11. Progettazione: Architetture SW

IS 2024-2025

Laura Semini, Jacopo Soldani
Corso di Laurea in Informatica
Dipartimento di Informatica, Università of Pisa

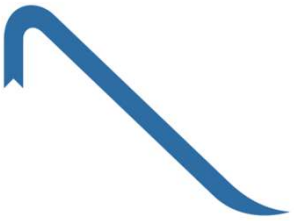
PROGETTARE PRIMA DI PRODURRE

«The architect's two most important tools are: the eraser in the drafting room and the wrecking bar on the site» (Frank Lloyd Wright)

Vale anche per il software (il codice è «più duro» dei modelli)

Tipico della produzione industriale, per

- complessità dei prodotti
- organizzazione e riduzione delle responsabilità
- controllo preventivo della qualità



PROGETTAZIONE

Costituisce la fase «**ponte**» fra la specifica e la codifica

Si passa da

- **che cosa** deve essere fatto a
- **come** deve essere fatto

Il risultato della progettazione si chiama **architettura** (o progetto) del software

LIVELLO DI ASTRAZIONE/DETTAGLIO

Progetto di alto livello (o **architetturale**)

- scompone un sistema complesso in più **sottosistemi**
- identifica e specifica le **parti del sistema** e le loro **inter-connessioni**

Progettazione di **dettaglio**

- indica come la specifica di ogni parte sarà realizzata

ARCHITETTURA SOFTWARE

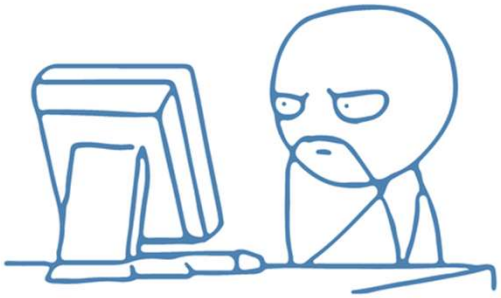
L'architettura di un sistema software (in breve, **architettura software**) è la **struttura del sistema**, costituita dalle **parti** del sistema, dalle **relazioni** tra le parti e dalle loro **proprietà visibili**

In altre parole, un'architettura software

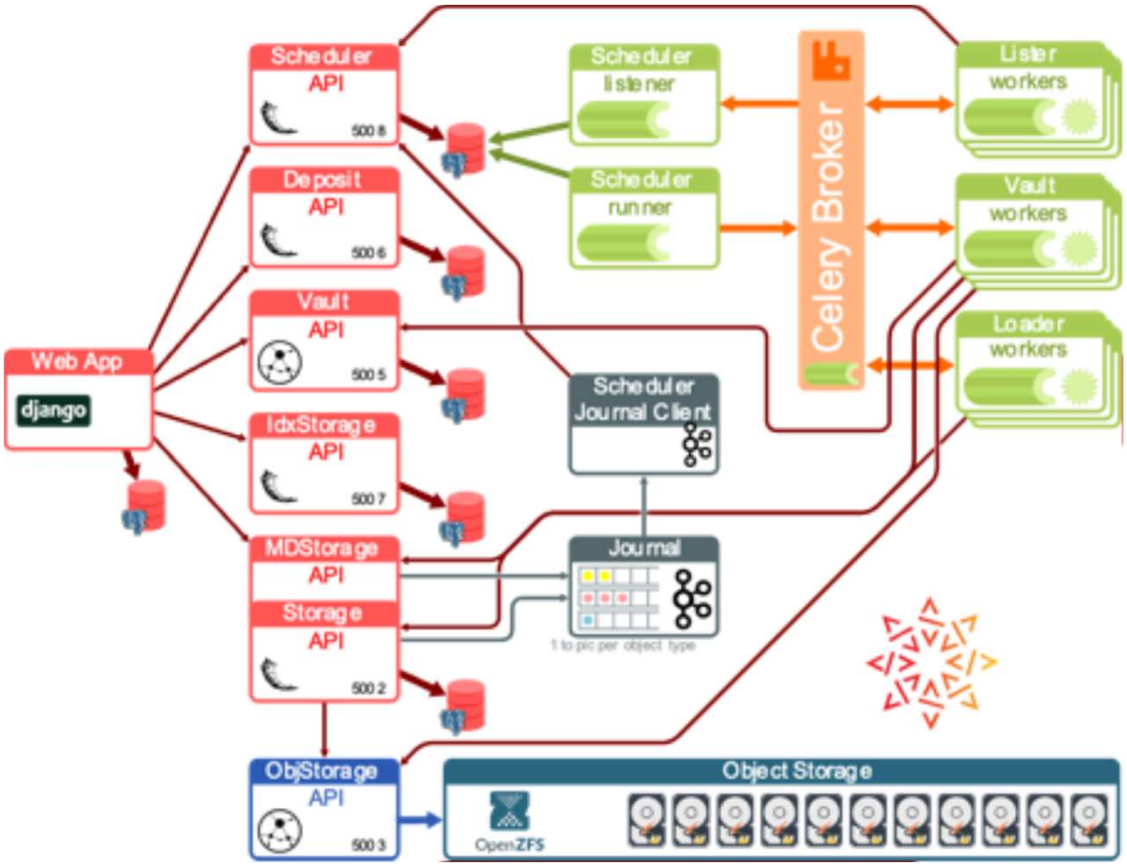
- definisce la struttura del sistema software
- specifica le comunicazioni tra componenti
- considera aspetti funzionali e non funzionali

Fornisce un'**astrazione** del sistema mediante un **artefatto complesso**

Come si rappresenta?

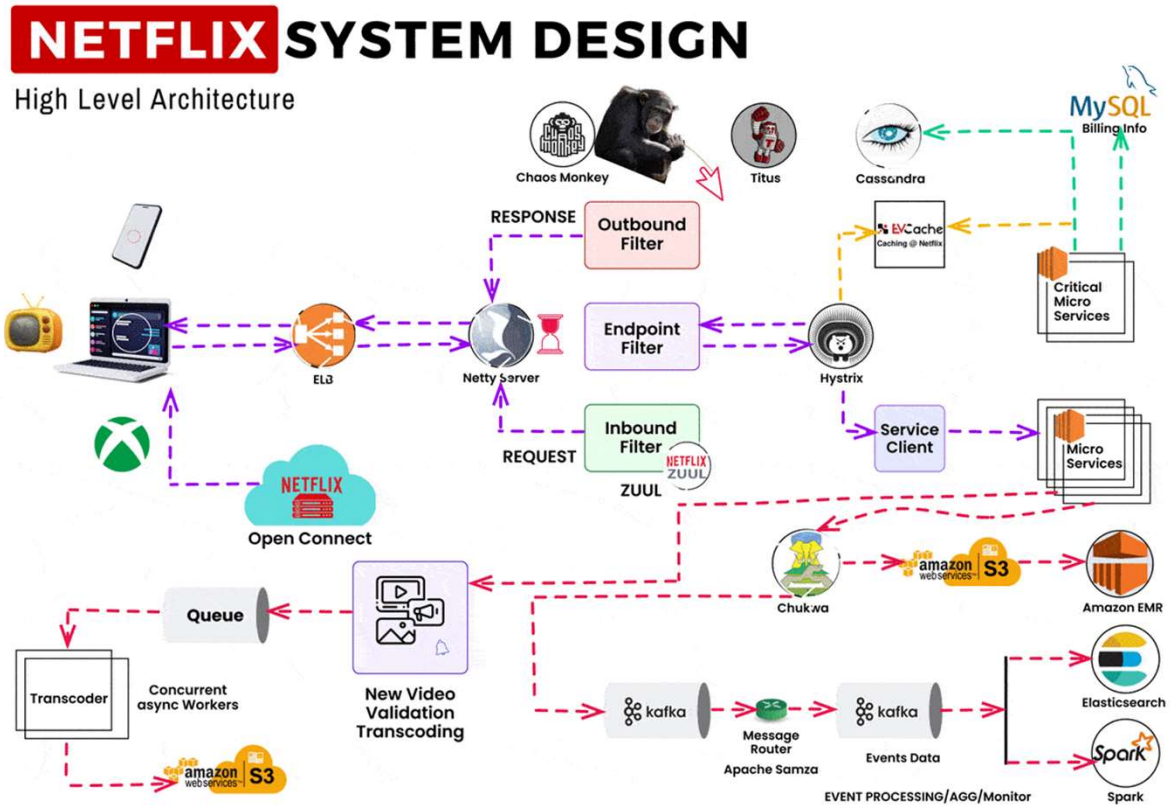


ESEMPI DI ARCHITETTURE SOFTWARE



https://docs.softwareheritage.org/devel/architecture/overview.html

ESEMPI DI ARCHITETTURE SOFTWARE (CONT.)



https://medium.com/@saddy.devs/netflix-architecture-72bb8572a102

UN'ANALOGIA CON L'INGEGNERIA EDILE

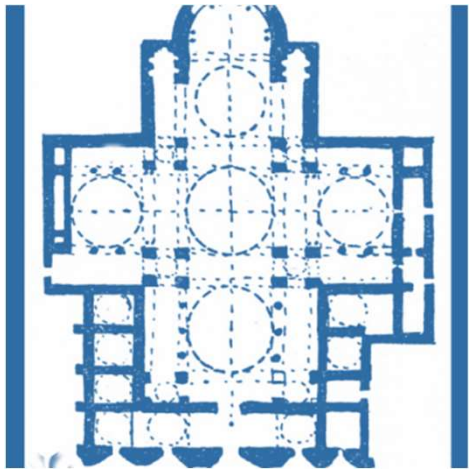
Il **disegno di progetto** rappresenta l'architettura immaginata dal progettista

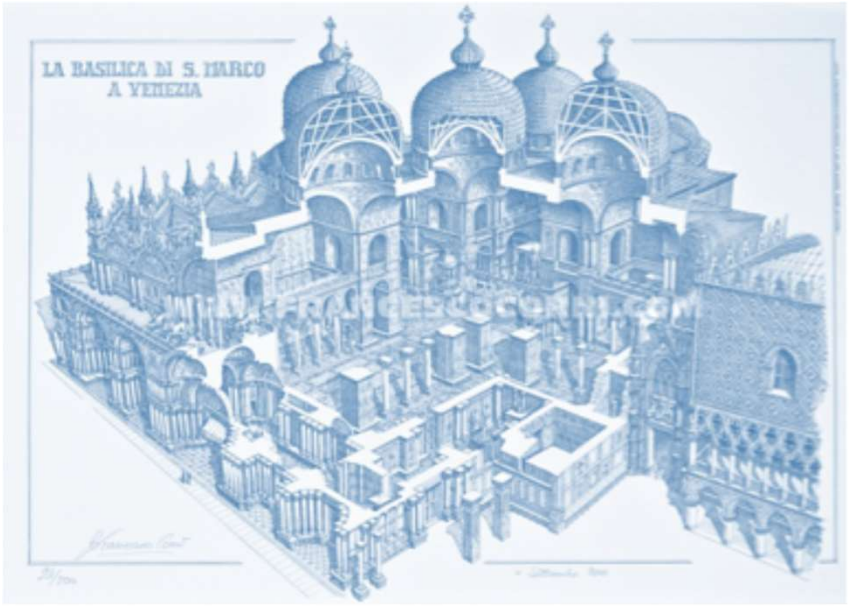
- linguaggio grafico **standard**
- diversi **punti di vista** (piante, prospetti, sezioni)

pianta
vista dall'alto
proiettata su piano xy

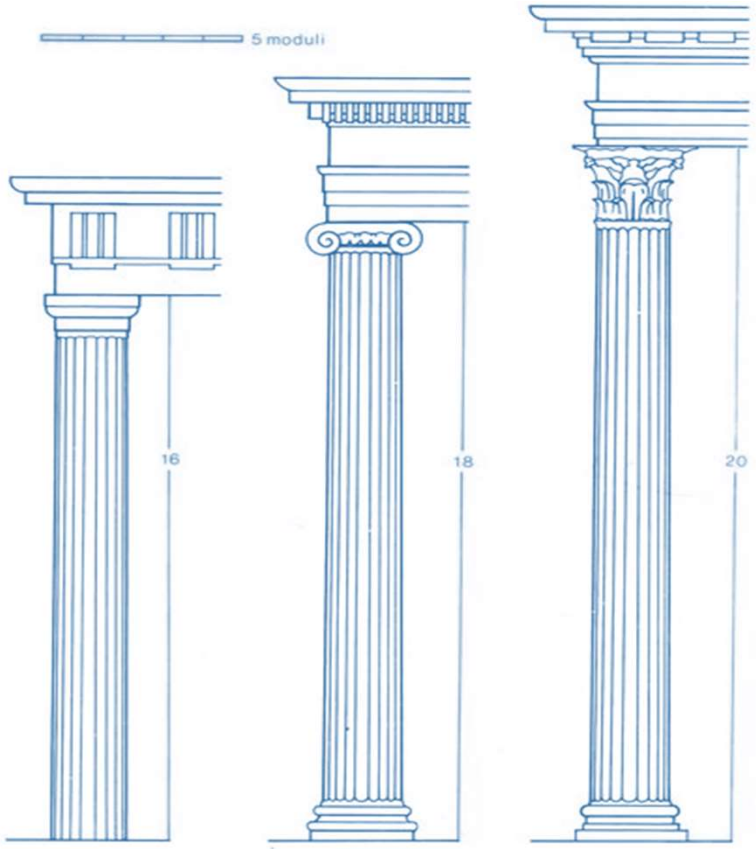
prospetto
vista frontale
proiettata su piano xz (o yz)

DIVERSE VISTE: PIANTA, PROSPETTO, SPACCATO





STESSA VISTA, MA STILI DIVERSI



La costruzione si basa su **stili** noti

- Scelta dello stile secondo diversi criteri
- In figura, considerazioni estetiche

TORNIAMO AL SOFTWARE

Viste e stili si applicano anche al software!

Tre **viste** interessanti (aka. tre astrazioni interessanti)

- vista **comportamentale**
- vista **strutturale**
- vista (**logistica**) di **deployment** (dislocazione)

(Gli stili li vediamo tra poco)

VISTA COMPORTAMENTALE

La vista comportamentale, aka. **component-and-connector** (C&C) descrive un sistema software come **composizione di componenti** software

- componenti e loro **interfacce**
- caratteristiche dei **connettori**
- struttura del sistema in esecuzione (flusso dei dati, dinamica, parallelismo, replicazioni, ecc.)

A cosa serve?

- Consente l'analisi delle **caratteristiche di qualità** a tempo d'esecuzione (prestazioni, affidabilità, disponibilità, sicurezza, ecc.)
- Consente di documentare lo **stile** dell'architettura

VISTA STRUTTURALE

La vista **strutturale** descrive la struttura di un sistema software come insieme di **unità di realizzazione**/codice (come classi e package, ad esempio)

A cosa serve?

