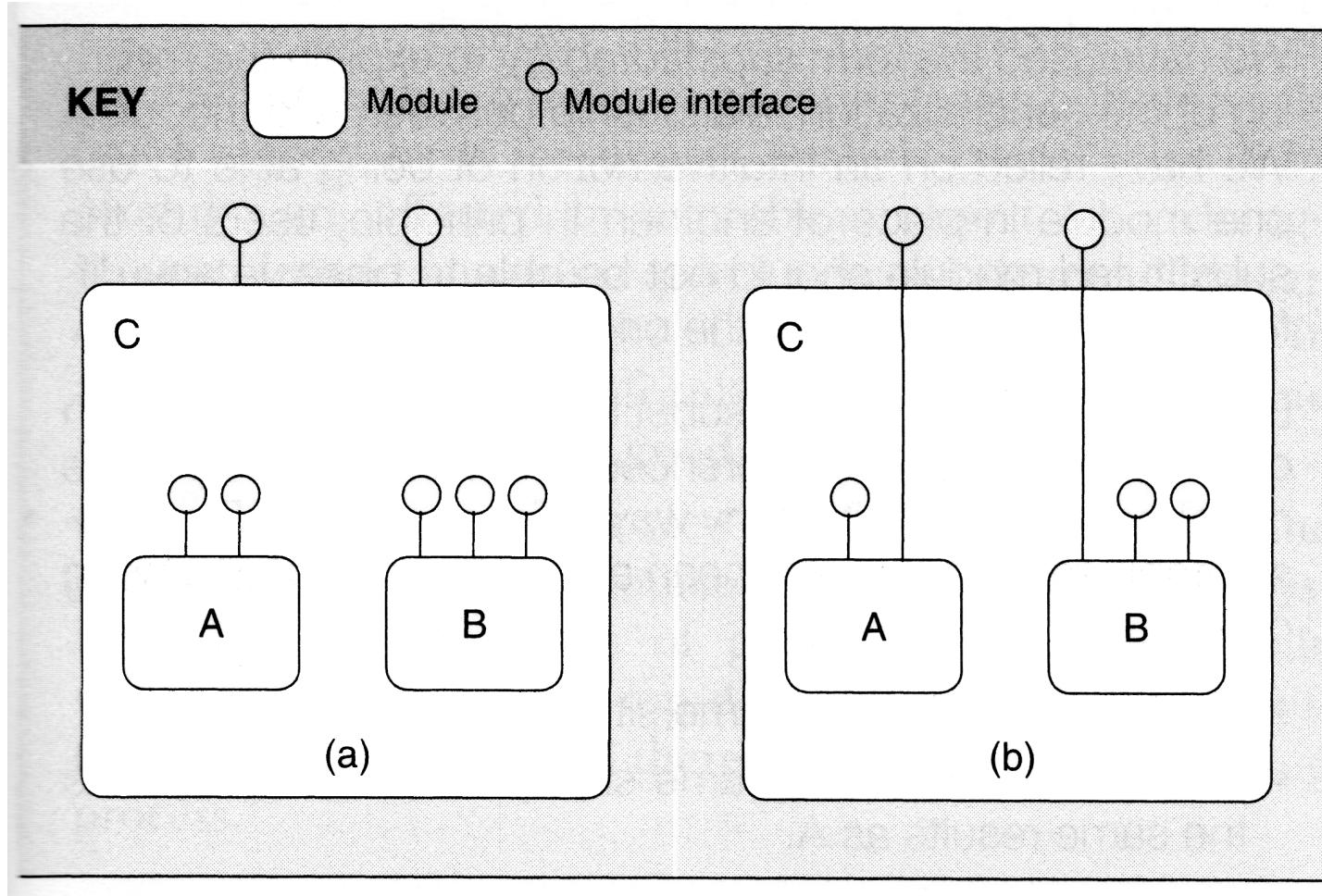
# Conceptos Básicos

## Module Viewtype

Elementos	Módulo
Relaciones	Is-part-of Is-a Depends-on
Propiedades	Nombre Responsabilidades del módulo
Topología	No tiene restricciones
Usos	<ul style="list-style-type: none"><li>•Provee una maqueta del código fuente</li><li>•Trazabilidad de requerimientos</li><li>•Análisis de Impacto</li><li>•Permite explicar la funcionalidad del sistema a los stakeholders</li></ul>
Notaciones	<ul style="list-style-type: none"><li>•Informales</li><li>•UML (Clases, paquetes)</li></ul>

# Conceptos Básicos

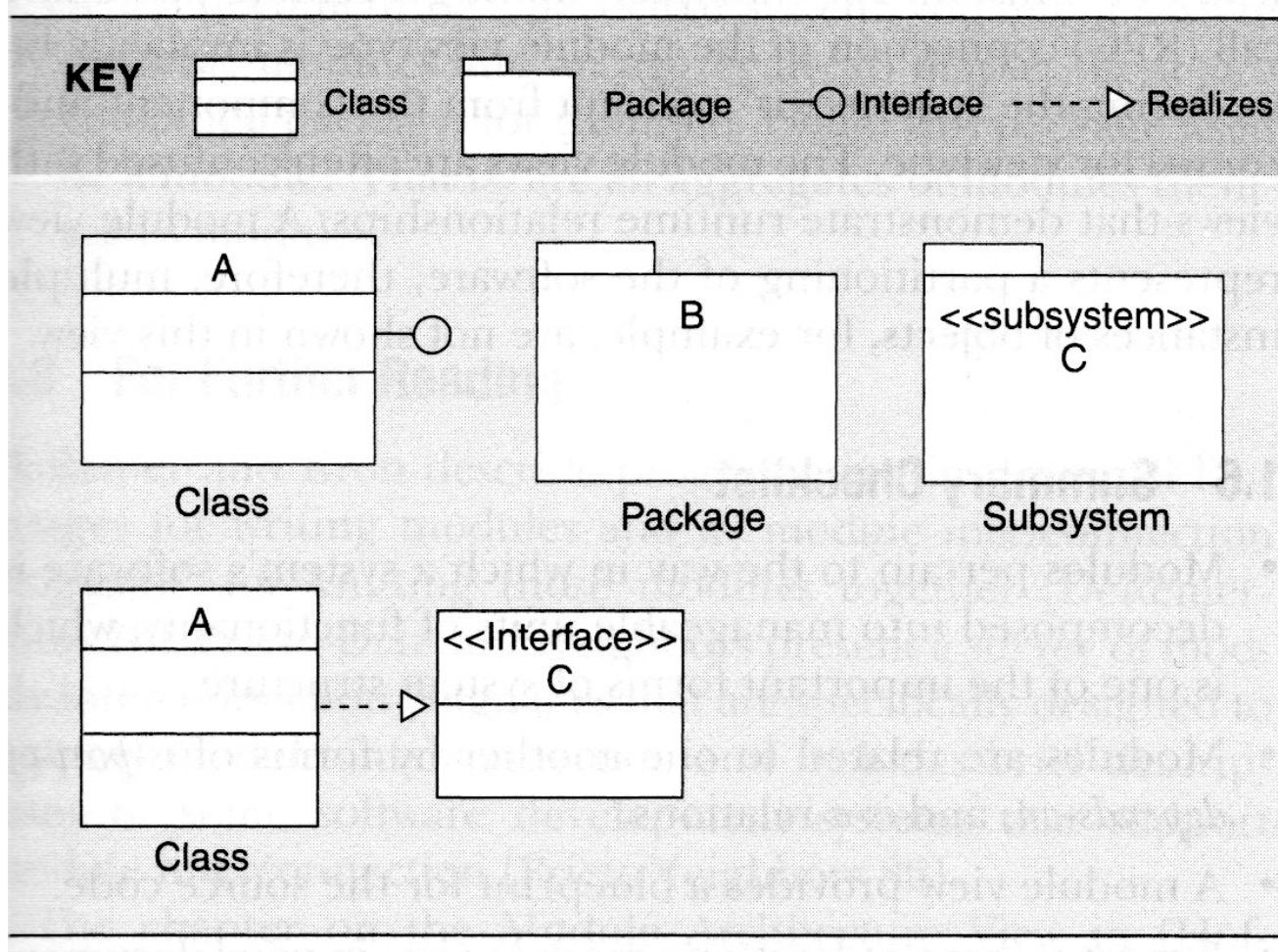
Ejemplos de notaciones para el tipo de vista *Module*



