Fase di Progettazione: Architetture Software

Roberta Gori, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

Progettare prima di produrre

- "The architect's two most important tools are: the eraser in the drafting room and the wrecking bar on the site" [Frank Lloyd Wright]
  - Anche nel software: nonostante l'apparente omogeneità dei materiali tra progetto e realizzazione: il codice è più "duro" dei modelli.
- Tipico della produzione industriale
- Alcune motivazioni
  - Complessità del prodotto
  - Organizzazione e riduzione delle responsabilità
  - Controllo preventivo della qualità

La progettazione

- Costituisce la fase ponte fra la specifica e la codifica
- La fase in cui si decide come passare da “che cosa” deve essere fatto a “come” deve essere fatto
- Il suo prodotto si chiama architettura (o progetto) del sw

Progettazione di alto livello e di dettaglio

- Progettazione di alto livello (o architetturale)
  - Scopo la scomposizione di un sistema in sottosistemi:
    - identificazione e specifica delle parti del sistema e delle loro inter-connessioni
- Progettazione di dettaglio
  - decisione su come la specifica di ogni parte sarà realizzata

Architettura software, def.

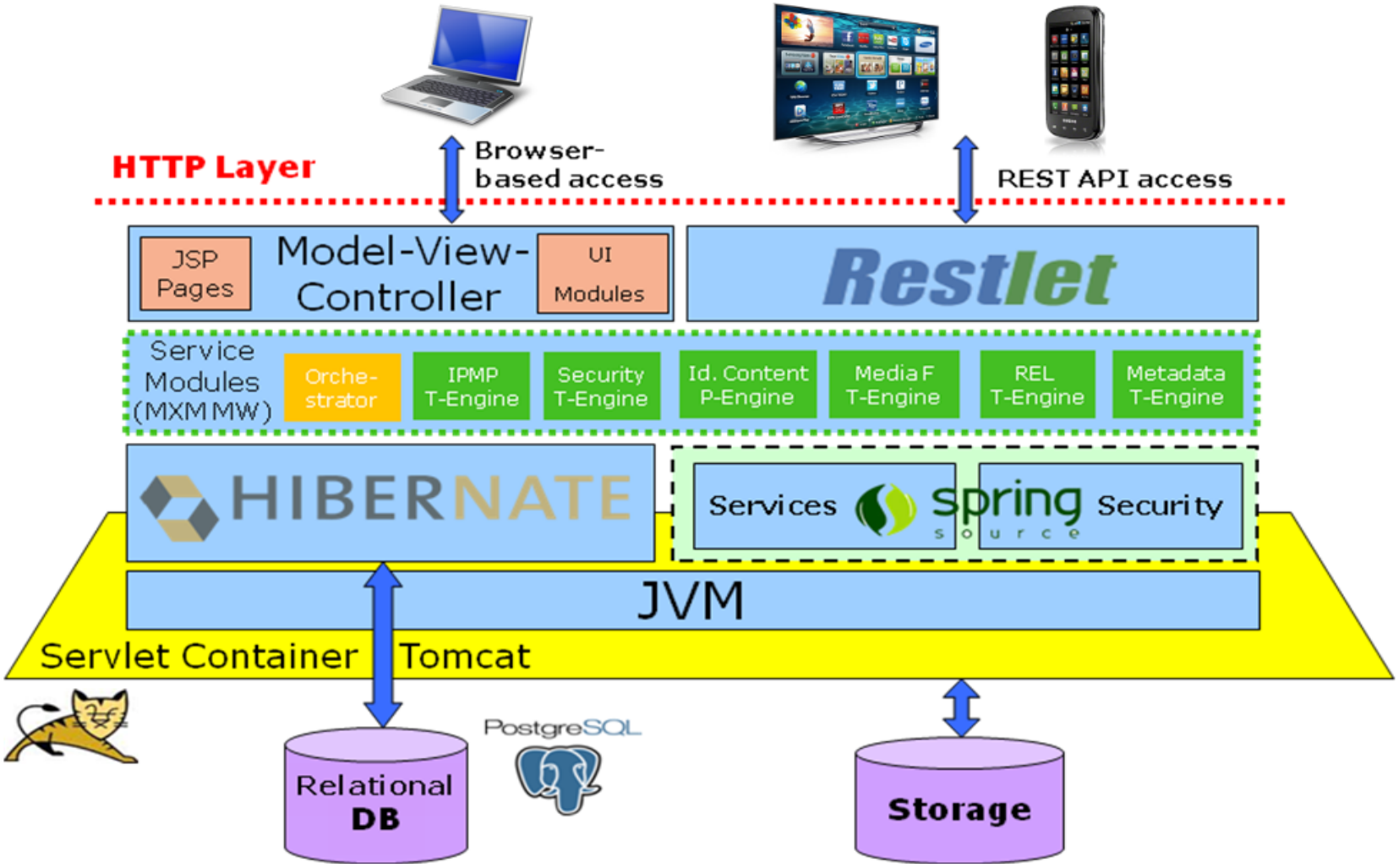
- L'architettura di un sistema software (in breve architettura software) è
  - la struttura del sistema,
    - costituita
      - dalle parti del sistema,
      - dalle relazioni tra le parti e
      - dalle loro proprietà visibili.

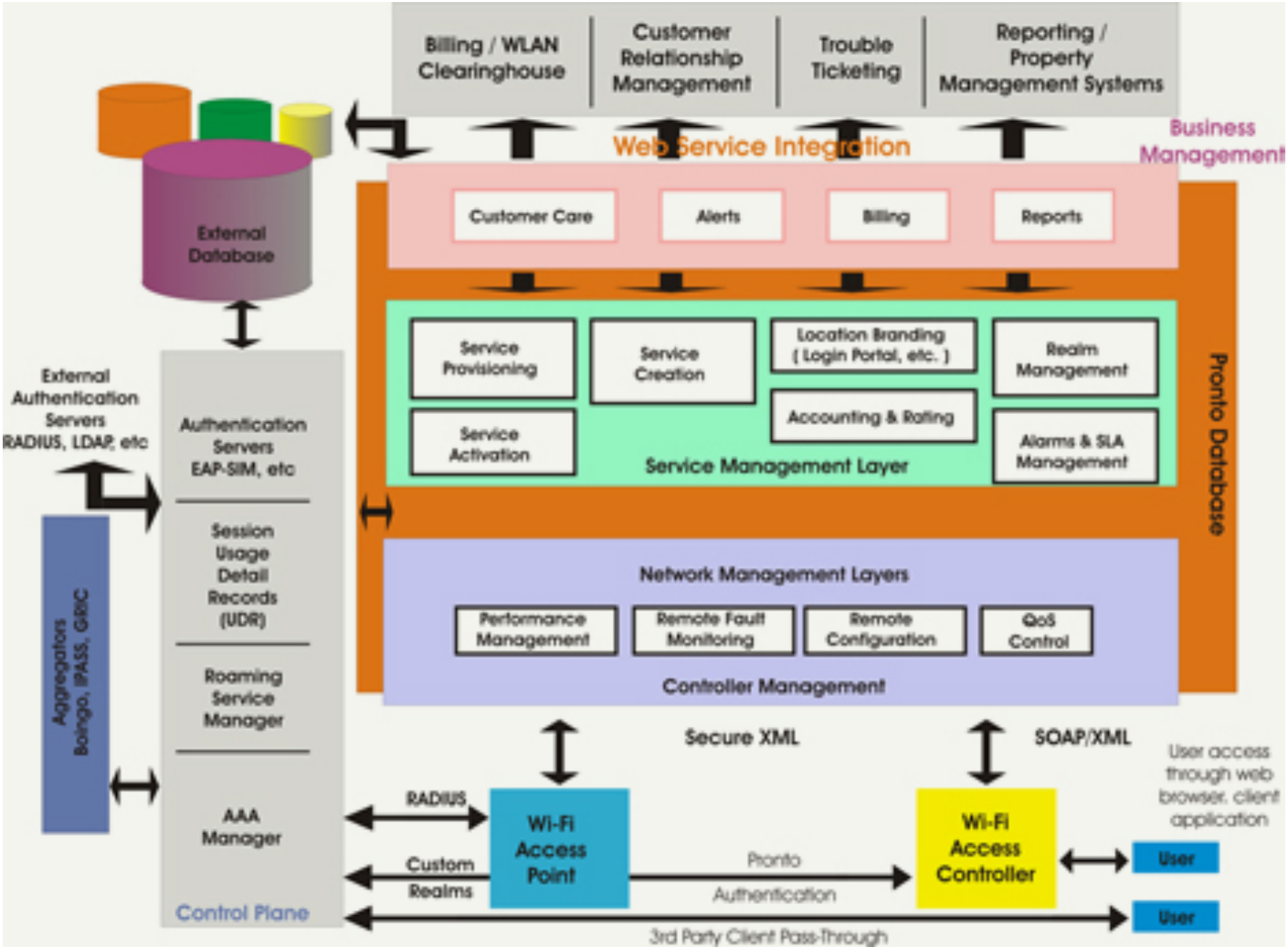
in altre/altrui parole

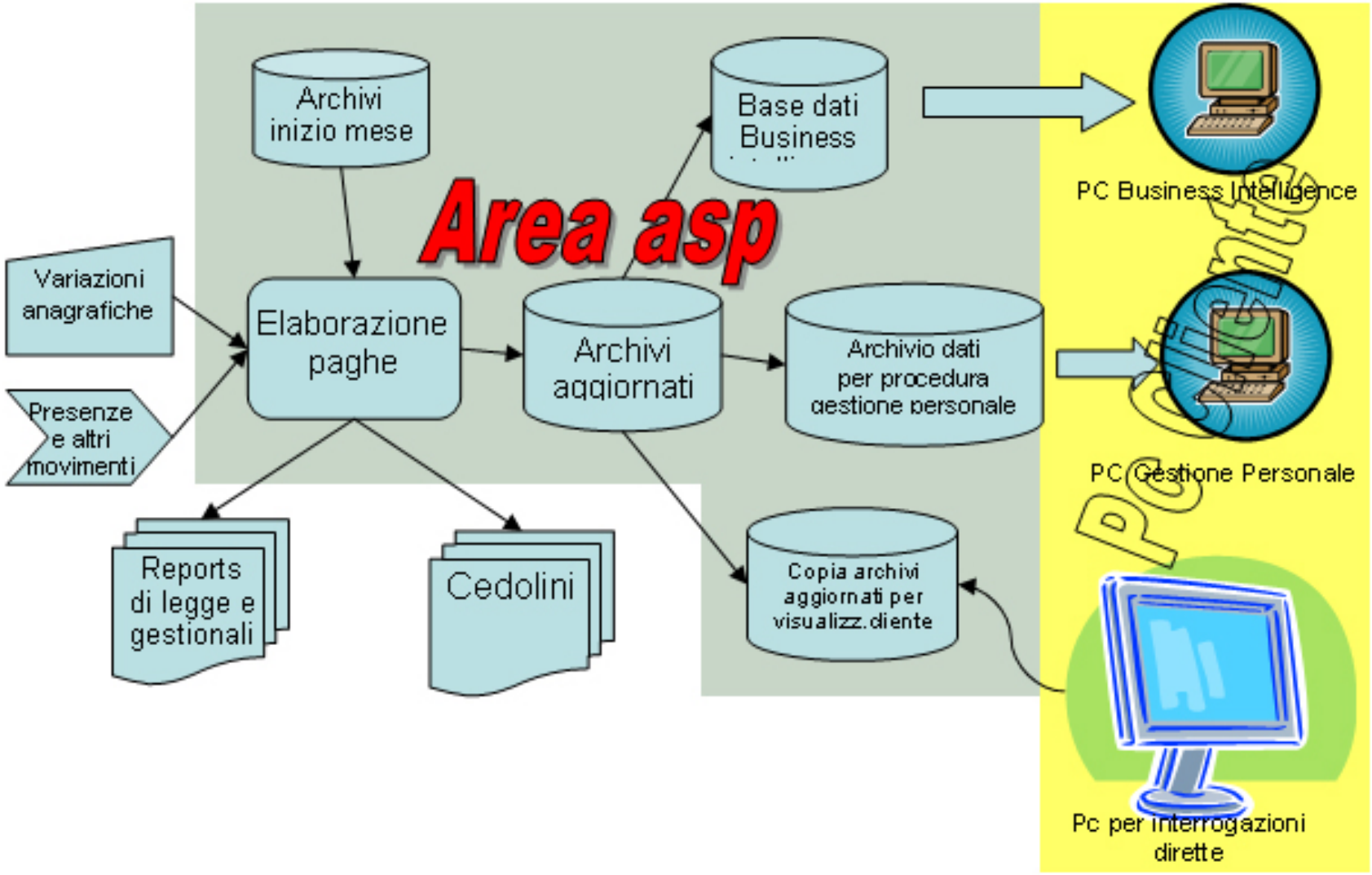
- L'architettura
  - definisce la struttura del sistema sw
  - specifica le comunicazioni tra componenti
  - considera aspetti non funzionali
  - è un'astrazione
  - è un artefatto complesso

    → problema di rapresentazione

Esempio di "descrizione" di una architettura...