- Analizzare **dipendenze** tra packages
- Progettare **test** di unità e di integrazione
- Valutare la **portabilità**

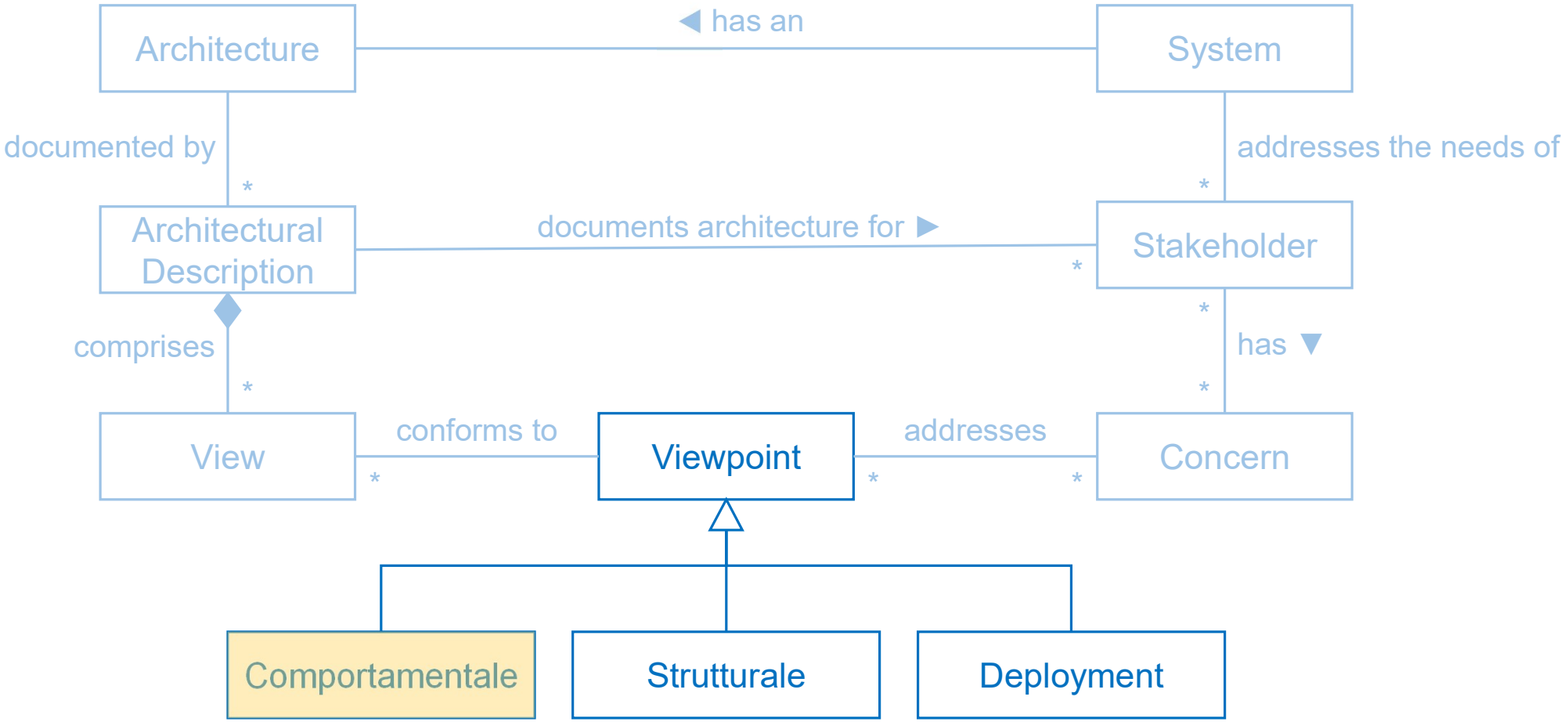
VISTA LOGISTICA (O DI DEPLOYMENT)

La vista di **deployment** descrive l'allocazione del software su ambienti di esecuzione

A cosa serve?

- Permette di valutare prestazioni e affidabilità

APPROFONDIAMO



VISTA COMPORTAMENTALE

Un **componente** software è un'unità di software indipendente e riusabile

- Unità concettuale di **decomposizione** di un sistema a tempo d'esecuzione (per esempio, processo, oggetto, servizio, deposito dati, ecc.)
- 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

Un **sistema software** è una **composizione** di componenti software

- Basata sulla "**connessione**" di più componenti
- Realizzata con **interfacce** dei componenti e **connettori**

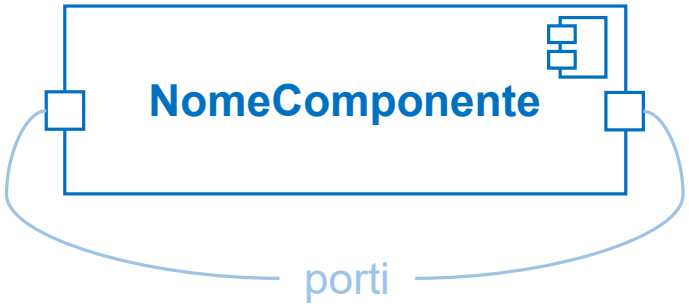
I COMPONENTI IN UML

Un componente è un **classificatore**

Tre rappresentazioni equivalenti
(la terza è ridondante, ma è quella usata da Visual Paradigm)

I **porti** identificano i **punti di interazione** di un componente

- un componente può avere più porti (p.e. uno per connessione)
- un porto fornisce o richiede una o più interfacce (omogenee)
- i porti possono avere nome e/o molteplicità (con l'usuale sintassi 1..n)



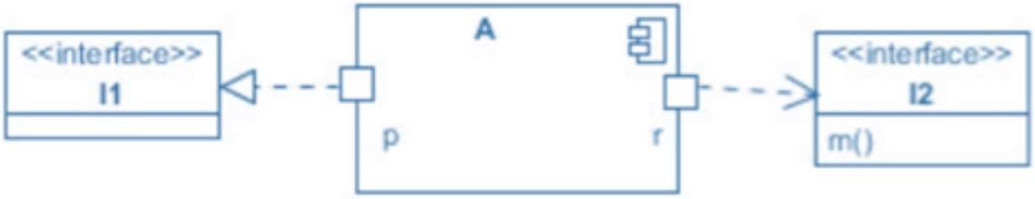
PORTI E INTERFACCE: NOTAZIONE

porti **senza** specifica di interfacce

porto con **interfaccia offerta** (forma sintetica)

porto con **interfaccia richiesta** (forma sintetica)

porto con **interfaccia offerta** (forma estesa

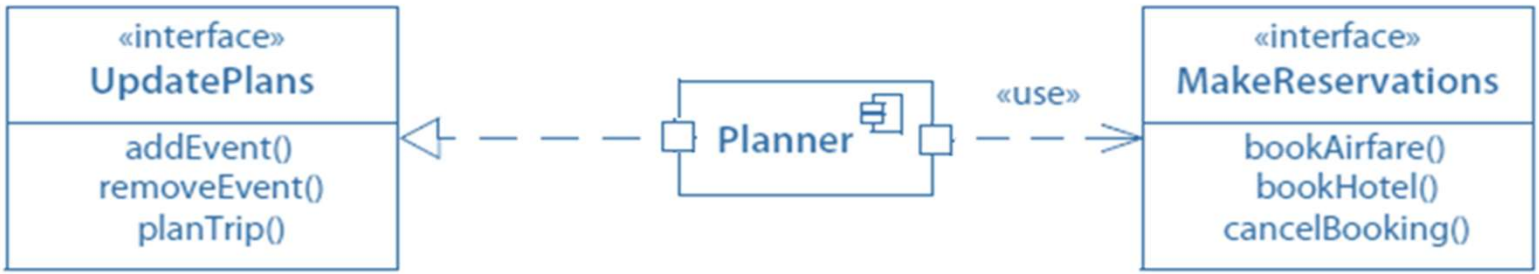


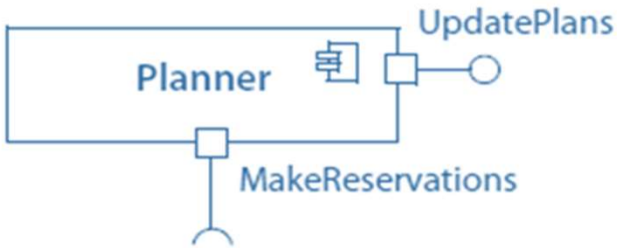
porto con **interfaccia richiesta** (forma estesa

INTERFACCE: DESCRIZIONE SINTETICA VS ESTESA

Le interfacce possono essere descritte in modo

- **sintetico**, con **lollipop** e **forchette**
- **esteso**, per mostrare le **operazioni richieste/offerte**



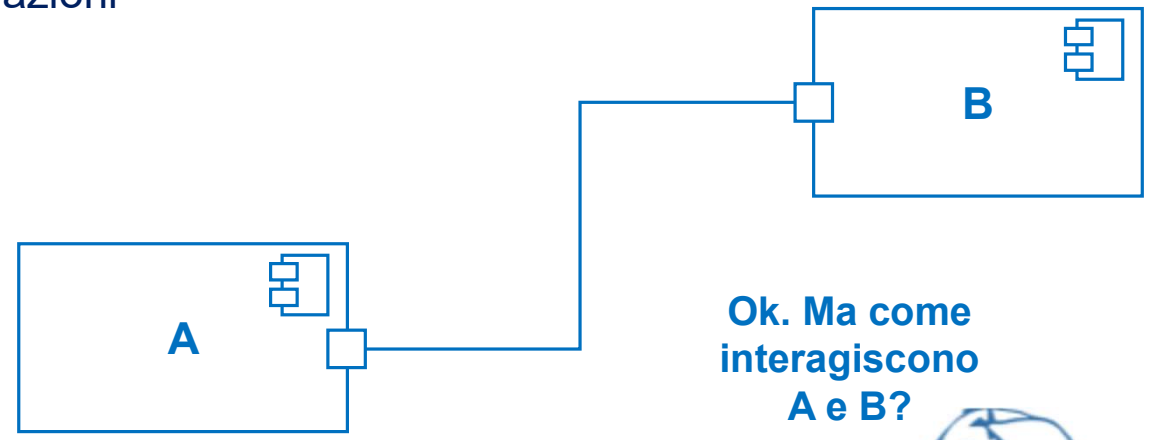


CONNETTORI

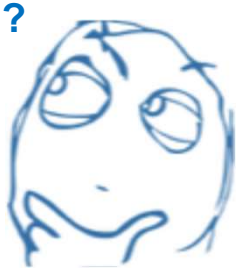
I **connettori** sono canali di interazione che **collegano porti** di componenti (ad esempio, per rappresentare protocolli, flussi di informazione, accessi ai depositi, ecc.)

In UML

- non hanno un descrittore specifico
- si rappresentano come associazioni



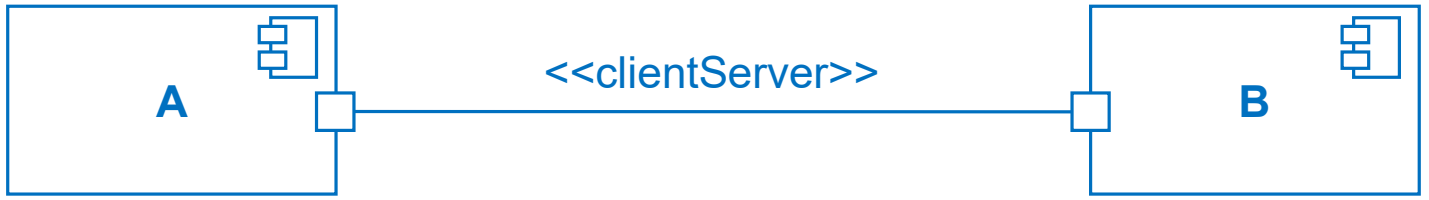
Ok. Ma come interagiscono A e B?



CONNETTORI (CONT.)

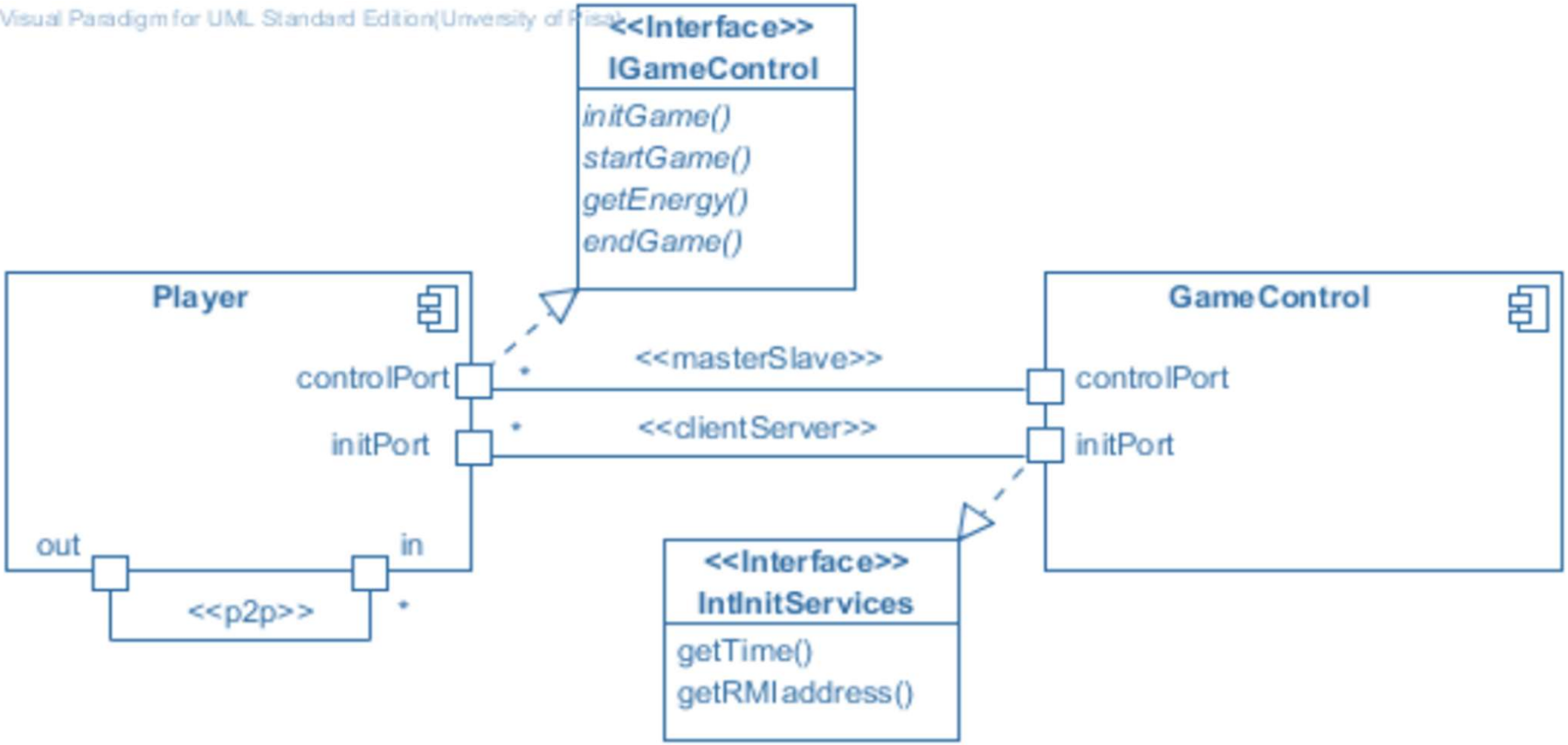
Aggiungiamo informazione sull'interazione, indicando lo **stile** della connessione

- Usando uno **stereotipo**
  ad esempio, <<clientServer>>, <<dataAccess>> (specializzazione di <<clientServer>>), <<pipe>>, <<peer2peer>> (o <<p2p>>), <<publishSubscribe>>



- Indicando i **ruoli** delle componenti

ESEMPIO


Visual Paradigm for UML Standard Edition(University of
<<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 architetturale** caratterizza una famiglia di architetture con caratteristiche comuni

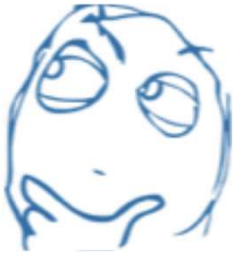
- client-server → caratterizzato dal tipo di interazione tra i componenti
- microservizi (cfr. https://martinfowler.com/articles/microservices.html)

Funzionalità e interazioni tra componenti spesso seguono 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)

Ok. Ma quali «stili» esistono?