Tomado de “Documenting Software Architectures” pag 45

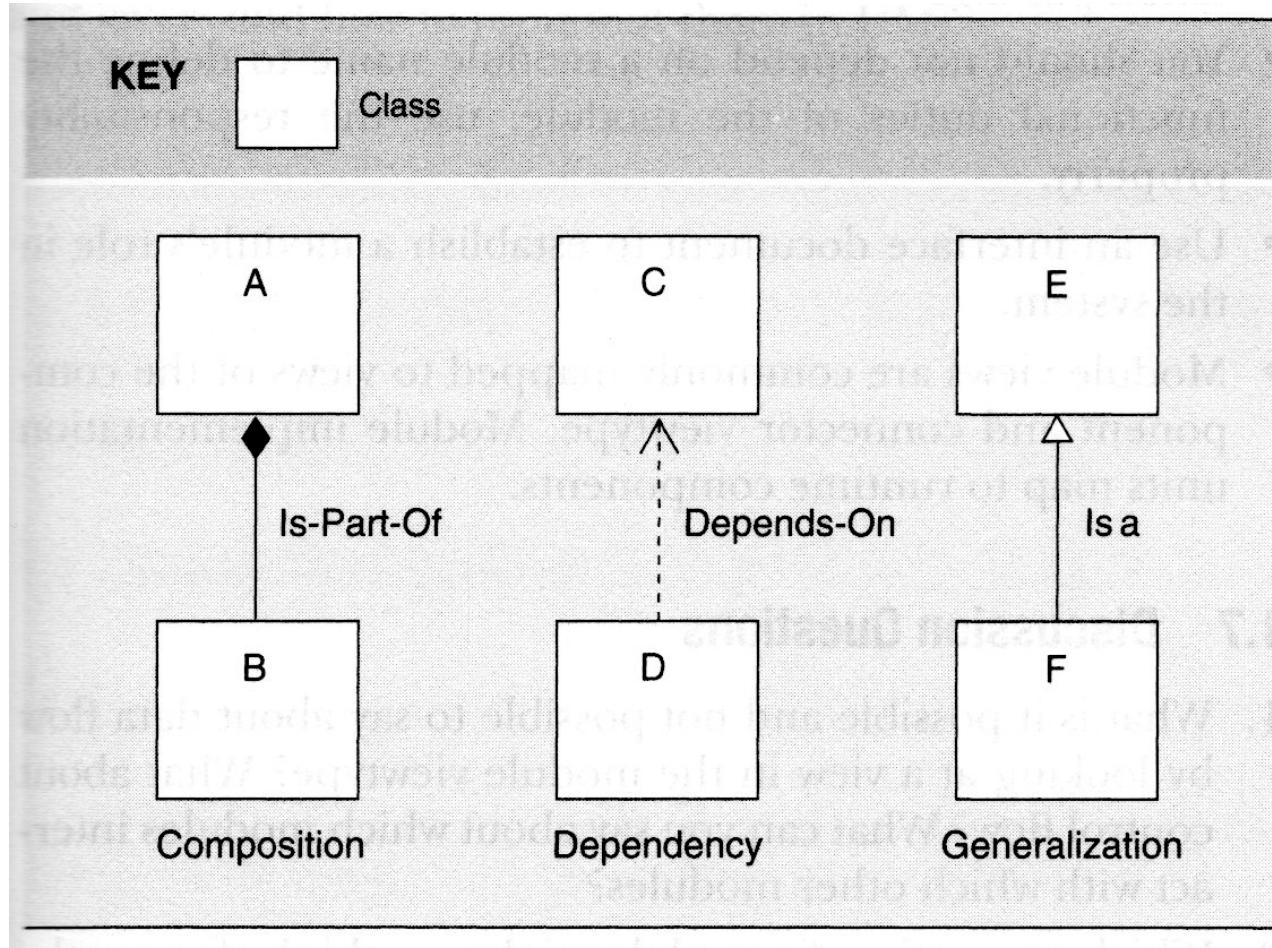
# Conceptos Básicos

Ejemplos de notaciones para el tipo de vista *Module*



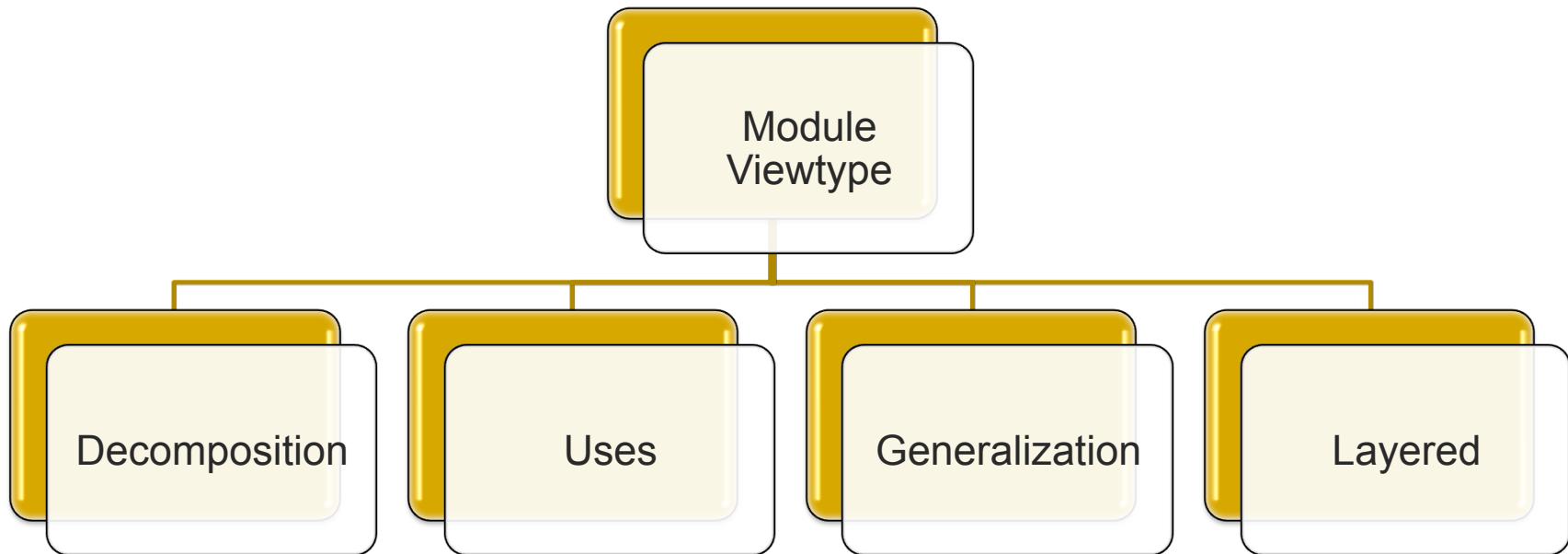
# Conceptos Básicos

Ejemplos de notaciones para el tipo de vista *Module*



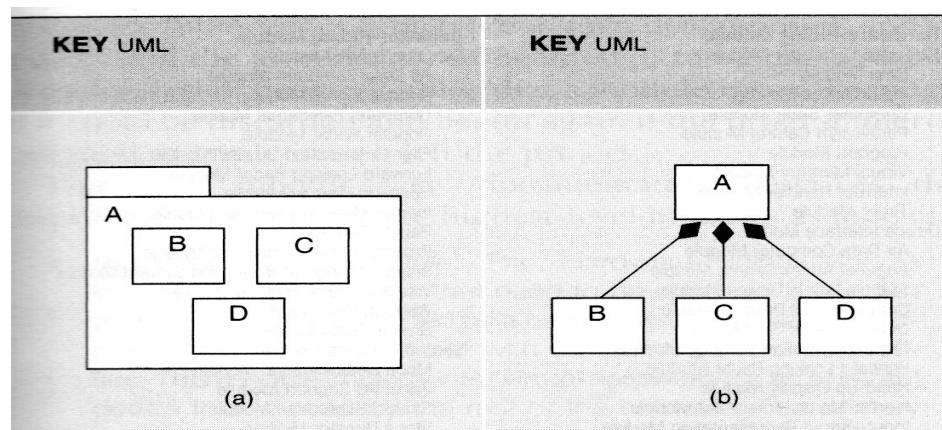
Tomado de “Documenting Software Architectures” pag 49