Billing / WLAN Clearinghouse
Customer Relationship Management
Trouble Ticketing
Reporting / Property Management Systems
Business Management
Web Service Integration
External Database
Customer Care
Alerts
Billing
Reports
Service Provisioning
Service Creation
Location Branding (Login Portal, etc.)
Realm Management
Accounting & Rating
Service Activation
Alarms & SLA Management
Service Management Layer
Pronto Database
External Authentication Servers RADIUS, LDAP, etc
Authentication Servers EAP-SIM, etc
Session Usage Detail Records (UDR)
Roaming Service Manager
AAA Manager
Aggregators Boingo, IPASS, GRIC
Control Plane
Network Management Layers
Performance Management
Remote Fault Monitoring
Remote Configuration
QoS Control
Controller Management
Secure XML
SOAP/XML
User access through web browser, client application
RADIUS
Custom Realms
Wi-Fi Access Point
Pronto Authentication
Wi-Fi Access Controller
User
User
3rd Party Client Pass-Through

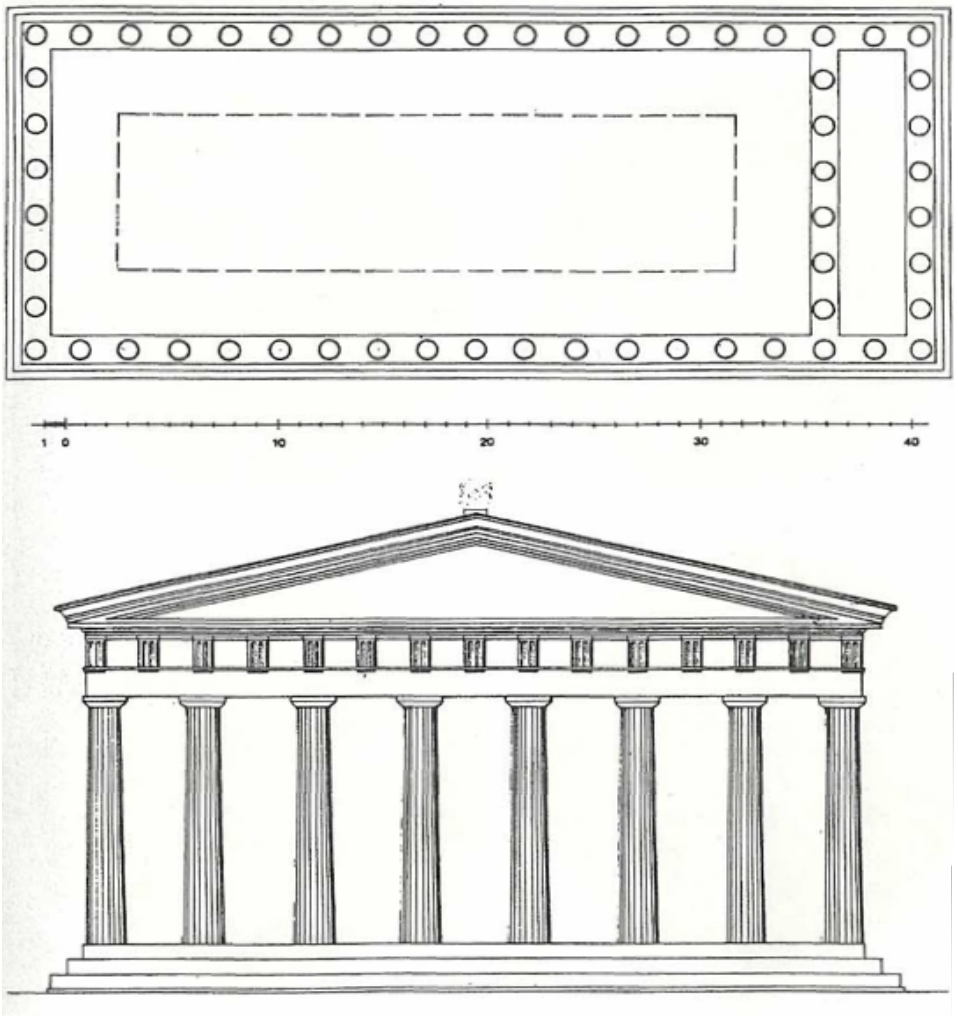

Area asp
Archivi inizio mese
Base dati Business
Variazioni anagrafiche
Elaborazione paghe
Archivi aggiornati
Archivio dati per procedura gestione personale
Presenze e altri movimenti
Reports di legge e gestionali
Cedolini
Copia archivi aggiornati per visualizz.diente
PC Business Intelligence
PC Gestione Personale
Pc per interrogazioni dirette

Per orientarci: una possibile analogia con l'ingegneria edile

- Il Disegno di Progetto permette di rappresentare agli altri l'architettura immaginata dal progettista:

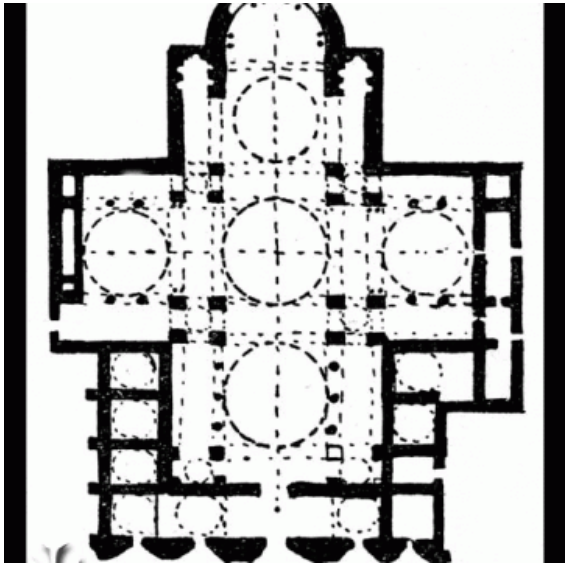
- si realizza graficamente tramite piante, prospetti e sezioni

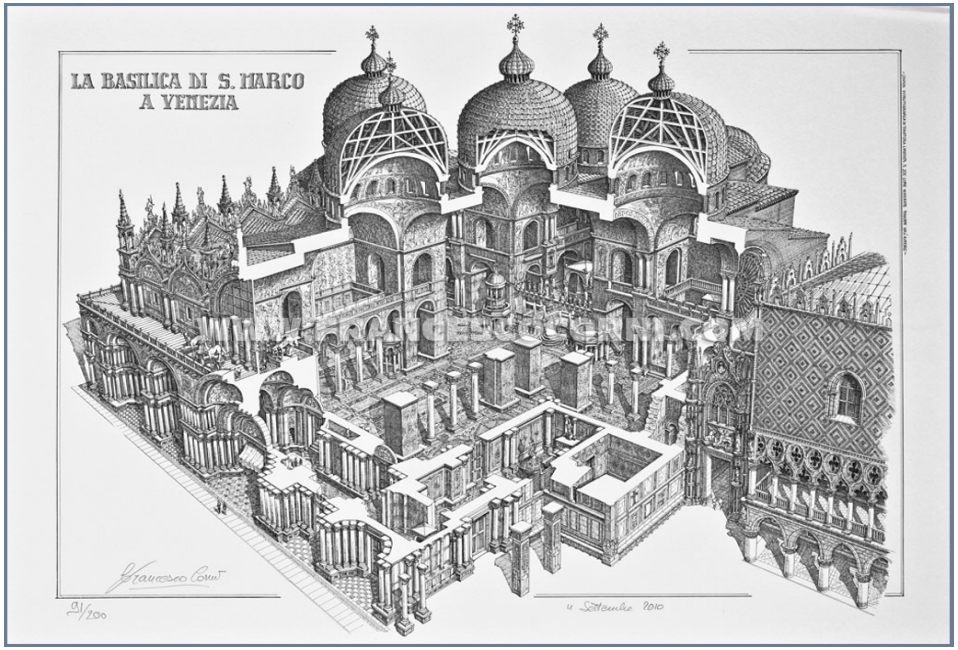
Una descrizione chiara a tutti



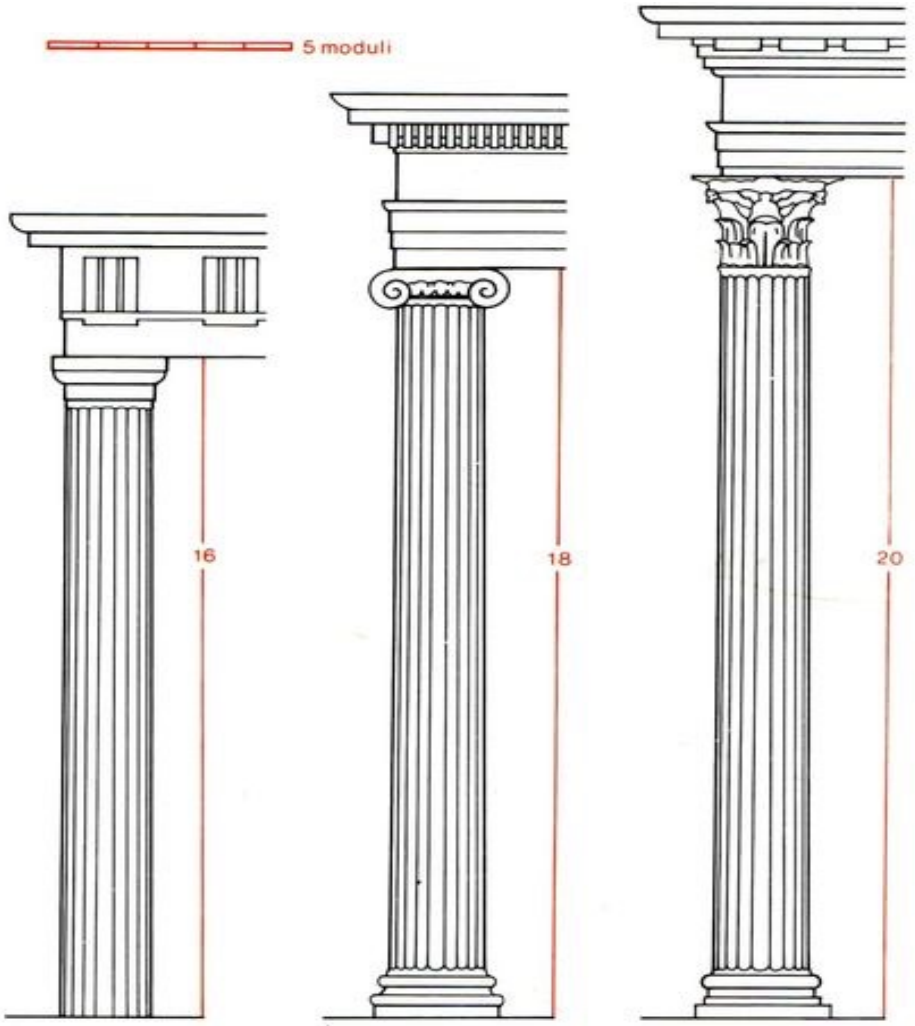
- Pianta
  - vista dall'alto
  - proiezione sul piano xy
- Prospetto
  - vista frontale
  - proiezione sul piano xz (o yz)

Proiezioni (prospetto e pianta) e spaccati: un monumento, diverse viste





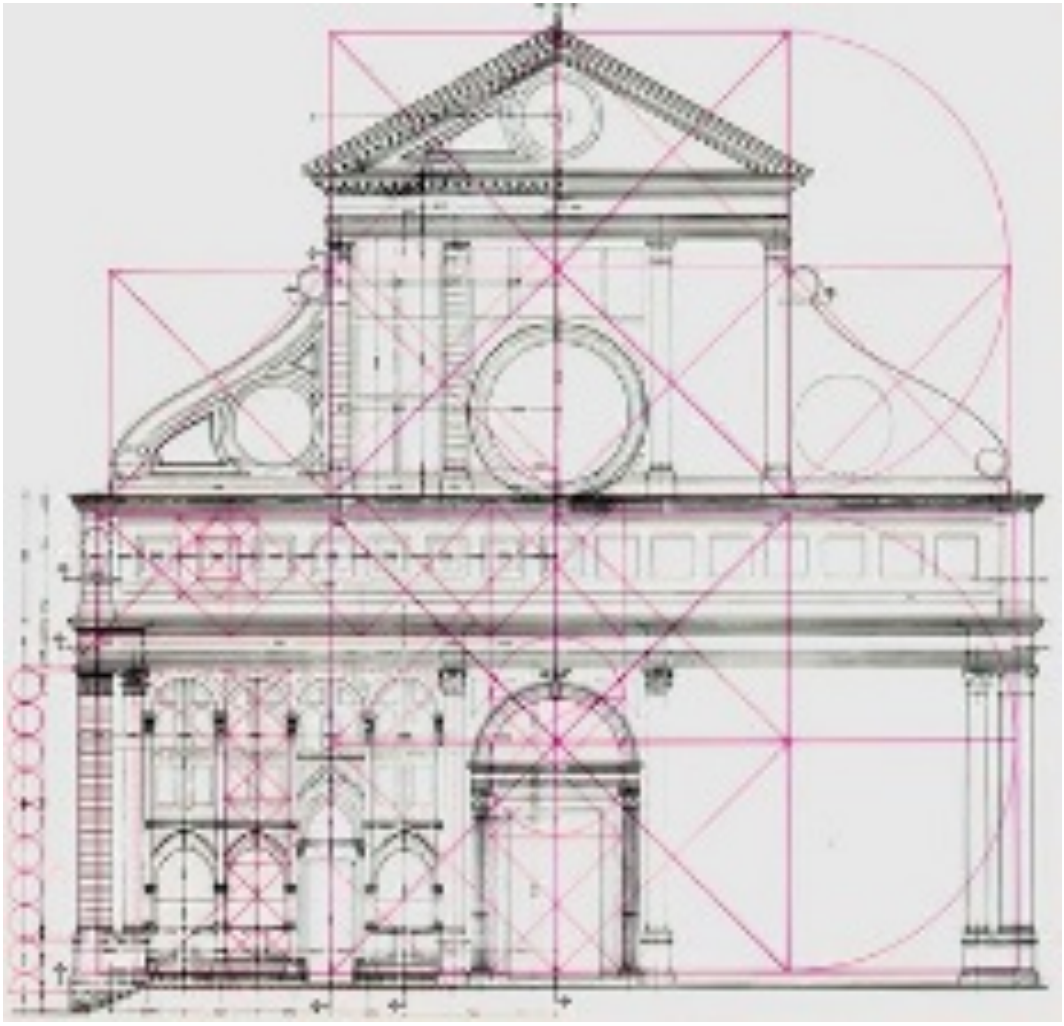
Ancora analogia: stili diversi



nella costruzione di una colonna posso seguire uno degli stili noti

Nel caso dell'architettura il criterio di scelta e' guidato principalmente dall'estetica

Pattern da applicare: schemi di progettazione (e.g. schemi proporzionali [Vitruvio, Alberti])



