Progettazione: architetture software

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 passa da
  - “che cosa” deve essere fatto a
  - “come” deve essere fatto

- Il suo prodotto si chiama architettura (o progetto) del sw

Progettazione: livello di astrazione

- Progettazione di alto livello (o architetturale)
  - Scopo è 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

Definizione di architettura software

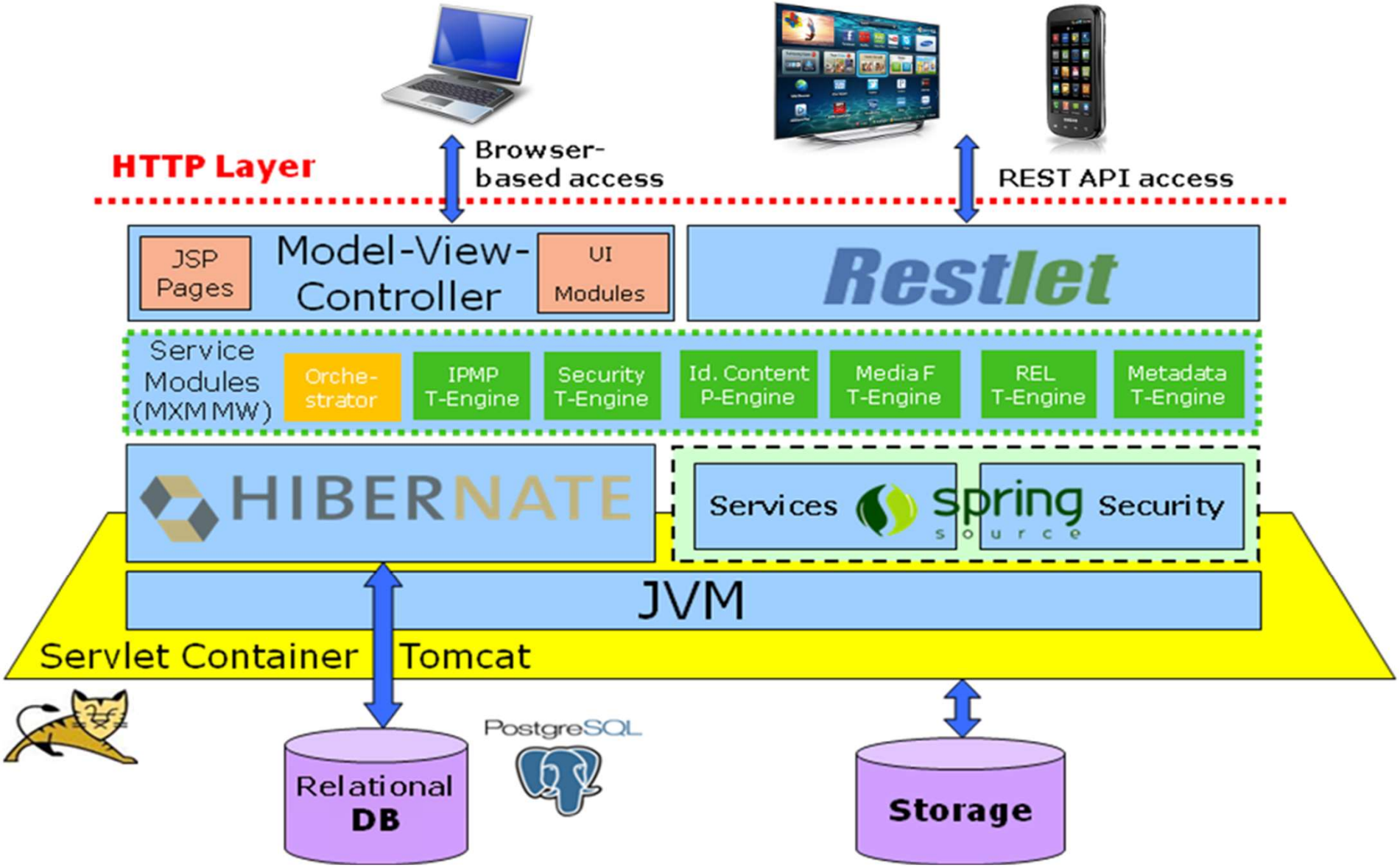
- L’architettura di un sistema software (in breve architettura software) è la struttura del sistema, costituita:
  - dalle parti del sistema,
  - dalle relazioni tra le parti
  - dalle loro proprietà visibili

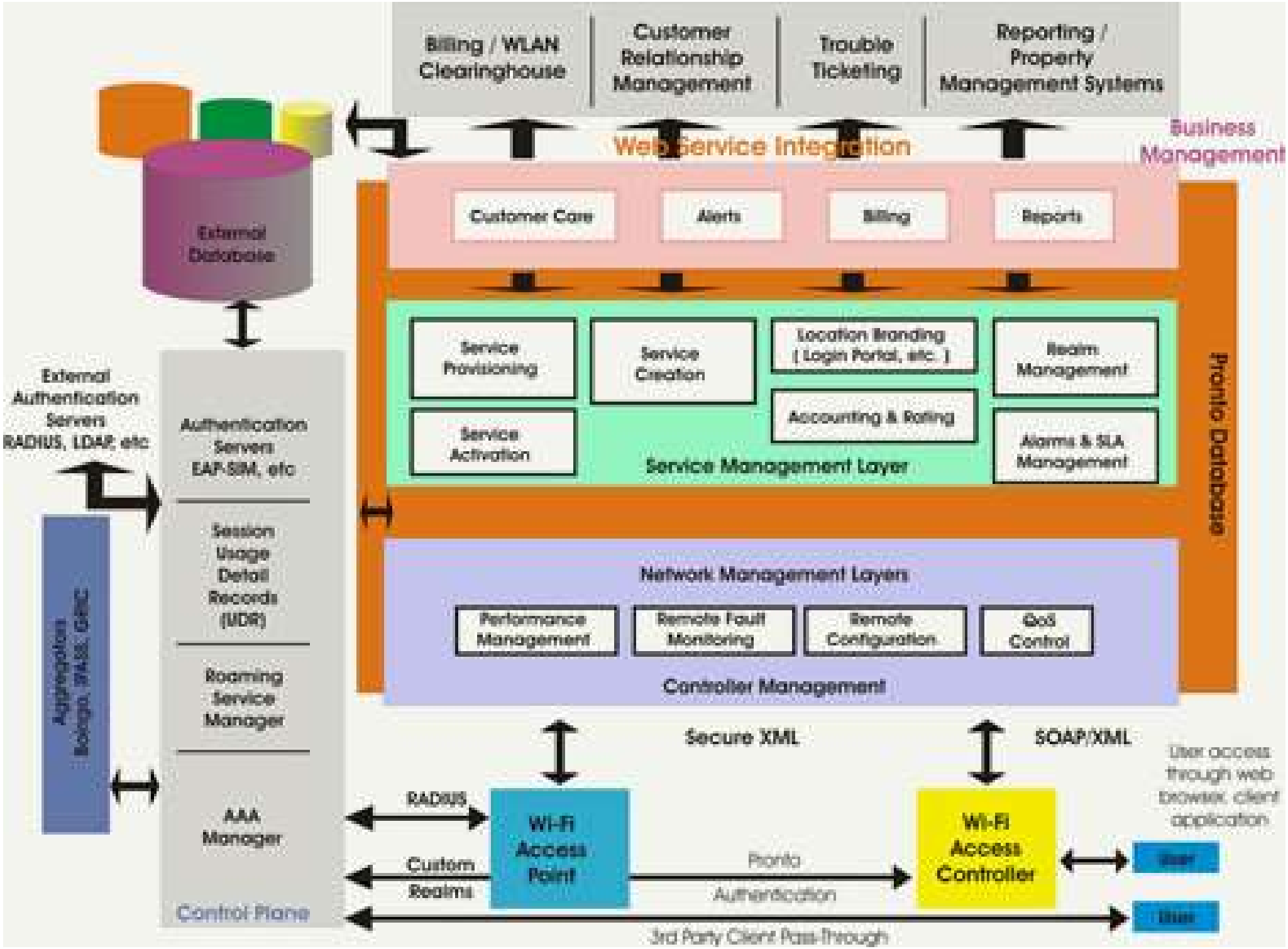
in altre/altrui parole

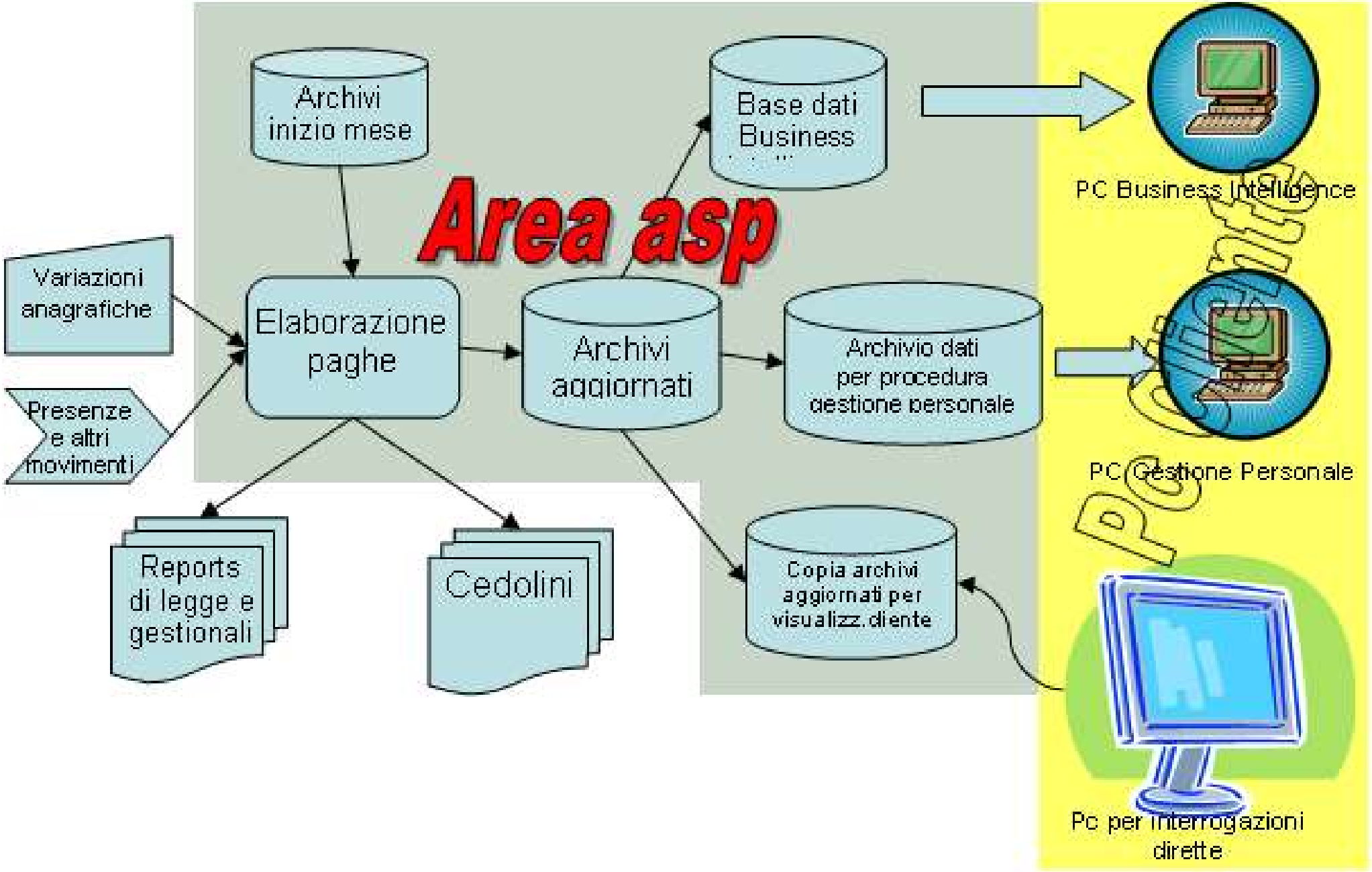
- L’architettura
  - definisce la struttura del sistema sw
  - specifica le comunicazioni tra componenti
  - considera aspetti non funzionali
  - è un’astrazione
  - è un artefatto complesso

→ problema di rappresentazione: Come si modella?

Esempi di descrizioni che si possono trovare




Billing / WLAN Clearinghouse
Customer Relationship Management
Trouble Ticketing
Reporting / Property Management Systems
Web Service Integration
Business Management
External Database
Customer Care
Alerts
Billing
Reports
Service Provisioning
Service Creation
Location Branding (Login Portal, etc.)
Realm Management
Service Activation
Accounting & Rating
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
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
3rd Party Client Pass-Through

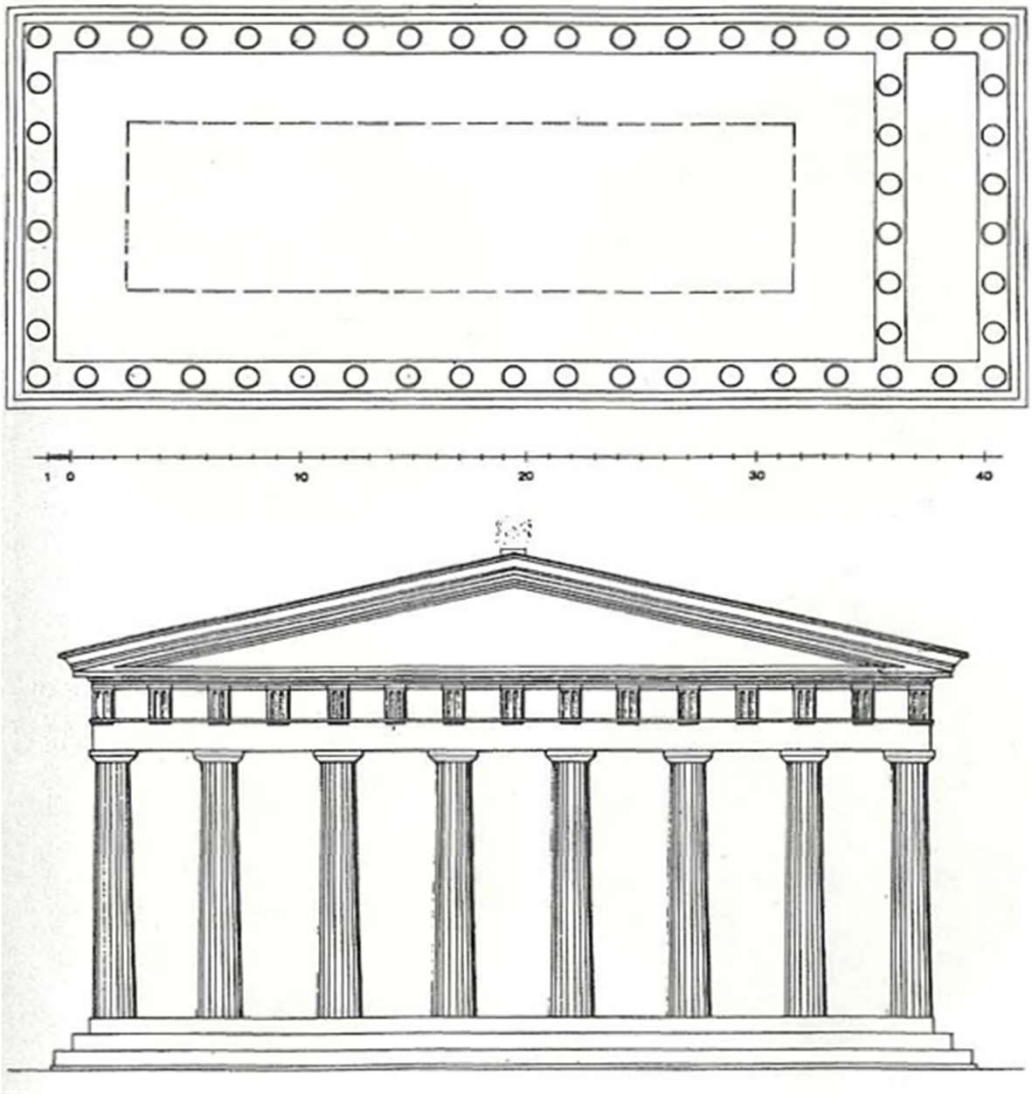

Area asp
Archivi inizio mese
Base dati Business
Variazioni anagrafiche
Elaborazione paghe
Archivi aggiornati
Archivio dati per procedura gestione personale
Presenze e altri movimenti
Reports di legge e gestionali
Cedolini
Copia archivi aggiornati per visualizz. diente
PC Business Intelligence
PC Gestione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