# Conceptos Básicos



# Conceptos Básicos

Decomposition Style	
Elementos	Módulos
Relaciones	is-part-of
Propiedades	
Propiedades de las relaciones	Visibilidad
Topología	No se permiten ciclos Un módulo solo es parte de otro módulo
Notación	UML



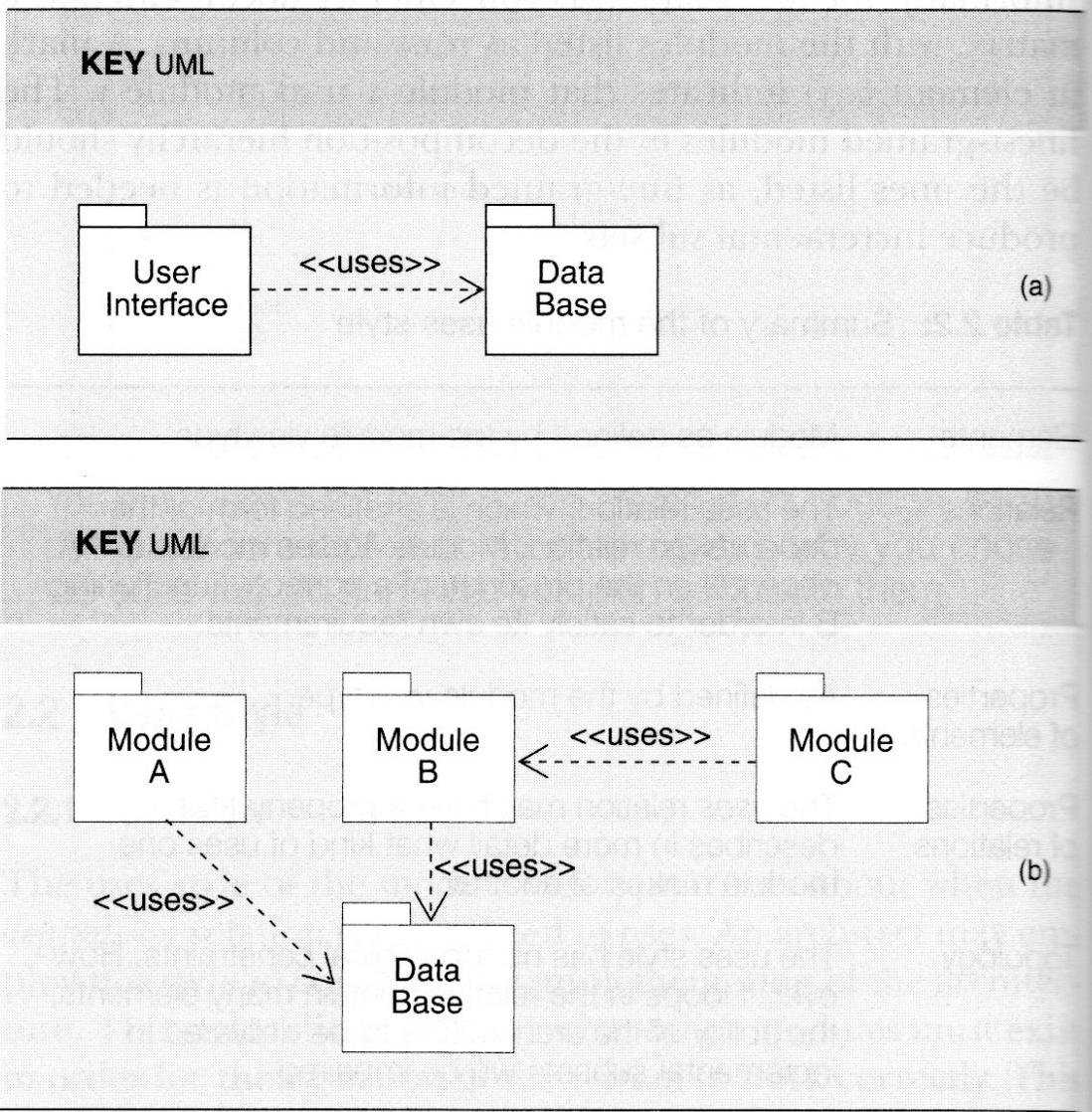
Tomado de "Documenting Software Architectures" pag 57

# Conceptos Básicos

## Uses Style

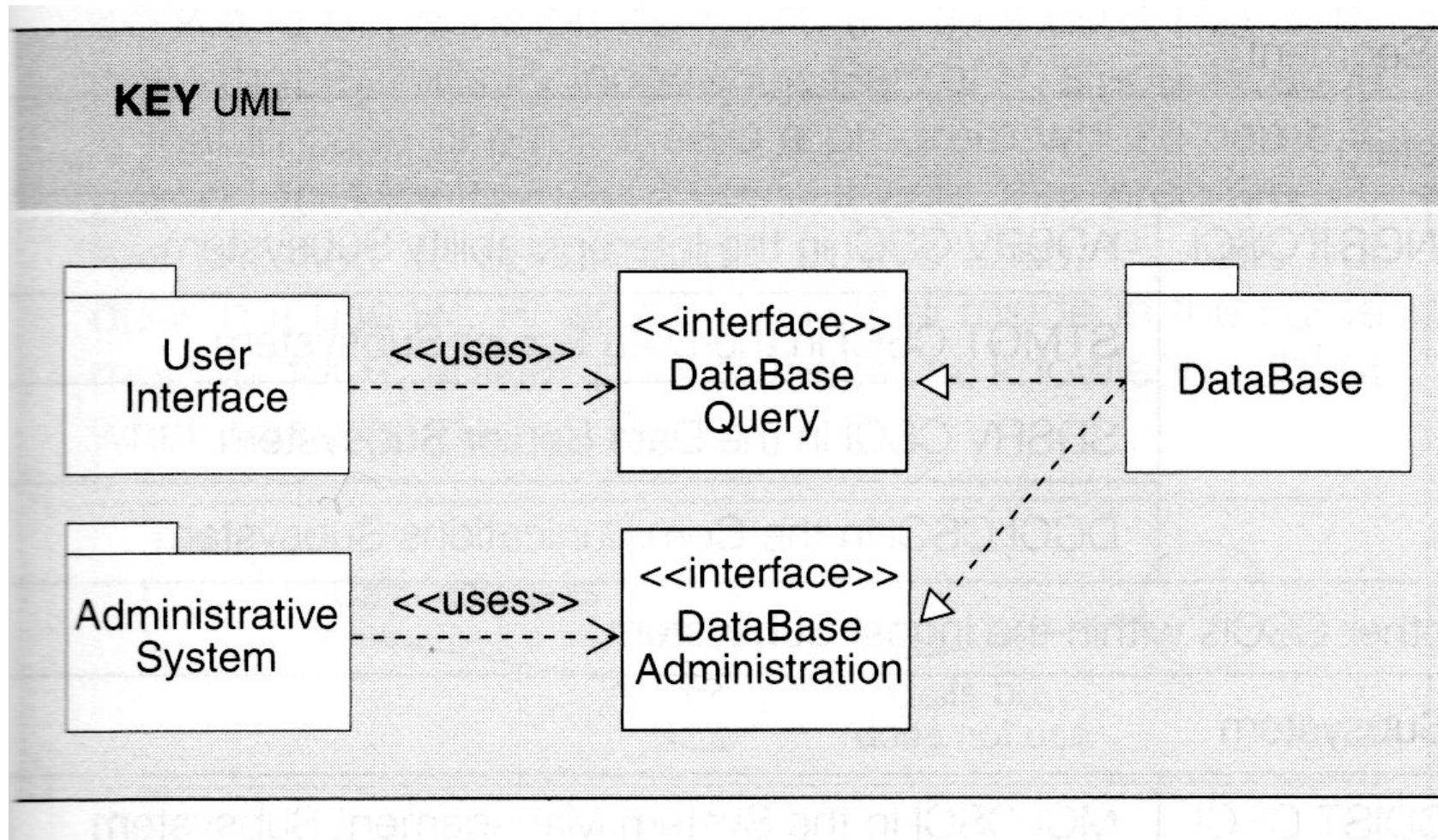
Elementos	Módulos
Relaciones	depends-on
Propiedades	
Propiedades de las relaciones	
Topología	
Notación	UML