STILE – PIPES & FILTERS

Lo stile **pipes and filters** consiste in un flusso di elaborazione di dati, che viaggiano lungo le **pipe** e vengono processati dai **filter** e

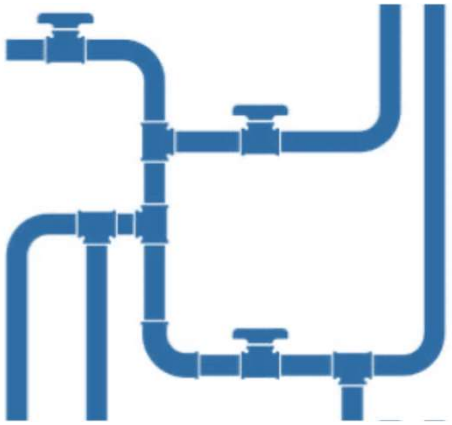
i filter passano i dati in uscita ai filtri adiacenti attraverso le pipe

I **componenti** sono di tipo **filter** (filtro)

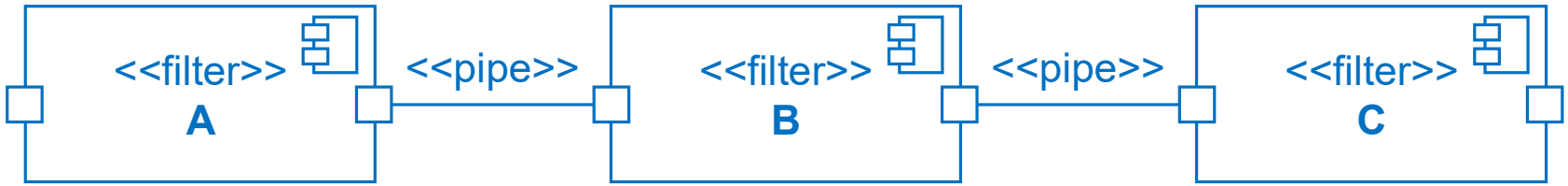
- trasformano uno o più flussi di dati dai porti d'ingresso in uno o più flussi sui porti d'uscita

I connettori sono di tipo **pipe** (condotta)

- canale di comunicazione unidirezionale bufferizzato
- preserva l'ordine dei dati dal ruolo d'ingresso a quello d'uscita

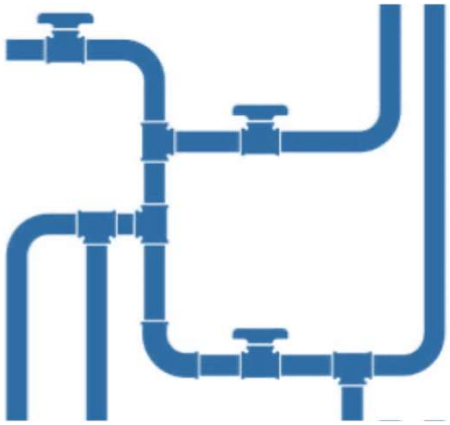


STILE – PIPES & FILTERS (CONT.)

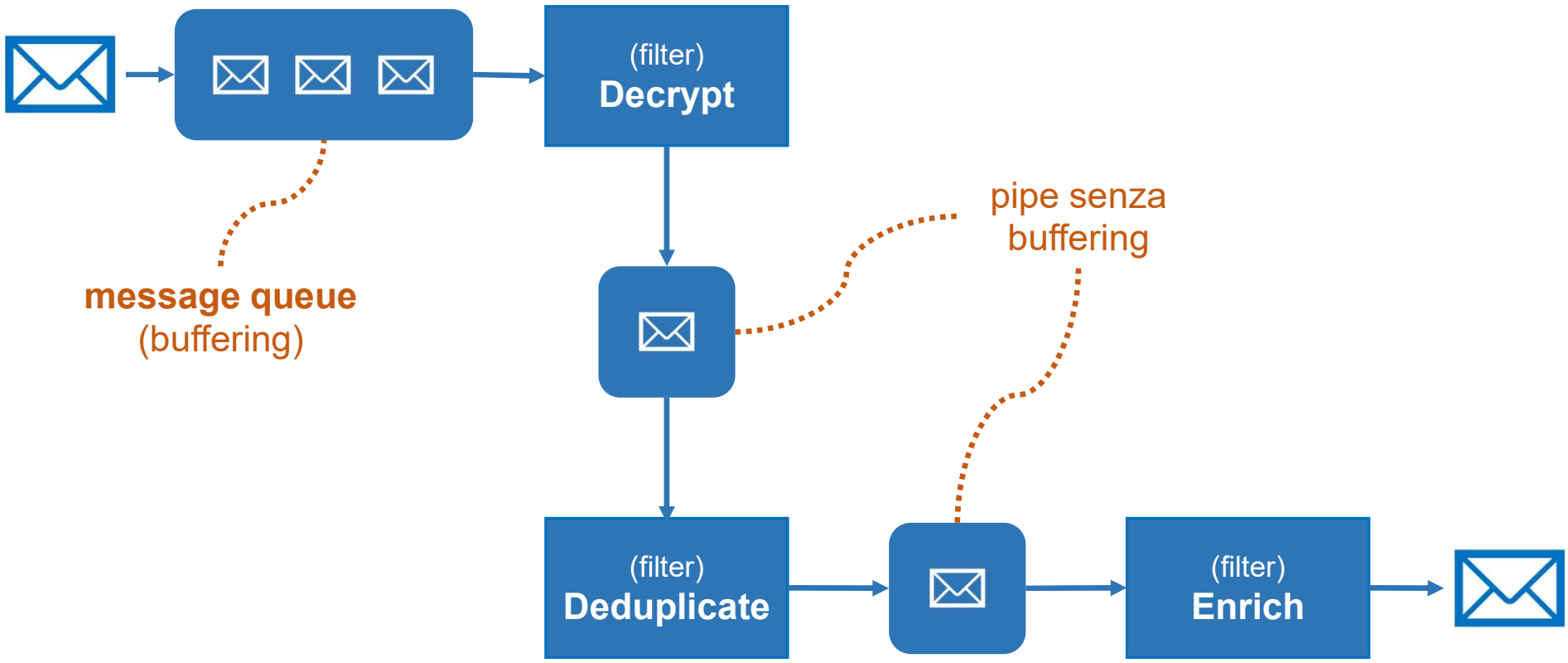


Gli elementi del possono variare nelle funzioni che svolgono, ad esempio

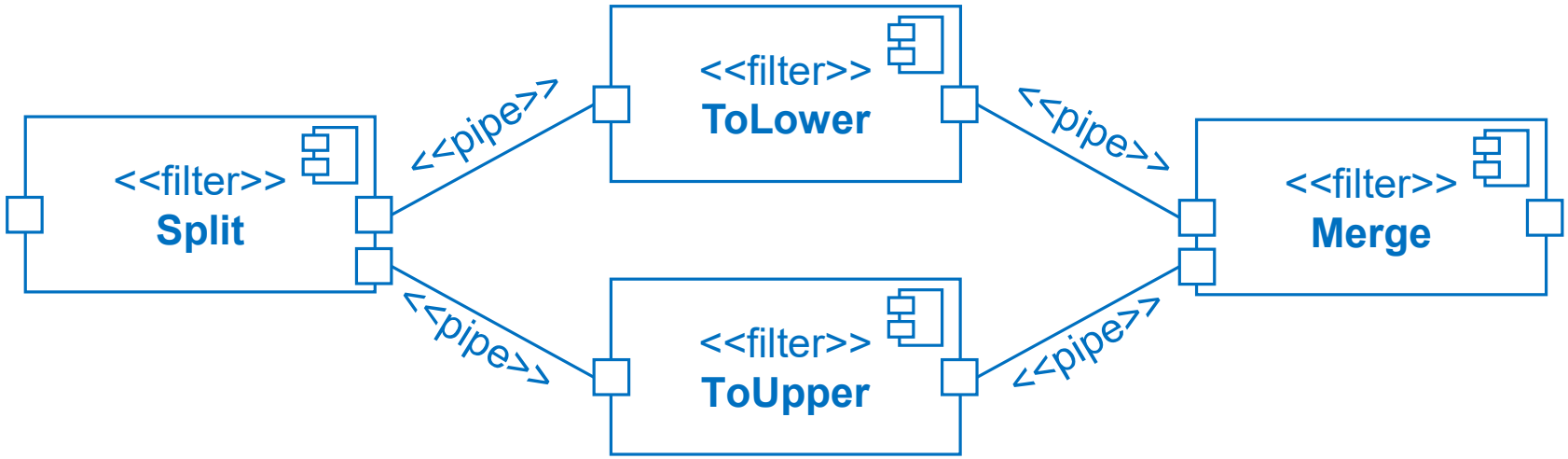
- pipe con supporto per il buffering dei dati
- biforcazioni



ESEMPIO, CON BUFFERING



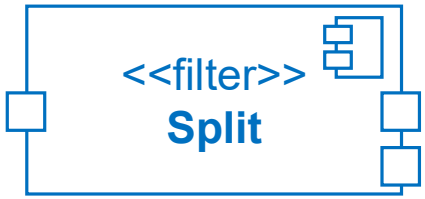
ESEMPIO, CON BIFORCAZIONE



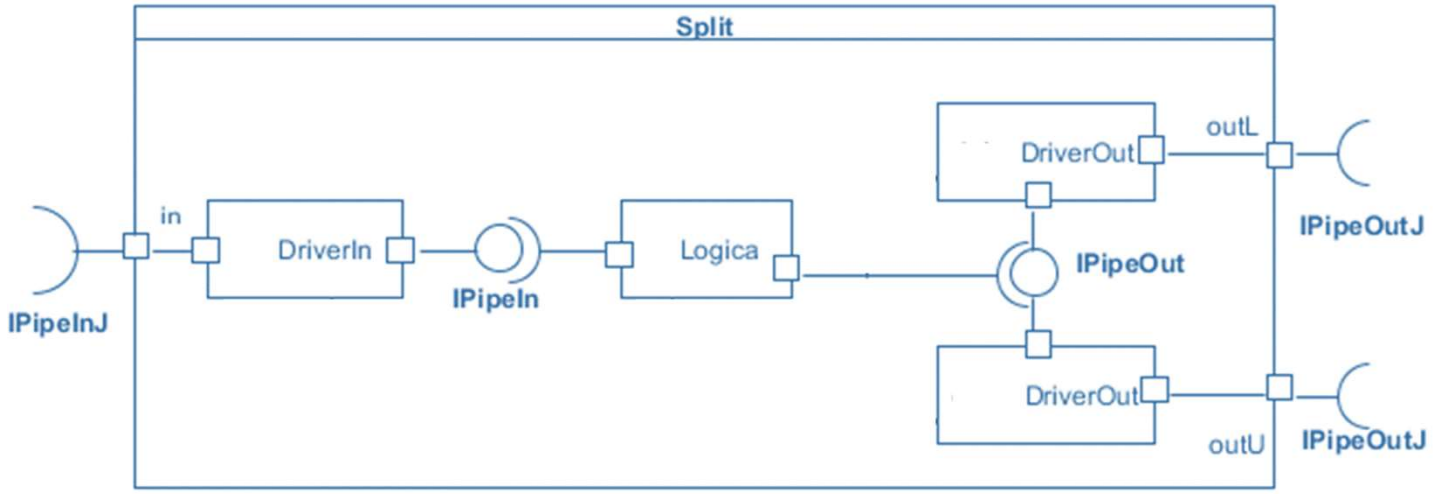
La stringa «ciao» diventa «claO» // stessa cosa per «CIAO» e «Ciao»

COME SI REALIZZA UN FILTER?

Ad esempio, il filtro



si può realizzare come segue



STILE – CLIENT-SERVER

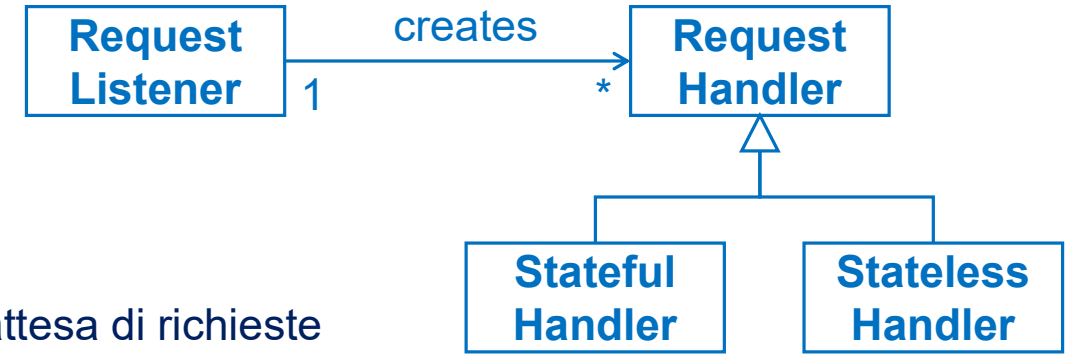
Sistema formato da due componenti // dispiegabili su macchine diverse, collegate in rete

Il **server** offre un servizio // ad esempio, gestione e accesso a dati

- aspetta le richieste di un client ad un porto
- più clienti servibili dallo stesso porto

Il **client** invia richieste al server e attende una risposta

COME SI REALIZZA UN SERVER?