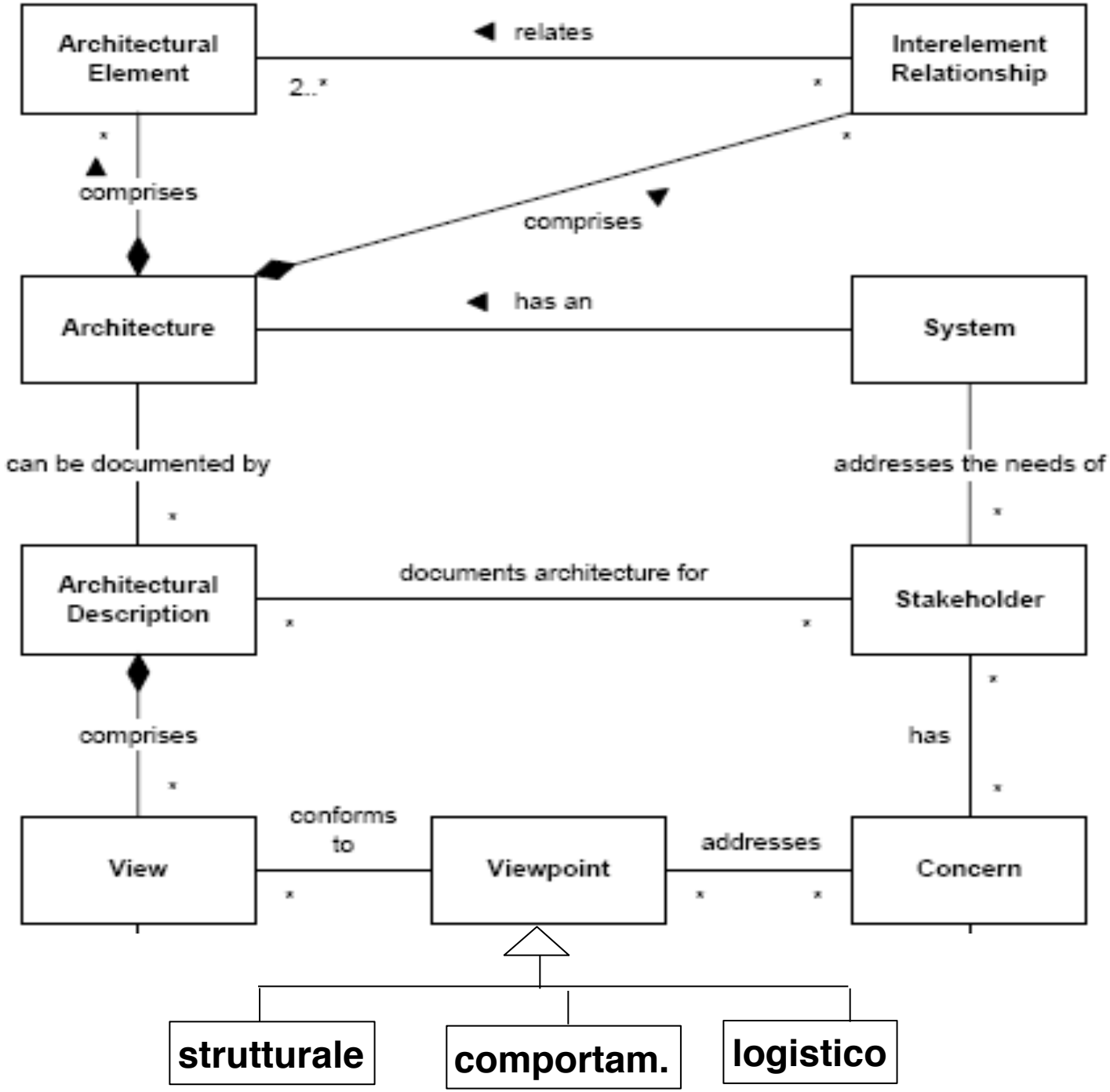
S. Maria Novella
L.B. Alberti

Torniamo al nostro caso: le viste

- Una vista su una architettura software è una sua astrazione secondo un aspetto di interesse
- Tre punti di vista simultanei sul sistema
  - vista comportamentale (component-and-connector (C&C))
    - la struttura come insieme di unità con comportamenti e interazioni, a tempo d'esecuzione
  - vista strutturale
    - la struttura come insieme di unità di realizzazione (codice)
  - vista logistica (problemi di allocazione)
    - le relazioni con strutture diverse dal software nel contesto del sistema

Utilità delle diverse viste sull'AS (prox. lezione)

- Comportamentali
  - Permette di valutare molti aspetti di qualità (scalabilità, efficienza, …)
- Strutturali
  - Progettare test di unità e di integrazione
  - Per esempio una vista strutturale a livelli permette di valutare la portabilità
- Logistiche
  - Per esempio una vista logistica di "deployment" permette di valutare prestazioni e affidabilità


Architectural Element
relates
Interelement Relationship
2..*
*
*
comprises
*
comprises
Architecture
has an
System
can be documented by
*
addresses the needs of
*
Architectural Description
documents architecture for
Stakeholder
*
*
*
comprises
*
has
*
View
conforms to
Viewpoint
addresses
Concern
*
*
*
*
strutturale
comportam.
logistico

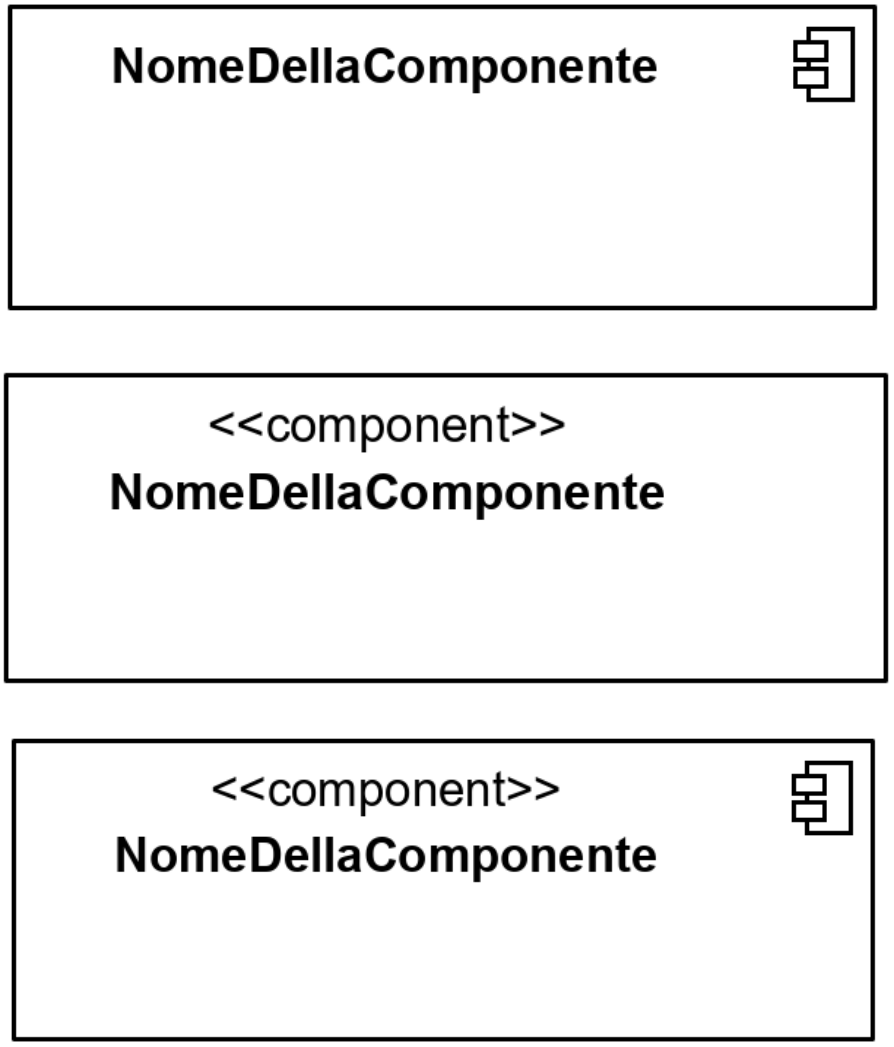
Viste di tipo comportamentale

Aka C&C aka componenti e connettori

Entità : i componenti

- Un componente software
  - unita' concettuali di decomposizione di un sistema a tempo d'esecuzione,
    - Per esempio: processi, oggetti, serventi, depositi di dati, …
  - incapsula un insieme di funzionalità e/o di dati di un sistema
  - restringe l'accesso a quell'insieme di funzionalità e/o dati tramite delle interfacce definite
  - ha un proprio contesto di esecuzione
  - può essere distribuito e installato in modo (possibilmente) indipendente da altri componenti

Componenti: notazione UML



- Un componente è un classificatore
- Si rappresenta in uno di questi modi
  - Tutti equivalenti
  - il terzo modo è ridondante, ma è quello usato da Visual Paradigm

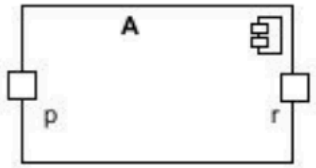
I connettori

- I connettori sono canali di interazione tra componenti che collegano i porti (prox lucido)
  - protocolli, flussi d'informazione, accessi ai depositi, …

Porti

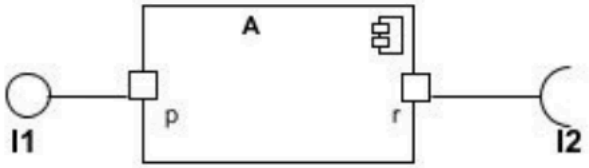
- I porti identificano i punti di interazione di un componente
  - un componente può avere più porti, uno per ogni tipo di connessione con altri componenti
  - un porto fornisce o richiede una o più interfacce (omogenee)
  - un porto è rappresentato con un «quadratino»
  - un porto può o avere associata una molteplicità con l'usuale sintassi (1..n)

Porti e interfacce: notazioni diverse

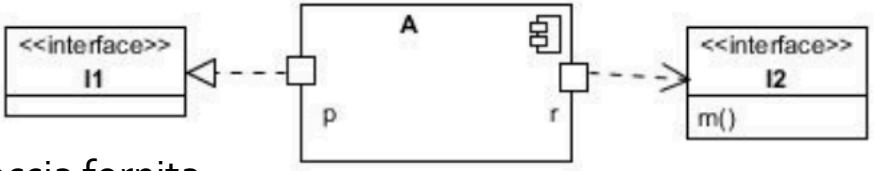


porti senza specifica di interfacce

porto con interfaccia fornita
(in forma sintetica con lollipop)



porto con interfaccia richiesta
(in forma sintetica con forchetta)

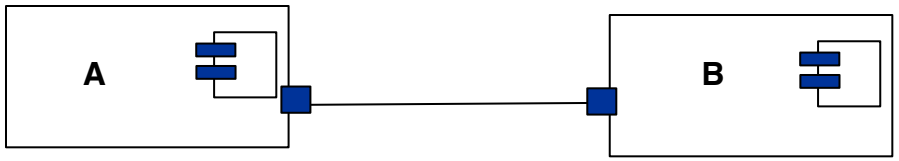


porto con interfaccia fornita
in forma estesa

porto con interfaccia richiesta
in forma estesa

Connettori graficamente

- I connettori collegano i porti



- Aggiungeremo poi informazione sullo stile della connessione

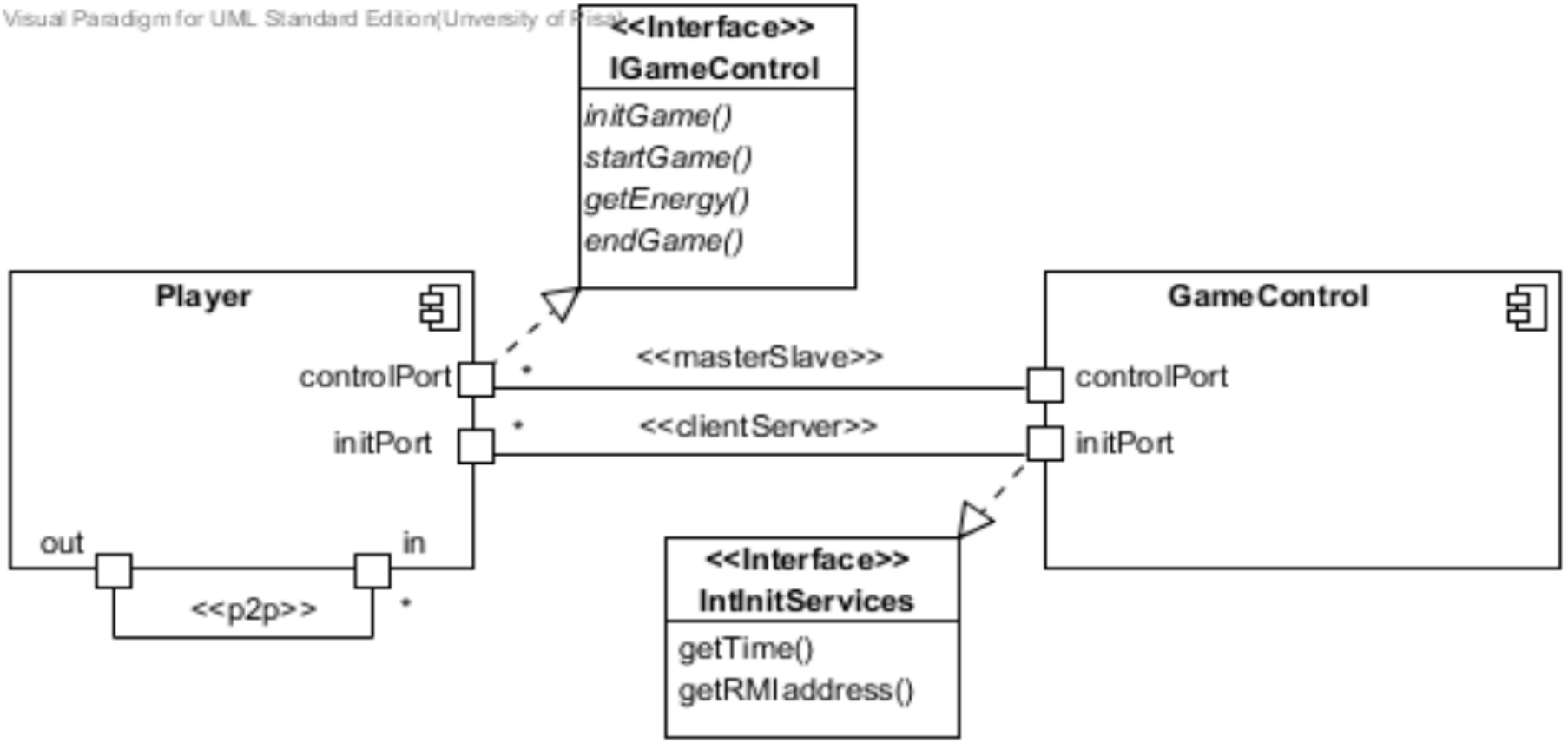
Componenti e Connettori

- Un sistema software è una composizione di componenti software
  - la composizione è basata sulla "connessione" di più componenti
  - realizzata sulla base delle interfacce dei componenti e
  - mediante l'ausilio di connettori

Viste di tipo comportamentale (C&C)