# Conceptos Básicos



Tomado de “Documenting Software Architectures” pag 66

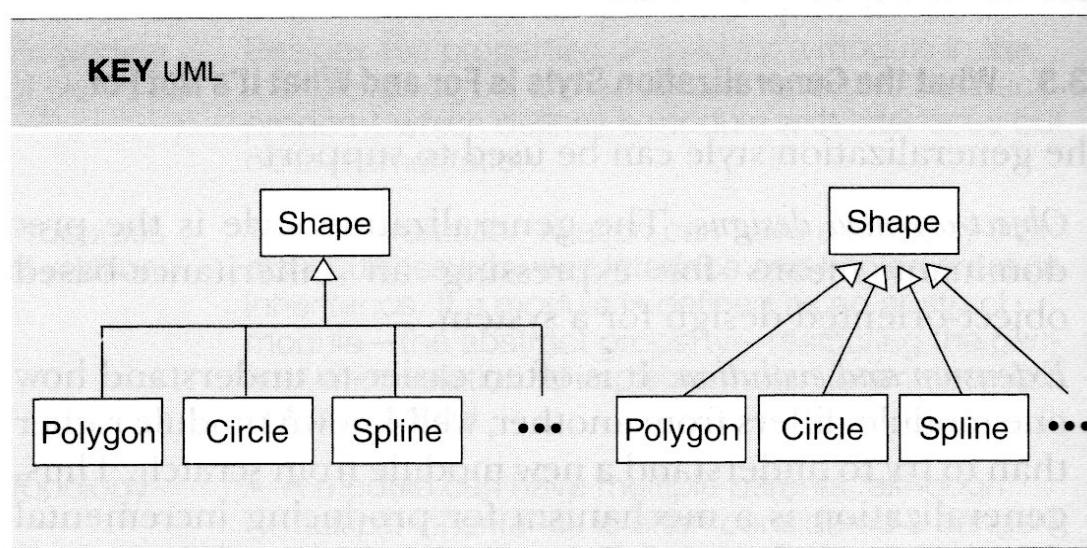
# Conceptos Básicos



Tomado de "Documenting Software Architectures" pag 67

# Conceptos Básicos

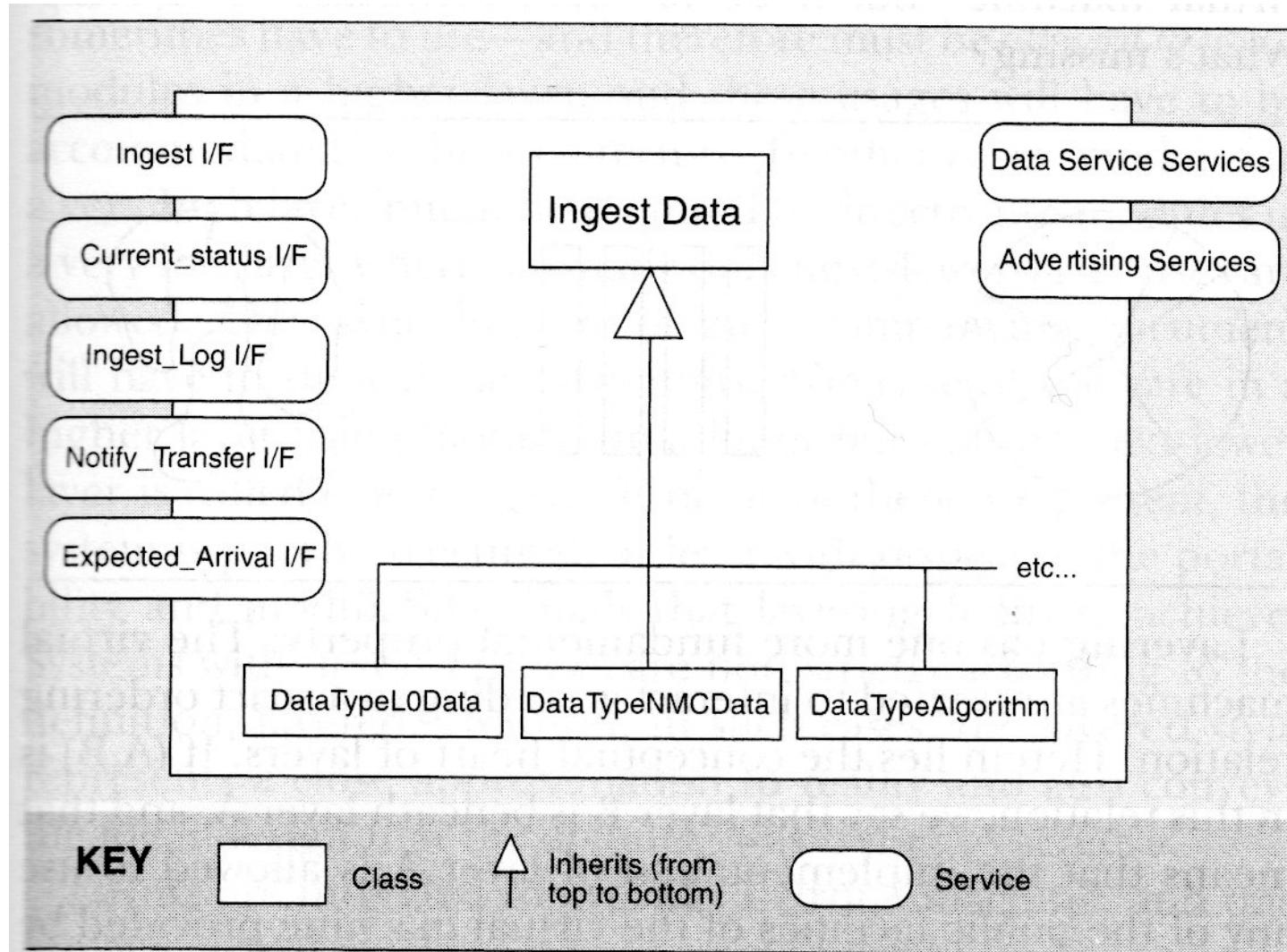
Generalization Style	
Elementos	Módulos
Relaciones	Generalization
Propiedades	
Propiedades de las relaciones	Diferenciar entre interfaces e implementación
Topología	Se permite herencia múltiple
Notación	UML



Tomado de "Documenting Software Architectures" pag 74



# Conceptos Básicos



Tomado de “Documenting Software Architectures” pag 77

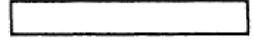
# Conceptos Básicos

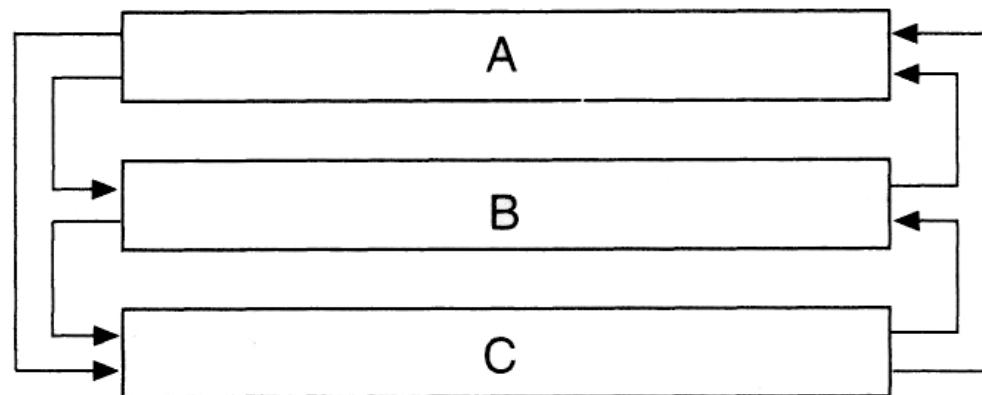
## Layered Style

Elementos	Capas
Relaciones	Autorizado para usar
Propiedades	Nombre de la capa
Propiedades de las relaciones	Diferenciar entre interfaces e implementación
Topología	No se permite el intercambio de posiciones en la jerarquía
Notación	Informales Segmentadas Anillos UML (mediante paquetes)