Un **RequestListener** (anche multi-thread) in attesa di richieste

Per ogni richiesta, un **RequestHandler** per gestirla

- elabora la richiesta e invia la risposta al client
- se **stateless**, gestisce ogni richiesta in modo indipendente
- se **stateful**, consente richieste composite che consistono di più richieste atomiche (mantenendo un record delle richieste di un client, chiamato **sessione**)

STILE – MASTER-SLAVE

Si tratta di un caso particolare di **client-server** che risponde ad esigenze precise

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

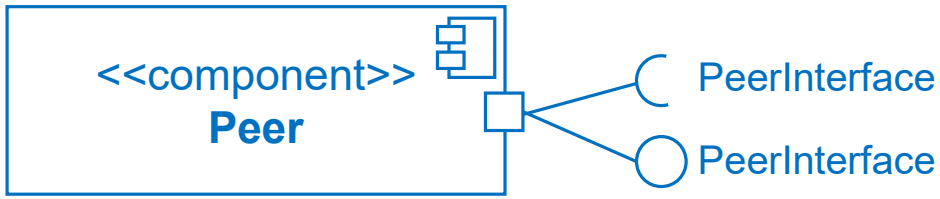
Usato, ad esempio, nella replica di database

- database master come fonte autorevole
- database slave come repliche sincronizzate con il master

STILE – P2P

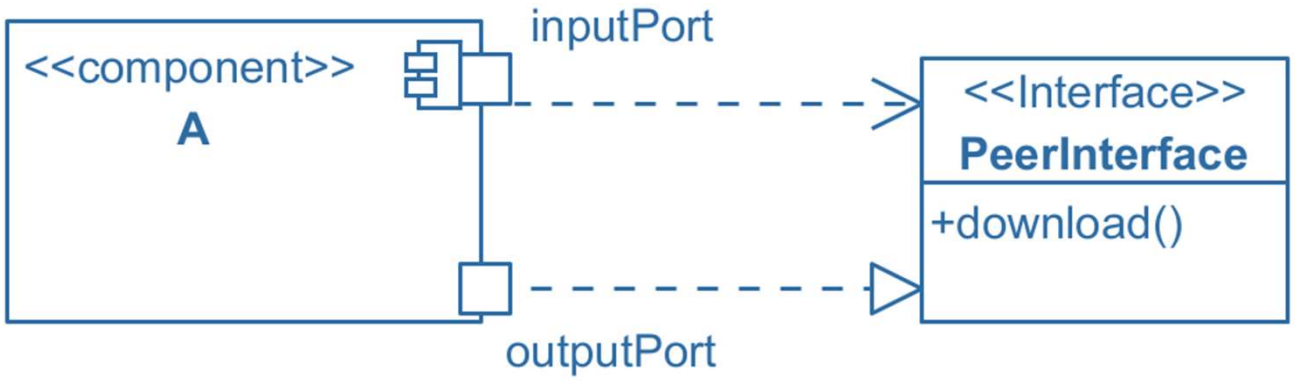
Anche P2P (**peer-to-peer**) è un caso particolare di **client-server**

- Tutti i componenti agiscono sia da client sia da server
- Scambio di servizi alla pari
- Esempio: programmi di scambio audio-video // winmx, kazaa, eMule



Nota: Solo un componente nello schema perché tutti i peer sono istanze di tale componente

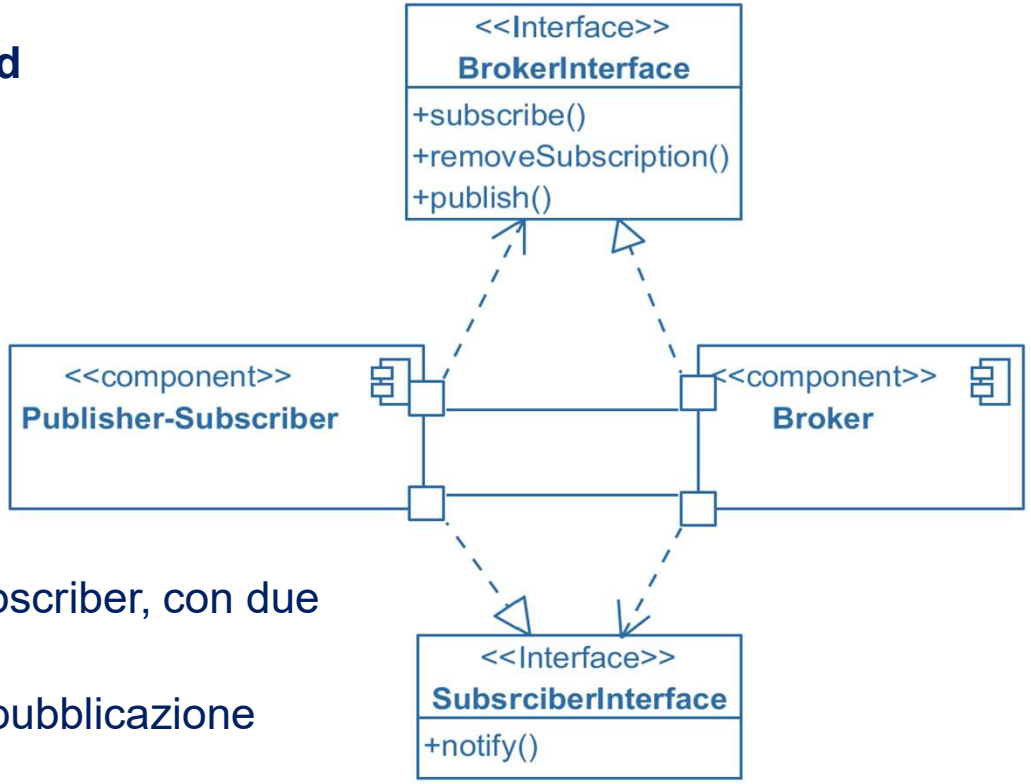
STILE – P2P (CONT.)



STILE – PUBLISH-SUBSCRIBE

I componenti interagiscono in modo **event-based**

- **Publisher:** produce classi di eventi
- **Subscriber:** si abbona alle classi di eventi che ritiene rilevanti
- **Broker**: «smista» gli eventi pubblicati



Un componente può essere sia publisher sia subscriber, con due **connettori diversi**

- Un connettore per richieste di sottoscrizione/pubblicazione
- Un connettore per diffondere i dati

STILE – PUBLISH-SUBSCRIBE (CONT.)

Mittenti (**publisher**) e destinatari (**subscriber**) dialogano attraverso un tramite (**broker**)

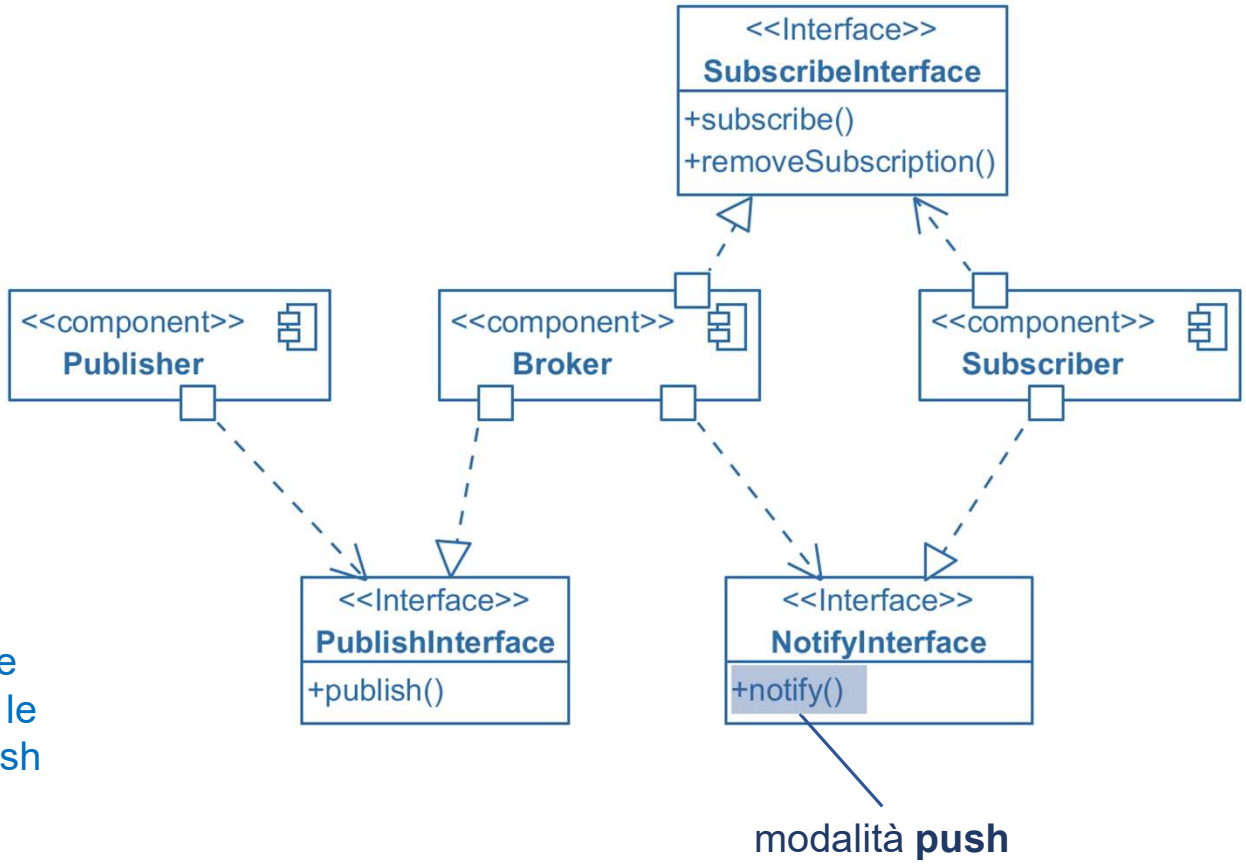
- Il **publisher** si limita a pubblicare messaggi sul broker (senza conoscere l'identità dei destinatari)
- Il **subscriber** si «abbona» al broker per determinati messaggi (per esempio, solo quelli generati da un publisher o aventi certe caratteristiche)
- Il **broker** inoltra ogni messaggio ricevuto da un publisher ai subscriber interessati

I **publisher** non sanno quanti/quali **subscriber** ci siano (e viceversa)

- Questo contribuisce alla **scalabilità** del sistema

STILE – PUBLISH-SUBSCRIBE (CONT.)

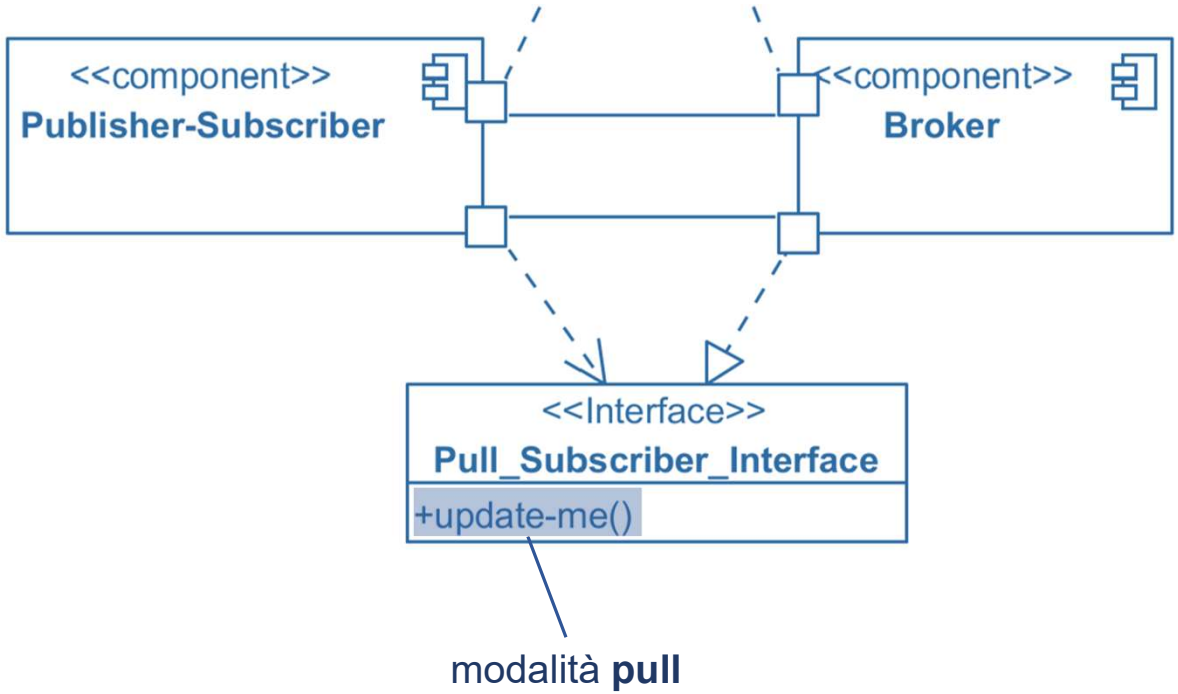
Publisher e subscriber possono essere componenti distinte



Ecco come funzionano le notifiche push

STILE – PUBLISH-SUBSCRIBE (CONT.)

La diffusione dei messaggi può avvenire anche in modo **pull**



PUSH O PULL?

Modello **push** → il **broker** invia **attivamente** i messaggi ai consumatori

- Controlla la frequenza con cui i dati vengono trasferiti
- Deve decidere se inviare un messaggio immediatamente o se accumulare più dati e inviare
- Complicato trattare con diversi tipi di consumatori

Modello **pull** → il **consumatore** si assume la responsabilità di recuperare i messaggi dal broker

- Il consumatore deve tenere traccia del «prossimo messaggio successivo»
- Migliora scalabilità (meno oneri per i broker) e flessibilità (consumatori diversi con esigenze e capacità diverse)
- Se non ci sono messaggi nel broker, i consumatori potrebbero comunque essere occupati in attesa del loro arrivo

ESEMPI DI PUBLISH-SUBSCRIBE

- **AMQP** (Advanced Message Queuing Protocol): protocollo per comunicazioni di tipo publish-subscribe (ma anche punto-a-punto)
    - **RabbitMQ**: RabbitMQ è un middleware di messaggistica open-source basato su AMQP, ampiamente utilizzato per l'implementazione di architetture publish-subscribe.

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

- **Apache Kafka**: piattaforma open-source per streaming distribuito di dati
    - Offre supporto per il modello publish-subscribe
    - Utilizzato per elaborare eventi in tempo reale e trasmettere di dati tra applicazioni

ESEMPI DI PUBLISH-SUBSCRIBE (CONT.)

- **DDS** (Data Distribution Service): middleware standard basato sul paradigma publish-subscribe, per lo sviluppo di livelli middleware per la comunicazione machine-to-machine
  - Mantenuto da OMG (Object Management Group)
  - Consente di implementare comunicazione affidabile tra sensori, controllori e attuatori
  - Fornisce un API per serializzazione/deserializzazione di dati built-in o custom attraverso un linguaggio di definizione dell'interfaccia (IDL) dedicato
  - Usato, per esempio, da
    - NASA
    - Siemens per gli impianti eolici
    - Volkswagen e Bosch per i sistemi di parcheggio autonomo

STILE – MODEL-VIEW-CONTROLLER

Model

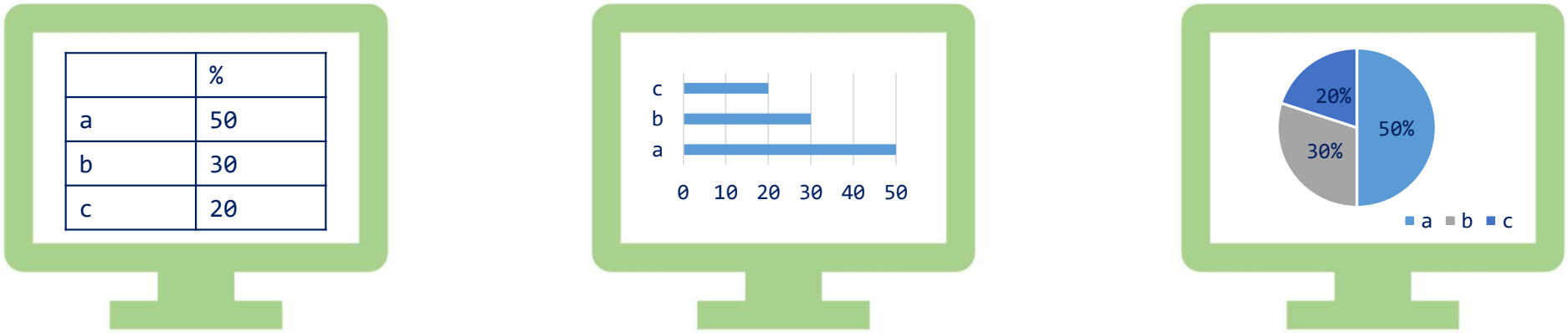
- Nucleo funzionale: implementa la **business logic** dell'applicazione
- Rappresenta i dati su cui opera l'applicazione stessa

View

- **Presentazione** del **model** all'utente // p.e., attraverso un'interfaccia
- Ci possono essere più viste per un modello

ESEMPIO DI MODEL CON VIEW DIVERSE

a = 50%, b = 30%, c = 20%



STILE – MODEL-VIEW-CONTROLLER

Model

- Nucleo funzionale: implementa la **business logic** dell'applicazione
- Rappresenta i dati su cui opera l'applicazione stessa

View

- **Presentazione** del **model** all'utente // p.e., attraverso un'interfaccia
- Ci possono essere più viste per un modello