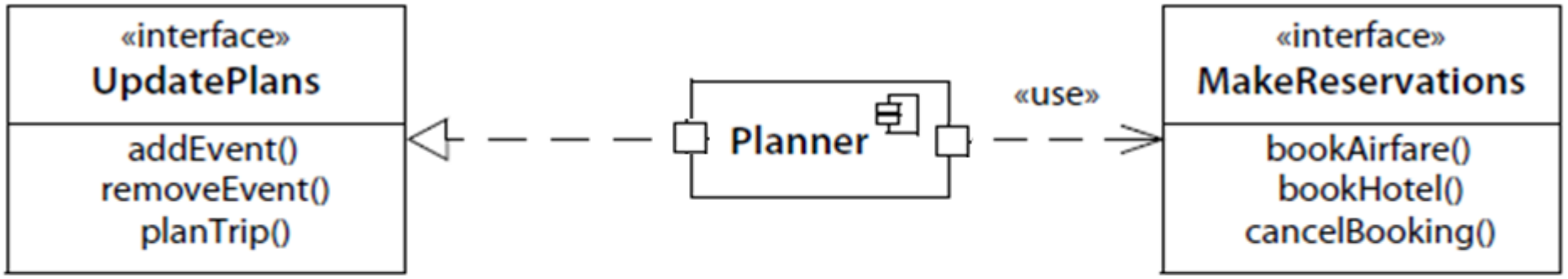
- La vista C&C descrive un sistema software come composizione di componenti software
  - specifica i componenti con le loro interfacce
  - descrive le caratteristiche dei connettori
  - descrivere la struttura del sistema in esecuzione
    - flusso dei dati, dinamica, parallelismo, replicazioni, …
- Utile per:
  - analisi delle caratteristiche di qualità a tempo d'esecuzione
    - prestazioni, affidabilità, disponibilità, sicurezza, …

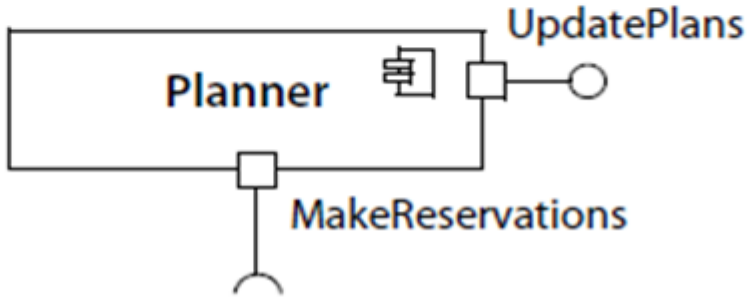
Un esempio


Visual Paradigm for UML Standard Edition(University of Pisa)
<<Interface>>
IGameControl
initGame()
startGame()
getEnergy()
endGame()
Player
controlPort
initPort
out
in
<<p2p>>
<<masterSlave>>
<<clientServer>>
GameControl
controlPort
initPort
<<Interface>>
IntInitServices
getTime()
getRMIaddress()

Interfacce: descrizione estesa e descrizione sintetica

Le interfacce sono descritte in modo sintetico con lollipop e forchette, oppure in modo esteso, per mostrare le operazioni richieste/offerte.

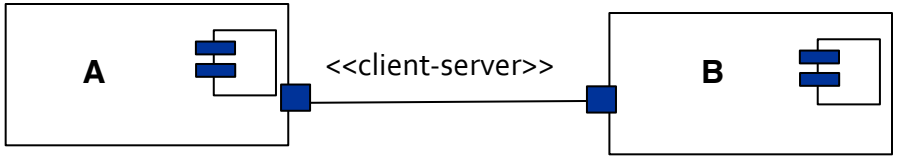




Conettori: notazione UML

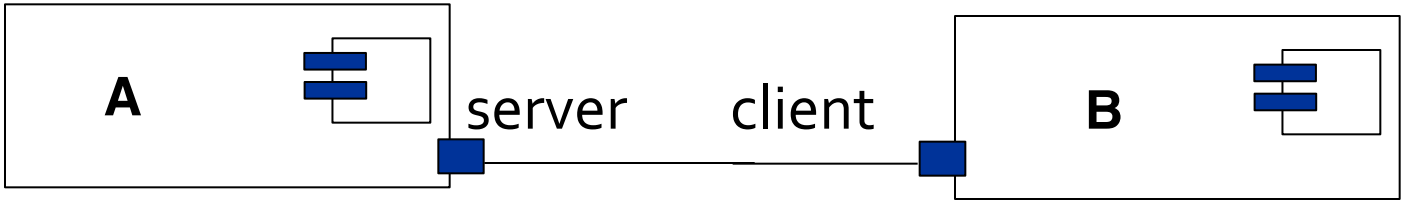
- Un connettore non ha un descrittore specifico, si usa il simbolo di associazione (linea). Collega tra loro due porti.

- Per documentare il protocollo di interazione si usa uno stereotipo:



  - <<clientServer>> ; <<dataAccess>> (specializzazione di clientServer) ; <<pipe>> ; <<peer2peer>> (<<p2p>>); <<publish-subscribe>>

- Oppure si indicano i ruoli delle componenti:



Stili (o schemi) architetturali

- Uno stile è una proprietà di una architettura
- Uno stile caratterizza una famiglia di architetture con caratteristiche comuni
  - stile a macchine virtuali: i moduli definiscono macchine virtuali
  - stile client-server: caratterizzato da particolari interazioni tra componenti

Stili (o schemi) architetturali e vista C&C

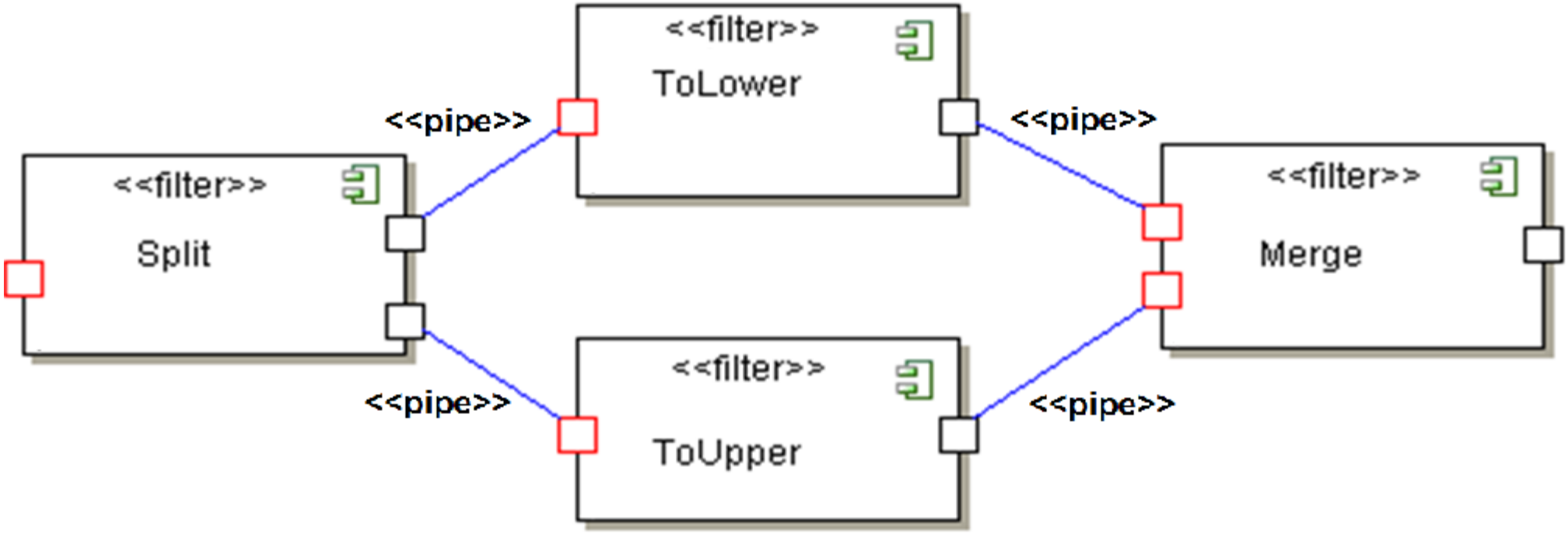
- Le funzionalità di componenti interagenti e le caratteristiche delle interazioni tra componenti spesso rispondono a stili (schemi) standard
- Nella vista C&C uno stile architetturale è caratterizzato da:
  - caratteristiche generali delle componenti in gioco
  - particolari interazioni tra le componenti,
    - e quindi dalle caratteristiche dei porti e dei connettori
- Vedremo gli stili di uso comune
  - Le loro caratteristiche
  - Un modo per documentarli

Stili comportamentali: Condotte e filtri (pipe & filters)

- Componenti
  - sono di tipo filtro: trasformano uno o più flussi di dati dai porti d'ingresso in uno o più flussi sui porti d'uscita
- Connettori
  - sono di tipo condotta (pipe): canale di comunicazione unidirezionale bufferizzato che preserva l'ordine dei dati dal ruolo d'ingresso a quello d'uscita
- Usi
  - pre-elaborazione in sistemi di elaborazione di segnali
  - analisi dei flussi dei dati, e.g. dimensioni dei buffer

Esempio: pipe and filter

Esempio: Pipe and filter



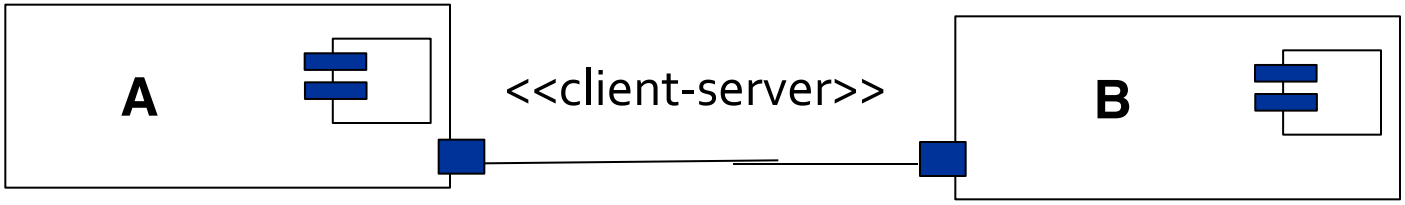
Per esempio «ciao» o «CIAO» o «Ciao» viene trasformato in «ciaO»

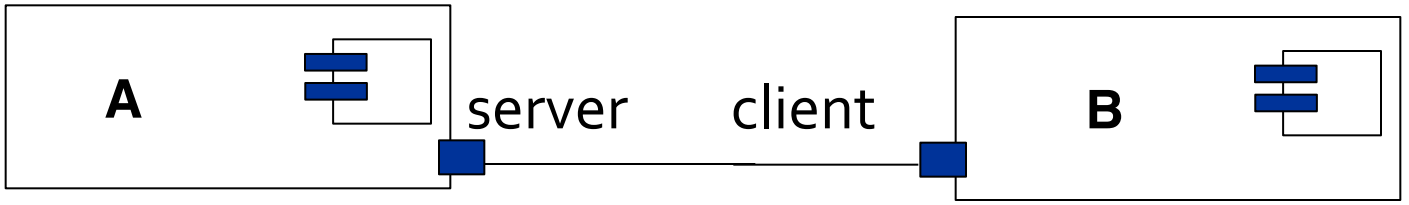
Stile client-server

- l sistema è formato da due componenti : il client e il server
  - spesso, ma non necessariamente, eseguiti su macchine diverse collegate in rete
- Il server svolge le operazioni necessarie per realizzare un servizio: ad esempio gestisce una banca dati, gestisce l'aggiornamento dei dati e la loro integrità,….
  - Aspetta le richieste dei client a un porto
- Il client invia al server le richieste ed attende una risposta

Stili comportamentali: Client-server

- Cliente-servente (Client-server)
  - le componenti clienti chiedendo servizi alle componenti serventi



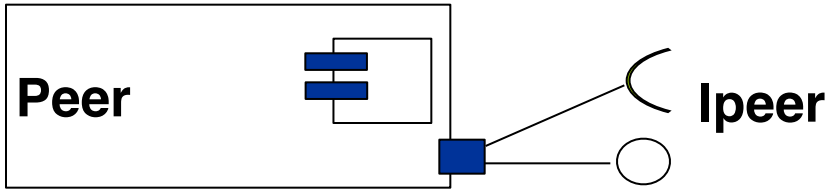


Stili comportamentali: master-slave

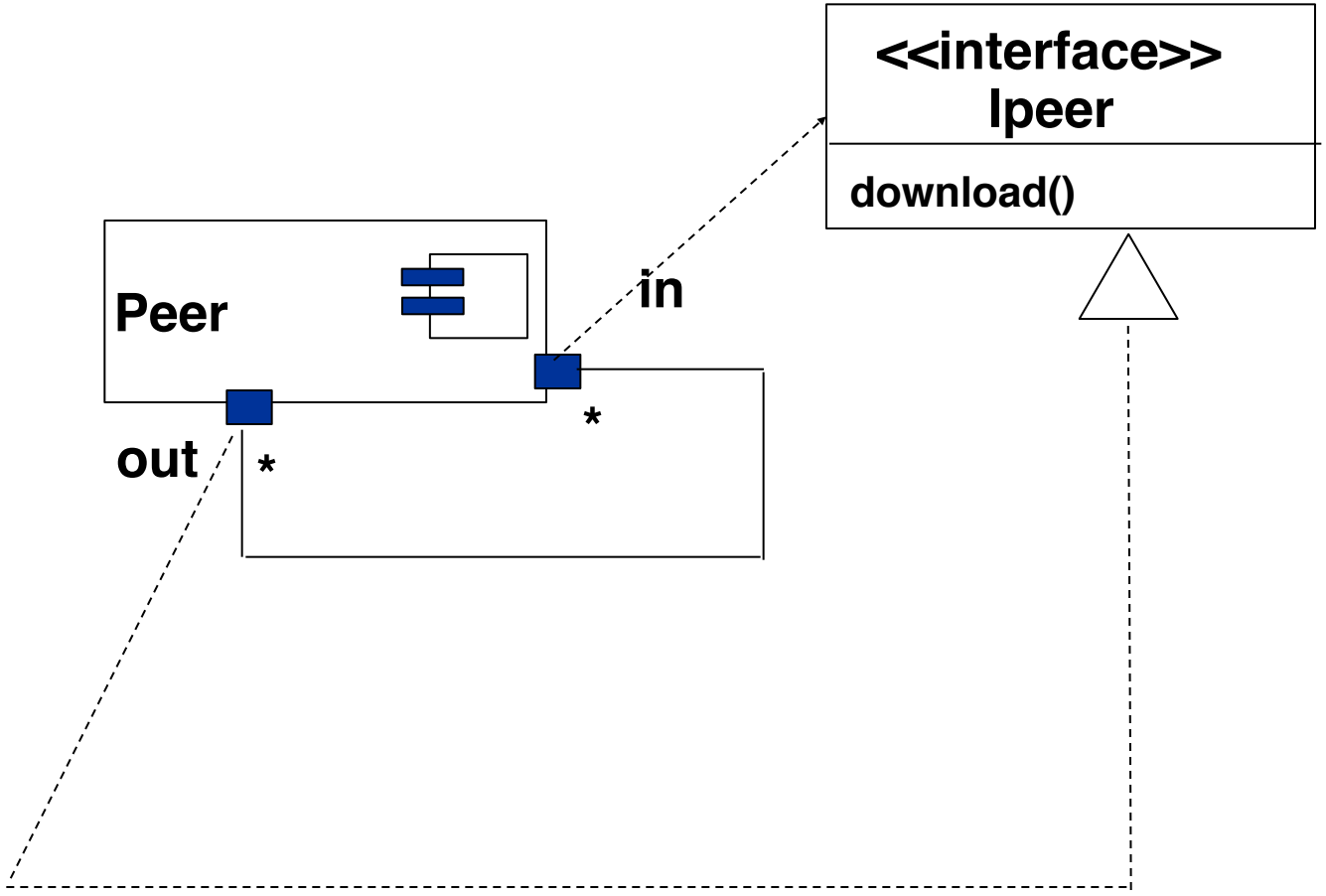
- Un caso particolare del client-server, ma risponde a esigenze diverse
- Il servente (slave) serve un solo cliente (master)
- Architettura usata per esempio nella replica di database, il database master è considerato come fonte autorevole e i database slave sono sincronizzati con esso.