Ejemplo de notación informal para representar un estilo por capas

---

**KEY** (informal notation)     Layer     $x \rightarrow y$      $x$  is allowed to use  $y$

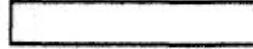


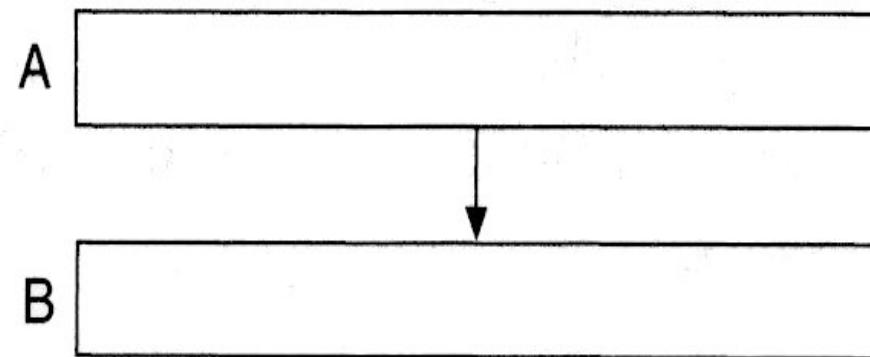
---

Nota algo raro en este ejemplo?

# Conceptos Básicos

Ejemplo de notación informal para representar un estilo por capas

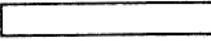
KEY (informal notation)     Layer     $x \rightarrow y$      $x$  is allowed to use  $y$

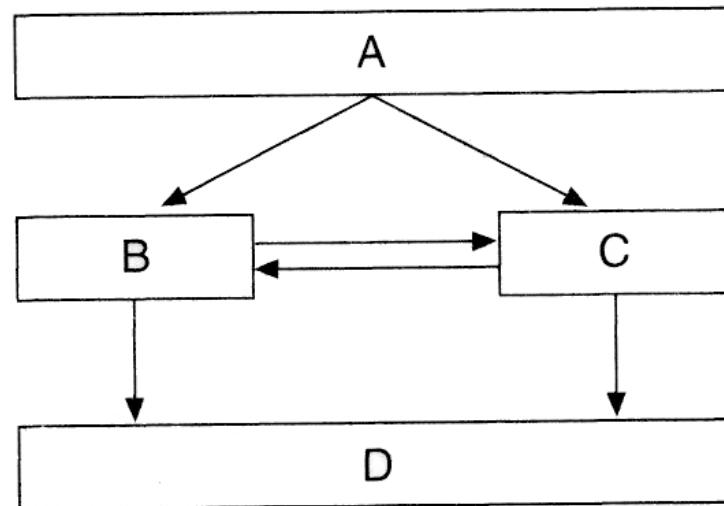


Tomado de "Documenting Software Architectures" pag 83

# Conceptos Básicos

Ejemplo de notación informal para representar un estilo por capas

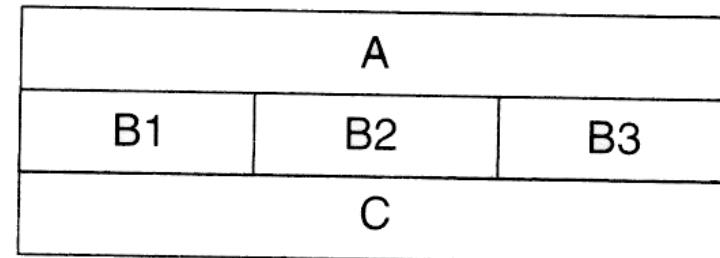
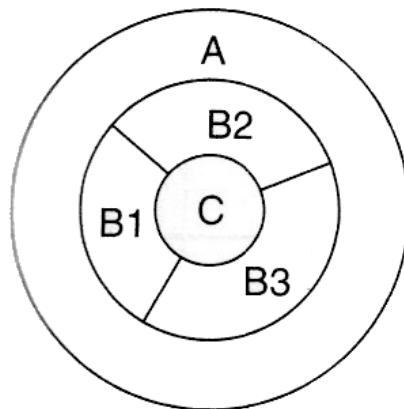
**KEY** (informal notation)     Layer     $x \rightarrow y$      $x$  is allowed to use  $y$



Tomado de “Documenting Software Architectures” pag 84

# Conceptos Básicos

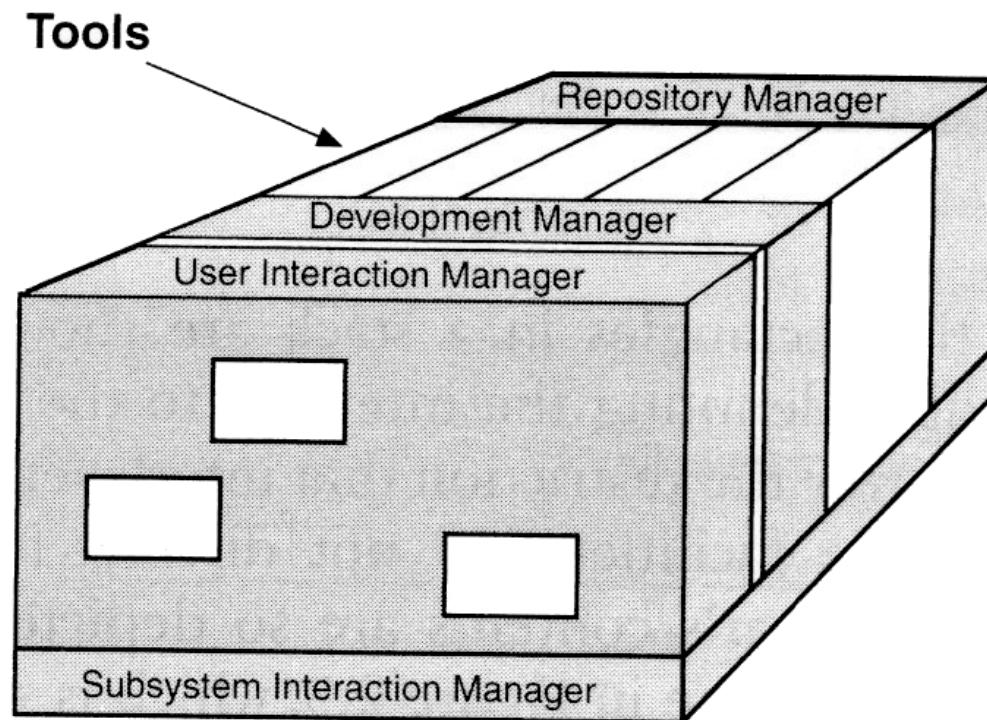
Ejemplo de notación informal para representar un estilo por capas



Tomado de "Documenting Software Architectures" pag 85

# Conceptos Básicos

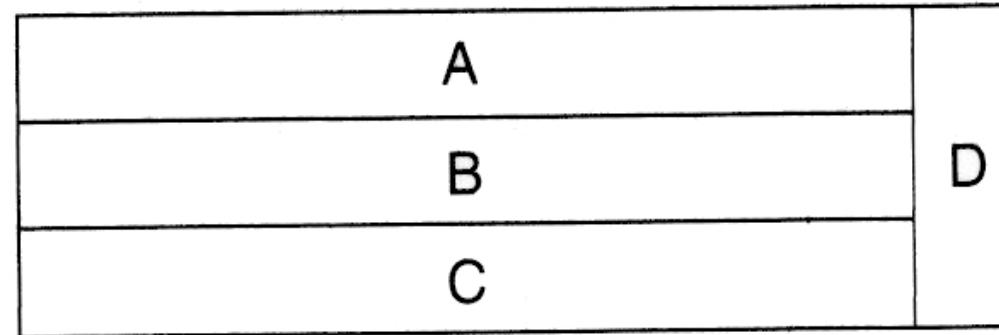
Ejemplo de notación informal para representar un estilo por capas