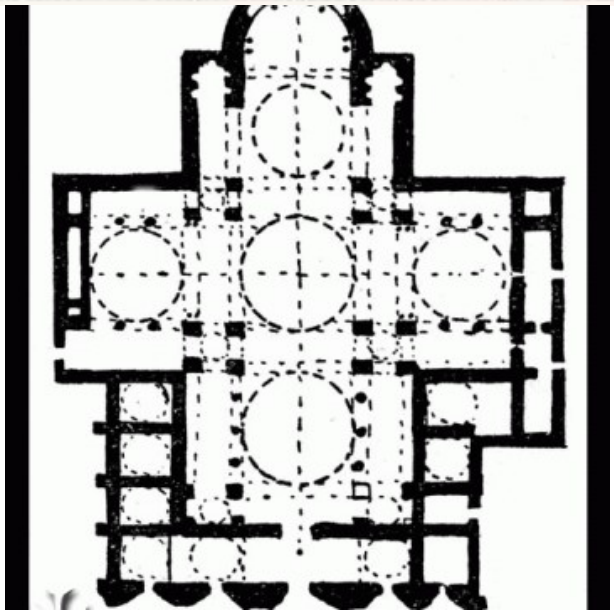
  - si realizza graficamente tramite piante, prospetti e sezioni
  - con un linguaggio grafico standard

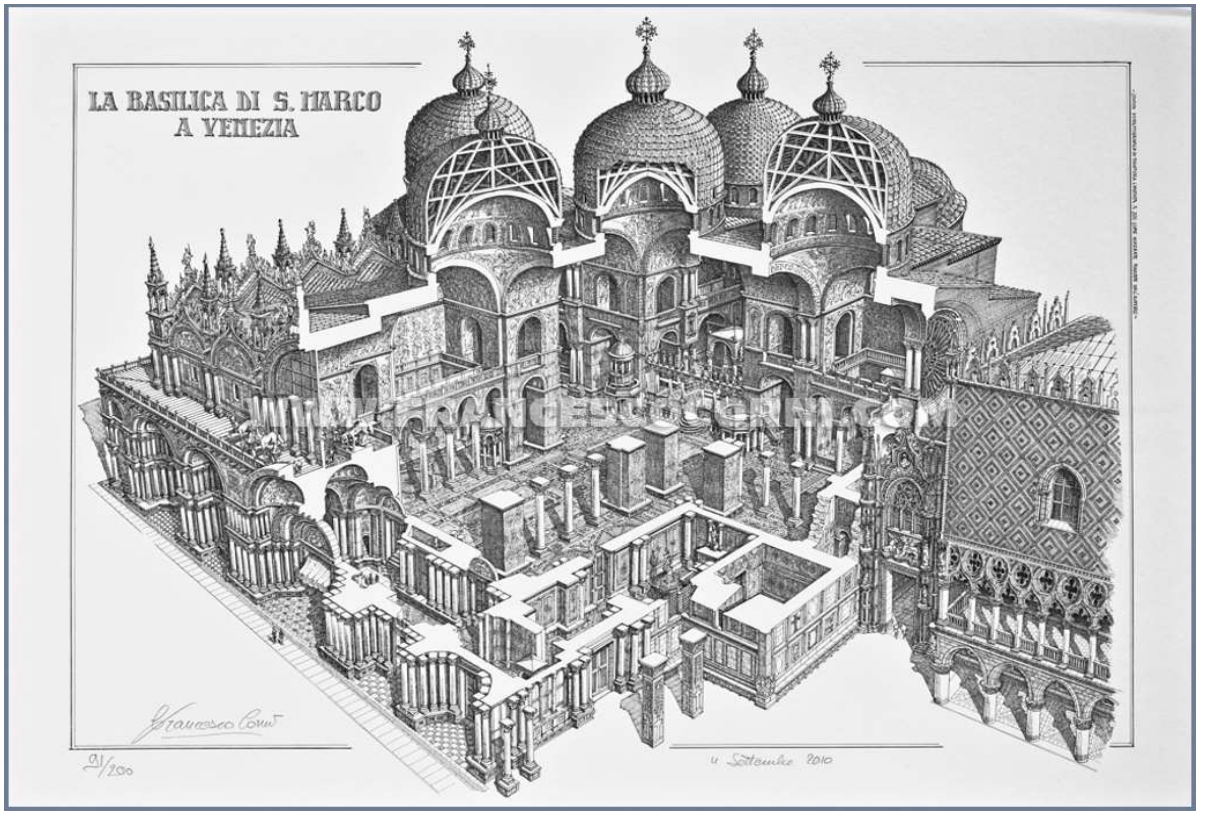
Linguaggio standard e punti di vista



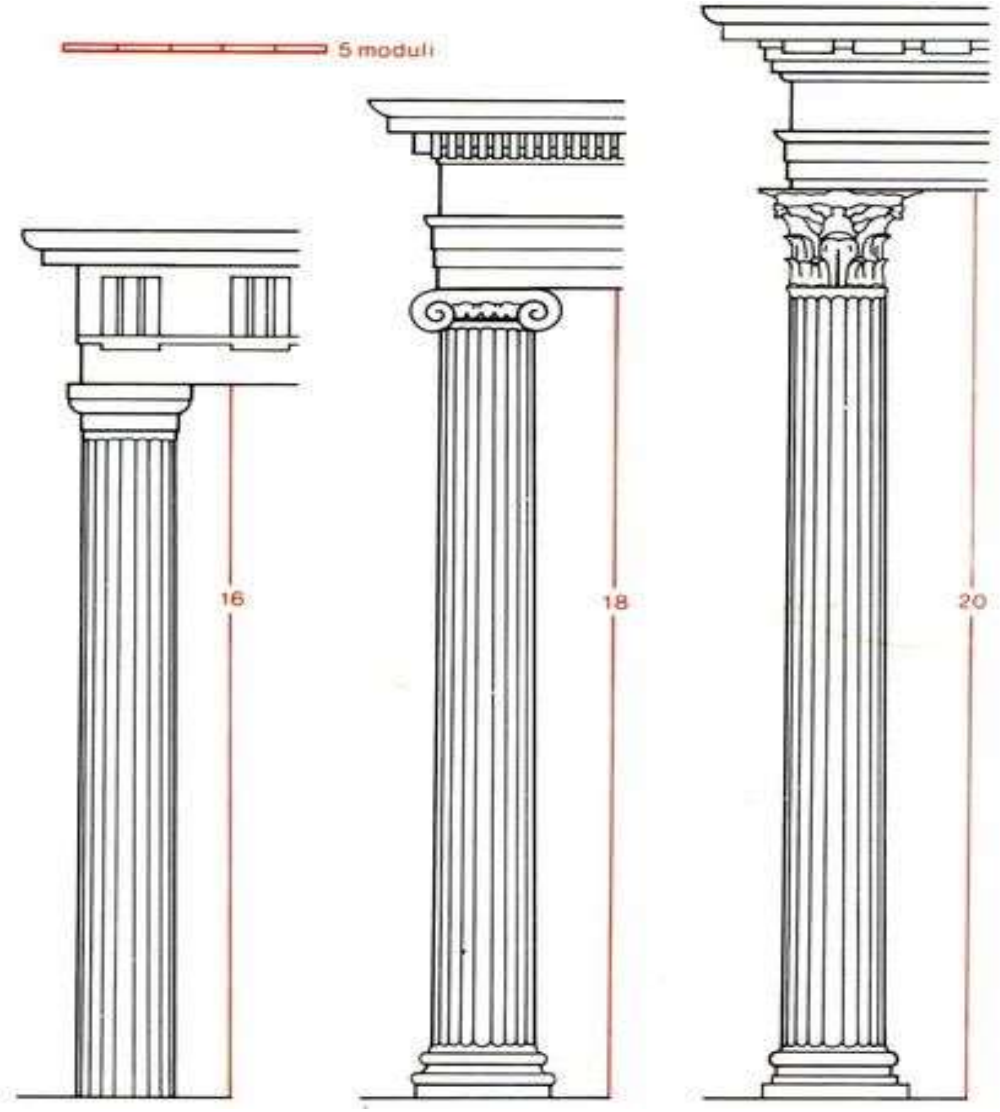
- Pianta
  - vista dall'alto
  - proiezione sul piano xy
- Prospetto
  - vista frontale
  - proiezione sul piano xz (o yz)

Diverse viste su un’architettura: proiezioni (prospetto e pianta) e spaccato





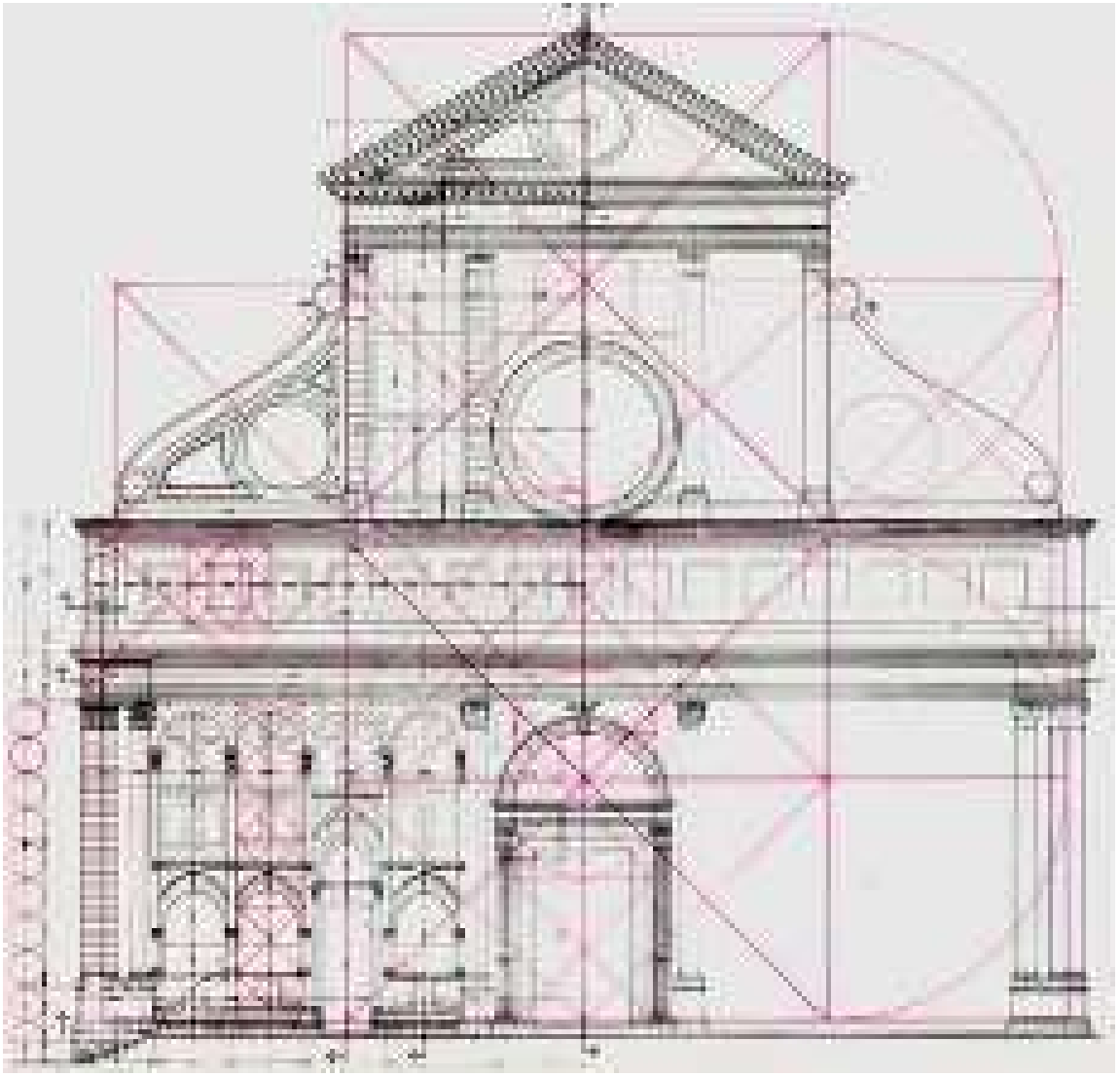
Ancora analogia: stessa vista, stili diversi



Nella costruzione di una colonna posso seguire uno degli stili noti

Nei casi in figura il criterio di scelta è guidato principalmente da considerazioni estetiche

Pattern: schemi di progettazione da applicare (esempi gli schemi proporzionali di Vitruvio o Alberti)



S. Maria Novella
L.B. Alberti

Torniamo al software: le viste

- 3 astrazioni interessanti → 3 punti di vista simultanei sul sistema sw
  - vista comportamentale
  - vista strutturale
    - vista logistica

Vista comportamentale

- Aka component-and-connector, aka 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, ...
- Utile anche per documentare lo stile dell'architettura

Vista Strutturale

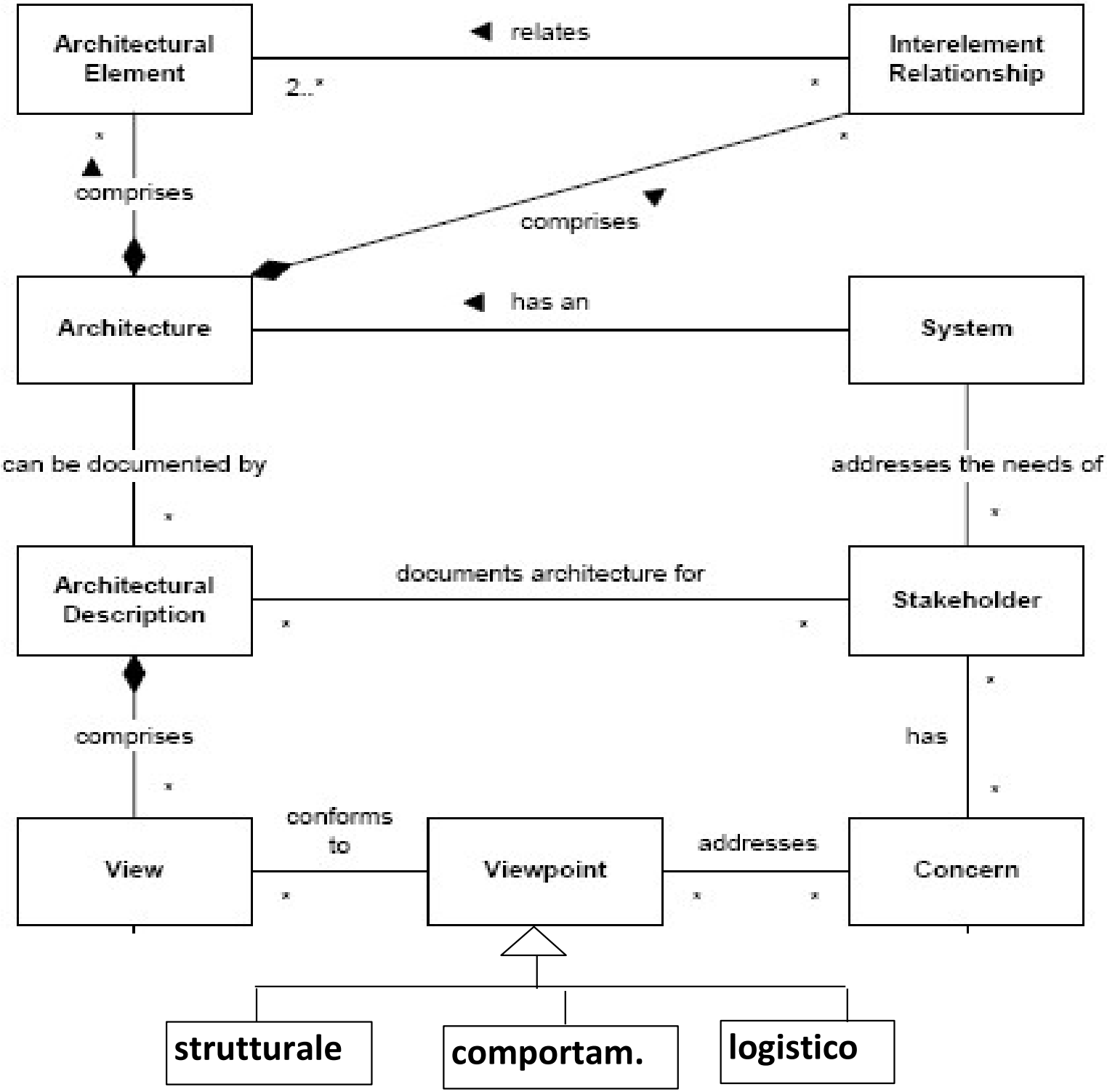
- Descrive la struttura del sistema come insieme di unità di realizzazione (codice)
  - Classi, packages…

- A cosa serve?
  - Analizzare dipendenze tra packages
  - Progettare test di unità e di integrazione
  - Valutare la portabilità

Vista Logistica (di allocazione)

- Aka vista di deployment
- Descrive l'allocazione del sw su ambienti di esecuzione