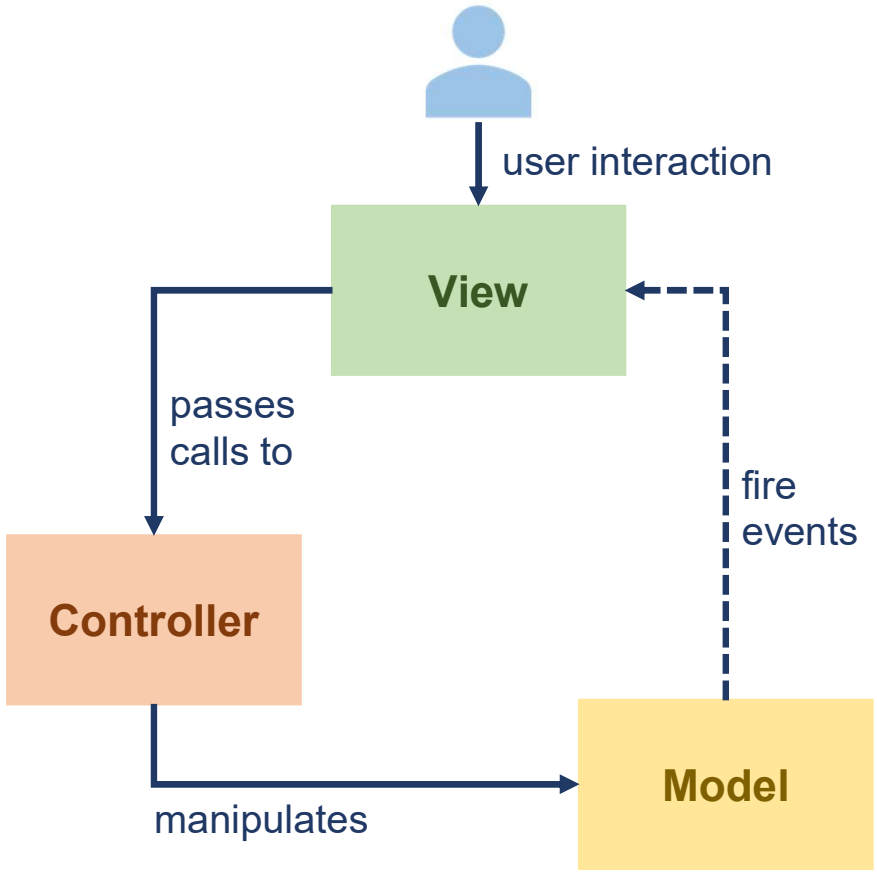
Controller

- **Controllo** dell'input dell'utente
- Traduce eventi in richieste/operazioni da eseguire su **model**

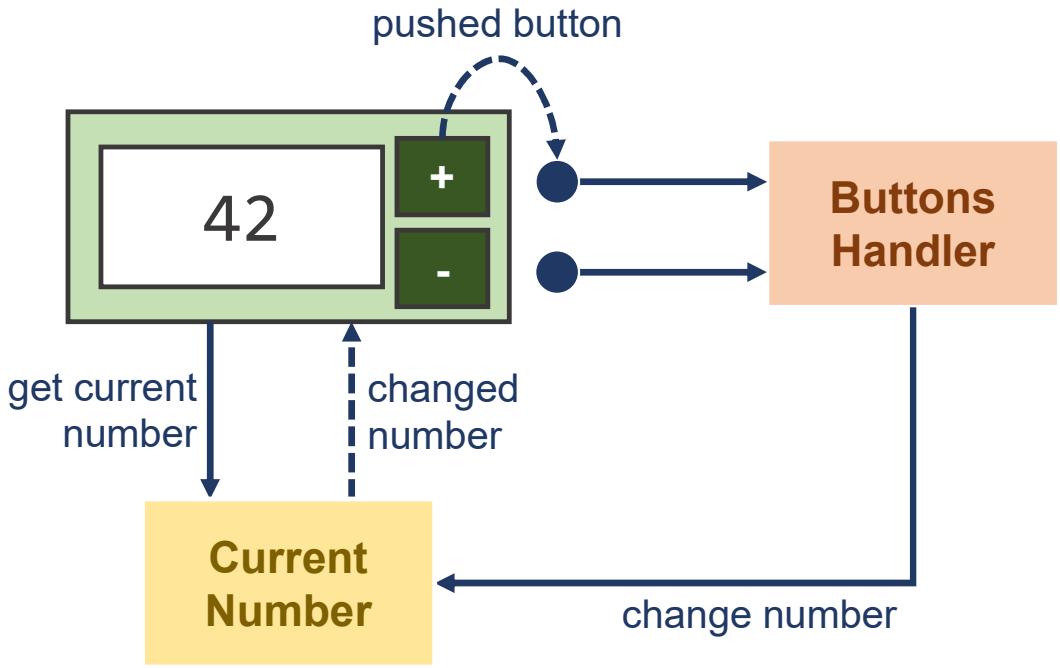
Nota: business logic, presentazione e controllo sono isolati tra loro, consentendone sviluppo, test e manutenzione in modo indipendente

STILE – MODEL-VIEW-CONTROLLER (CONT.)

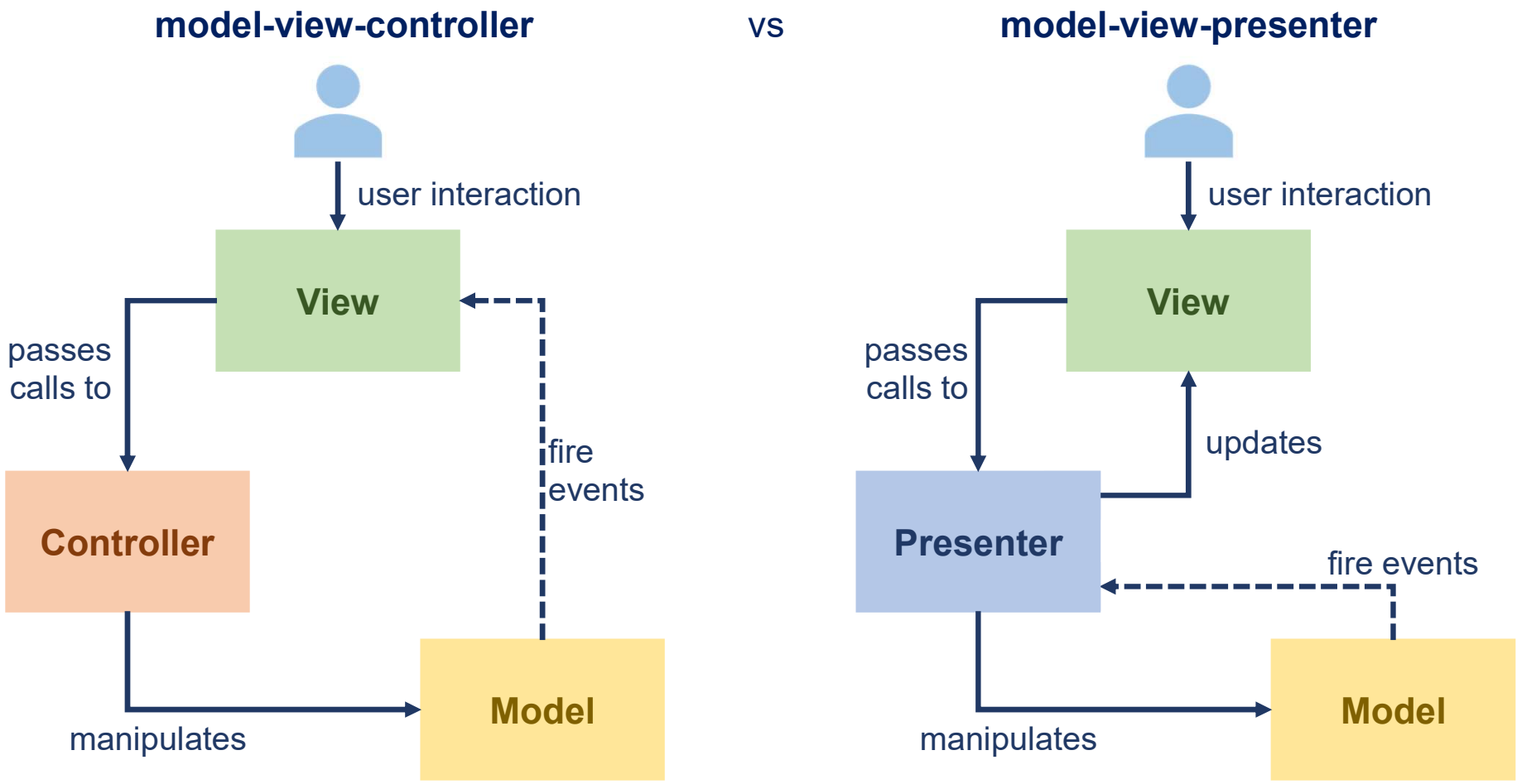
- L'utente interagisce con la **view**
- Il **controller** riceve e interpreta le azioni dell'utente
- Il **controller** chiede al **model** di cambiare stato
- Il **model** notifica la **view** quando cambia stato
- La **view** chiede lo stato al **model**



ESEMPIO DI MODEL-VIEW-CONTROLLER



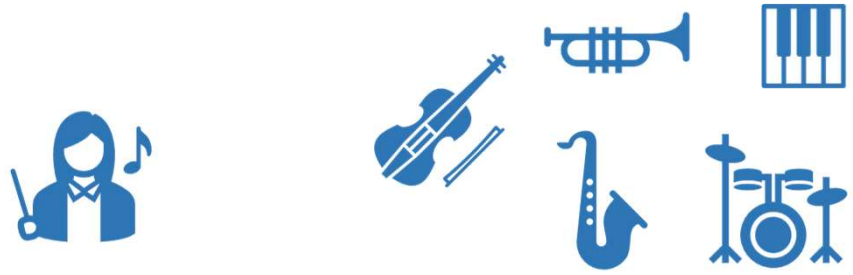
STILE – MODEL-VIEW-PRESENTER



STILE – PROCESS COORDINATOR

Un componente funziona da **process coordinator** (coordinatore), mentre gli altri sono passivi

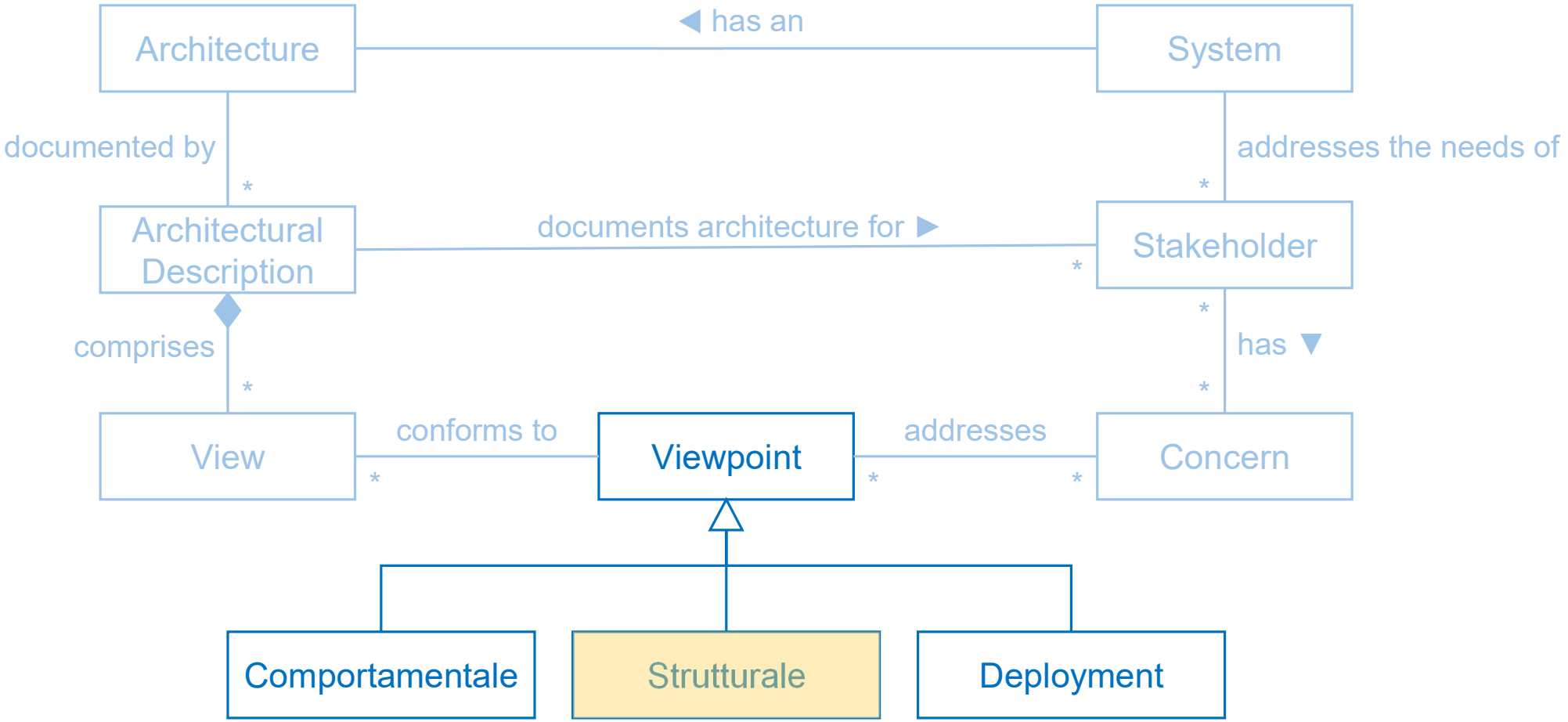
- I componenti (escluso il coordinatore) non conoscono il loro ruolo nel processo complessivo
- Ogni componente fornisce un insieme di funzionalità



Il coordinatore è responsabile della **sequenza di passi** necessari a realizzare un processo

- Riceve la richiesta
- Invoca le funzionalità offerte dagli altri componenti secondo un ordine prefissato
- Fornisce una risposta

APPROFONDIAMO



VISTA STRUTTURALE

Diagramma con elementi e relazioni

- Elementi = **moduli**, ovvero unità di software che realizzano un insieme coerente di responsabilità (ad esempio, classi, collezioni di classi, package)
- Relazioni tra elementi, del tipo **parte di**, **eredita da**, **dipende da**, **può usare**

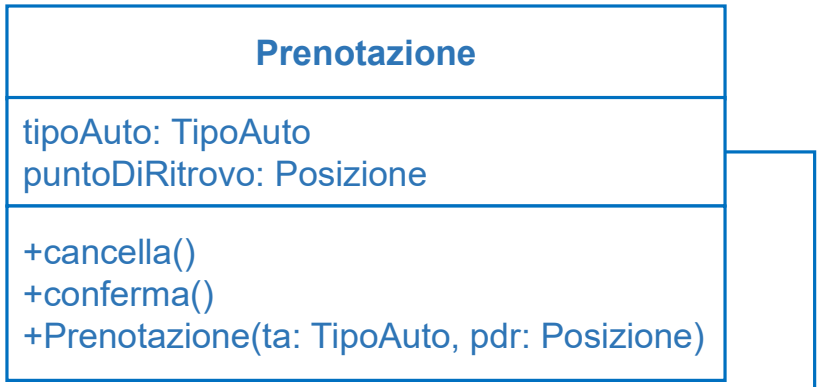
Utile per

- **costruzione**: schema del codice, directory, file sorgente
- **analisi**: tracciabilità dei requisiti, impatto di eventuali modifiche
- **comunicazione**: se gerarchica, offre presentazione top-down per suddivisione responsabilità
- progettazione di **test** di unità e integrazione