Tomado de “Documenting Software Architectures” pag 85

# Conceptos Básicos

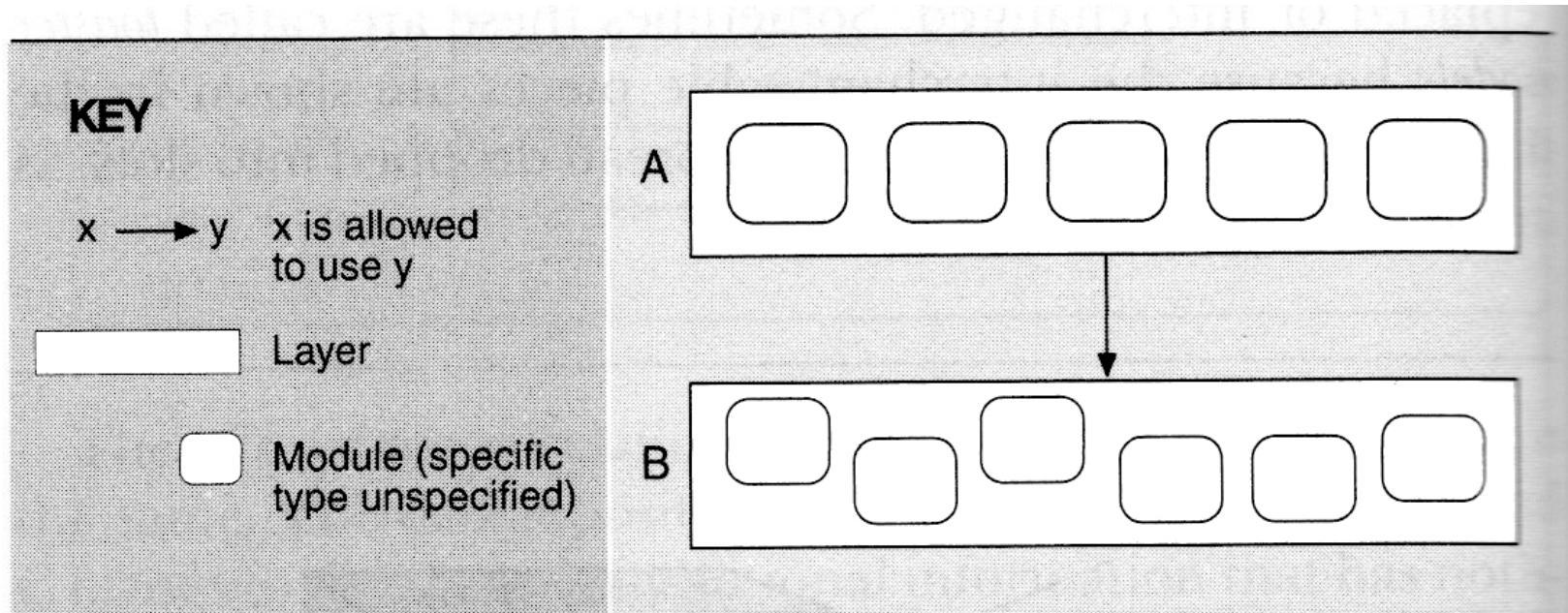
Ejemplo de notación informal para representar un estilo por capas



Tomado de "Documenting Software Architectures" pag 86

# Conceptos Básicos

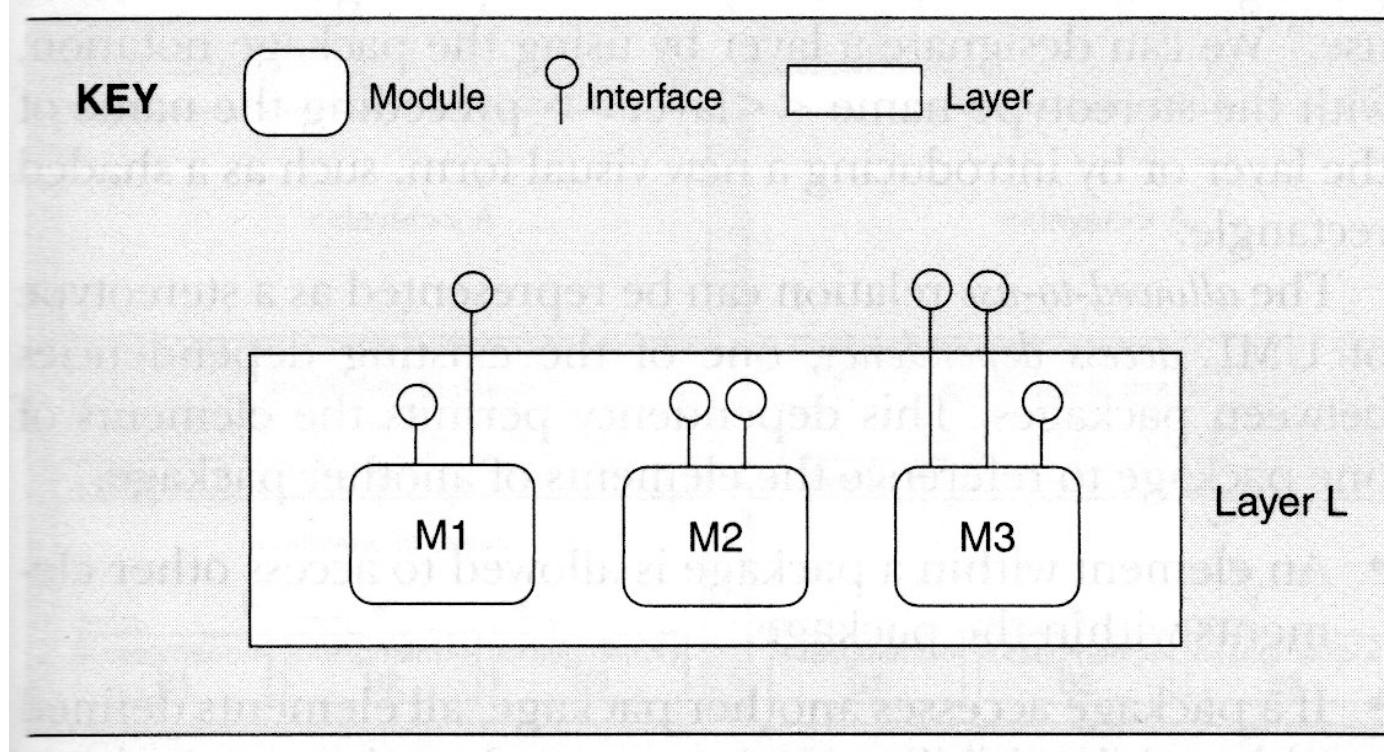
Ejemplo de notación informal para representar un estilo por capas