- A cosa serve?
  - 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

Vista di tipo comportamentale

Aka C&C aka componenti e connettori

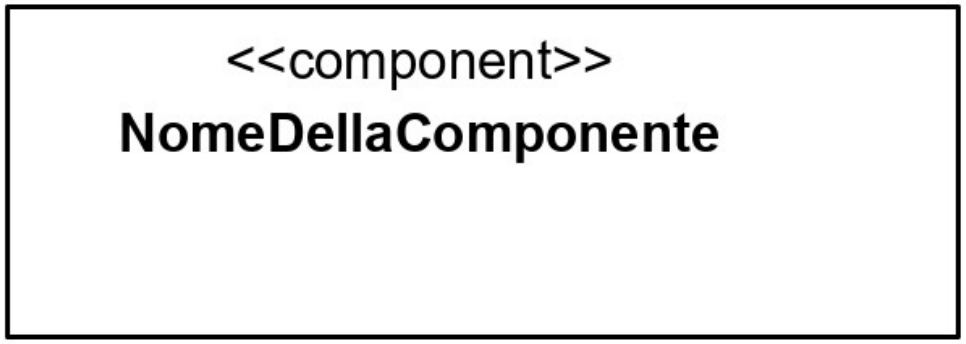
Componenti

- Componente software
  - Una componente software è un'unità di software indipendente e riusabile che esegue una specifica funzione o compito all'interno di un'applicazione più ampia.
  - È un’unità concettuale di decomposizione di un sistema a tempo d’esecuzione,
    - Per esempio: processo, oggetto, servizio, deposito 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 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

Come descrivere componenti in UML

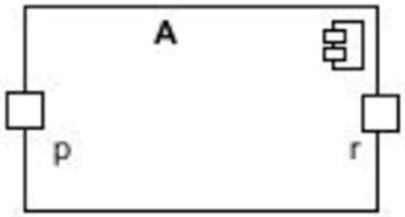


- Un componente è un classificatore
- Si rappresenta in uno di questi modi
  - Sono tutti equivalenti tra loro
  - il terzo modo è ridondante, ma è quello usato da Visual Paradigm

Porti

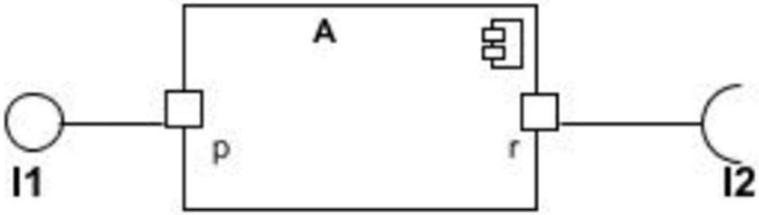
- I porti identificano i punti di interazione di un componente
  - un componente può avere più porti, uno pe ogni tipo di connessione con altri componenti
  - un porto fornisce o richiede una o più interfacce (omogenee)
  - UML: un porto è rappresentato con un «quadratino», può avere un nome e/o avere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

Porti e interfacce: notazione

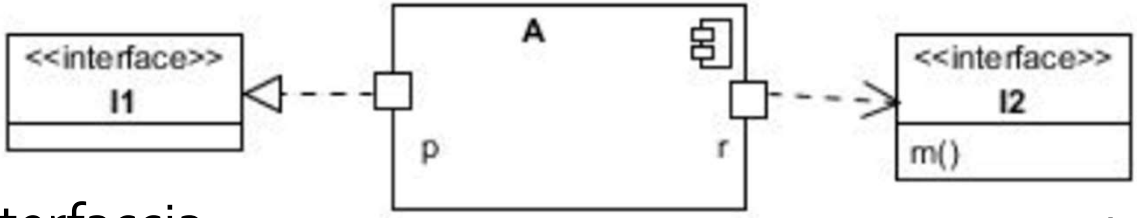


porti senza specifica di interfacce

porto con interfaccia fornita
(in forma sintetica: con lollipop)



porto con interfaccia richiesta
(in forma sintetica: con forchetta)

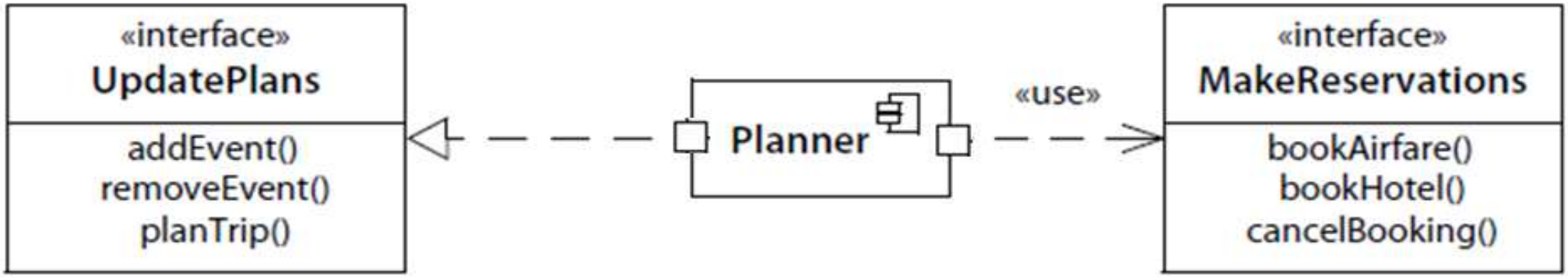


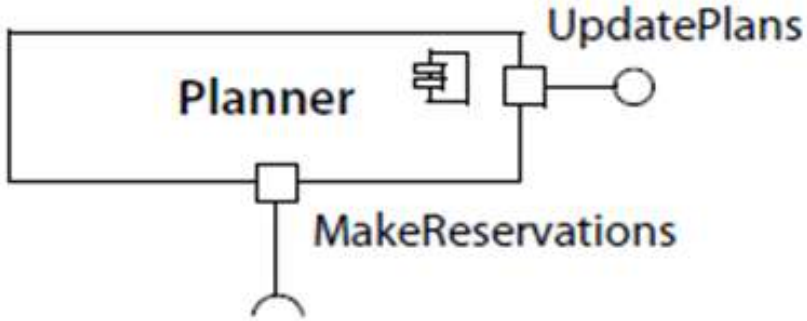
porto con interfaccia fornita descritta
in forma estesa

porto con interfaccia richiesta descritta
in forma estesa

Interfacce: descrizione sintetica vs estesa

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



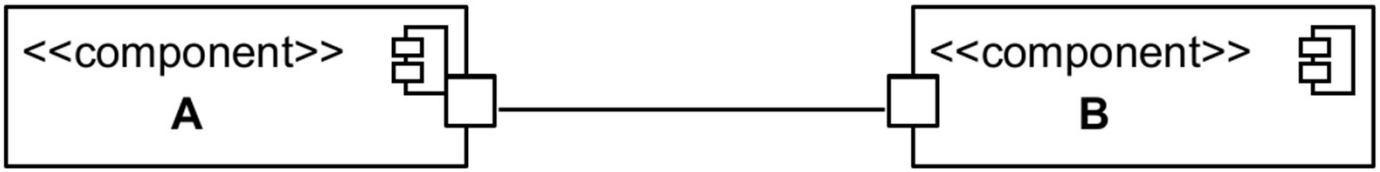


Connettori

- I connettori sono canali di interazione tra componenti che collegano i porti
- protocolli, flussi d’informazione, accessi ai depositi, ...

Connettori in UML

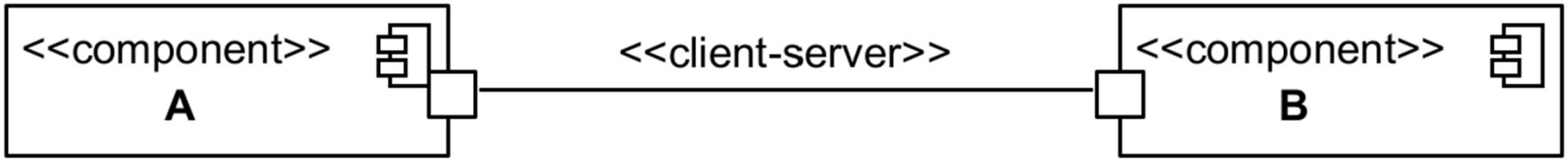
- I connettori collegano i porti



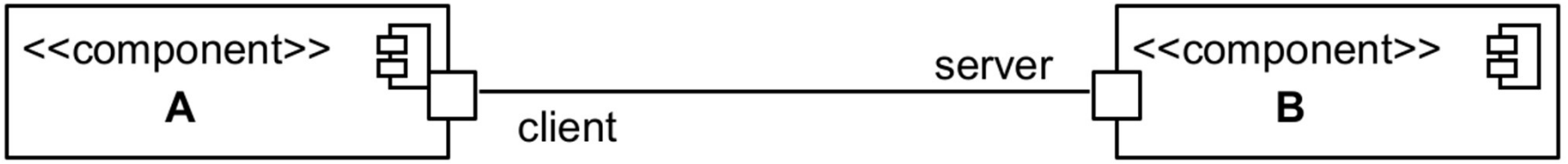
- In UML i connettori non hanno un descrittore specifico, si modellano con un'associazione
- Aggiungiamo informazione sul protocollo di comunicazione, descritto sinteticamente indicando lo stile della connessione (stile di quel frammento di architettura)

Connettori in UML: stile del protocollo

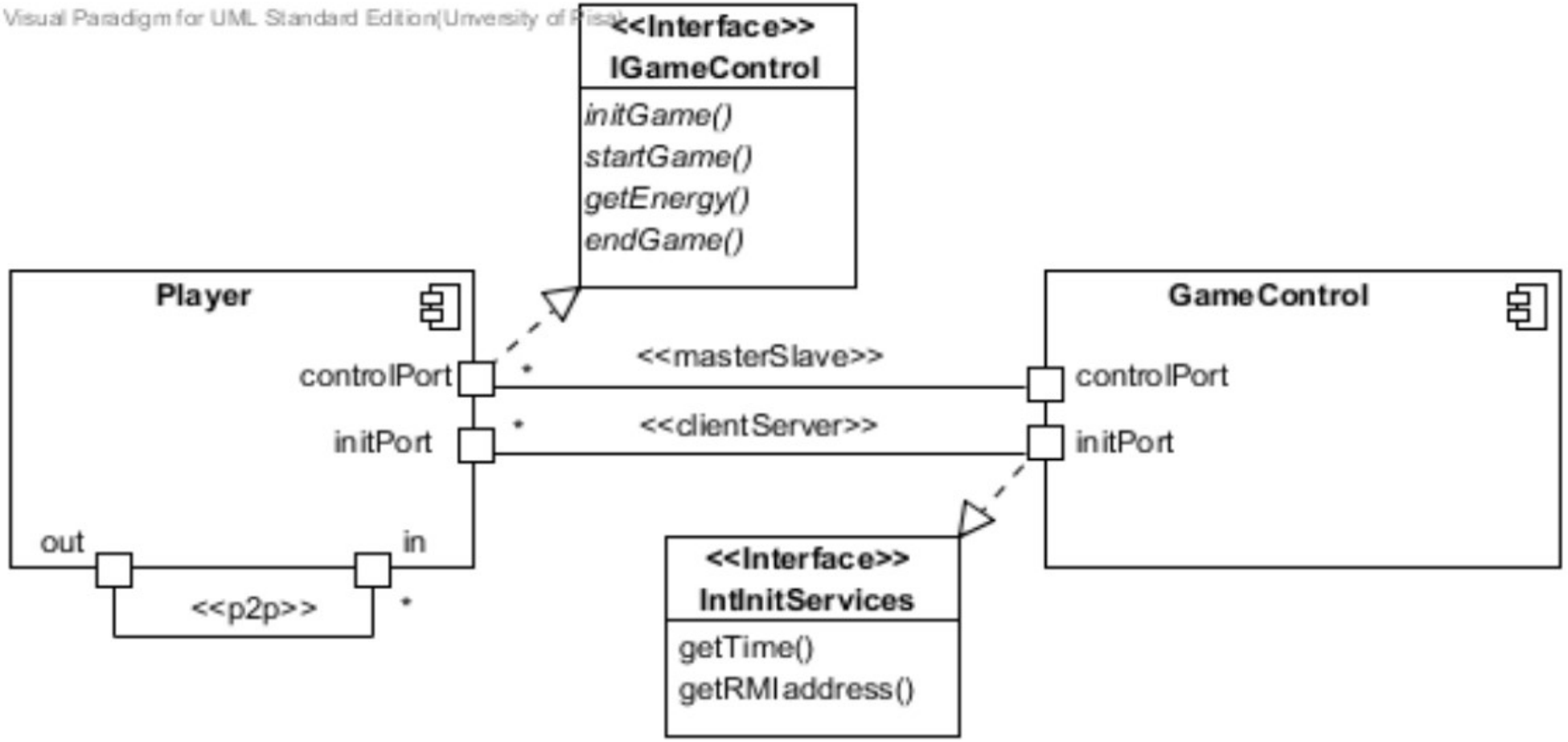
- Per documentare il protocollo di interazione si usa uno stereotipo:
  - Esempi: <<clientServer>> ; <<dataAccess>> (specializzazione di clientServer) ; <<pipe>> ; <<peer2peer>> (<<p2p>>); <<publish-subscribe>>



- Oppure si indicano i ruoli delle componenti:



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()

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

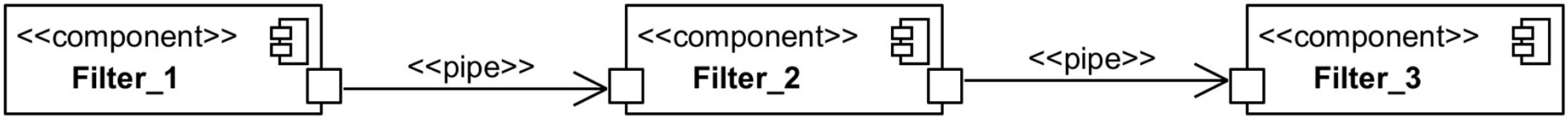
Stili (o schemi) architetturali e vista C&C

- 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 architetturali: condotte (pipe) e filtri

- 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

Pipe and filter

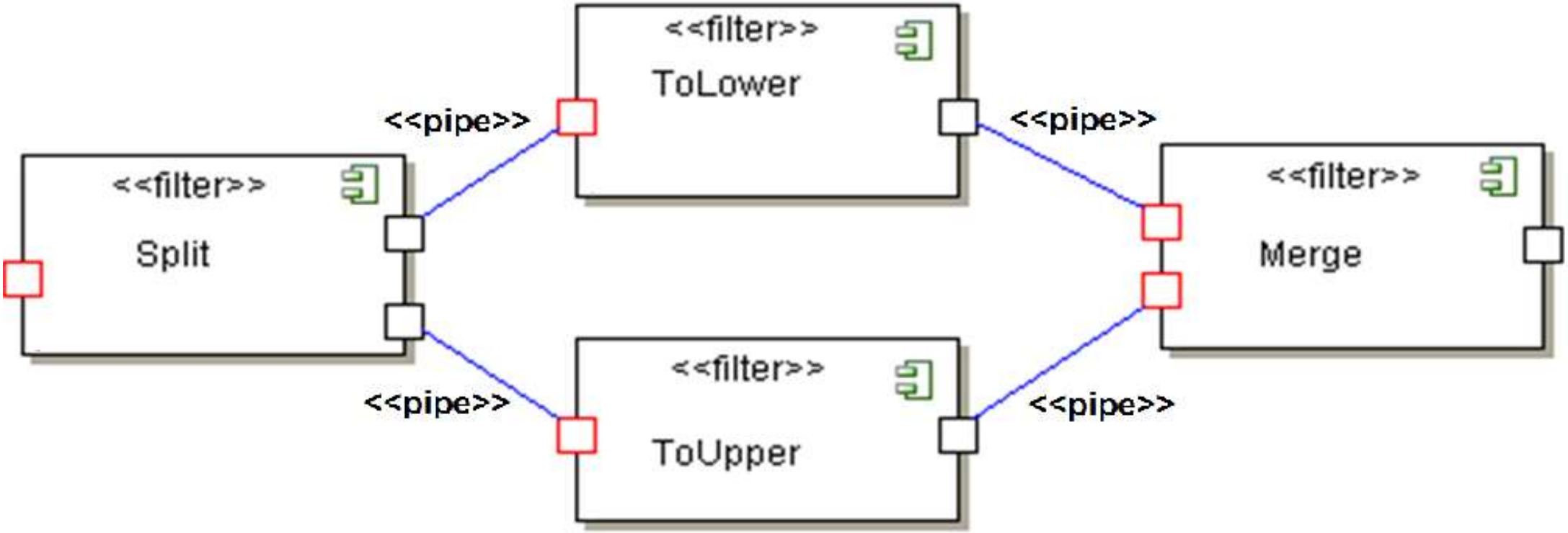


- Lo stile Pipes and Filters consiste in una catena di filtri per l'elaborazione dei dati, collegati attraverso delle pipe:
  - i filtri passano i dati in uscita ai filtri adiacenti attraverso le pipe.
- Gli elementi del modello Pipes and Filters possono variare nelle funzioni che svolgono
  - Esempi: pipe con supporto per il buffering dei dati, biforcazioni.