Non serve per analisi dinamiche, fatte invece con viste comportamentali/di deployment

VISTA STRUTTURALE, IN UML

Classi (con specifica delle operazioni più dettagliata, rispetto alla descrizione del dominio)



Package

Relazioni tra classi e/o package (ad esempio, contenimento)

VISTA STRUTTURALE DI DECOMPOSIZIONE

Relazione «**parte di**»

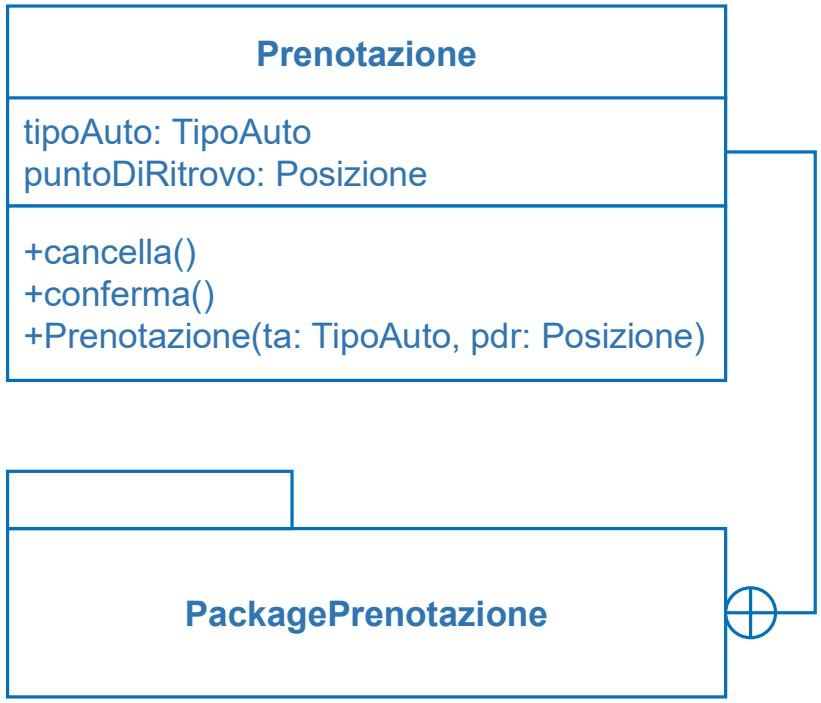
- una classe fa parte di (è contenuta in) un package
- un package fa parte di uno più grande

Criteri per raggruppare

- Incapsulamento per modificabilità
- Supporto alle scelte costruisci/compra
- Moduli comuni in linee di prodotto

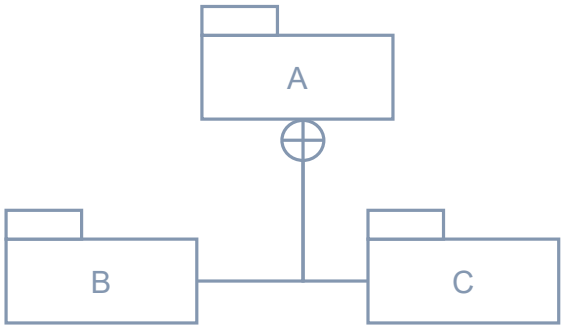
A cosa serve?

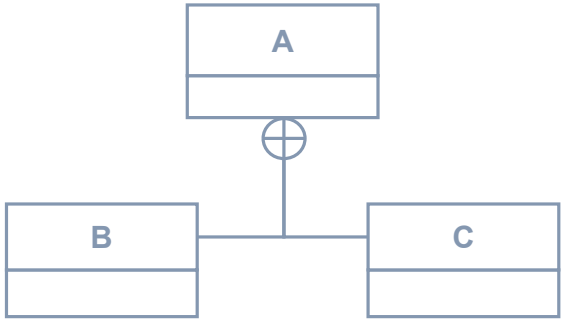
- Apprendimento del sistema
- Allocazione del lavoro (come punto di partenza)

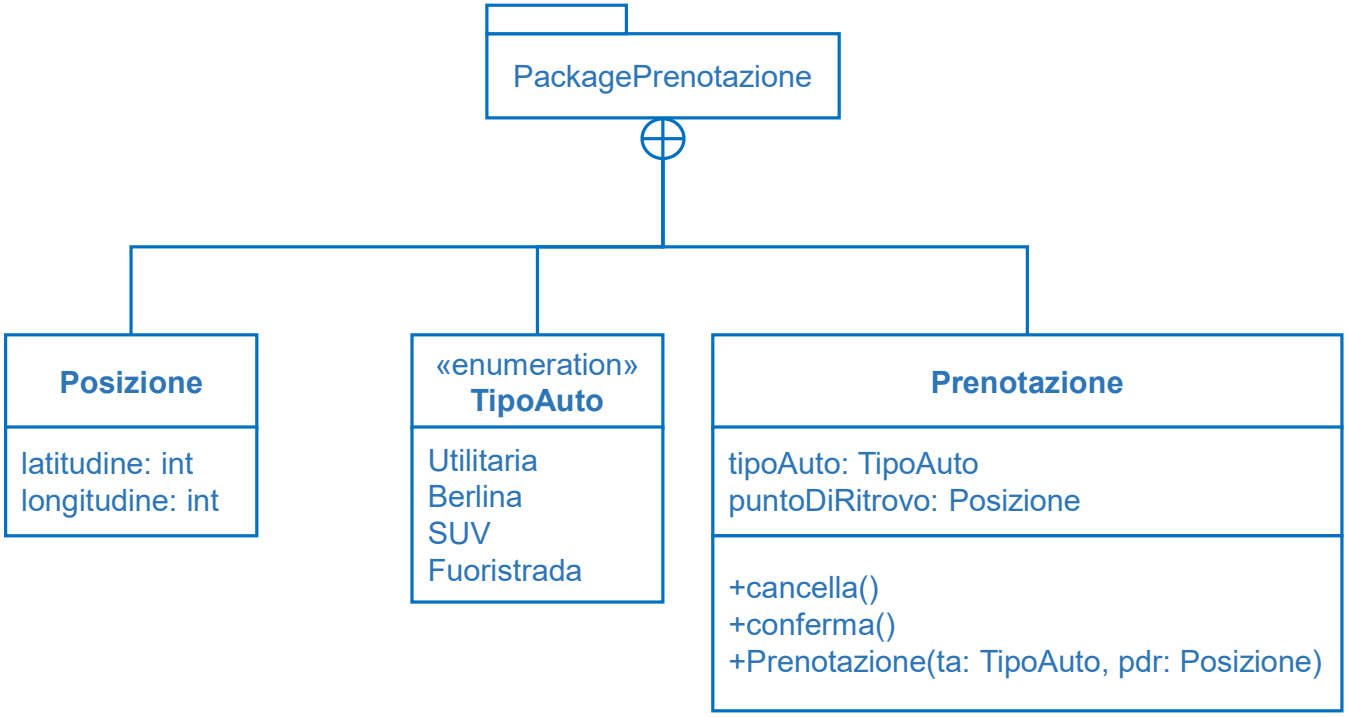


DECOMPOSIZIONE, IN UML

La relazione «**parte di**» si rappresenta con ⊕

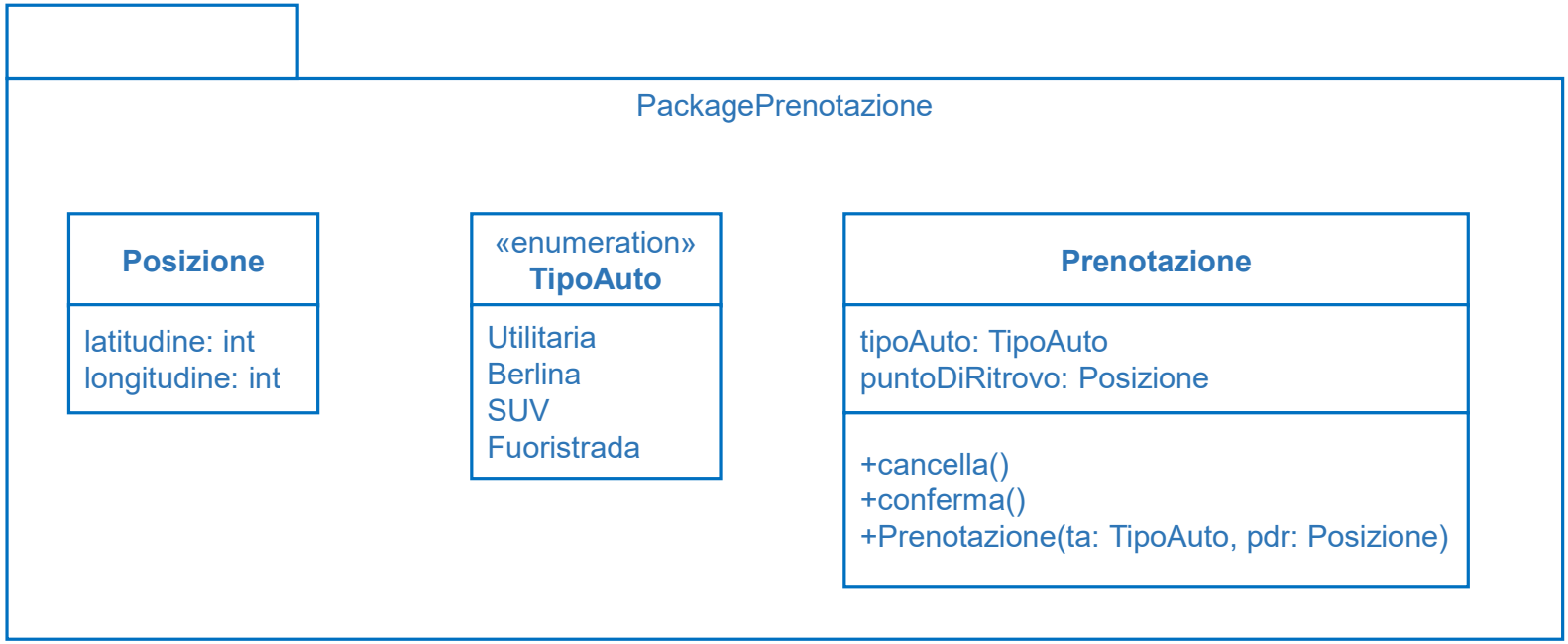






DECOMPOSIZIONE, IN UML (CONT.)

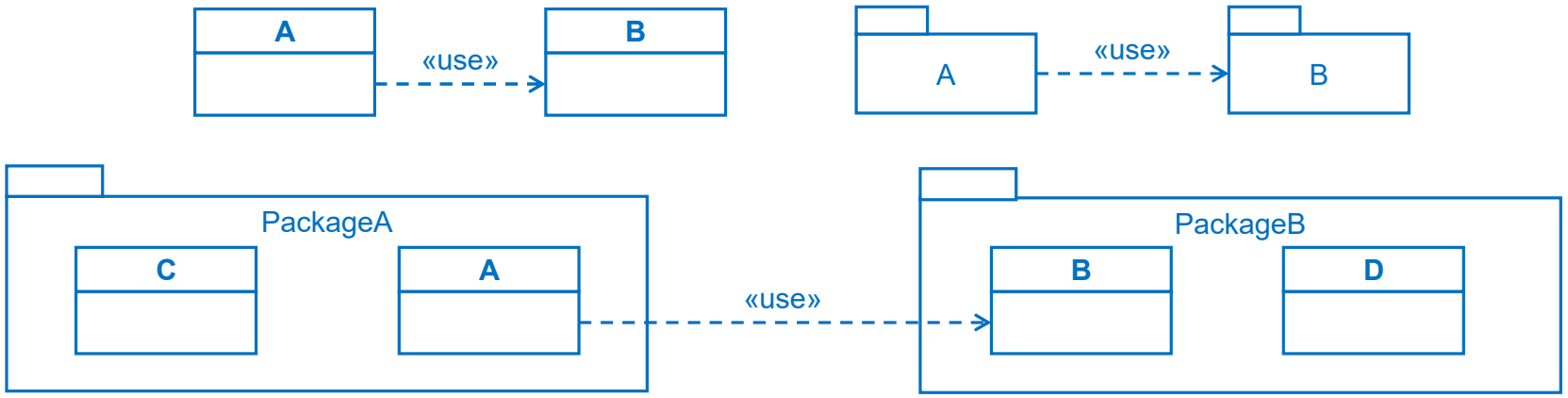
La relazione «**parte di**» si può rappresentare anche con l'inclusione grafica (in un package)



VISTA STRUTTURALE D'USO

Relazione «**use**»

- Il modulo A usa il modulo B se dipende dalla presenza di B per soddisfare i suoi requisiti
- Cicli permessi (ma pericolosi)



A cosa serve?

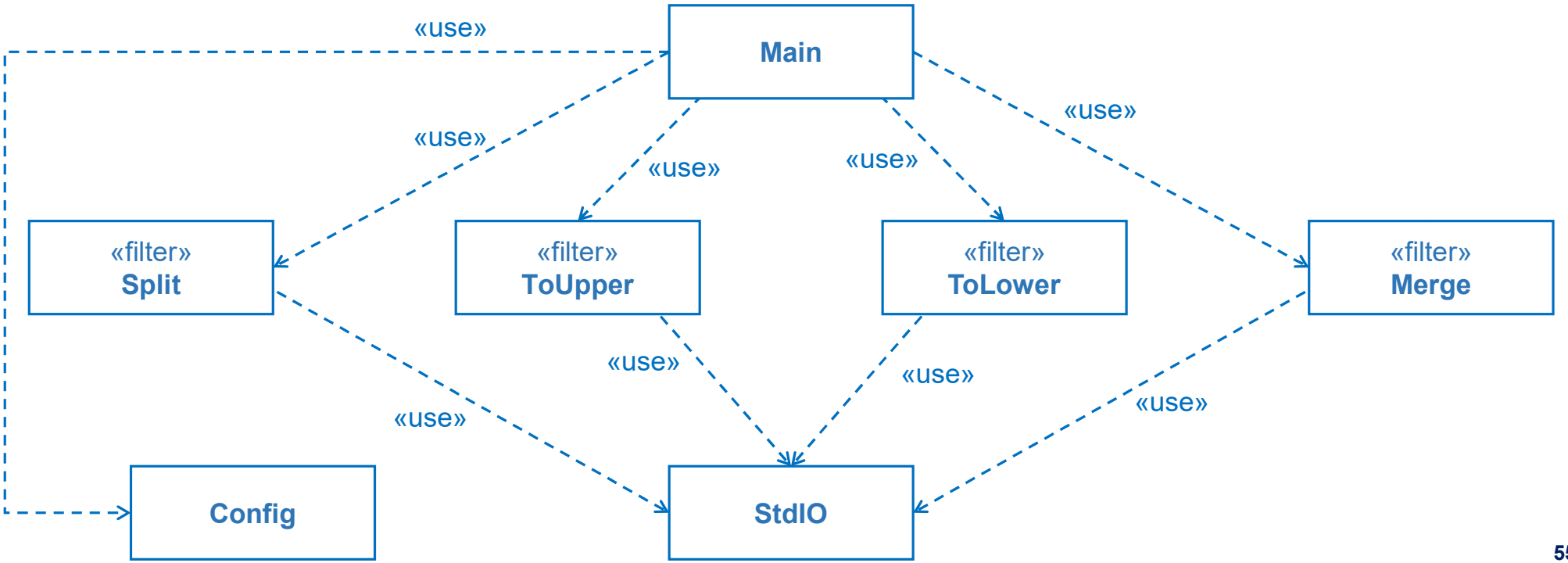
- Pianificazione di sviluppo incrementale
- Test di unità e integrazione

Non confondiamo invocazione e dipendenza: se A segnala un errore a B, ma funziona anche senza B (lo invoca, ma non lo usa), A non cliente di B in una dipendenza «**use**»

ESEMPIO DI VISTA STRUTTURALE D'USO

Riprendiamo l'esempio di pipes and filter con biforcazione

- I filtri si chiamano tra loro (ma non si usano)
- Il main li configura per metterli in comunicazione via StdIO (realizzazione del connettore pipe)
- Tre diversi «strati» → vista a strati?