Stili comportamentali: peer2peer, caso particolare di client-server

- Da pari a pari (peer to peer)
  - tutti i programmi agiscono sia da client sia da server. Es: i programmi di scambi audio e video (WinMx, Kazaa, eMule, ecc.)
  - Scambio di servizi alla pari



P2P



Stili comportamentali: Publish-subscribe

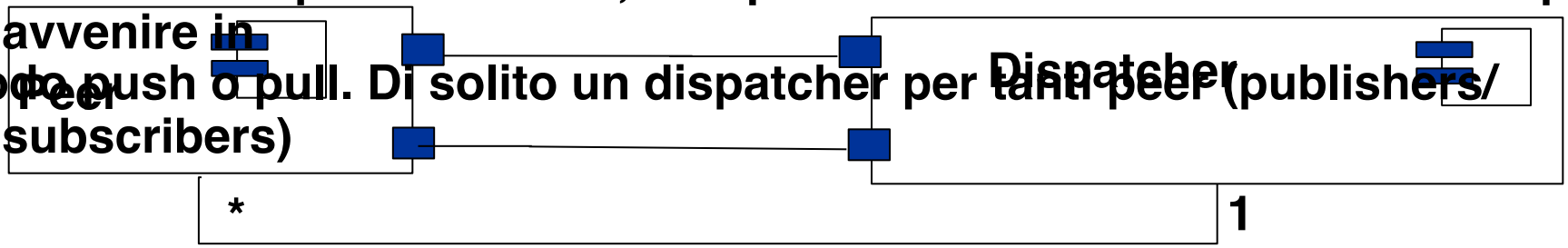
- Le componenti interagiscono annunciando eventi: ciascuna componente si "abbona" a classi di eventi rilevanti per il suo scopo
- Ciascuna componente, volendo, può essere sia produttore che consumatore di eventi
- Disaccoppia produttori e consumatori di eventi e favorisce le modifiche dinamiche del sistema

Publish-subscribe

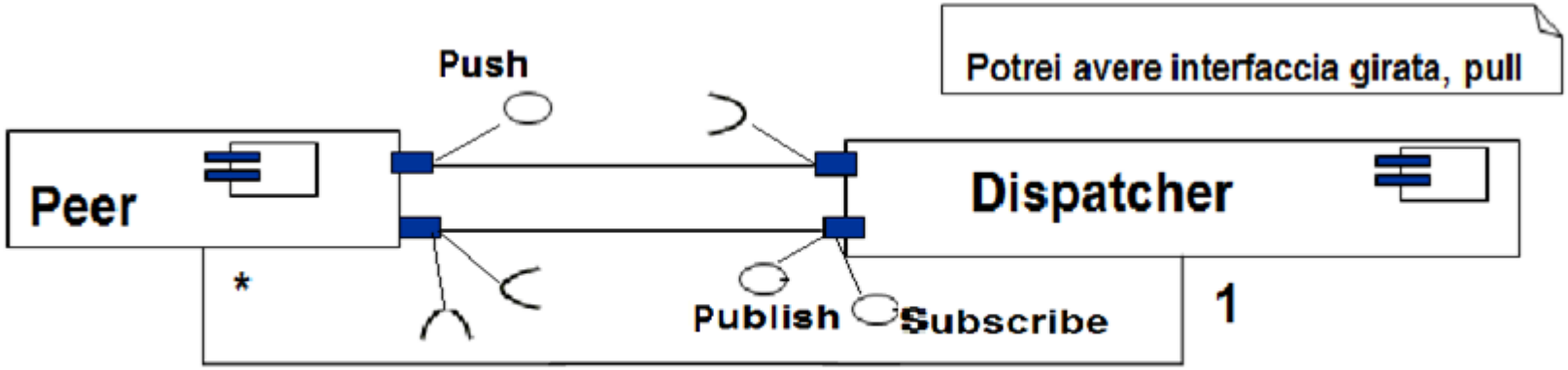
- In questo stile, mittenti e destinatari di messaggi dialogano attraverso un tramite, che può essere detto dispatcher o broker. Il mittente di un messaggio (detto publisher) non deve essere consapevole dell'identità dei destinatari (detti subscriber); esso si limita a "pubblicare" (in inglese to publish) il proprio messaggio al dispatcher. I destinatari si rivolgono a loro volta al dispatcher "abbonandosi" (in inglese to subscribe) alla ricezione di messaggi. Il dispatcher quindi inoltra ogni messaggio inviato da un publisher a tutti i subscriber interessati a quel messaggio.
- In genere, il meccanismo di sottoscrizione consente ai subscriber di precisare nel modo più specifico possibile a quali messaggi sono interessati. Per esempio, un subscriber potrebbe "abbonarsi" solo alla ricezione di messaggi da determinati publisher, oppure aventi certe caratteristiche.
- Questo schema implica che ai publisher non sia noto quanti e quali sono i subscriber e viceversa. Questo può contribuire alla scalabilità del sistema.

Publish-subscribe

Si possono immaginare due diversi connettori, uno per le richieste di sottoscrizione e
per richieste di pubblicazione, uno per diffondere i dati. La diffusione può avvenire in
modo push o pull. Di solito un dispatcher per tanti peer (publishers/ subscribers)



Operazioni: subscribe(), unsubscribe() offerte da interfaccia Subscribe,
publish() offerta da interfaccia Publish
notify() (offerta da Push), letMeKnow() (offerta da Pull – non in diagramma)



Publish-subscribe: middleware e reti

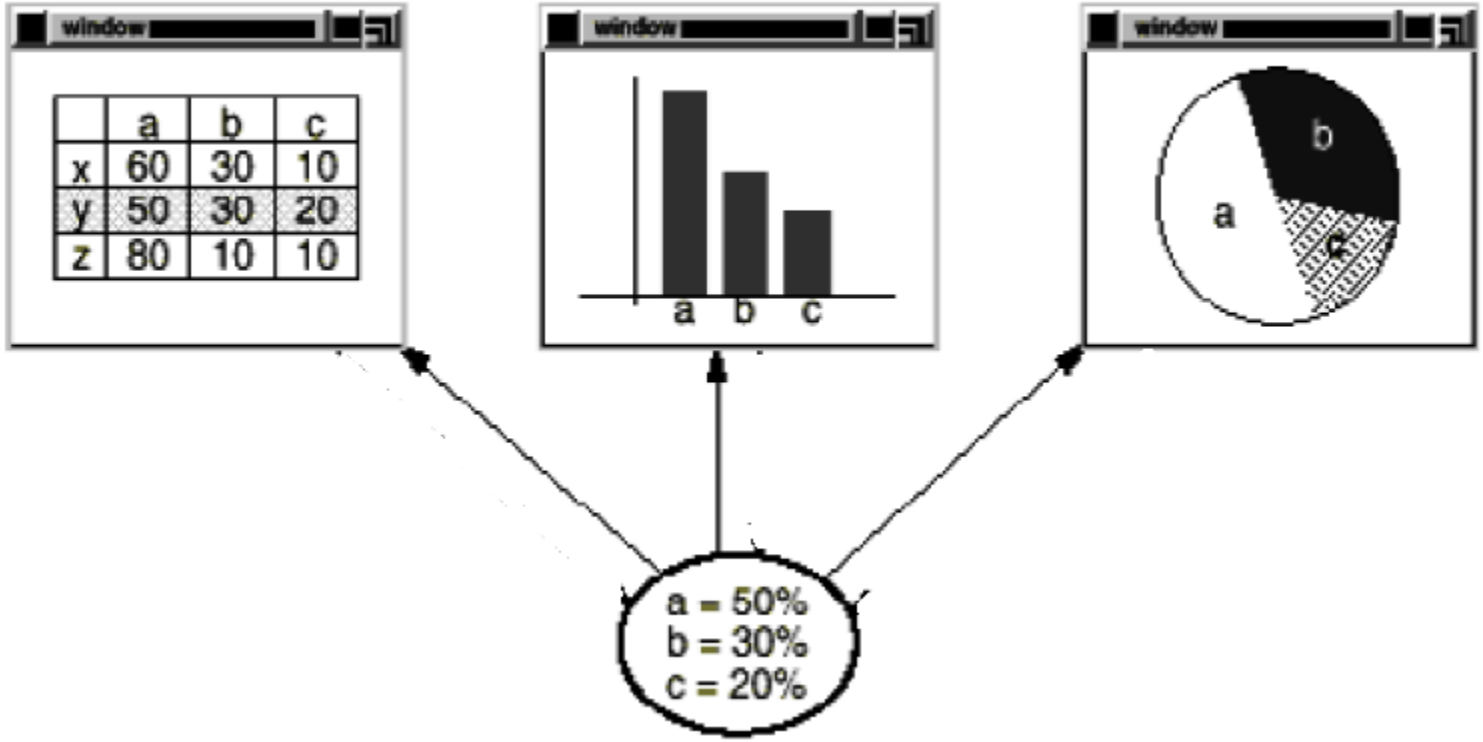
- Reti: multicast con algoritmi di flooding

- Il Data Distribution Service for Real Time Systems (DDS) è uno standard emanato dall'Object Management Group (OMG) che definisce un middleware per la distribuzione di dati in tempo reale secondo il paradigma publish/subscribe.

Stili comportamentali: Model–View–Controller (MVC)

- Stile in cui si isola la logica di business dal controllo sull'input e dalla presentazione (vista sui dati), consentendo sviluppo indipendente, test e manutenzione di ciascuno.
- Modello
  - È la rappresentazione del dominio dei dati su cui opera l'applicazione. Quando un modello cambia il suo stato, notifica le sue viste associate in modo che si possano aggiornare.
- Vista
  - Rende il modello in una forma adatta all'interazione, in genere un elemento dell'interfaccia utente. Ci possono essere più viste per un singolo modello, per scopi diversi.
- Controllore
  - Riceve l'input e effettua chiamate agli oggetti del modello.

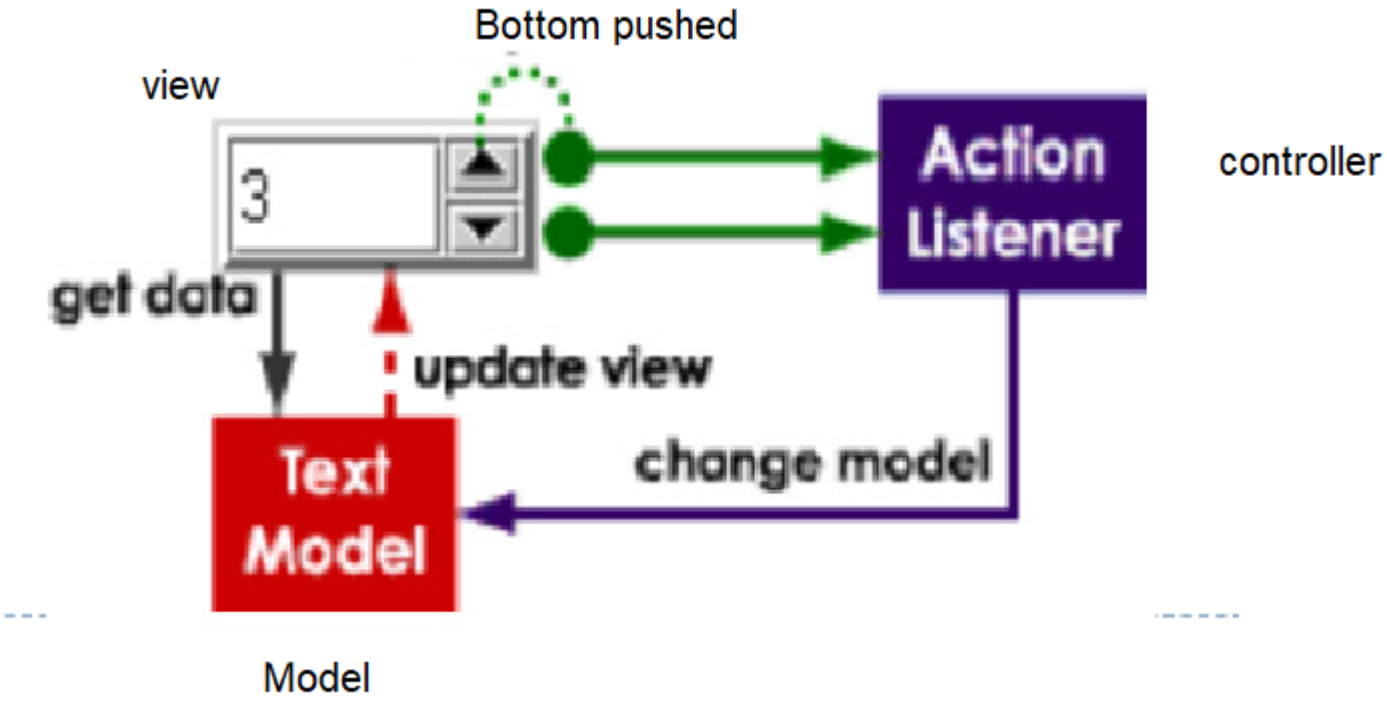
Un modello, diverse viste



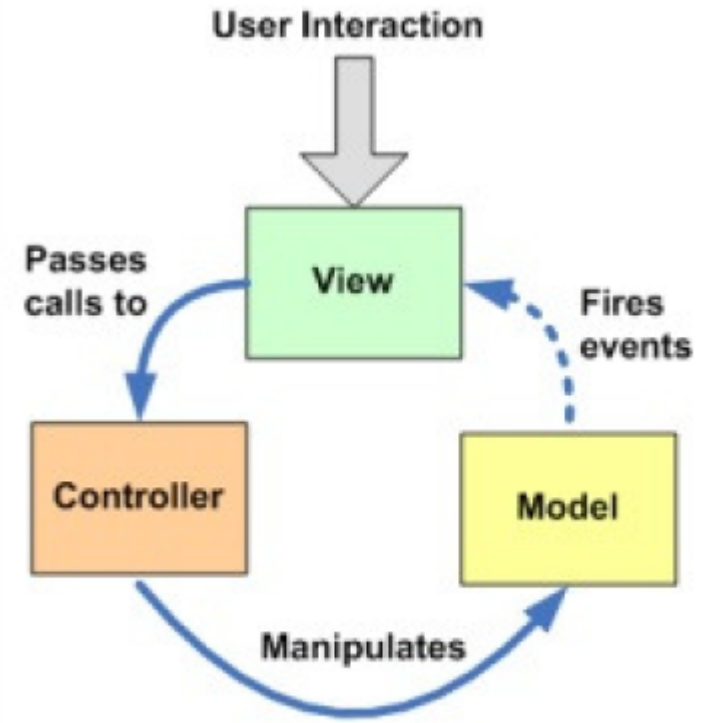
Model-View-controller

- L'utente interagisce con la vista
- Il controller riceve le azioni dell'utente e le interpreta
  - Se si fa clic su un pulsante, è compito del controllore capire che cosa significhi e come il modello dovrebbe essere manipolato in base a tale azione.
- Il controller chiede al modello di cambiare il suo stato
- Il modello notifica la vista quando il suo stato è cambiato
  - Quando qualcosa cambia nel modello, in risposta a qualche azione (es. fare clic su un pulsante) o per altri motivi (es. è iniziato il brano successivo nella playlist), i modello notifica alla vista che il suo stato è cambiato.
- La vista chiede lo stato al modello
  - Per esempio, se il modello notifica la vista che è iniziata un nuovo brano, la vista richiede il nome del brano al modella e lo mostra.