Pipe and filter: esempio

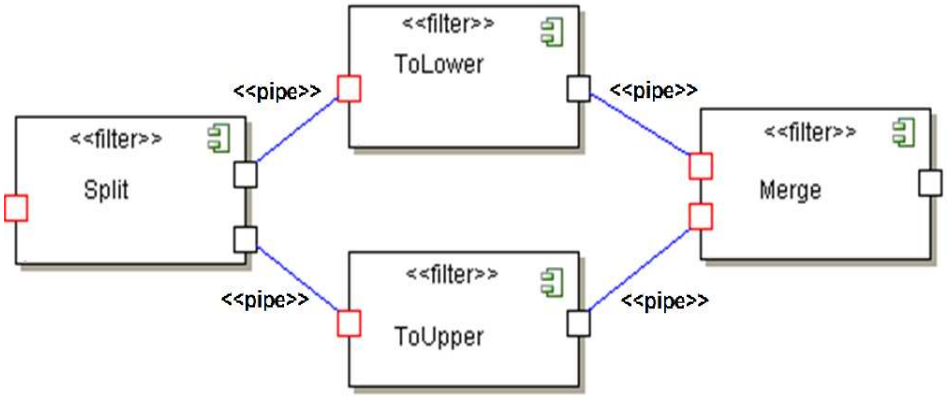
Esempio: Pipe and filter con biforcazione

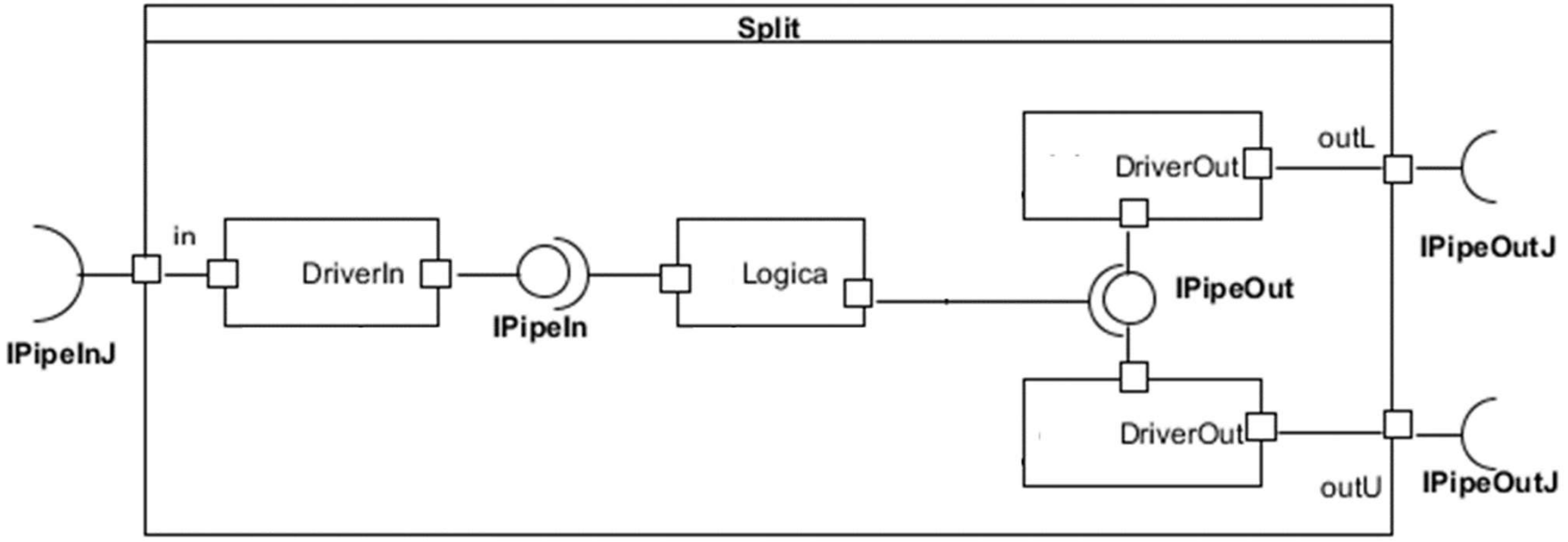
Per esempio «ciao» o «CIAO» o «Ciao» viene trasformato in «claO»



Come si realizza una pipe: esempio

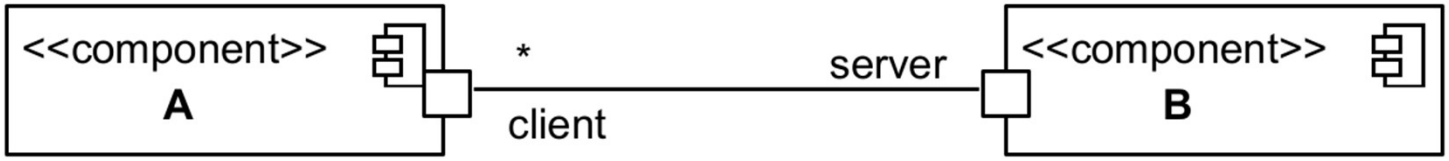
Esempio della componente Split (push vs pull)





Stile client-server

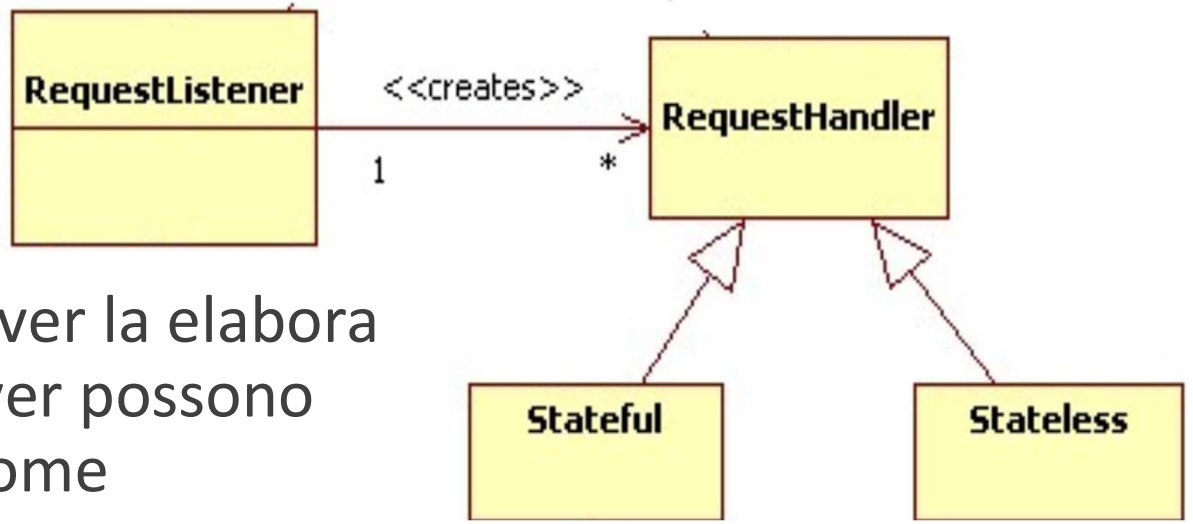
- l sistema è formato da due componenti: il client e il server
  - spesso, ma non necessariamente, eseguiti su macchine diverse collegate in rete



- 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
  - Ci possono essere tanti clienti
- Un client invia al server le richieste ed attende una risposta

Come si realizza un’architettura client server

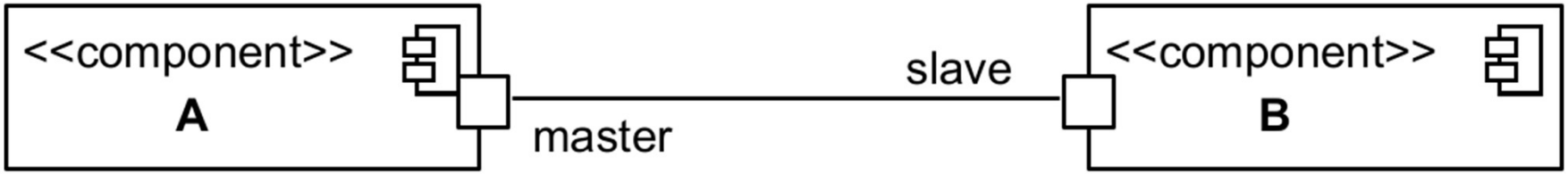
Lato server: uno (o più) thread in ascolto delle richieste, più un gestore delle richieste.



Quando riceve una richiesta, il server la elabora e invia una risposta al client. I server possono essere ulteriormente classificati come stateless o stateful.
I client di un server stateful possono fare richieste composite che consistono in più richieste atomiche. Ciò consente un'interazione più colloquiale o transazionale tra client e server. A tal fine, un server stateful conserva un record delle richieste di ciascun client corrente. Questo record è chiamato sessione.

Stile master-slave

- Un caso particolare del client-server, ma risponde a esigenze diverse

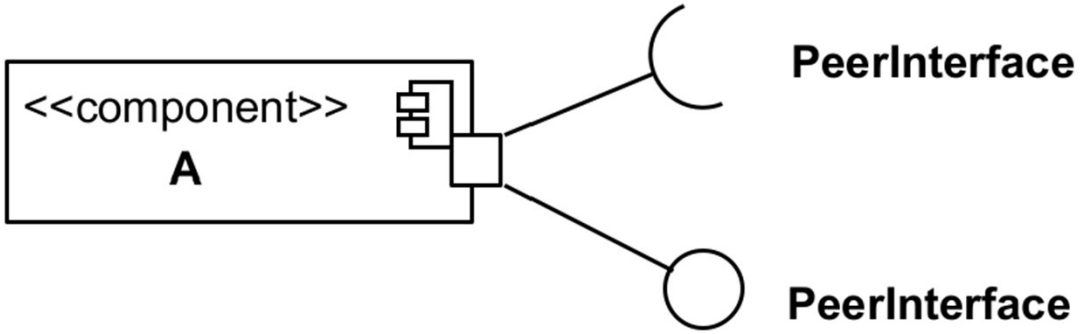


- Il servente (slave) serve un solo cliente (master)

- Architettura usata per esempio nella replica di database, il database master è considerato come fonte autorevole e i database slave sono sincronizzati con esso

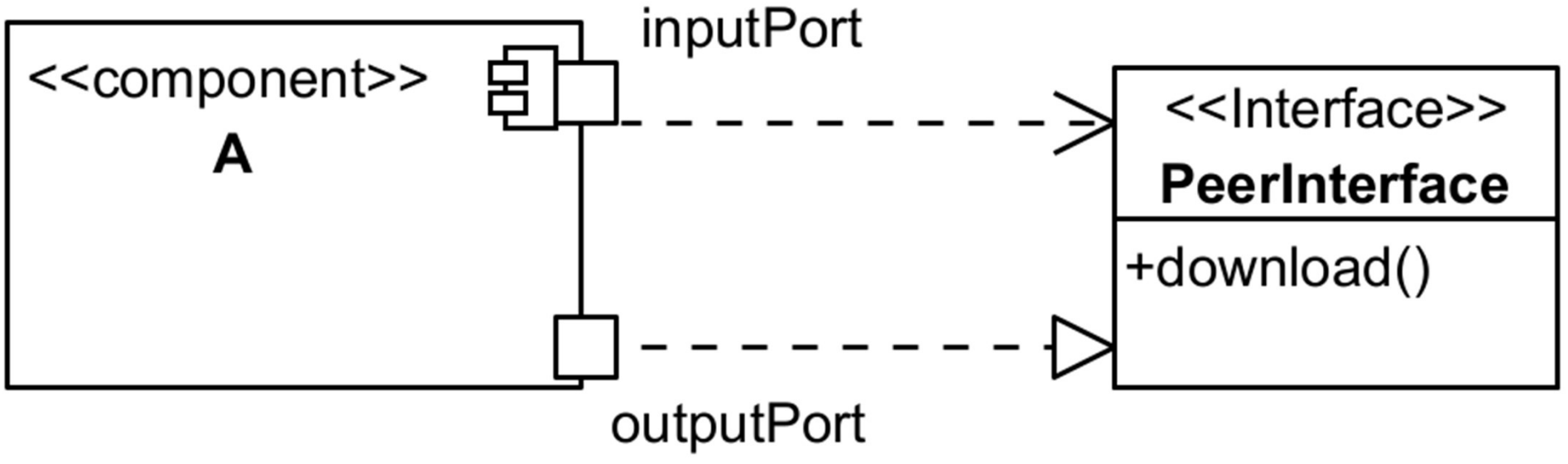
Stile P2P (peer to peer)

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



  - C’è una sola componente nello schema perché i peer sono tutti sue istanze

P2P


<<component>>
A
inputPort
outputPort
<<Interface>>
PeerInterface
+download()

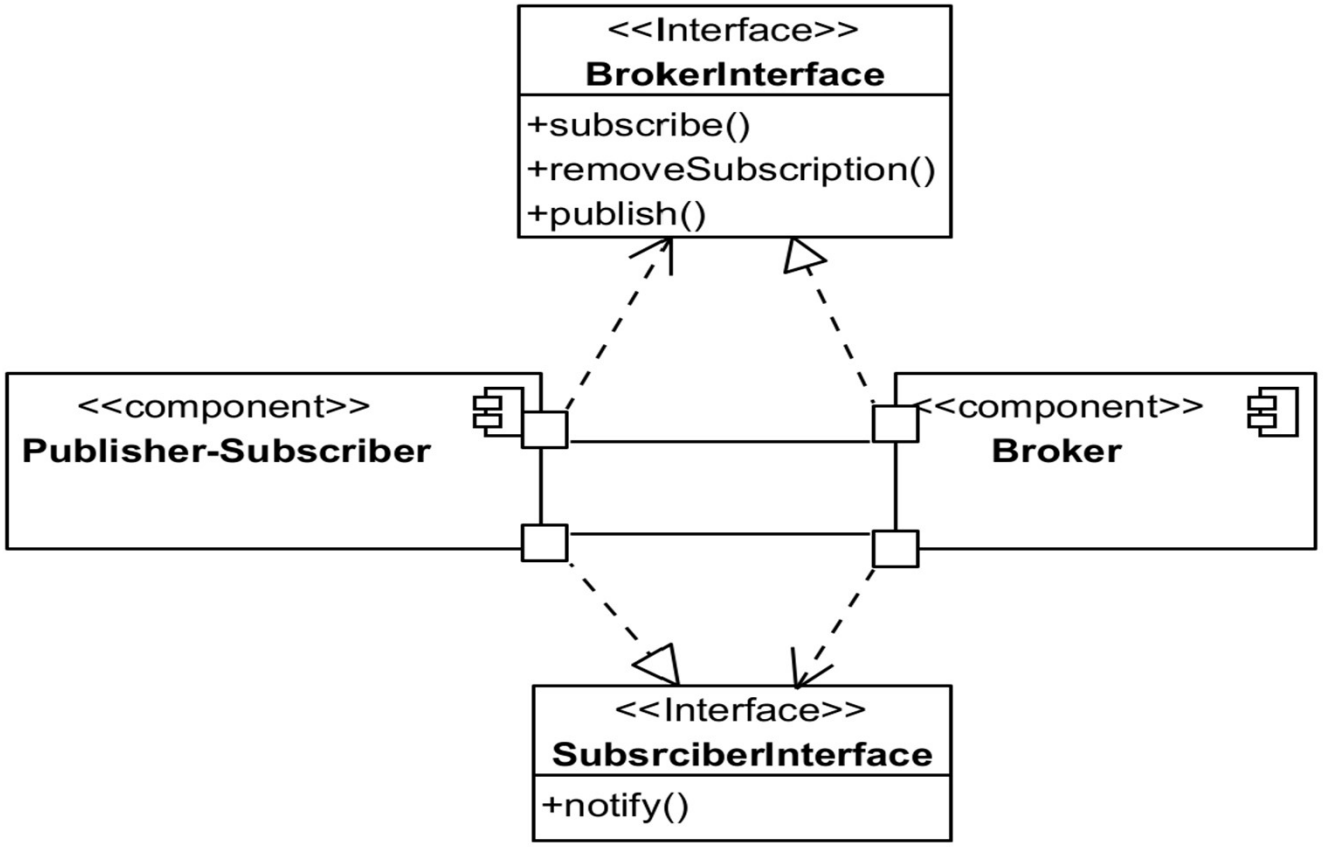
Stile Publish-Subscribe

- I componenti interagiscono annunciando eventi:
  - Un 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