Tomado de "Documenting Software Architectures" pag 86

# Conceptos Básicos

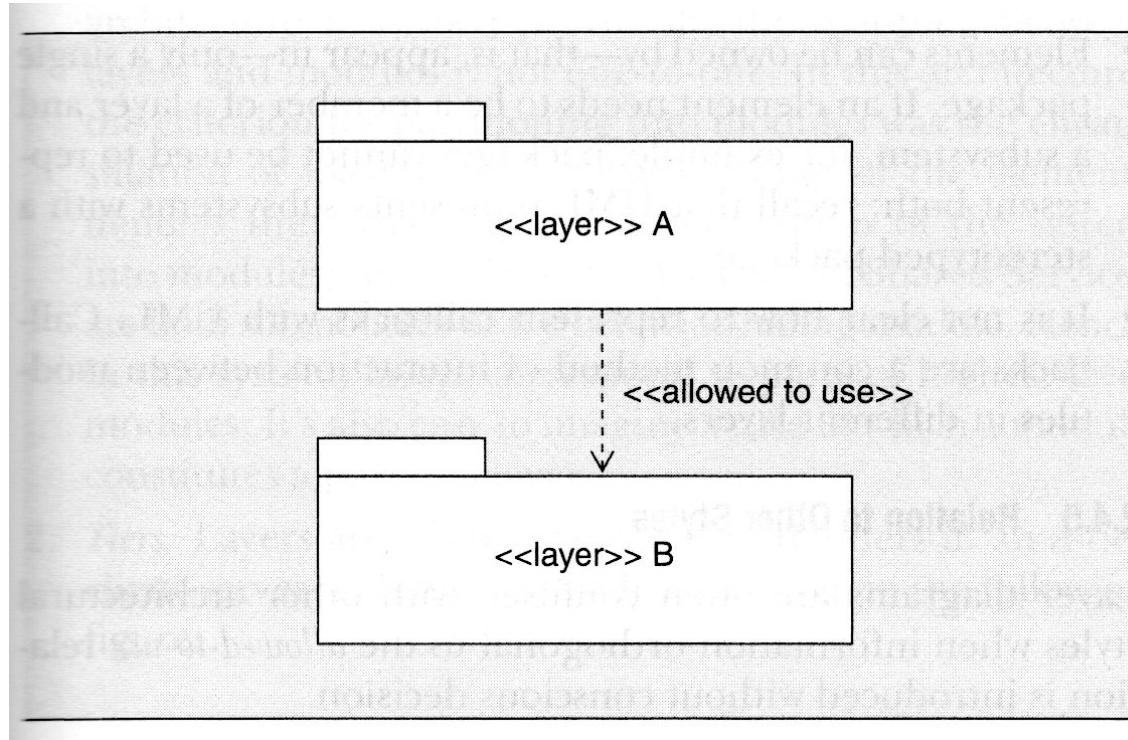
Ejemplo de notación informal para representar un estilo por capas



Tomado de “Documenting Software Architectures” pag 87

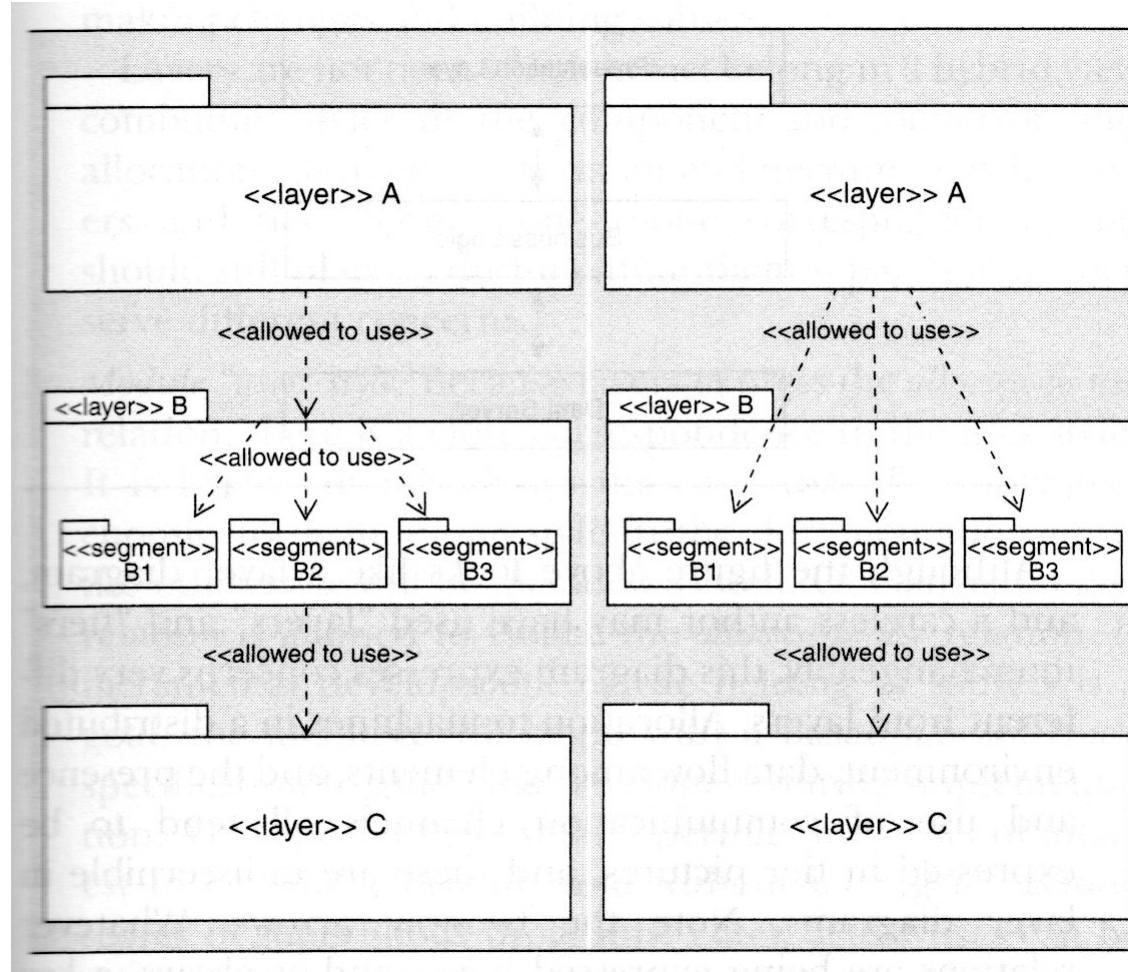
# Conceptos Básicos

Ejemplo de notación UML para representar un estilo por capas



Tomado de "Documenting Software Architectures" pag 87

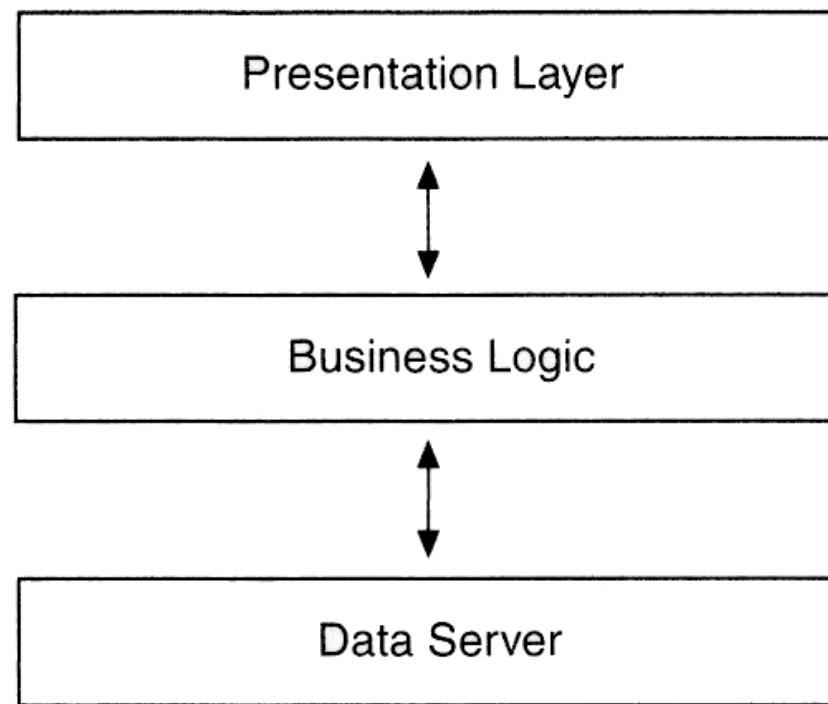
Ejemplo de notación UML para representar un estilo por capas



Tomado de “Documenting Software Architectures” pag 89

# Conceptos Básicos

Qué opina sobre este ejemplo de un estilo arquitectural por capas?