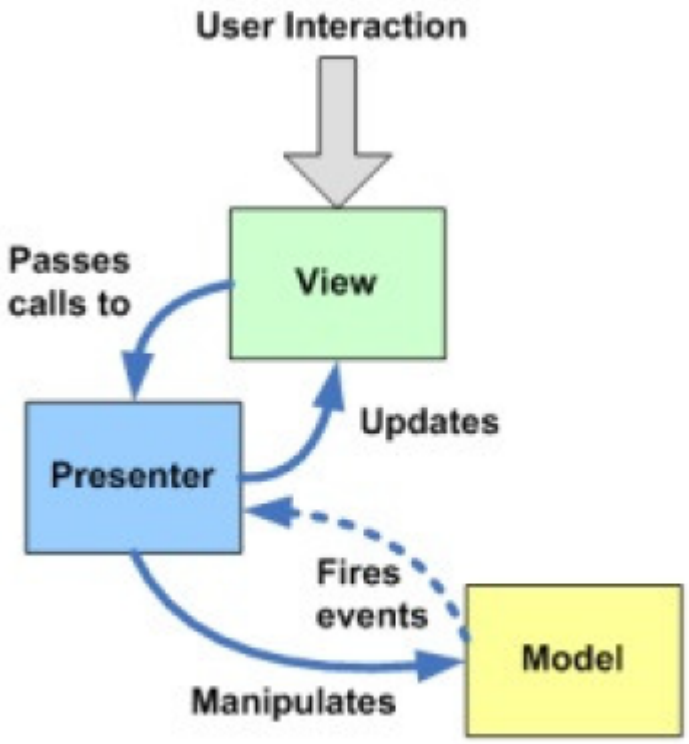
Esempio di MVC



MVC vs Model-view-presenter



Model-View-Controller



Model-View-Presenter

Stili Comportamentali: Coordinatore di processi

- Il Coordinatore di processi conosce la sequenza di passi necessari per realizzare un processo.
- Riceve la richiesta, chiama i servers secondo l'ordine prefissato, fornisce una risposta
- Normalmente usato per realizzare processi complessi
- Disaccoppiamento: i server non conoscono il loro ruolo nel processo complessivo né l'ordine dei passi del processo. Ogni server semplicemente definisce un servizio
- Comunicazione flessibile: sincrona o asincrona.

Esercizio

- Fornire rappresentazione UML per gli stili MVC, MVPresenter e Coordinatore di processi

Arrivata fino a qui ven 17

Viste di tipo strutturale

4 viste di tipo strutturale

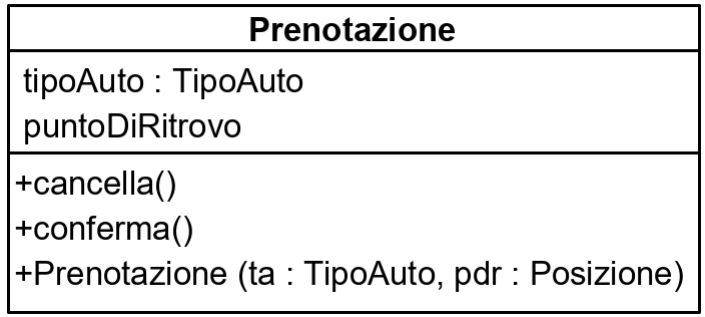
- Elementi
  - **Modulo**: unità di software che realizza un insieme coerente di responsabilità
    - Esempio: classe, collezione di classi, un livello.
- Relazioni tra elementi
  - **parte di, eredita da, dipende da, può usare**
    - Le 4 viste strutturali sono caratterizzate da questi 4 tipi di relazione

Viste di tipo strutturale (2)

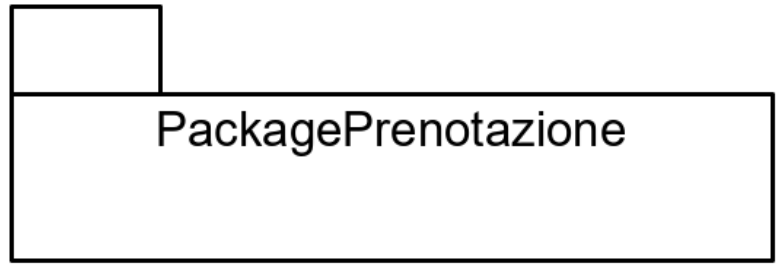
- A cosa servono
  - costruzione
    - la vista può fornire lo schema del codice e directory e file sorgente hanno una struttura corrispondente
  - analisi
    - tracciabilità dei requisiti
    - analisi d'impatto per valutare eventuali modifiche
  - comunicazione
    - se la vista è gerarchica, offre una presentazione top-down della suddivisione delle responsabilità nel sistema ai novizi
- Non utili per:
  - analisi dinamiche, fatte invece con viste comportamentali e logistiche

Notazione UML per documentare una vista strutturale

- classi
  - Rispetto alla descrizione del dominio, più spazio alla specifica delle operazioni



- packages



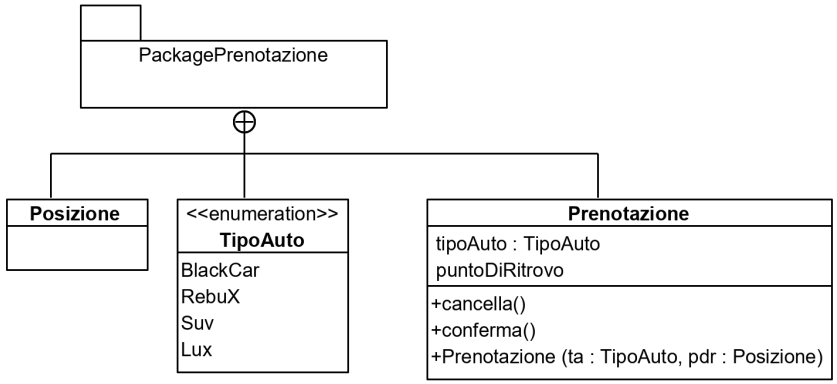
- relazioni tra classi, tra packag
  - contenimento, dipendenze, generalizzazione….

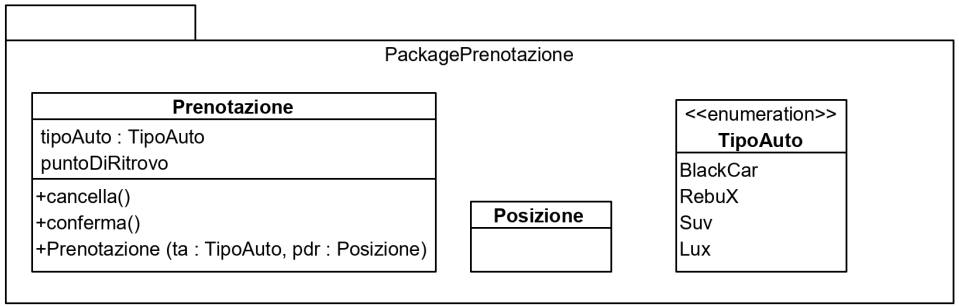
Prima vista strutturale: decomposizione

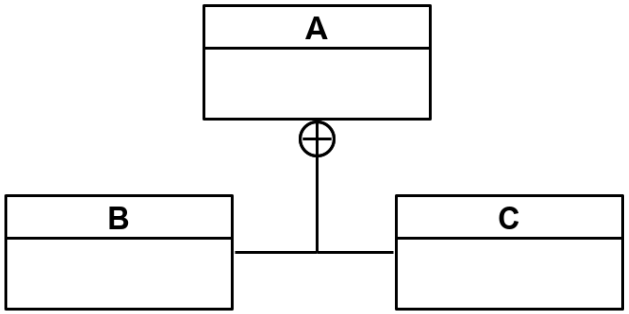
- Relazione "parte di"
  - Una classe fa parte di un package, un package fa parte di uno più grande
- Criteri per raggruppare
  - incapsulamento per modificabilità
  - supporto alle scelte costruisci/compra
  - moduli comuni in linee di prodotto
- A cosa serve questa vista
  - apprendimento del sistema
  - punto di partenza per l'allocazione del lavoro

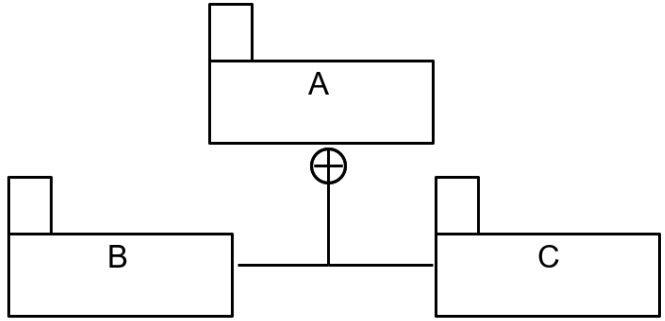
Notazione UML per la decomposizione

La relazione «parte di» è resa con ⊕ oppure con inclusione grafica (in un package)







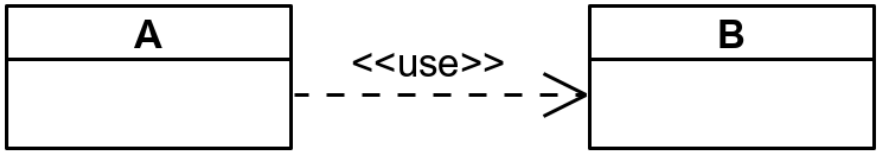


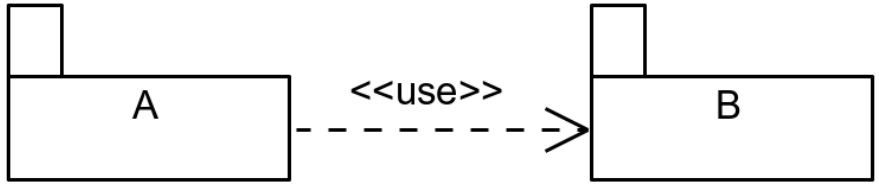
Seconda vista strutturale: d'uso

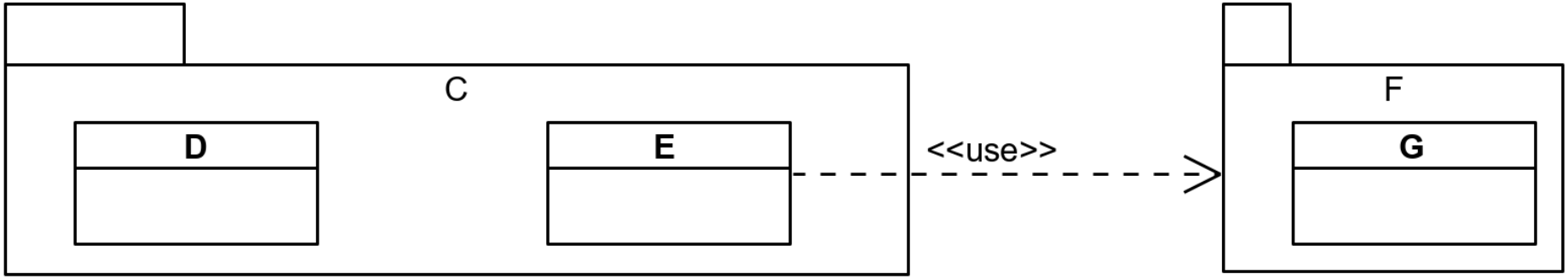
- Relazione "usa"
  - il modulo A usa il modulo B se dipende dalla presenza di B (funzionante correttamente) per soddisfare i suoi requisiti
    - Attenzione a non confondere invocazione con dipendenza: un modulo A che segnala un errore a B funziona correttamente indipendentemente da cosa fa il modulo B che riceve la segnalazione, per cui lo invoca, ma non lo usa, quindi non è suo cliente in una dipendenza.
  - cicli permessi ma pericolosi

- A cosa serve
  - pianificazione di sviluppo incrementale
  - test di unità e di integrazione