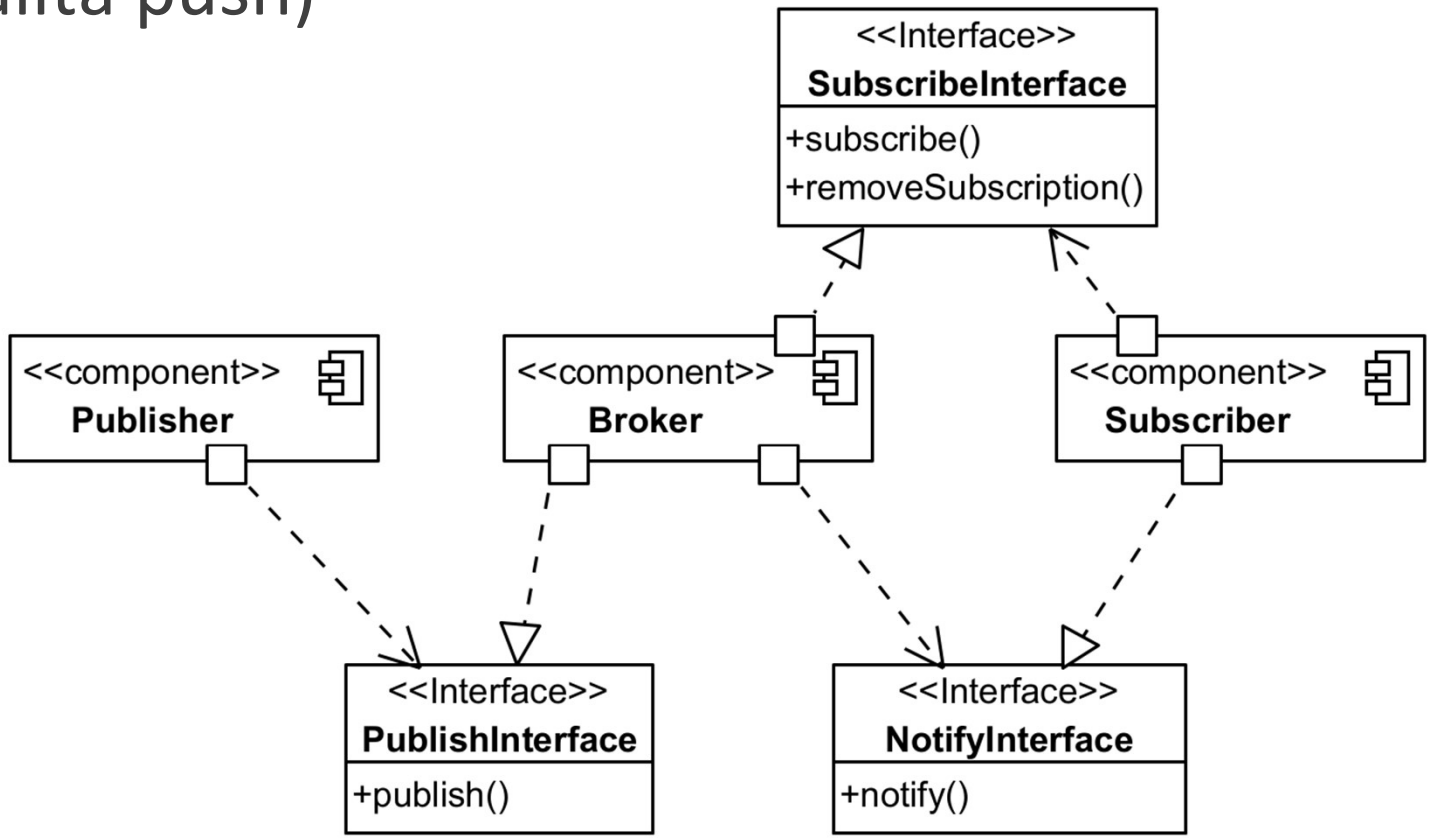
Stile Publish-Subscribe

- Un componente può essere in generale sia publisher che subscriber
- Due diversi connettori, uno per le richieste di sottoscrizione e per richieste di pubblicazione, uno per diffondere i dati



Stile Publish-Subscribe

Publisher e subscriber possono essere componenti distinte (Notify in modalità push)

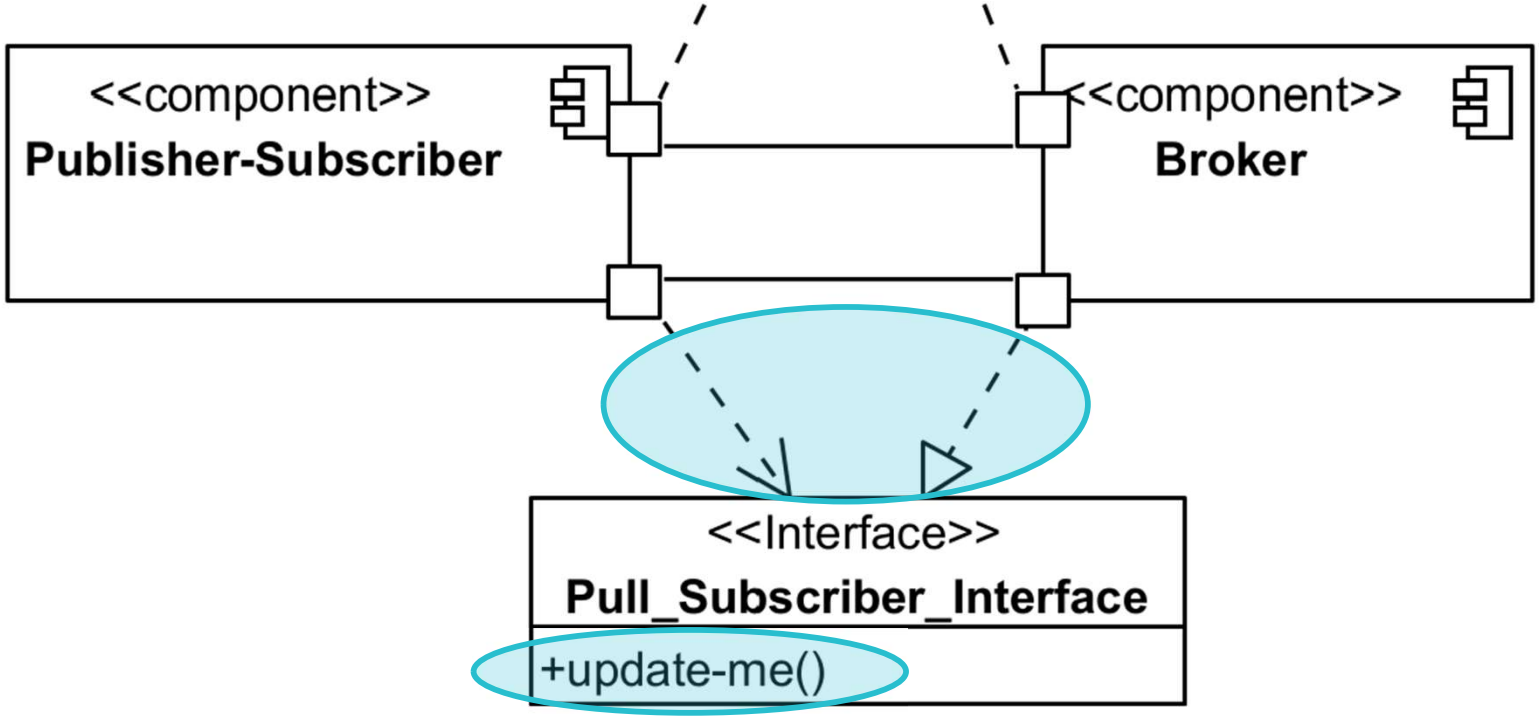


Stile Publish-Subscribe

- In questo stile, mittenti e destinatari di messaggi dialogano attraverso un tramite, che 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" (to publish) il proprio messaggio al dispatcher.
- I destinatari (subsribers) si rivolgono a loro volta al dispatcher "abbonandosi" (to subscribe) alla ricezione di determinati tipi di messaggi.
  - 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.
- Il dispatcher inoltra ogni messaggio ricevuto da un publisher a tutti i subscriber interessati ai messaggi di quel tipo.
- Questo schema implica che ai publisher non sia noto quanti e quali sono i subscriber e viceversa. Questo può contribuire alla scalabilità del sistema.

Stile Publish-Subscribe modalità pull

La diffusione può avvenire anche in modo pull



Modello push o pull ?

- Modello push
  - Il broker invia attivamente i messaggi ai consumatori
  - È complicato per il broker trattare con diversi tipi di consumatori, in quanto controlla la frequenza con cui i dati vengono trasferiti
  - Deve decidere se inviare un messaggio immediatamente o se accumulare più dati e inviare
- Modello pull
  - Il consumatore si assume la responsabilità di recuperare i messaggi dal broker
  - Il consumatore deve mantenere un valore che identifica il prossimo messaggio successivo da trasmettere ed elaborare
  - Pro: migliore scalabilità (minore onere per i broker) e flessibilità (consumatori diversi con esigenze e capacità diverse)
  - Contro: nel caso in cui il broker non disponga di dati, i consumatori potrebbero essere occupati in attesa dell'arrivo dei dati

Publish-Subscribe: middleware

- Reti: comunicazioni multicast con algoritmi di flooding

- **Advanced Message Queuing Protocol (AMQP)** è un protocollo che realizza sia comunicazioni punto-a-punto che publish-and-subscribe
  - **RabbitMQ:** RabbitMQ è un middleware di messaggistica open-source basato su AMQP. Supporta la pubblicazione e la sottoscrizione di messaggi ed è ampiamente utilizzato per l'implementazione di architetture publish-subscribe.

- **Message Queuing Telemetry Transport (MQTT):** MQTT è un protocollo ISO standard di messaggistica leggero, posizionato in cima a TCP/IP.

- **Apache Kafka:** Kafka è una piattaforma open-source di streaming distribuita che offre supporto per il modello publish-subscribe. È comunemente utilizzato per l'elaborazione di eventi in tempo reale e la trasmissione di dati tra applicazioni.

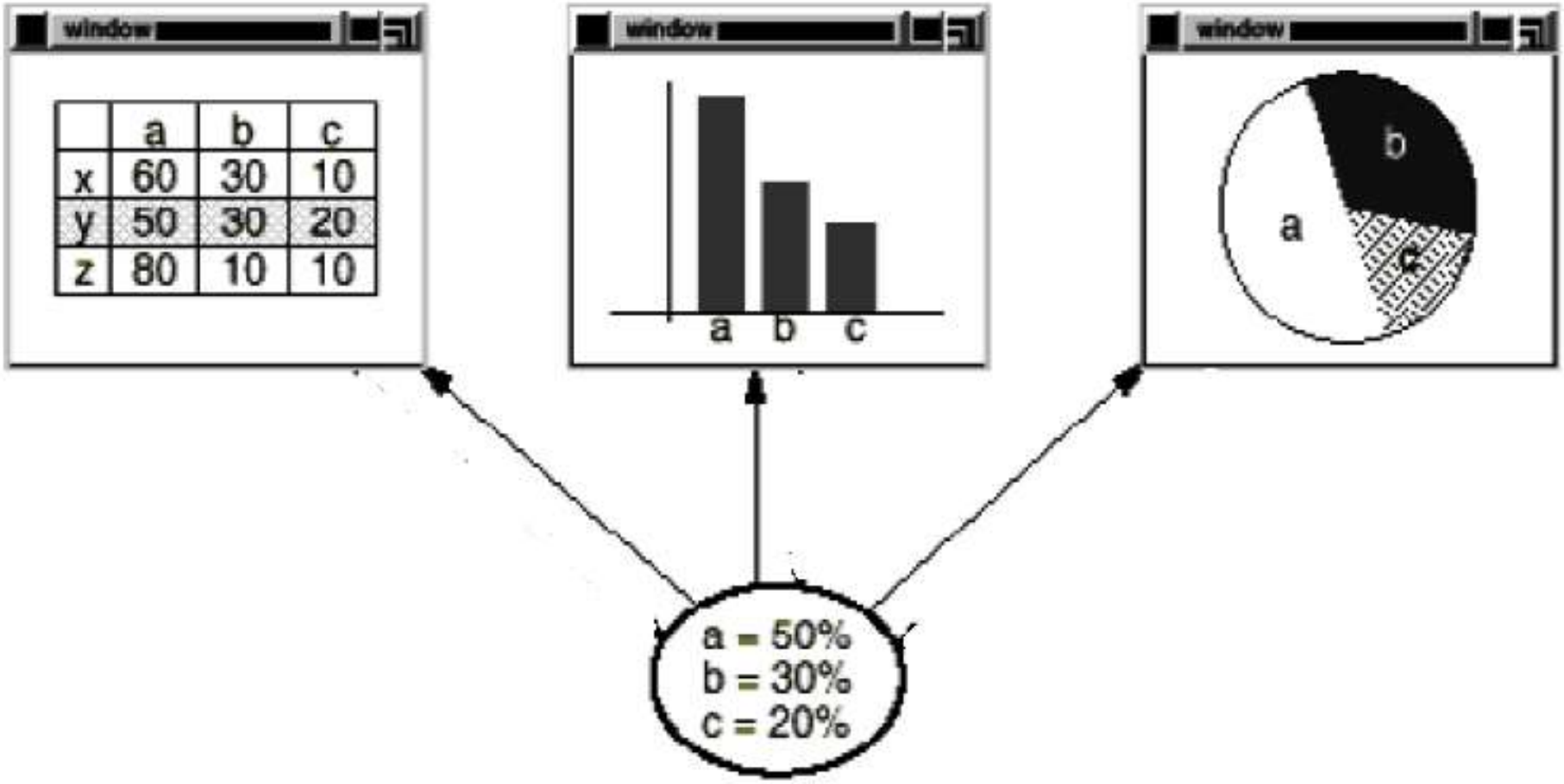
Publish-Subscribe: DDS

- Data Distribution Service (DDS) è un middleware standardizzato basato sul paradigma publish-subscribe, che aiuta lo sviluppo di livelli middleware per la comunicazione machine-to-machine.
  - E’ parte integrante di molti sistemi embedded e di applicazioni con requisiti in tempo reale.
  - Mantenuto dall'Object Management Group (OMG),7 DDS è utilizzato in tutte le classi di applicazioni critiche per implementare un livello di comunicazione affidabile tra sensori, controllori e attuatori.
  - Usato, per esempio da
    - NASA
    - Siemens per gli impianti eolici
    - Volkswagen e Bosch per i sistemi di parcheggio autonomo
  - Dal punto di vista del programmatore, DDS è una application programming interface (API). Supporta serializzazione e deserializzazione di qualsiasi tipo di dati built-in o custom attraverso un linguaggio di definizione dell'interfaccia (IDL) dedicato.

Stile 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
  - Il Modello fornisce il nucleo funzionale di un'applicazione, ed è la rappresentazione del modello 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: notifica alle viste la modifica dei dati; le viste recuperano le informazioni (e le mostrano all'utente).
- 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. I controllori traducono gli eventi in richieste per eseguire operazioni sugli elementi del Modello.