ALTRE VISTE STRUTTURALI: A STRATI

Elementi = **strati**

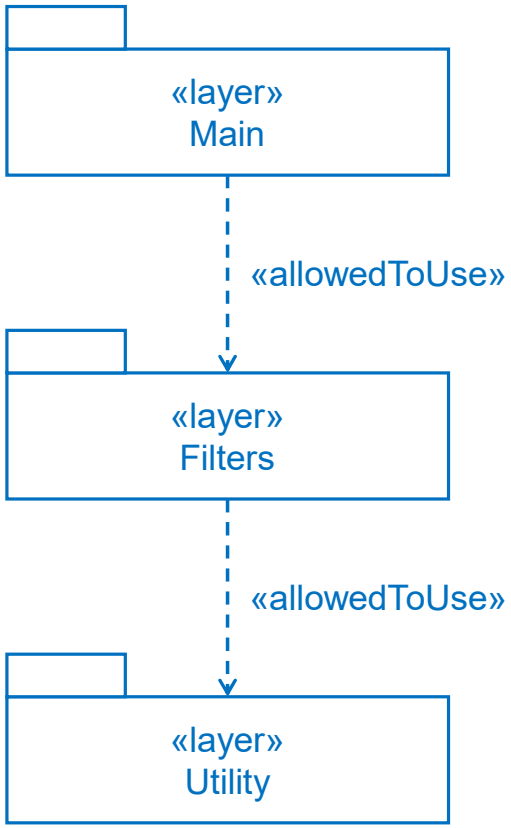
- Uno strato è un insieme coeso di moduli (a volte raggruppati in segmenti)
- Offre un'interfaccia pubblica per i suoi servizi

Relazione «**allowedToUse**»

- Caso particolare di relazione d'uso
- Antisimmetrica, non implicitamente transitiva

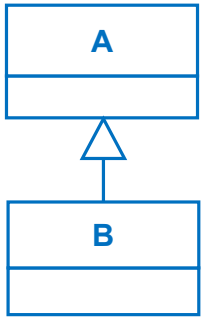
A cosa serve?

- Modificabilità e portabilità
- Controllo della complessità



ALTRE VISTE STRUTTURALI: GENERALIZZAZIONE

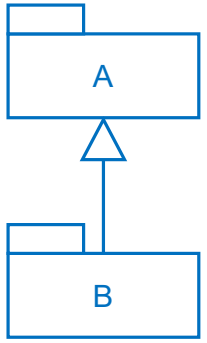
Elementi = **moduli** (classi o packages)



Relazione di generalizzazione

A cosa serve?

- Tra classi, a rappresentare la relazione tipo-sottotipo
- Tra package, rappresentare la relazione tra un framework[1] e una sua specializzazione



1. Collezione di classi, anche astratte, con relazioni d'uso tra loro

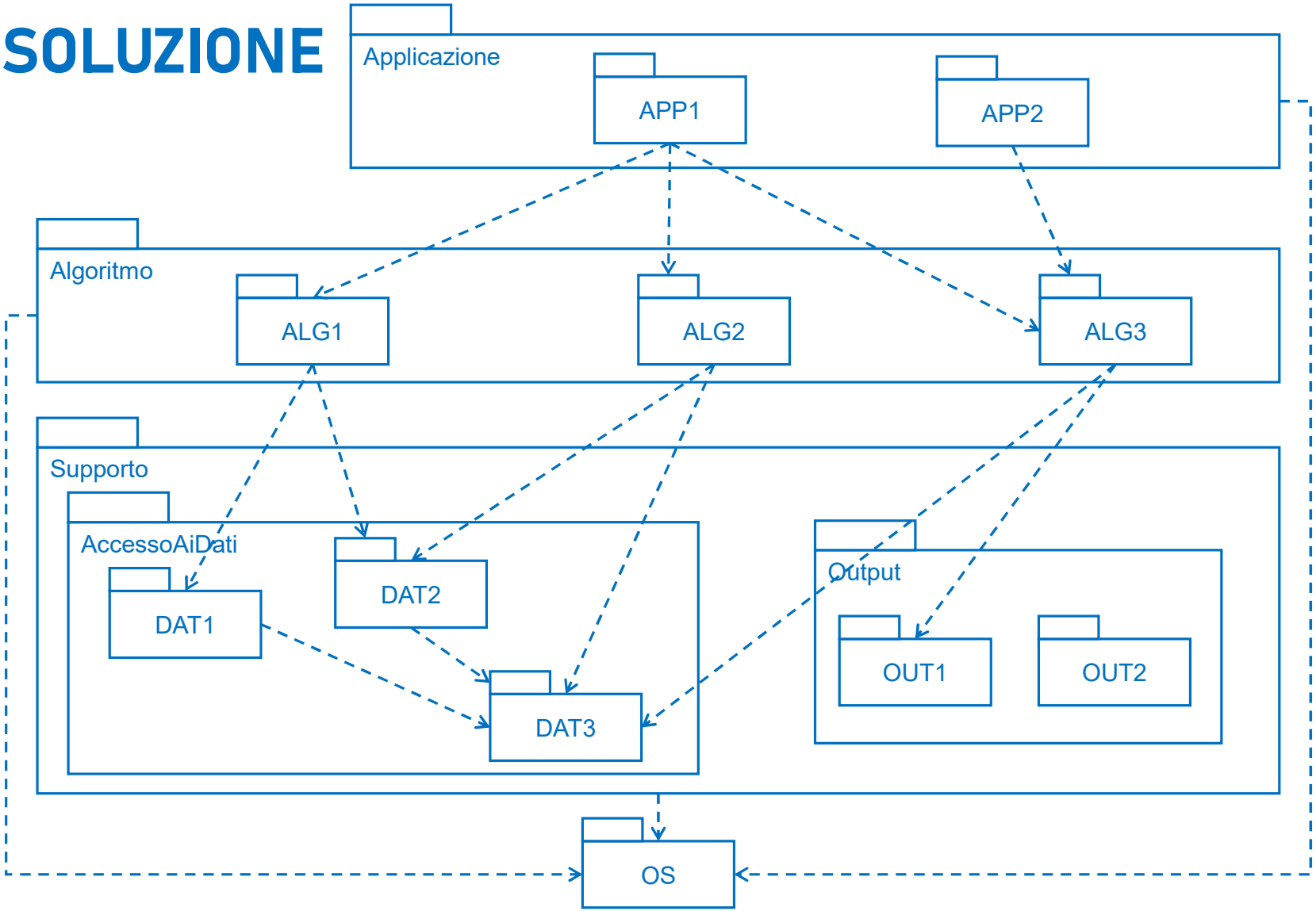
ESERCIZIO

Produrre una vista strutturale per un'applicazione i cui moduli sono nella tabella.

- Come organizzare i moduli in package di moduli?
- Possono tali moduli essere organizzati in livelli? Se si, incorporare i livelli nella vista strutturale

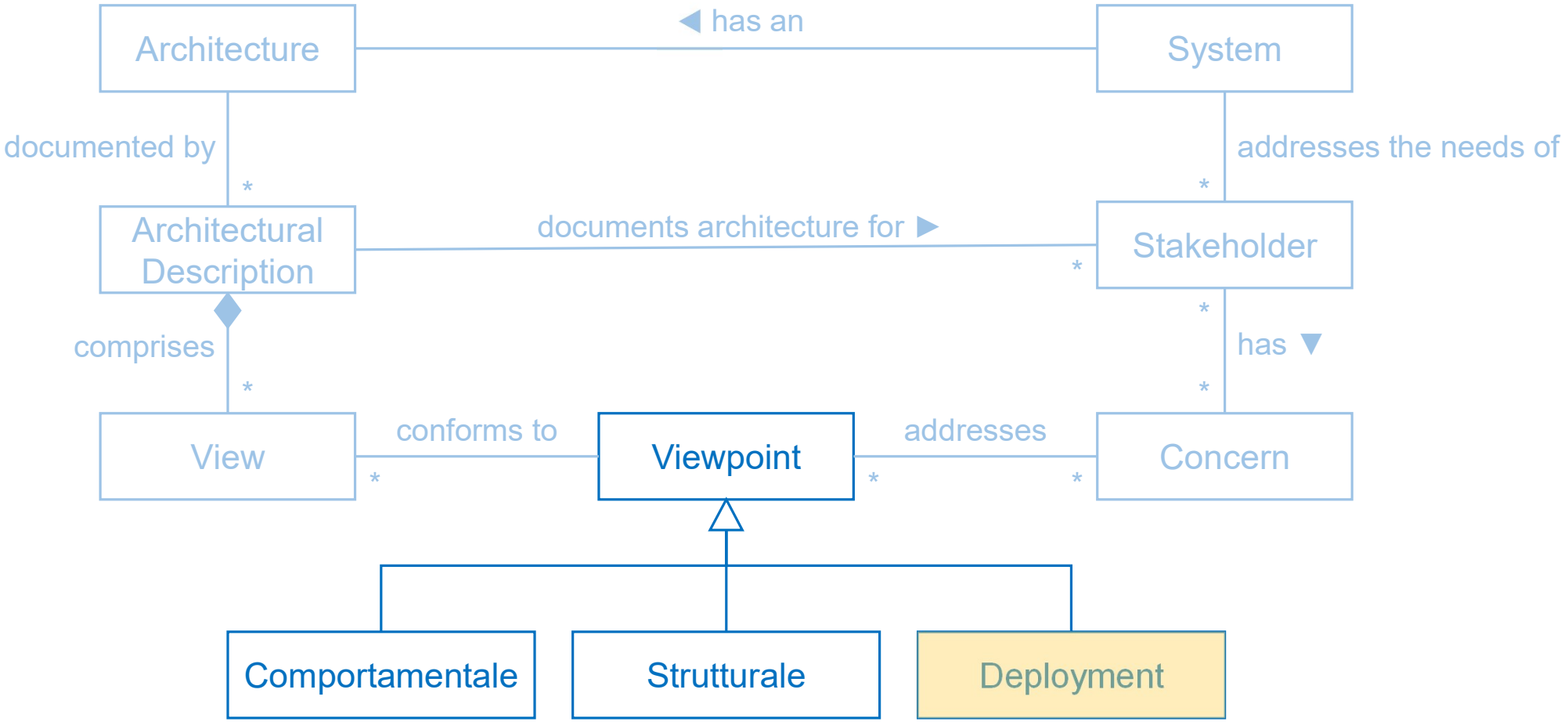
Modulo	Tipo di modulo	Moduli utilizzati
APP1	Applicazione	ALG1,ALG2,ALG3
APP2	Applicazione	ALG3
ALG1	Algoritmo	DAT1,DAT2
ALG2	Algoritmo	DAT2,DAT3
ALG3	Algoritmo	DAT3,OUT1
DAT1	Accesso ai dati	DAT3
DAT2	Accesso ai dati	
DAT3	Accesso ai dati	DAT3
OUT1	Output	
OUT2	Output	
Tutti i moduli elencati sopra usano moduli del sistema operativo		

SOLUZIONE



Quali sono le dipedenze «use»?
Quali sono quelle «allowedToUse»?

APPROFONDIAMO



VISTA DI DEPLOYMENT

Diagramma contenente i seguenti elementi

- **Artefatti** prodotti da un processo di sviluppo software o dal funzionamento di un sistema (ad esempio, codice sorgente, script, file binari, tabelle di database, o documenti)
- Nodi hardware o, più in generale, **ambienti di esecuzione**

Relazioni tra elementi

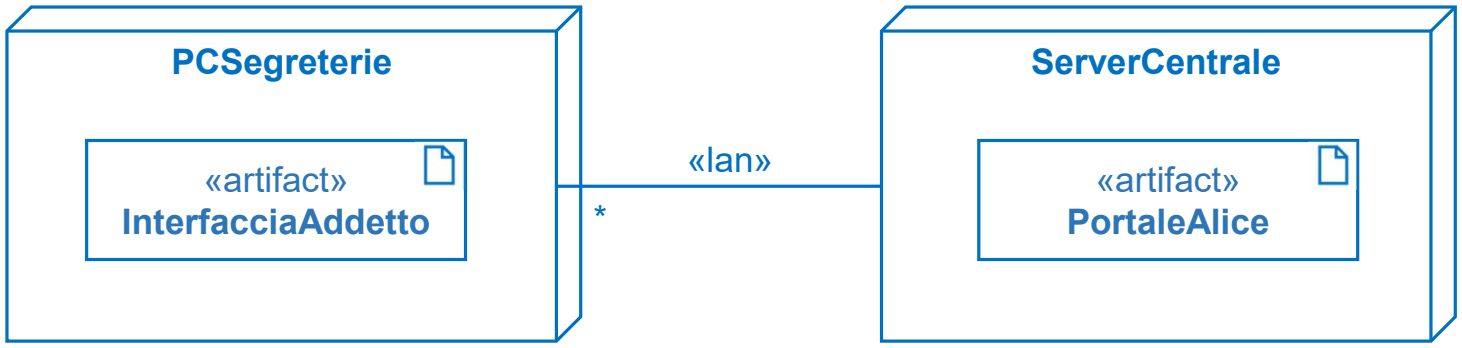
- **Dislocazione** di artefatti negli ambienti di esecuzione
- **Interconnessioni** tra gli ambienti di esecuzione

Utile per

- Analisi delle prestazioni
- Guida per l'installazione di un sistema

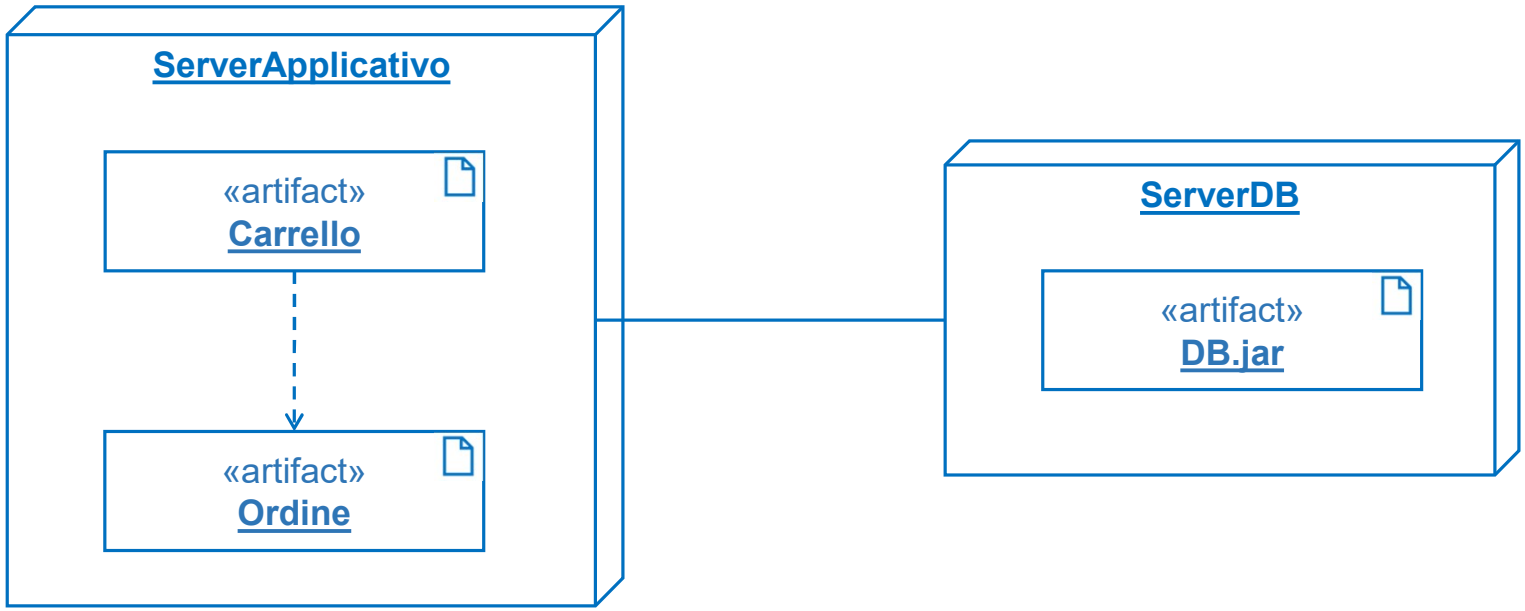
VISTA DI DEPLOYMENT, IN UML

- Gli **ambienti di esecuzione** si rappresentano come **parallelepipedi**
- Gli **artefatti** si rappresentano come elementi con lo stereotipo «**artifact**»
- La **dislocazione** di artefatti in ambienti di esecuzione si rappresenta con il **contenimento**
- Le **interconnessioni** si rappresentano con **relazioni** stereotipate



VISTA DI DEPLOYMENT, IN UML (CONT.)