Notazione UML per la dipendenza d'uso

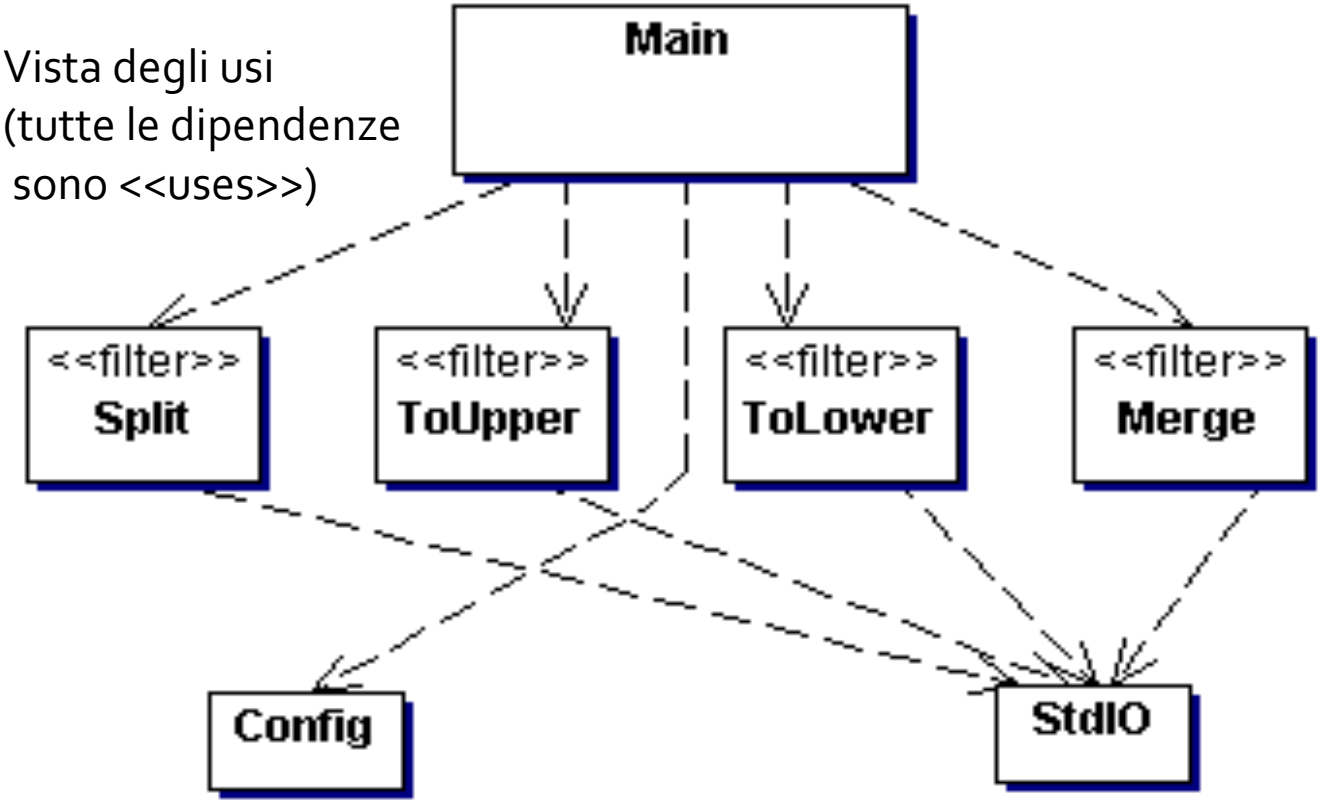






Vista strutturale degli usi (dell'esempio di cui abbiamo già dato vista C&C)

Vista degli usi
(tutte le dipendenze
sono <<uses>>)

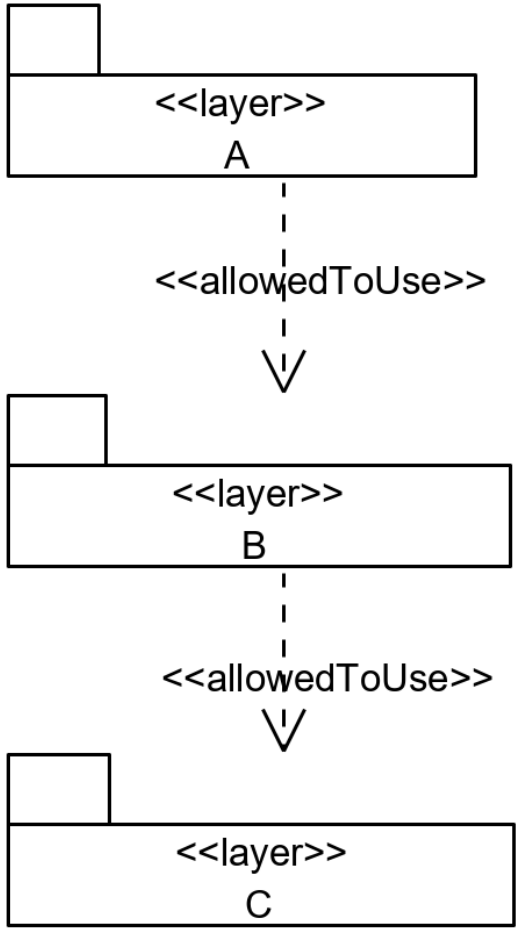


- i filtri si chiamano tra loro, ma non si usano…
- e il main li configura per metterli in comunincazione via StdIO (realizzazione del connettore pipe)
- suggerisce anche una vista a strati…

Terza vista strutturale: a strati (macchine virtuali)

- Elementi: strati
  - uno strato è un insieme coeso di moduli
    - a volte raggruppati in segmenti
  - offre un'interfaccia pubblica per i suoi servizi (macchina virtuale)
- Relazione: può usare
  - Caso particolare di relazione d'uso
    - antisimmetrica (a meno di rare eccezioni)
    - non implicitamente transitiva
- A cosa serve
  - modificabilità e portabilità
  - controllo della complessità

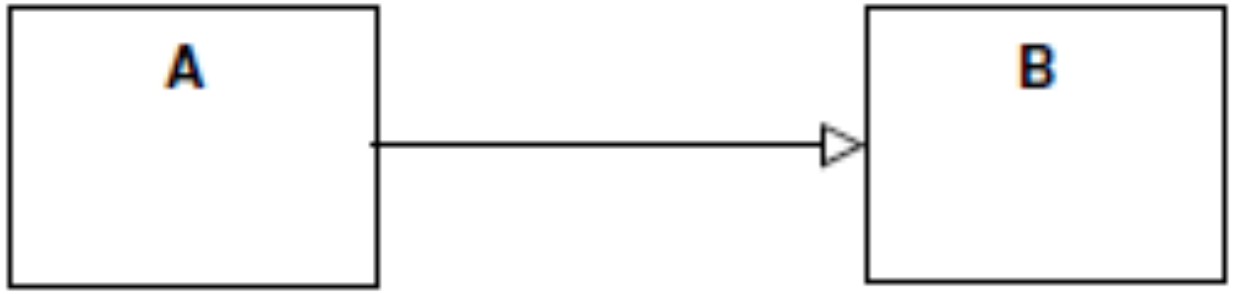
Notazione UML per gli strati (layers) e per la dipendenza «può usare»



Quarta vista strutturale: di generalizzazione

- Elementi: moduli (classi o packages)
- Relazione: generalizzazione
- A cosa serve
  - A rappresentare la relazione tipo-sottotipo (tra classi)
  - A rappresentare la relazione tra un framework (collezione di classi, anche astratte, con relazioni d'uso tra loro) e una sua specializzazione (tra packages)

Notazione UML per la generalizzazione



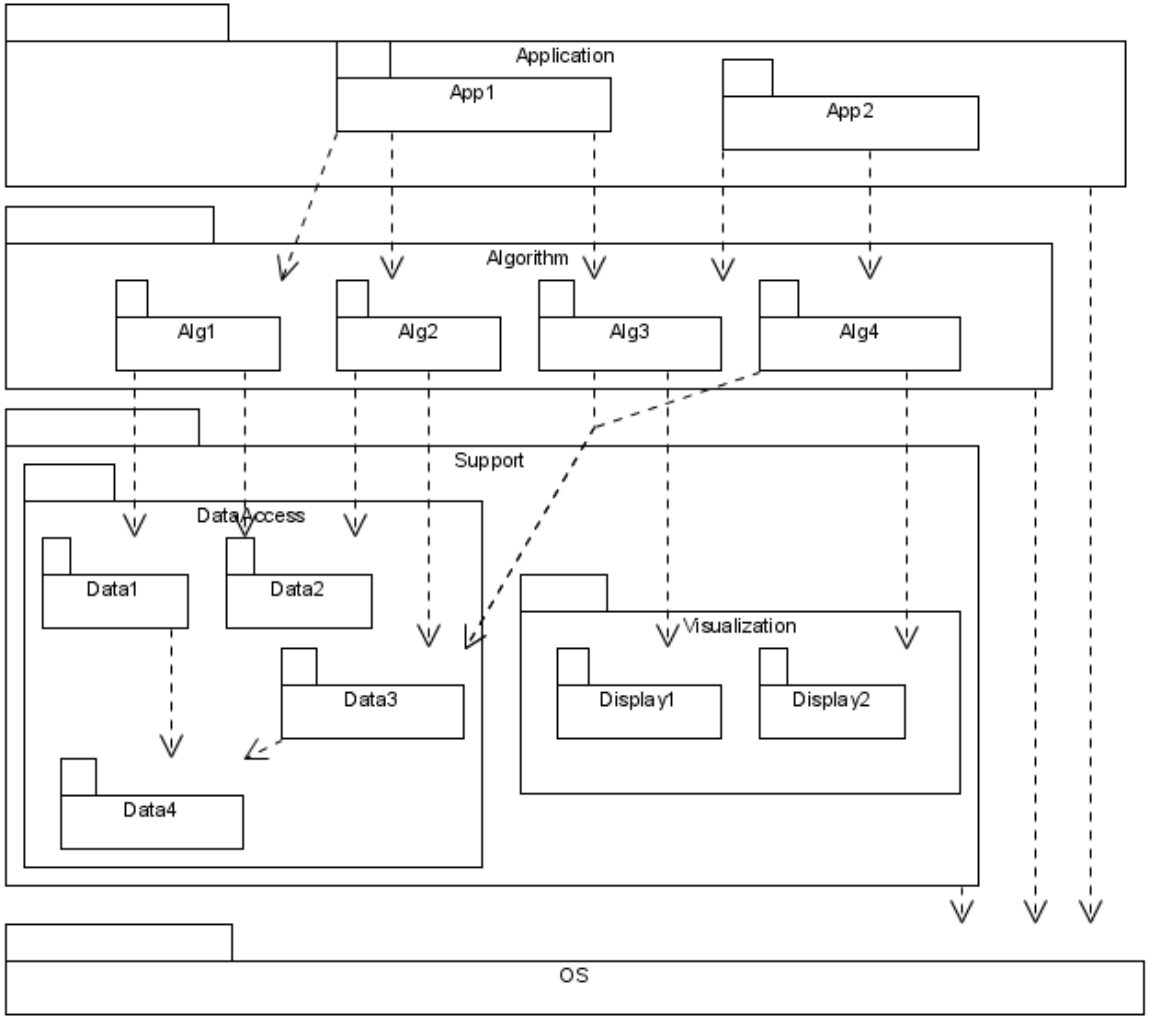
Esercizio

Module	Module Type	Invokes Module
APP-1	Application	ALGO-1, ALGO-2, ALGO-3
APP-2	Application	ALGO-3, ALGO-4
ALGO-1	Algorithm	DATA-1, DATA-2
ALGO-2	Algorithm	DATA-2, DATA-3
ALGO-3	Algorithm	DATA-3, DISP-1
ALGO-4	Algorithm	DATA-3, DISP-2
DATA-1	Data Access	DATA-4
DATA-2	Data Access	
DATA-3	Data Access	DATA-4
DATA-4	Data Access	
DISP-1	Display Output	
DISP-2	Display Output	
All Modules use Modules in the Operating System		

a.Sketch a graphical module view that illustrates the uses relationships implied by this table. Could these modules be grouped into packages of modules? Could these modules (or the packages) be grouped into layers? If so, try to incorporate your groupings into your view.
b.Sketch the dependency tree for the APP-1 application.
c.Suppose that we want to create a separate runtime component for each of the applications. Which modules would participate when the APP-2 component is running?
d.Assume that the original plan was to have the data access modules use the file system of the operating system for data management. Now, suppose that we decide to purchase a separate database system for data storage and retrieval. The database, of course, uses modules in the operating system. How would you incorporate this change into your graphical module view?

Soluzione parziale

Quali sono dipendenze d'uso, e quali sono «può usare» ?



Viste di tipo logistico

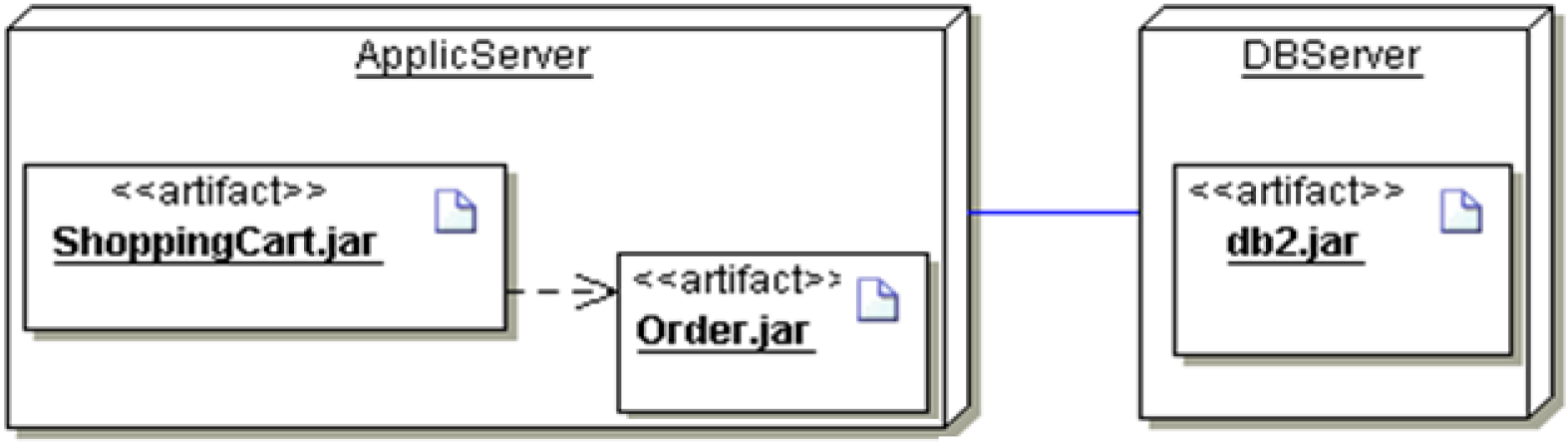
Vista logistica di dislocazione (aka di deployment, unica vista di tipo logistico che vediamo)

- Elementi
  - software: artefatti
  - dell'ambiente: hardware, ambiente di esecuzione
- Relazione
  - elemento software allocato a elemento dell'ambiente
- A cosa serve questa vista
  - analisi delle prestazioni
  - Guida per l'installazione

Un **artefatto** è un'informazione fisica che viene utilizzata o prodotta da un processo di sviluppo software o dal funzionamento di un sistema. Esempi di artefatti includono

- codice sorgente, script, file eseguibili binari,
- la tabella di un database,
- un deliverable di progetto, un documento word, un messaggio di posta.