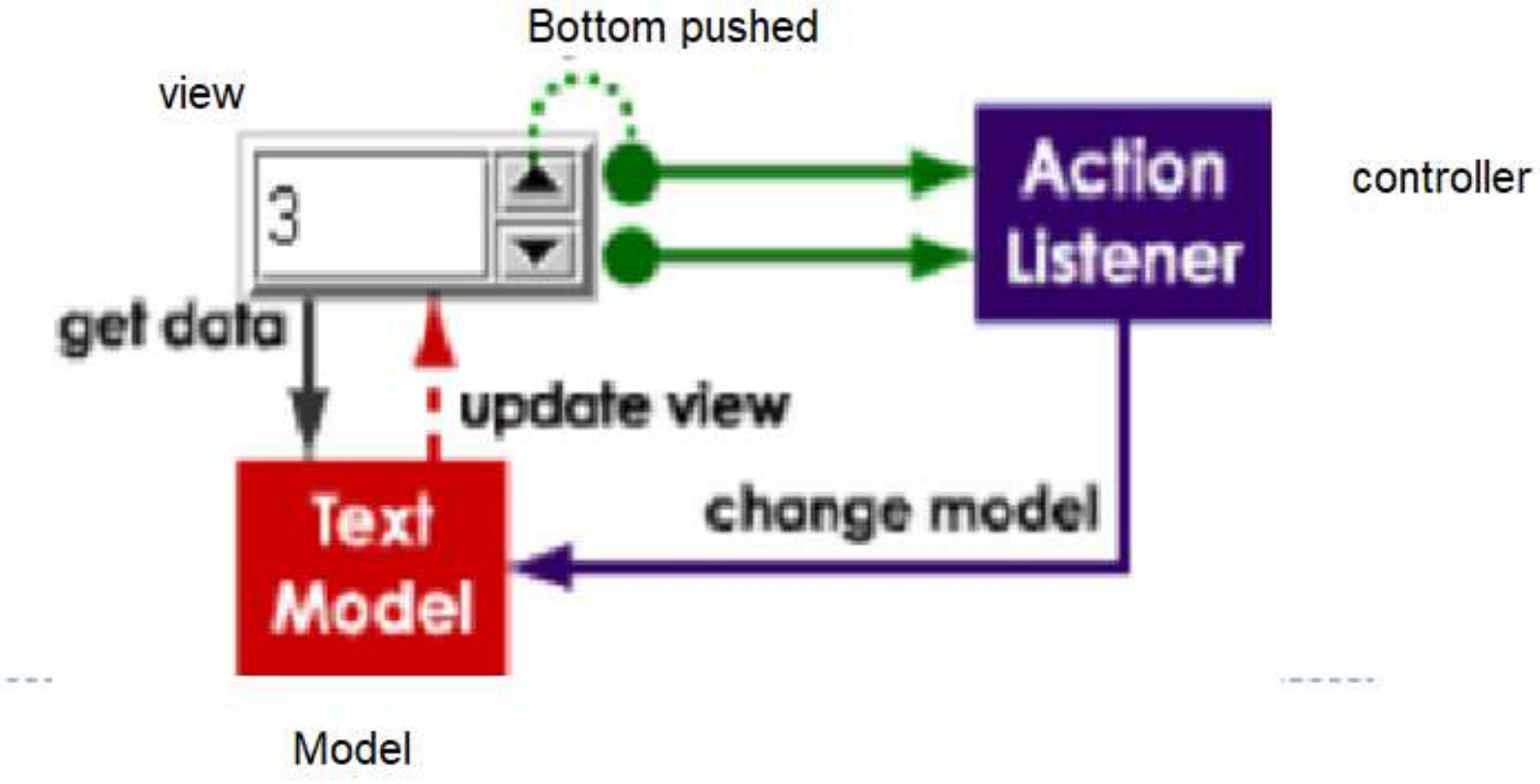
Un modello, diverse visualizzazioni



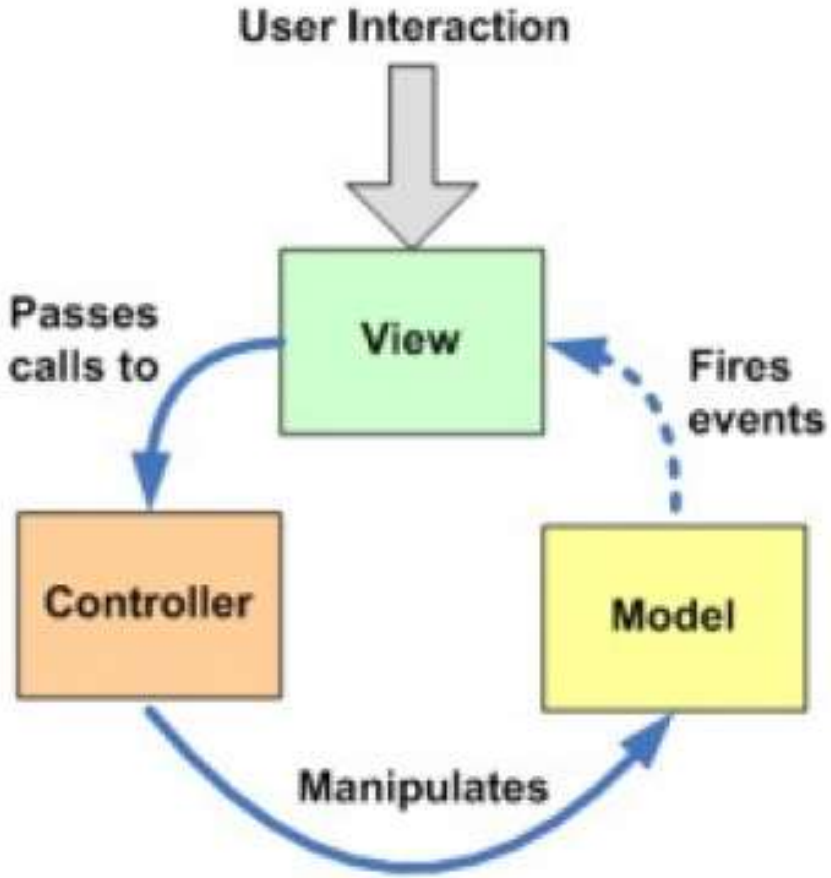
Model-View-Controller

- 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.

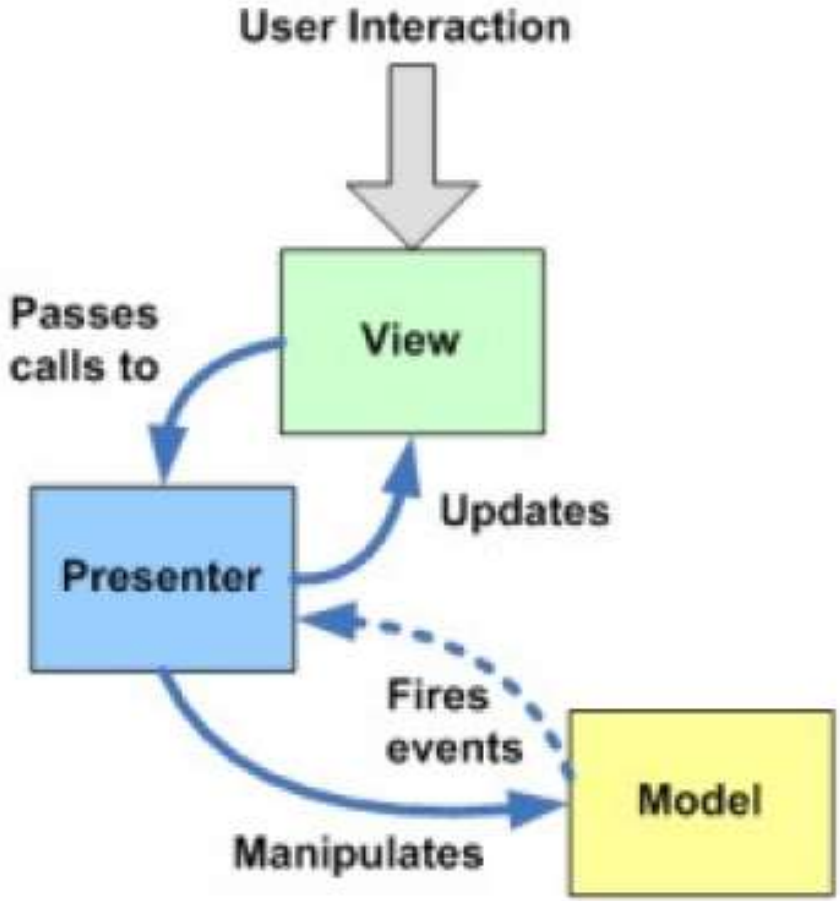
Model-View-Controller: esempio



Model-View-Controller vs Model-View-Presenter



Model-View-Controller



Model-View-Presenter

Stile: coordinatore di processi

- E’ uno stile di progettazione software che prevede l'uso di un componente dedicato noto come "coordinatore" o "process coordinator",
- Il Coordinatore 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

Homework

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

Viste di tipo strutturale

Viste di tipo strutturale

- Gli elementi (nodi del diagramma)
  - **Moduli**: unità di software che realizzano un insieme coerente di responsabilità
    - Esempio: classi, collezioni di classi, package.

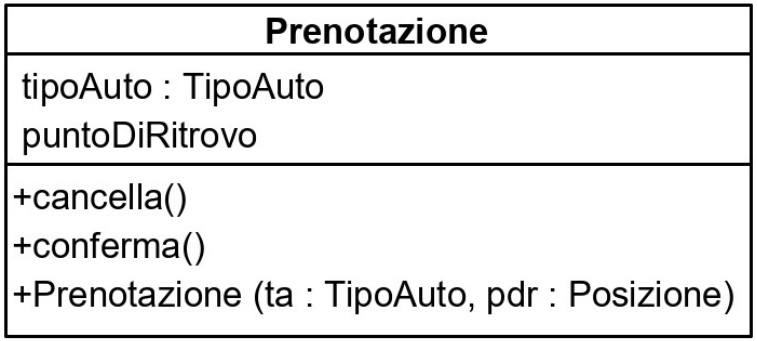
- Relazioni tra elementi
  - **parte di, eredita da, dipende da, può usare**

Viste di tipo strutturale

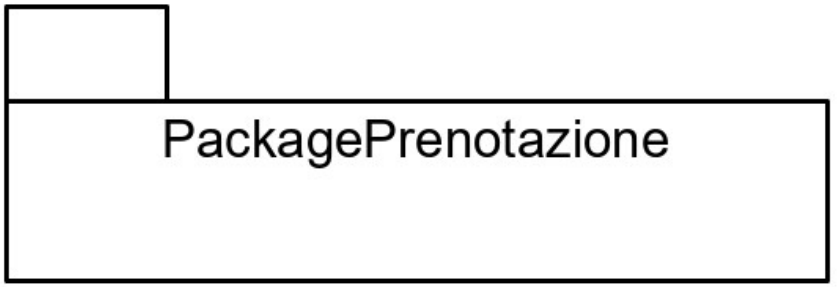
- A cosa servono
  - costruzione
    - la vista può fornire lo schema del codice e directory e file sorgente hanno una struttura corrispondente
  - analisi
    - tracciabilità dei requisiti
    - analisi d'impatto per valutare eventuali modifiche
  - comunicazione
    - se la vista è gerarchica, offre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

UML per documentare la vista strutturale

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



- packages



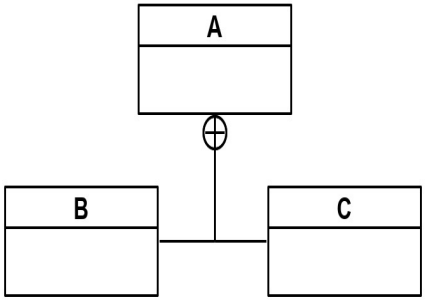
- relazioni tra classi, tra packages e tra classi e packages:
  - contenimento, dipendenze, generalizzazione….

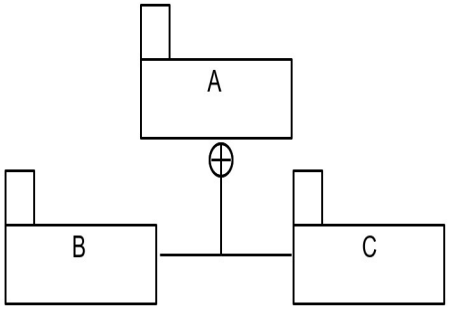
Vista strutturale di decomposizione

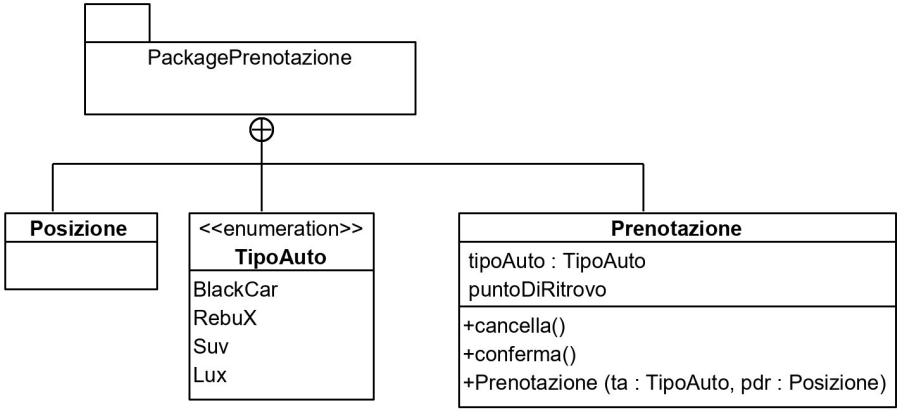
- Relazione: “parte di”
  - una classe fa parte di (è contenuta in) un package,
  - un package fa parte di uno più grande
- Criteri per raggruppare
  - incapsulamento per modificabilità
  - supporto alle scelte costruisci/compra
  - moduli comuni in linee di prodotto
- A cosa serve questa vista
  - apprendimento del sistema
  - punto di partenza per l’allocazione del lavoro

Notazione UML per documentare la vista di decomposizione

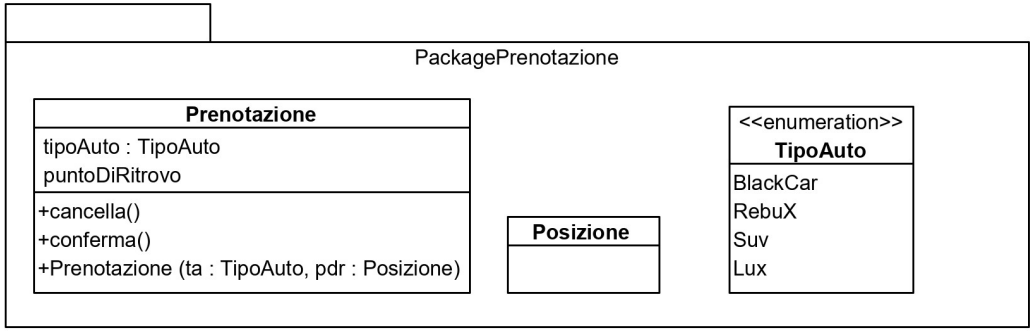
La relazione «parte di» è resa con ⊕







oppure con inclusione grafica (in un package)



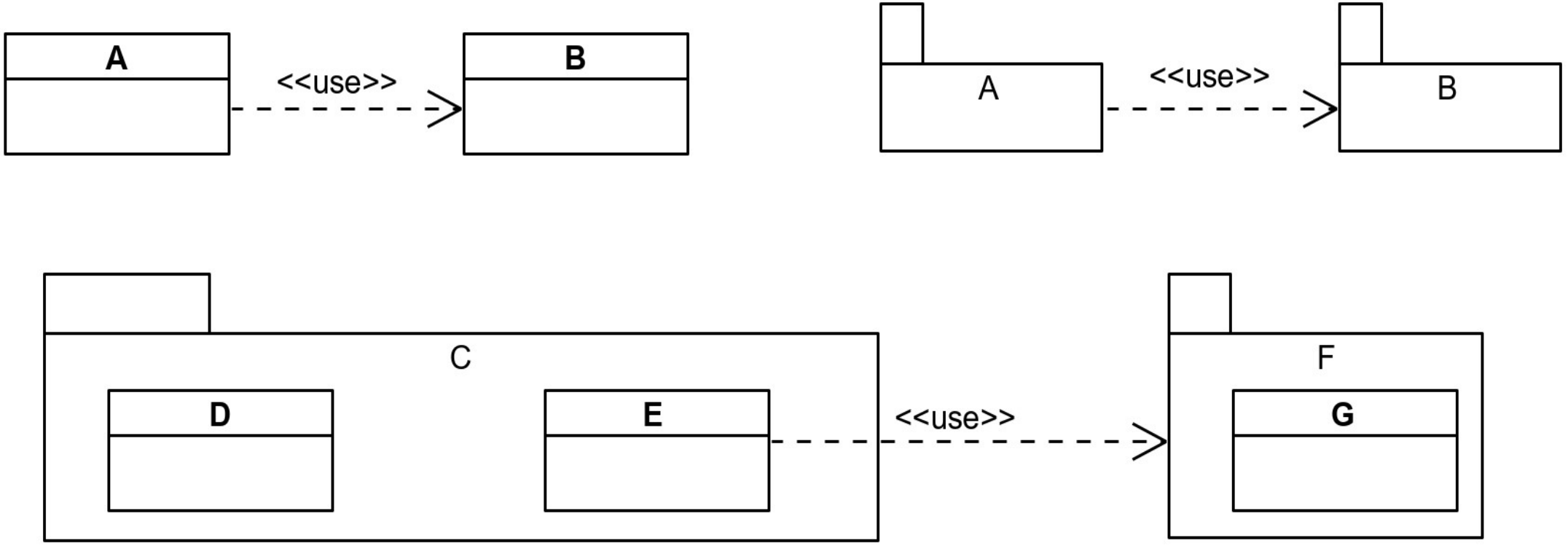
Vista strutturale d'uso

- 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 documentare la vista d’uso