È possibile modellare un diagramma di deployment anche a livello di **istanza**



ALCUNE PRECISAZIONI

Si parta di deployment di componenti, mentre in realtà si disloca un artefatto

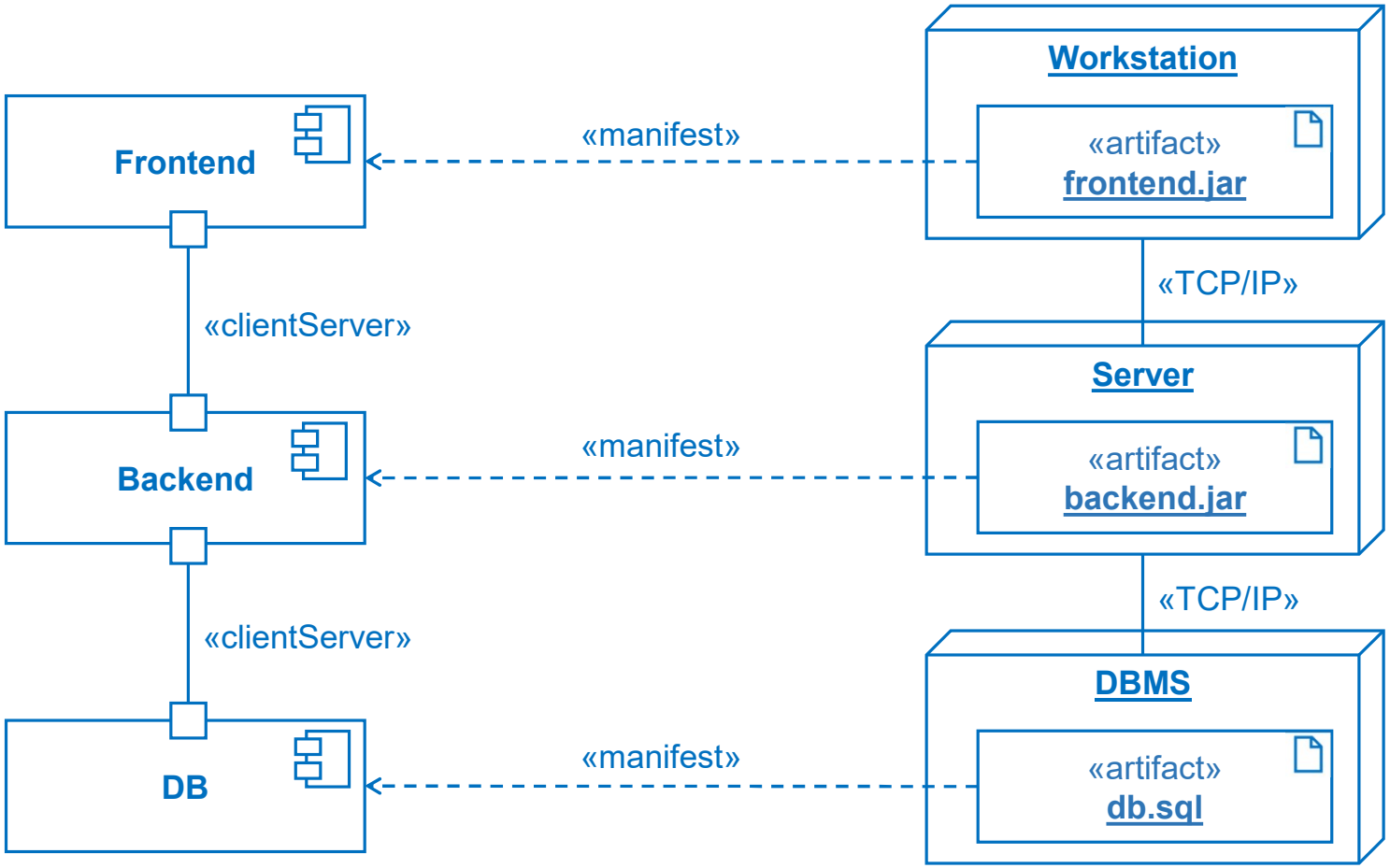
- Un artefatto «manifesta» un componente (ovvero ne fornisce un'implementazione)
- Un artefatto viene dislocato (deployed) su un ambiente di esecuzione
- L'installazione avviene nell'ambiente di esecuzione
- L'installazione comprende la configurazione e la registrazione del componente in tale ambiente

⇒ L'istanza di un componente (a runtime) è quindi creata a partire dell'artefatto

Alcuni esempi

VISTE IBRIDE

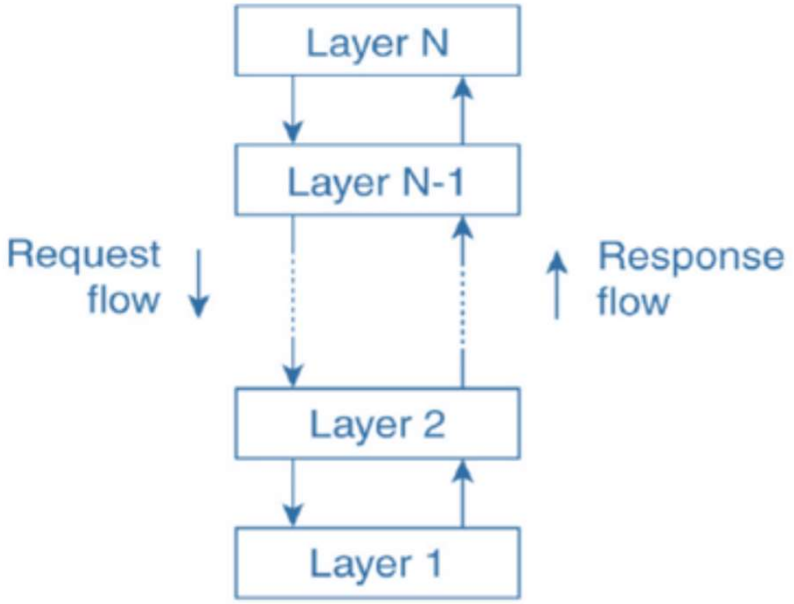
Vista **ibrida** (comportamentale + deployment) su di un'applicazione 3-tier



ALTRI ESEMPI

Architettura a livelli (dal web)

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



Descrizione e rappresentazione ambigue!

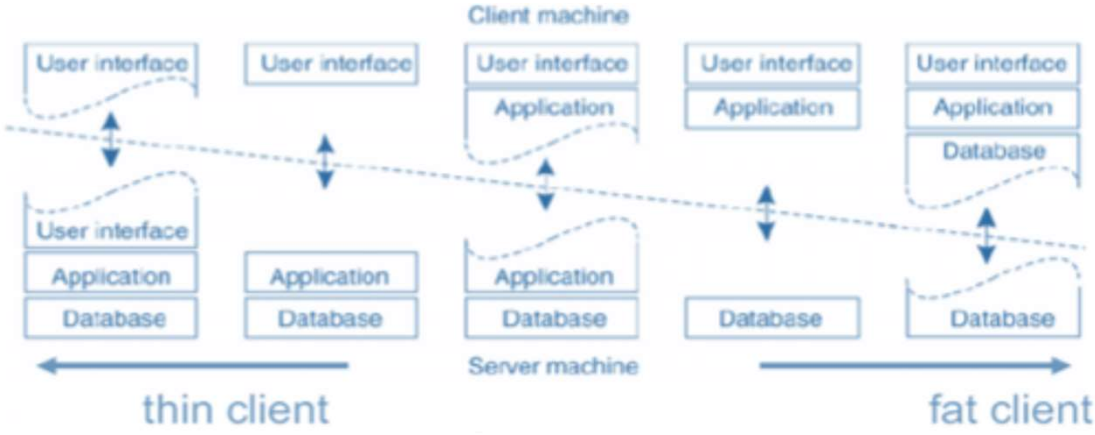
- Vista comportamentale per rappresentare le catene di client-server
- Vista strutturale per rappresentare i livelli

ALTRI ESEMPI (CONT.)

Architettura multi-livello (dal web)

- Mapping tra livelli logici (**layer**) e livelli fisici (**tier**)
- Da un livello a N livelli
  - 1-tier: configurazione mainframe e terminale (non è client-server)
  - 2-tier: due livelli fisici (macchina client, singolo server)
  - 3-tier: ciascun livello su una macchina separata

Diverse configurazioni two-tier

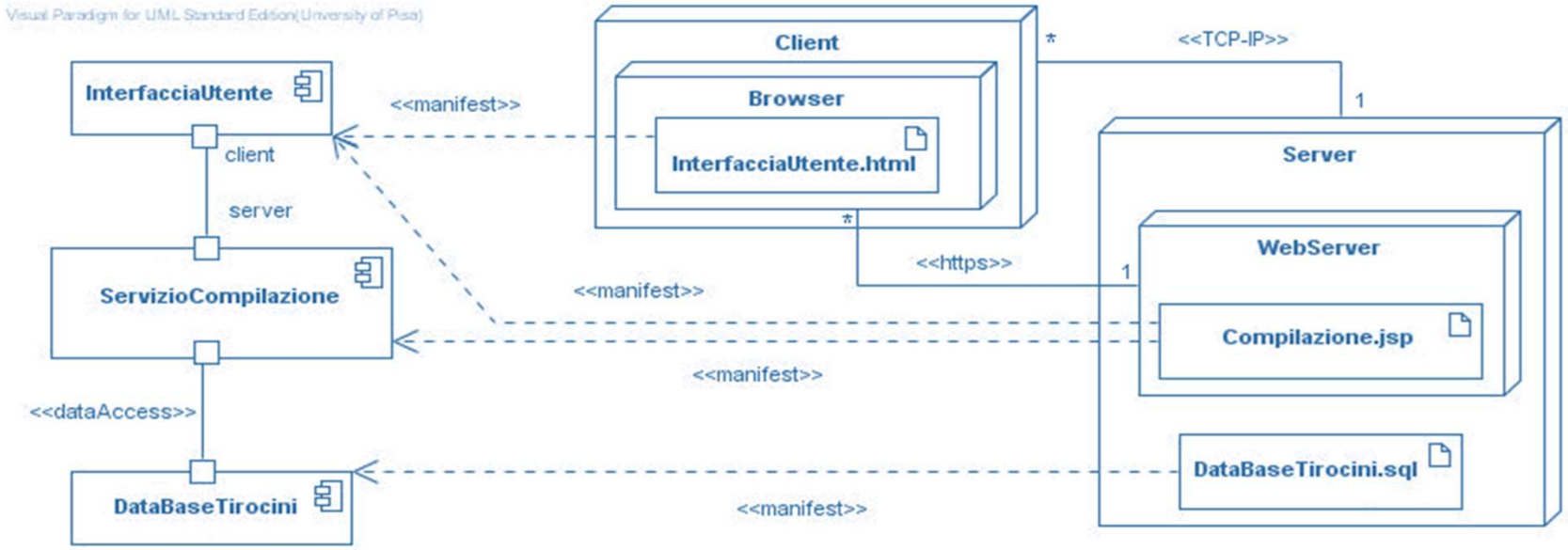


- All'aumentare del numero di livelli,
  - l'architettura guadagna in flessibilità, funzionalità e possibilità di distribuzione
  - si introducono problemi di prestazioni (più costi di comunicazione, più complessa da gestire/ottimizzare)

ALTRI ESEMPI (CONT.)

Vista ibrida (comportamentale + 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.

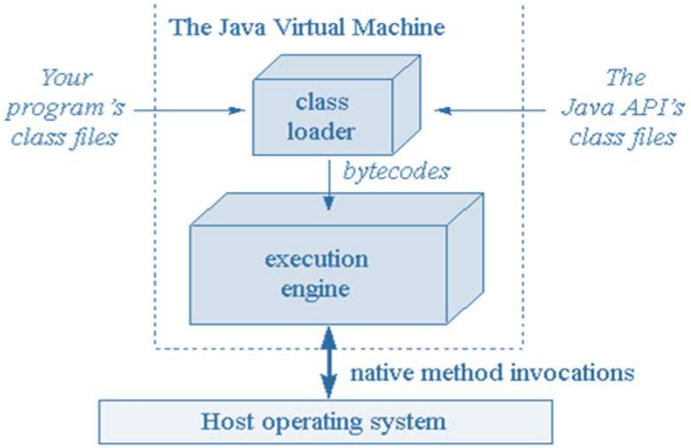



Q
A

HOMEWORK

1) Provate a fornire una rappresentazione UML per
   - Model-View-Controller
   - Model-View-Presenter
   - Coordinatore di processi

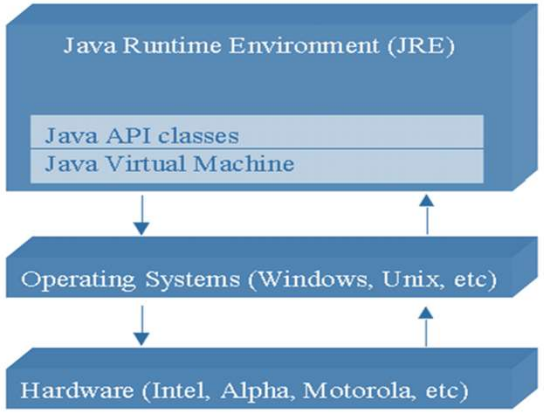
2) Quali viste utilizzare per descrivere le seguenti architetture?



Componente (JVM)

Due sotto-componenti (loader e engine)

Ambiente di esecuzione (SO)



Codie (API classes)

Componente (JVM)

Ambienti di esecuzione (JRE, SO, HW)

RIFERIMENTI

Contenuti

- **Sezioni 6.1-6.4** di "Software Engineering" (G. C. Kung, 2023)
- **Dispensa** di "Architetture Software e Progettazione di Dettaglio" (C. Montangero, L. Semini)

Approfondimenti

- Krutchen, P. *The 4+1 View Model of Architecture*. IEEE Software 12(6), 1995.