UML: diagrammi di dislocazione



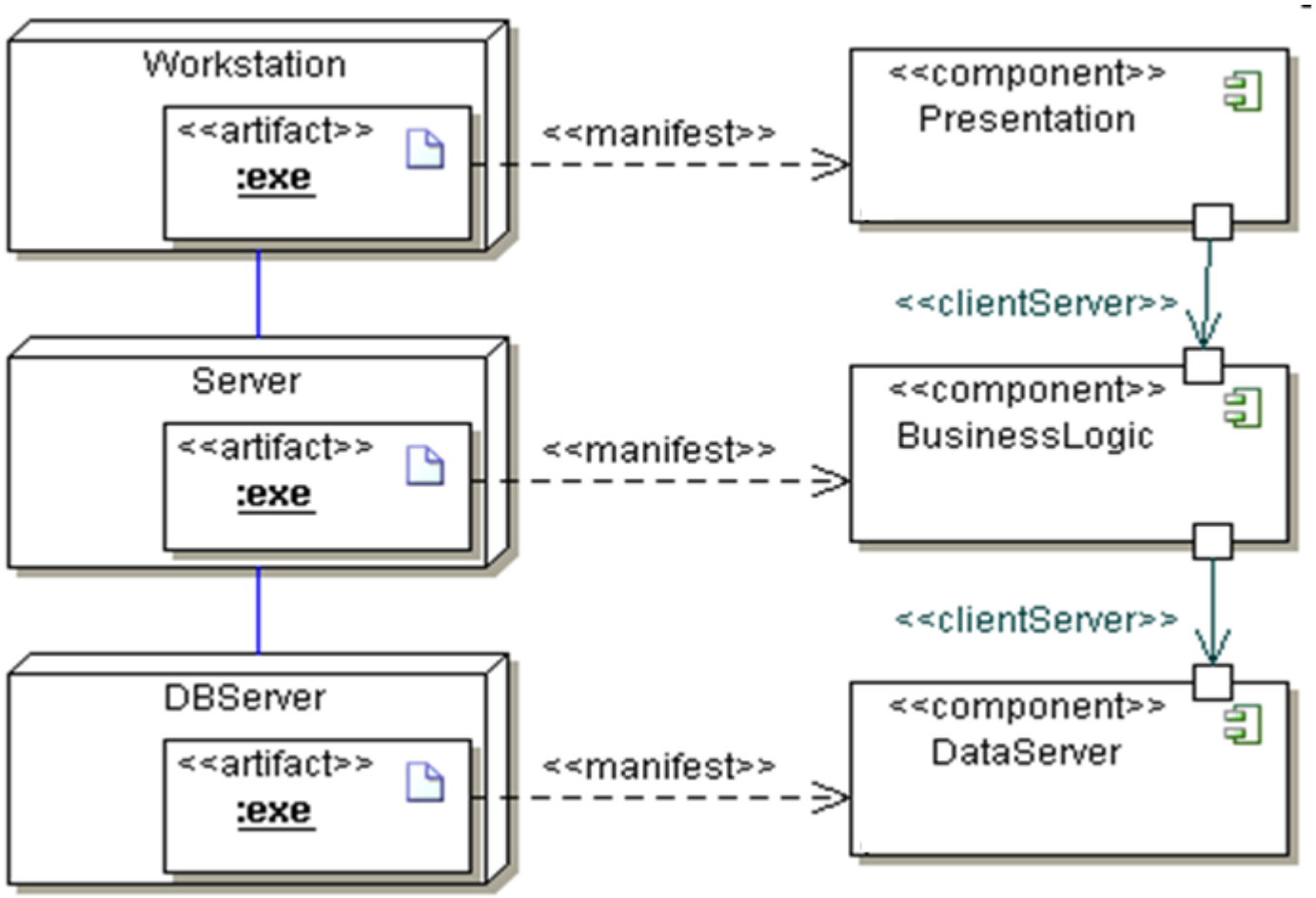
- Esempio a livello di istanza, un diagramma di dislocazione può essere anche a livello di classificatori
- **Nodo**: rappresenta un nodo hardware o, in generale, un ambiente di esecuzione.
- Le relazioni tra nodi sono connessioni fisiche o protocolli di comunicazione.
- **Artefatto**: informazione prodotta dallo sviluppo o esecuzione di sw

Viste ibride

Due o più viste in un diagramma, per collegare elementi

Vista ibrida: C&C e dislocazione

Esempio di architettura 3-tier



- Un artefatto «manifesta» un componente

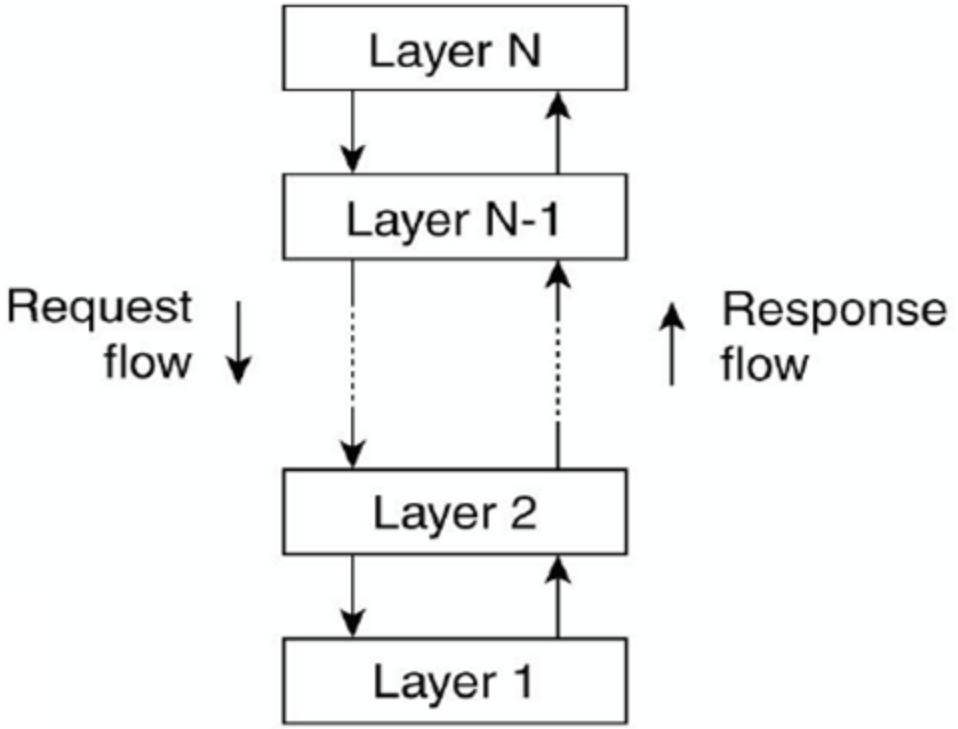
«Deployment» di componenti

- Si parta di deployment di componenti, mentre in realtà si disloca un artefatto
  - L'artefatto è una copia di un'implementazione di componente che è stata installata/rilasciata (deployed) su un particolare computer/ ambiente di esecuzione
- l'installazione avviene su un ambiente di esecuzione (nodo)
  - l'installazione comprende la configurazione e la registrazione del componente in tale ambiente
- Un artefatto «manifesta» un componente

Esempi

Architettura a livelli

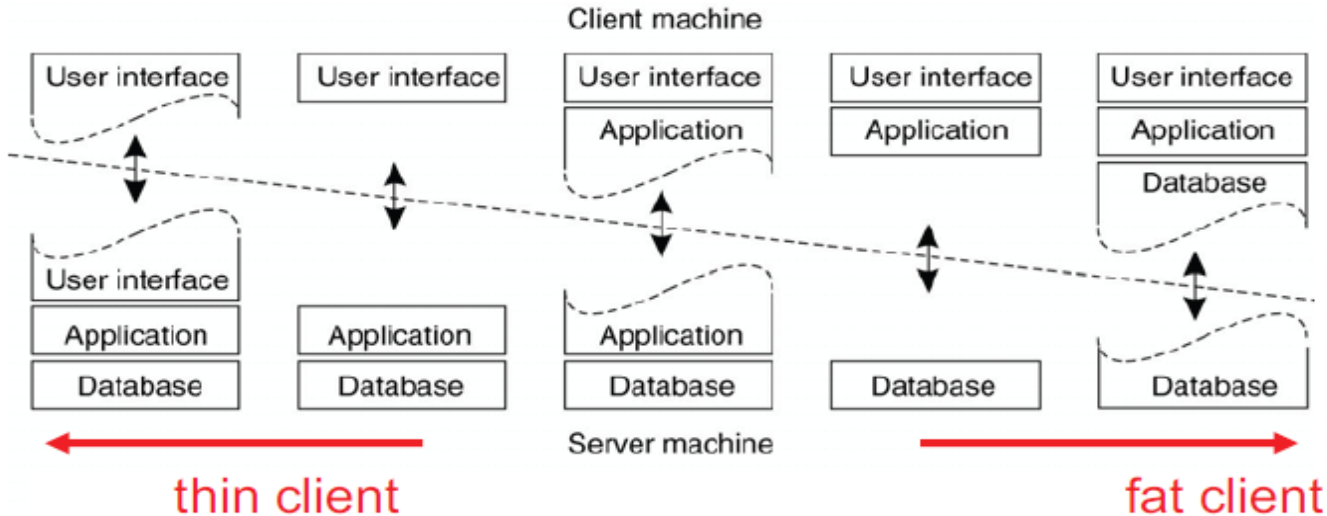
- Componenti organizzati in livelli (*layer*)
- Un componente a livello *i* può invocare un componente del livello sottostante *i-1*
- Le richieste scendono lungo la gerarchia, mentre le risposte risalgono



- Concetto ambiguo
  - Vista C&C: catene di client-server
  - Vista Strutturale: Layers

Architetture multilivello

- Mapping tra livelli logici (layer) e livelli fisici (tier)
- Architettura ad un livello (single-tier): configurazione monolitica mainframe e terminale “stupido” (non è C/S!)
- Architettura a due livelli (two-tier): due livelli fisici (macchina client/singolo server)
- Architettura a tre livelli (three-tier): ciascun livello su una macchina separata
- Diverse configurazioni two-tier

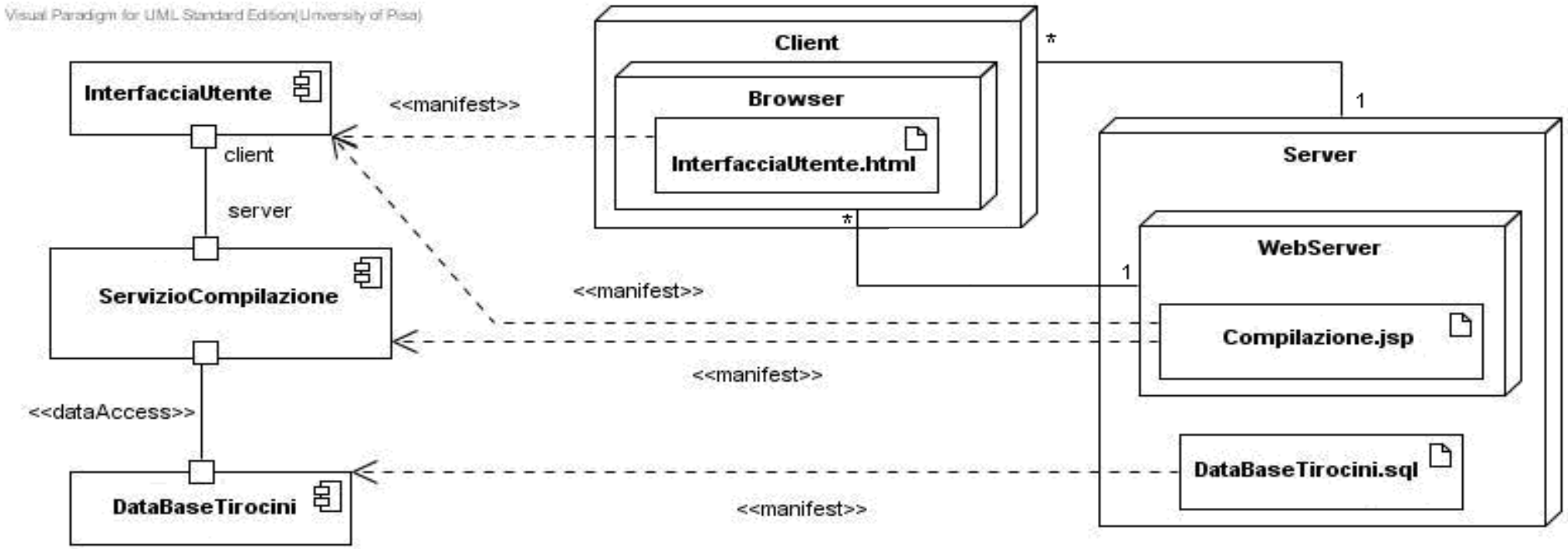


Architetture multilivello (cont'd)

- Da un livello ad N livelli
- Con l'introduzione di ciascun livello
  - L'architettura guadagna in flessibilità, funzionalità e possibilità di distribuzione
- Ma l'architettura multilivello potrebbe introdurre un problema di prestazioni
  - Aumenta il costo della comunicazione
  - Viene introdotta più complessità, in termini di gestione ed ottimizzazione

Esempio
Tirocini: 3-layers, 2-tier

Vista ibrida (C&C e dislocazione) dell'architettura del sotto-sistema di Compilazione Tirocini, assumendo che gli artefatti che manifestano le componenti citate siano: InterfacciaUtente.html, visualizzato da un browser di una macchina client e Compilazione.jsp (dislocata su un web server) e DataBaseTirocini.sql, mantenute su una macchina server.



Syllabus

- Dispensa di architettura:
  - Carlo Montangero e Laura Semini, *Architetture software e Progettazione di dettaglio, Note per il Corso di Ingegneria del software, 2014*
  - Su didawiki

Riferimenti

- Shaw, M e Garlan, D. Software Architecture. Prentice-Hall, 1996.
- Soni, D. et al. Software Architecture in Industrial Application. Proc. 17° Int. Conf. on Software Engineering, 1995.
- Parnas, D. Designing Software for the ease of Extention and Contraction. IEEE Transaction on SE 5(2), 1979.
- Krutchen, P. The 4+1 View Model of Architecture. IEEE Software 12(6), 1995.
- Magee, J. et al. Specifying Distributed Software Architecures. Proc. Fifth Europ. Software Eng. Conf. LNCS 989, Springer Verlag, 1995.
- Clemens, P. et al. Documenting Software Architectures. Addison Wesley, 2003.