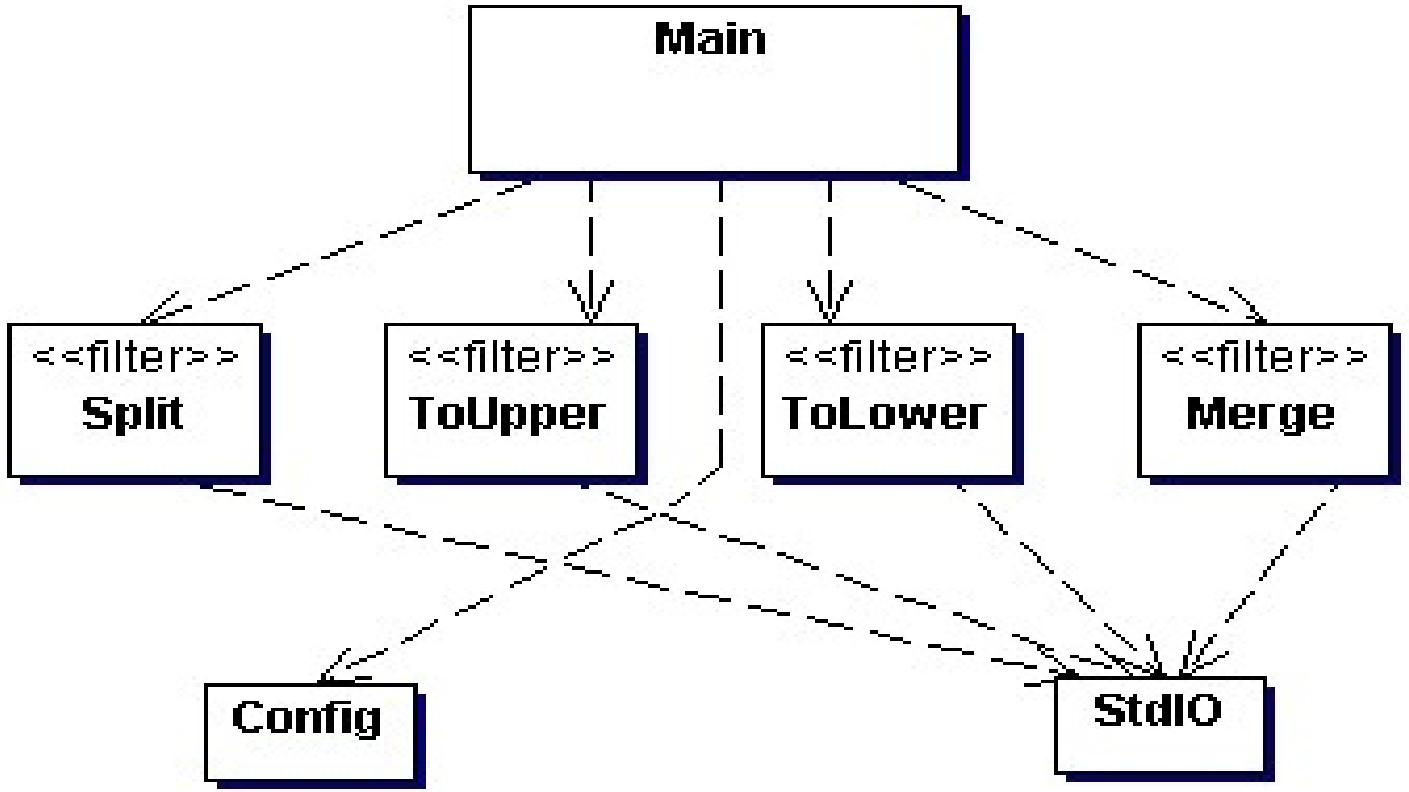
Usando una dipendenza etichettata con lo stereotipo «use»



Vista strutturale d'uso: esempio (Abbiamo già descritto la vista C&C)

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

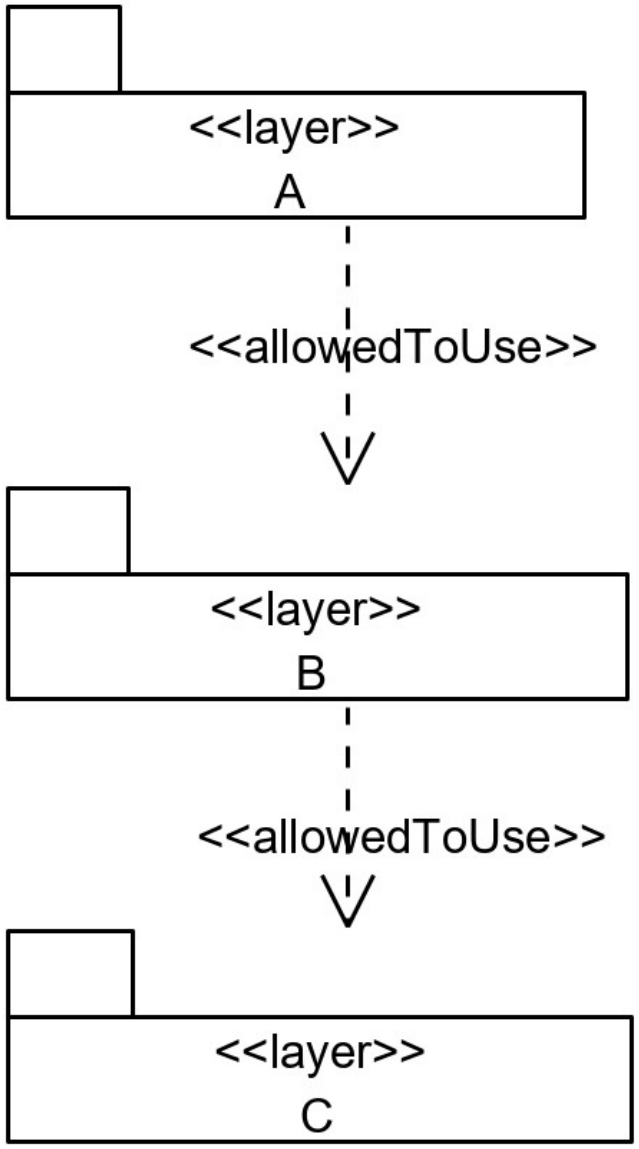
Vista degli usi
(tutte le dipendenze
sono <<uses>>)



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
    - non implicitamente transitiva
- A cosa serve
  - modificabilità e portabilità
  - controllo della complessità

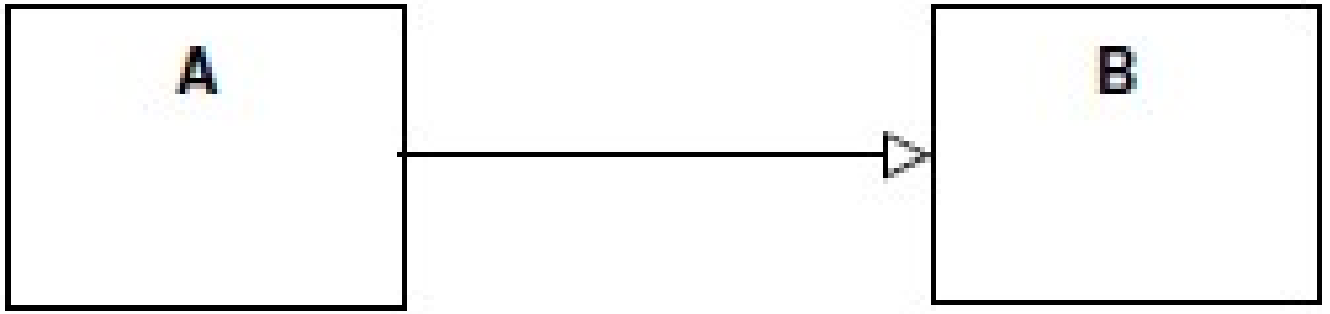
Notazione UML per documentare la vista a strati



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 documentare la vista di generalizzazione



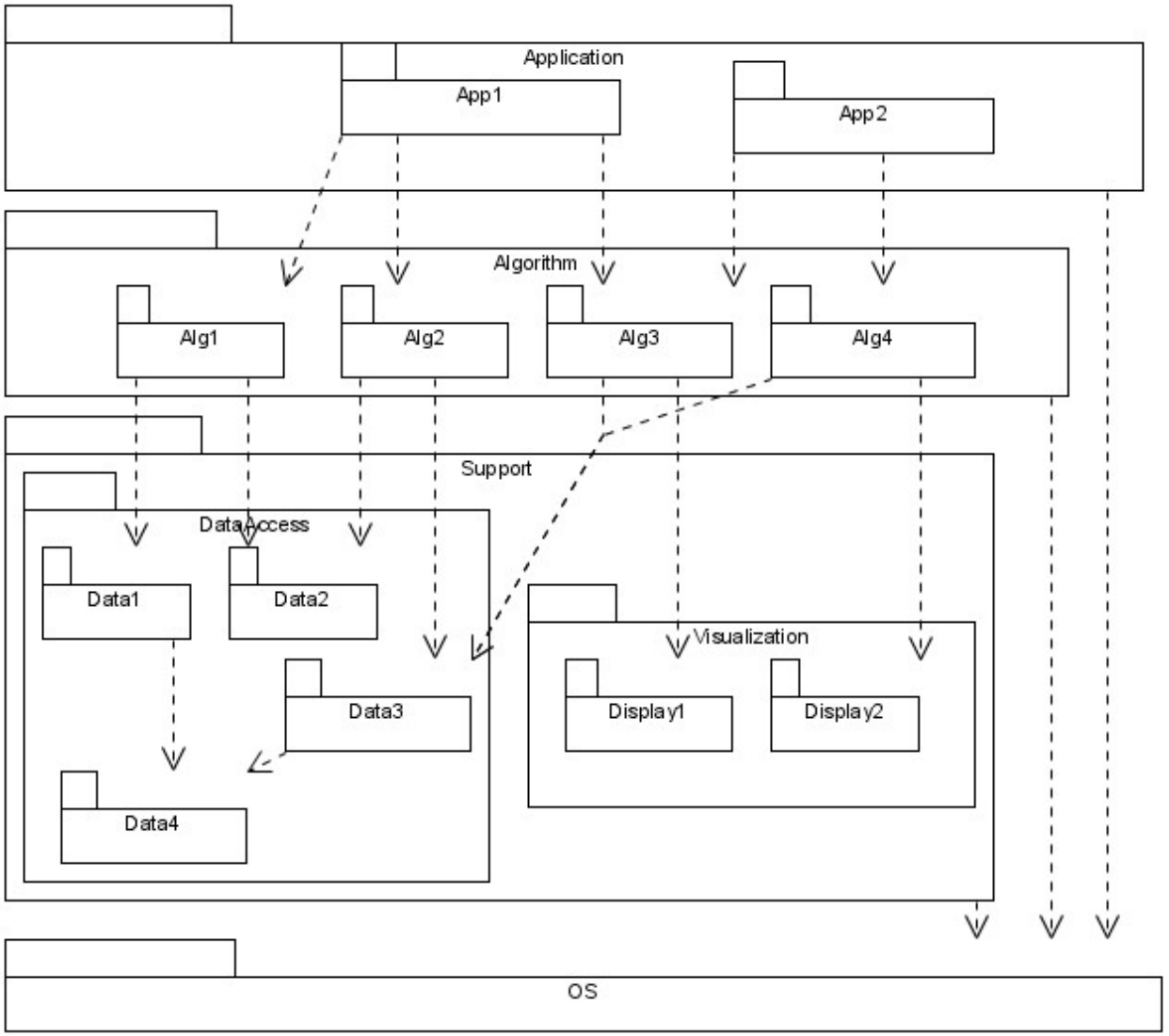
Esempio

Module	Module Type	Used 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

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



Vista (logistica) di dislocazione (deployment)

Vista di dislocazione (deployment)

- Elementi
  - Software: artefatti
    - 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.
  - 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

Notazione UML per documentare la vista di dislocazione

- Parallelepipedo: denota un nodo hardware o, in generale, un ambiente di esecuzione.
- Artefatti (sembrano classi con stereotipo «artifact»)
- Le relazioni tra nodi hw sono connessioni fisiche o protocolli di comunicazione.

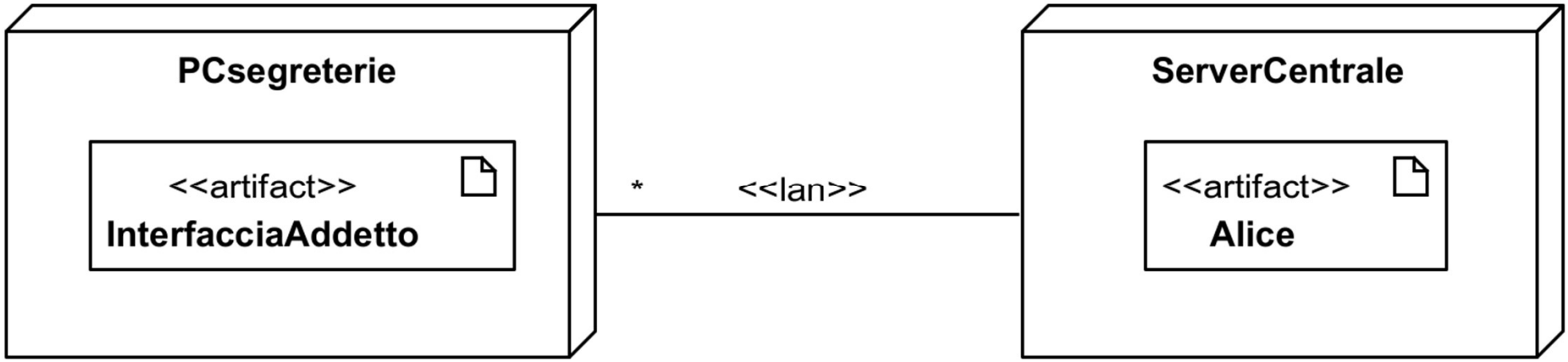
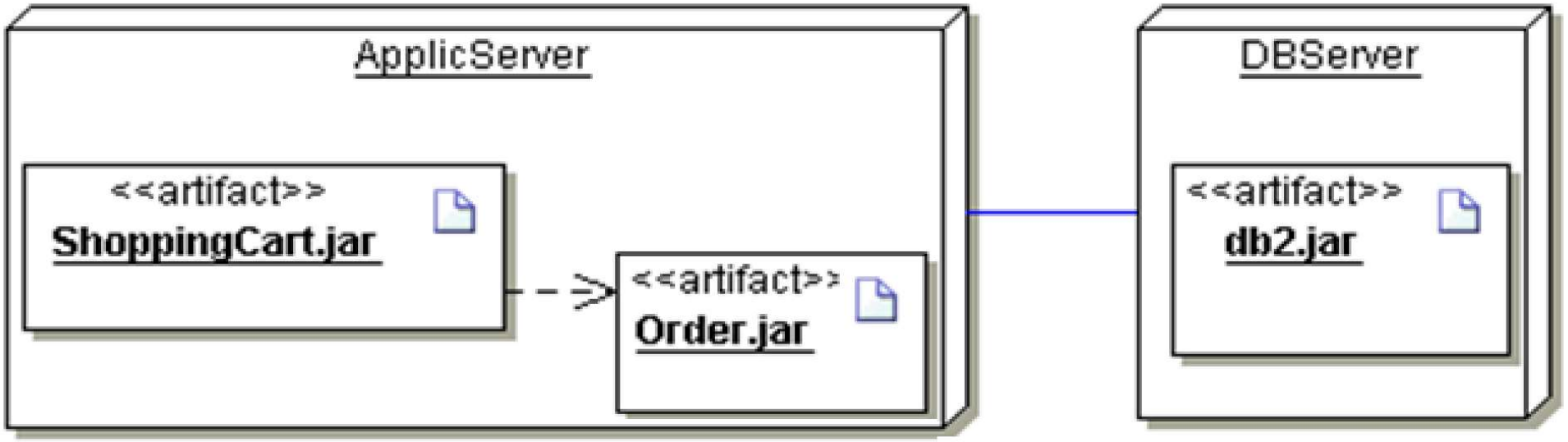
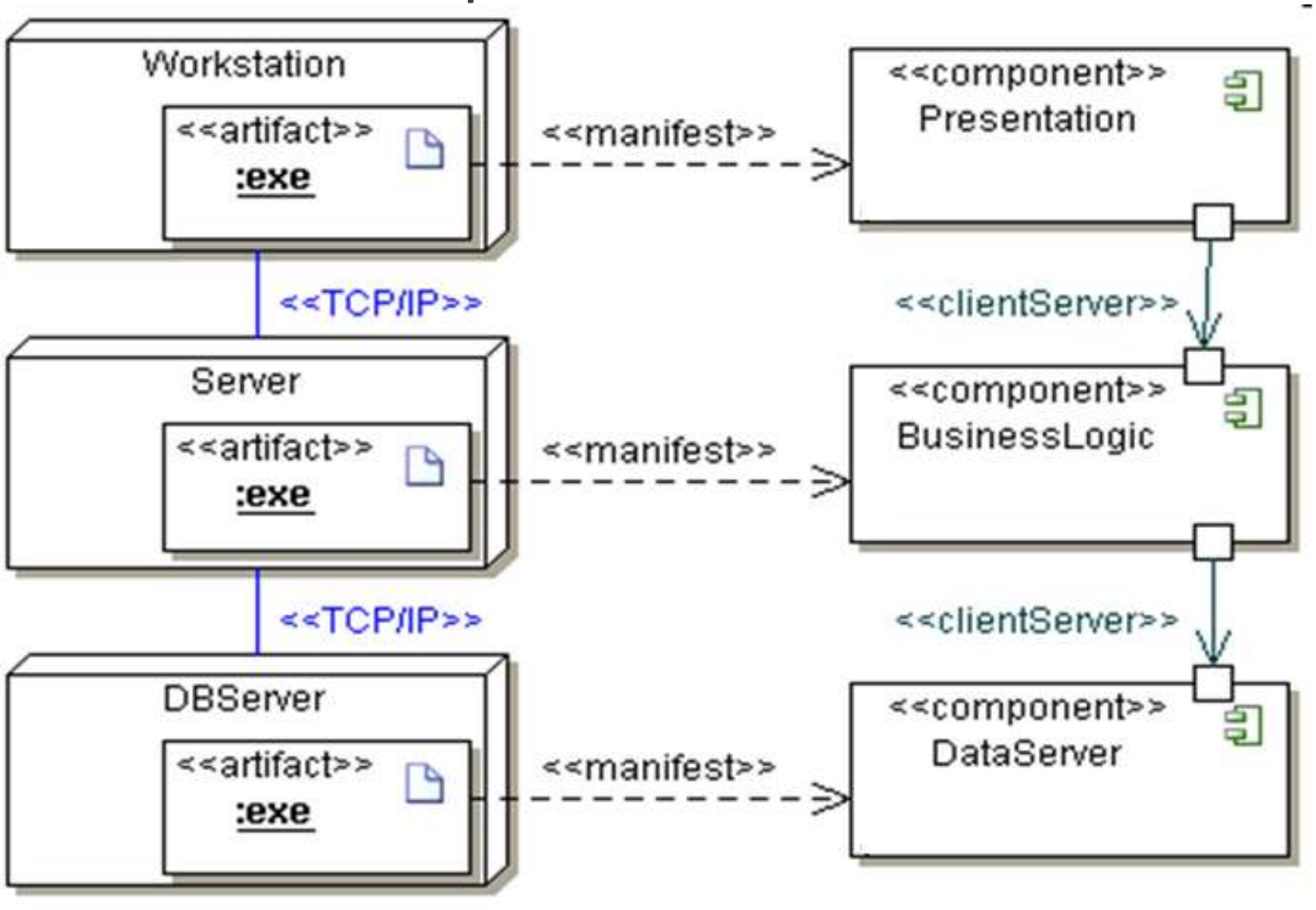