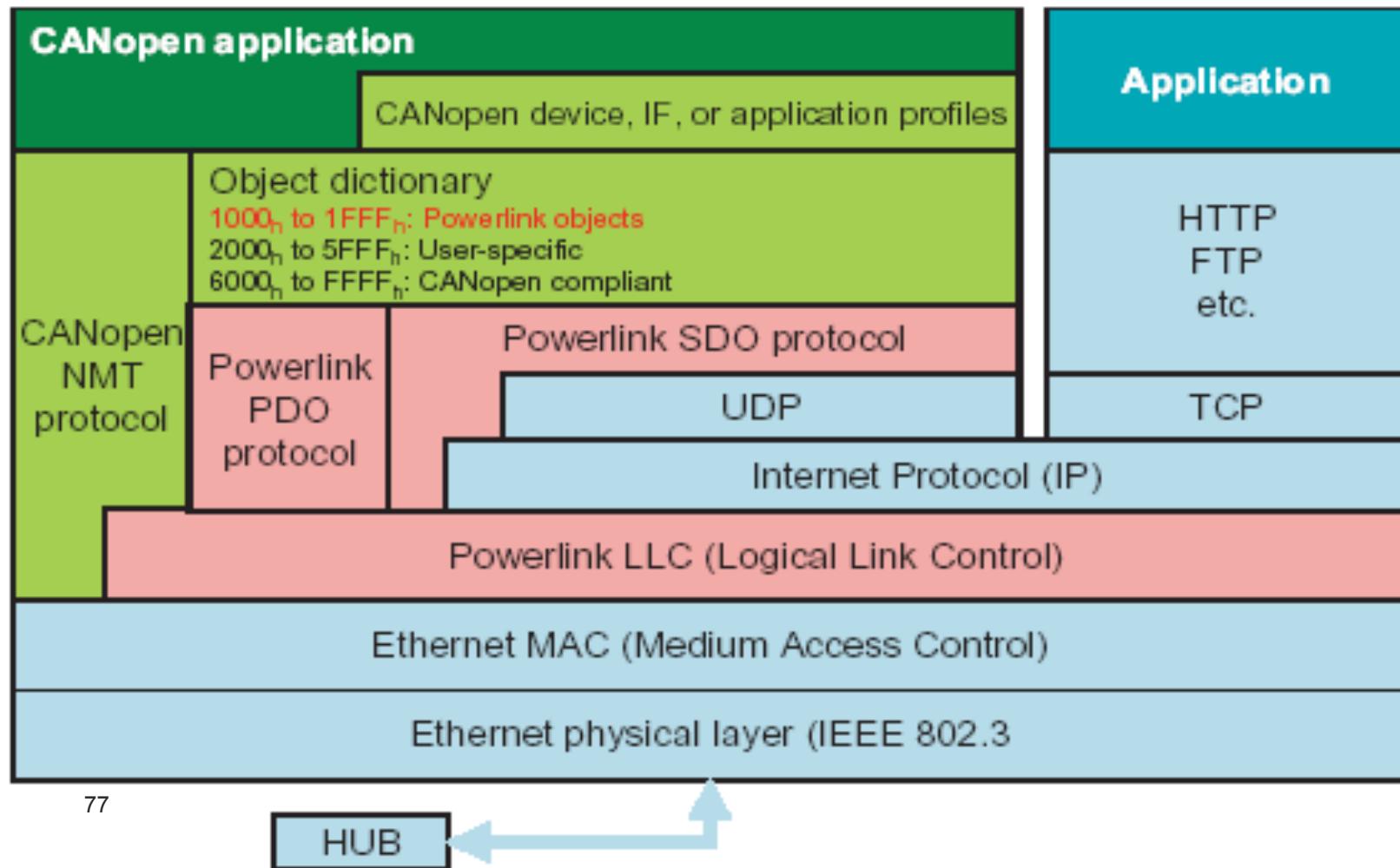


Tomado de “Documenting Software Architectures” pag 90



# Conceptos Básicos

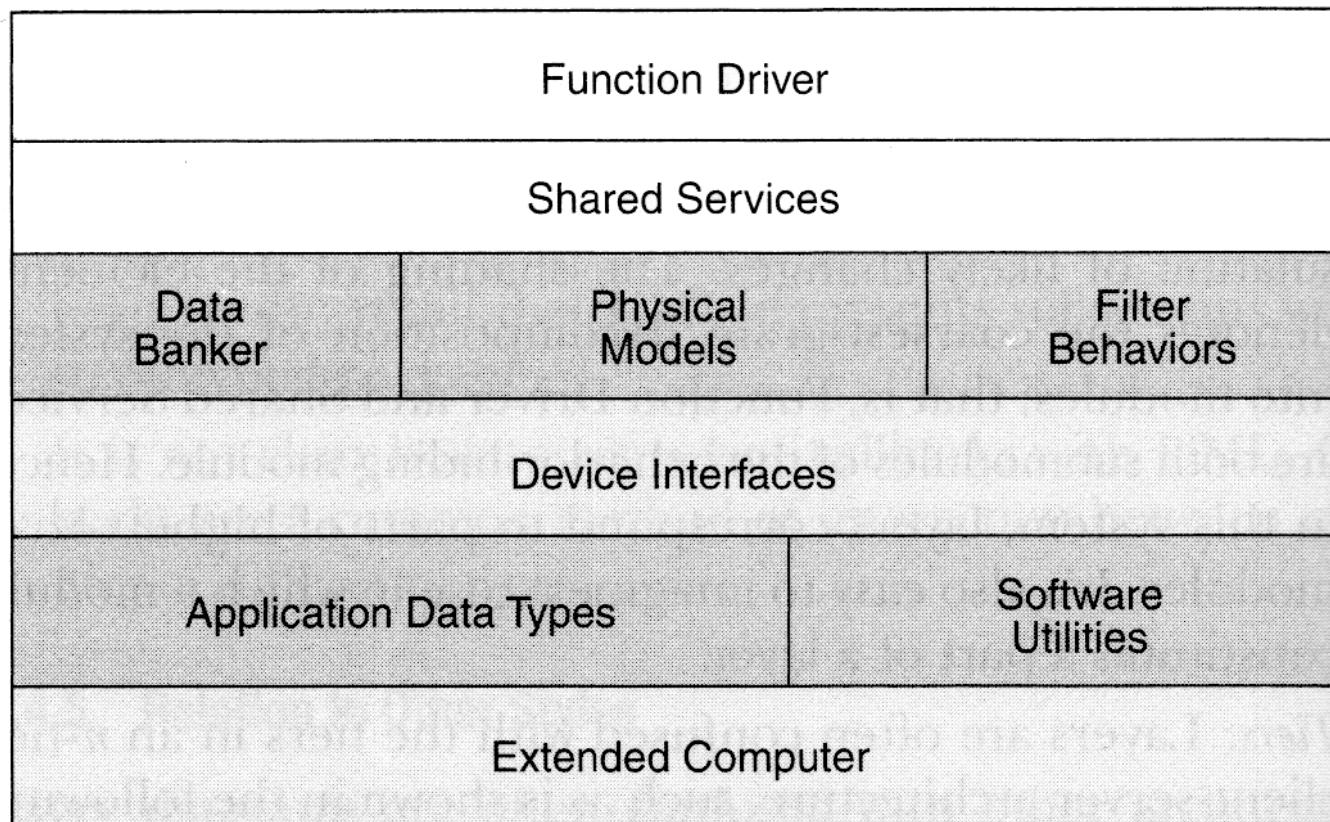
Ejemplo de notación informal para representar un estilo por capas



# Conceptos Básicos

Ejemplo de notación informal para representar un estilo por capas

**KEY**     Behavior-hiding module     Software decision-hiding module     Hardware-hiding module



Tomado de "Documenting Software Architectures" pag 90

# Referencias

- [1] Rozanski, N., Woods,E., “Software Systems Architecture”, Addison Wesley. 2005
- [2] CLEMENTS, P., KAZMAN, R., “Software Architecture in Practice”, Addison-Wesley, Second Edition, 2006.
- [3] Paul Clements et al, “Documenting Software Architectures: Views and Beyond”, Addison Wesley, 2002.
- [4] Paul Clements et al, “Evaluating Software Architectures”, Addison Wesley, 2002.
- [5] Erl, T., “SOA Principles of Service Design”, Prentice Hall, 2008
- [6] Geary, D., Horstmann, C., “Core JavaServer Faces” Second Edition, Prentice Hall, 2007[7] Richard Taylor, Nenad Medvidovic, Eric Dashofy. “Software Architecture Foundations, Theory and Practice, 2009.