Diagramma di deployment a livello di istanza: esempio



Vista ibrida C&C e deployment

Illustrata con un esempio di architettura 3-tier



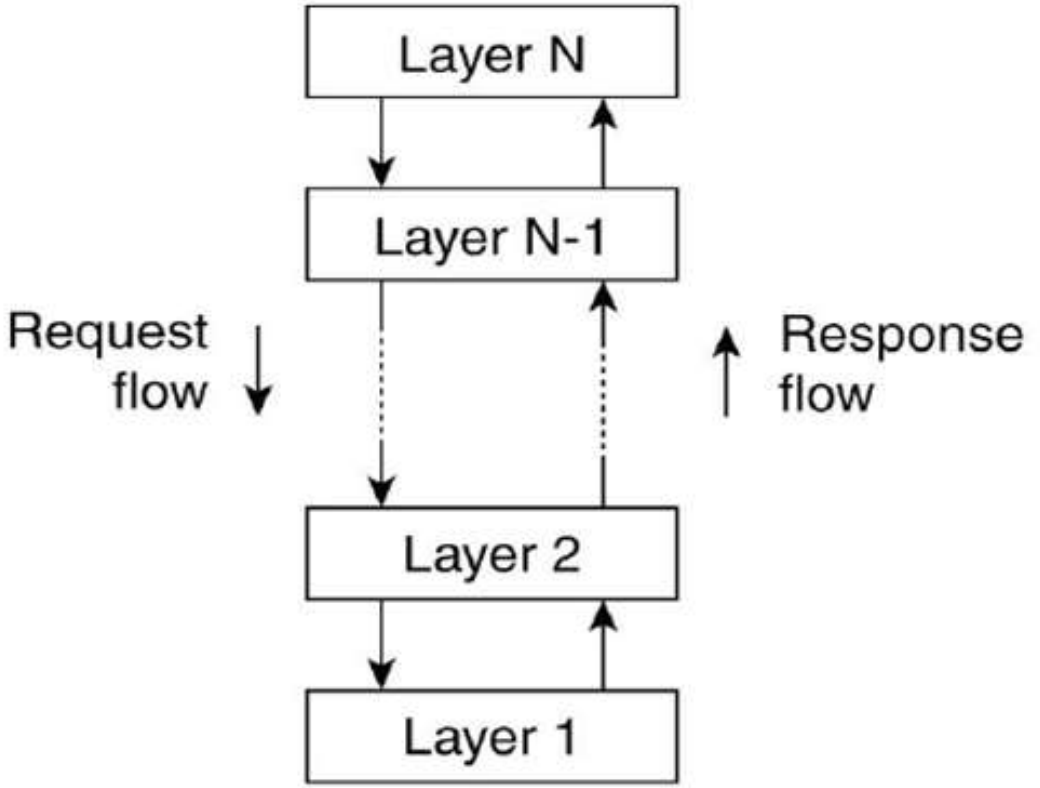
"Deployment" di componenti

- Normalmente 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 (in questo caso anche chiamato componente installato) «manifesta» un componente
- Un componente (che esiste a run time) è un'istanza creata a partire da un artefatto in un ambiente di esecuzione

Esempi

"architettura a livelli" (esempio da web)

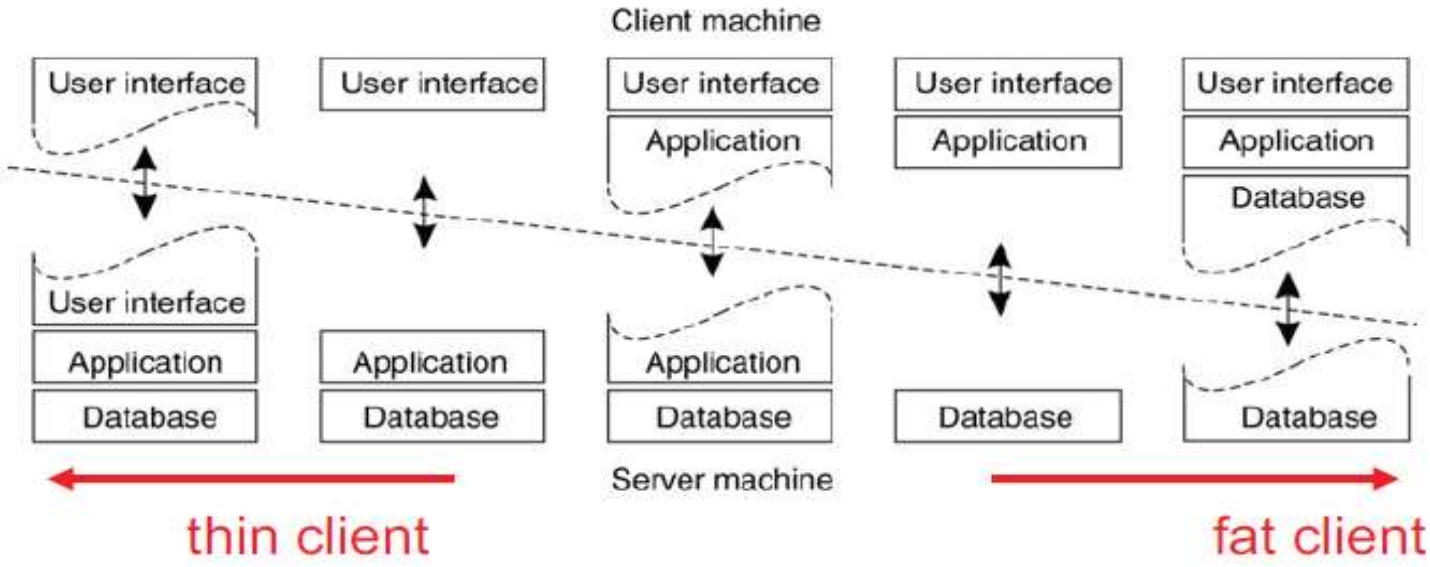
- Componenti organizzati in livelli (*layer*)
- Un componente a livello *i* può invocare un componente del livello sottostante *i-1*
- Le richieste scendono lungo la gerarchia, mentre le risposte risalgono



- In Realtà è un concetto ambiguo
  - Vista C&C: catene di client-server
  - Vista Strutturale: Layers

Architetture multilivello (esempio da web)

- 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

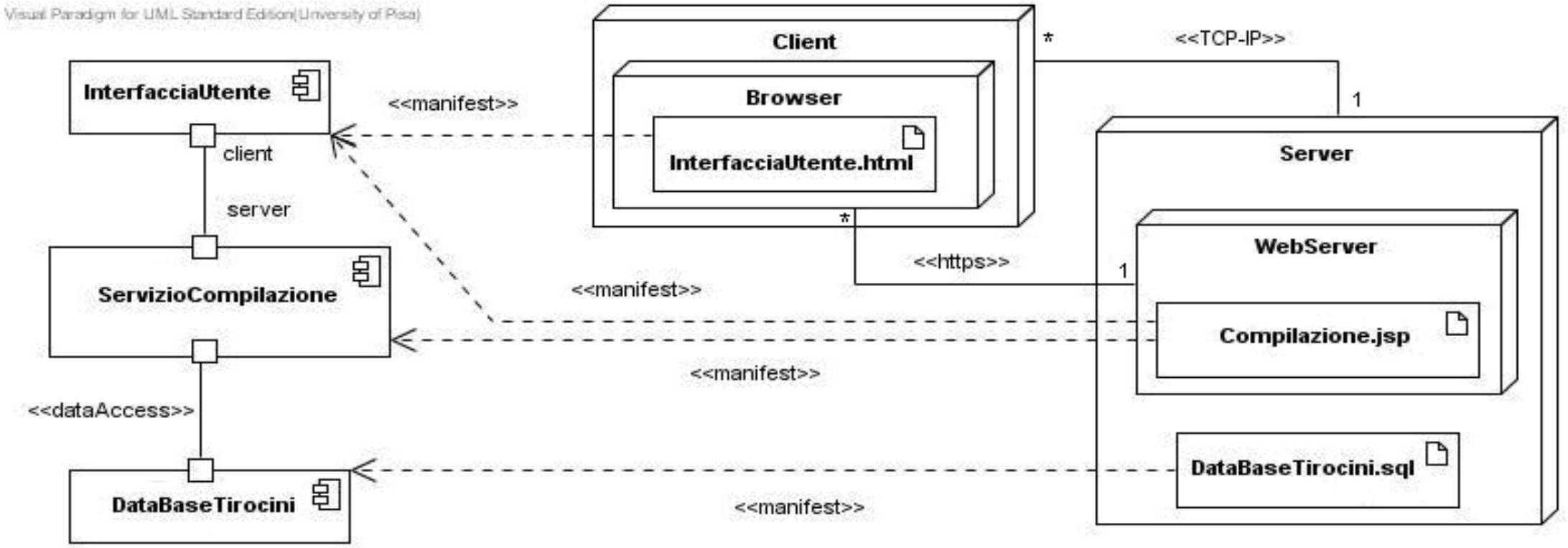


Architetture multilivello

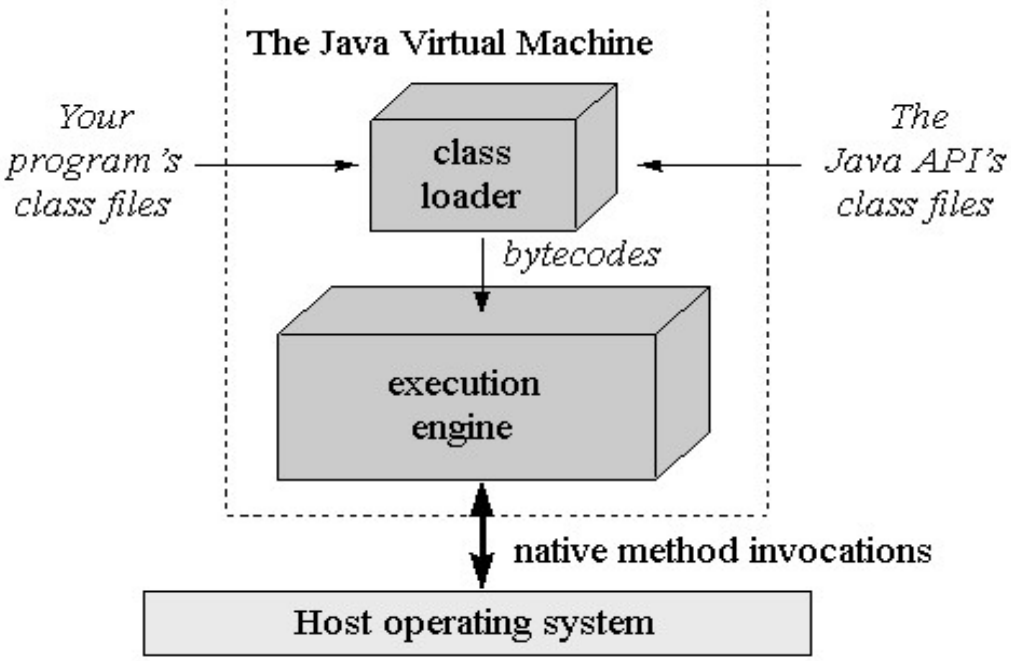
- Da un livello ad N livelli
- Con l'introduzione di ciascun livello
  - L'architettura guadagna in flessibilità, funzionalità e possibilità di distribuzione
- Ma l'architettura multilivello potrebbe introdurre un problema di prestazioni
  - Aumenta il costo della comunicazione
  - Viene introdotta più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

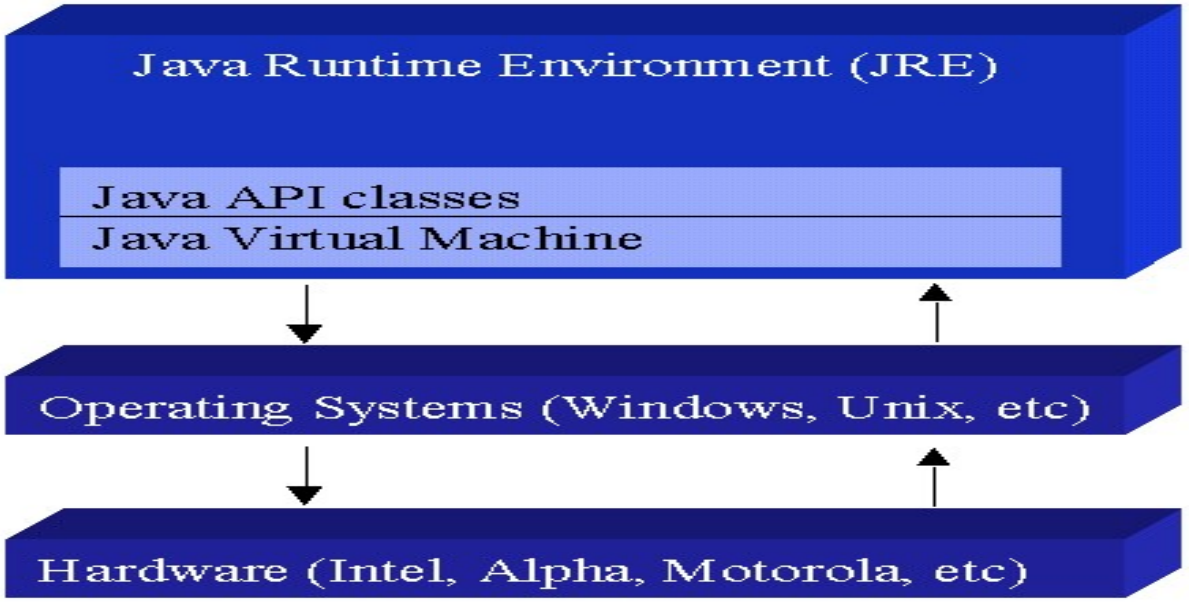
Esempio con 3 layers, 2 tiers (servizio per la gestione dei tirocini)

Vista ibrida (C&C e dislocazione) dell'architettura del sotto-sistema di Compilazione, assumendo che gli artefatti che manifestano le componenti citate siano: Compilazione.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.



Esercizio:
quali viste per descrivere queste architetture?





- Componente (JVM)
  - Sotto-Componente (loader) +
  - Sotto-Componente (engine)
- Ambiente di esecuzione (SO)

- Codice (API classes) +
- Componente (JVM) +
- Ambienti di esecuzione (JRE, SO, HW)

Syllabus

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

Approfondimenti

- 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.