UML 2.0: COMPONENT, PACKAGE AND DEPLOYMENT DIAGRAM

Ingegneria del Software 2023-2024

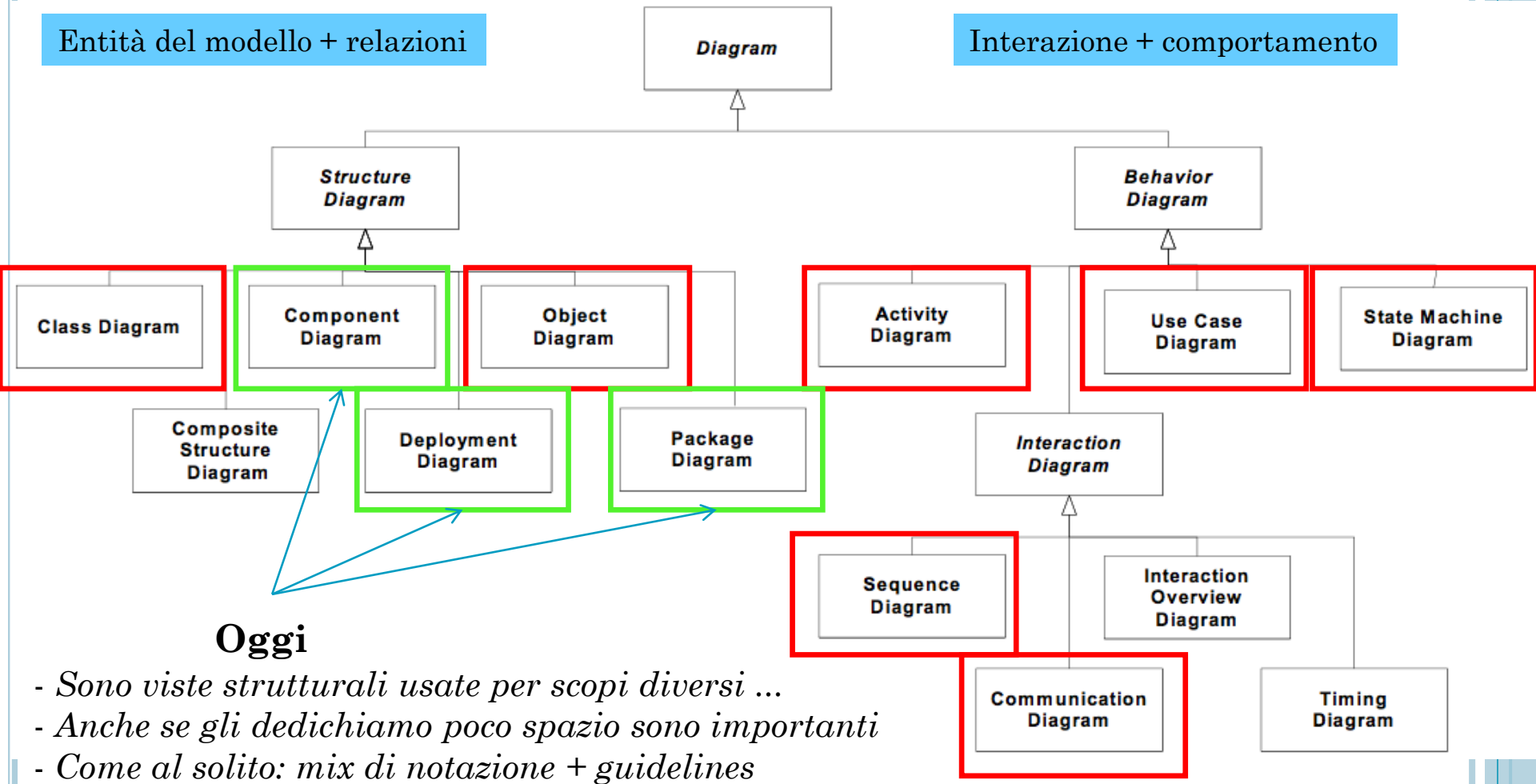
ORIENTIAMOCI …

Modellazione statica

Entità del modello + relazioni

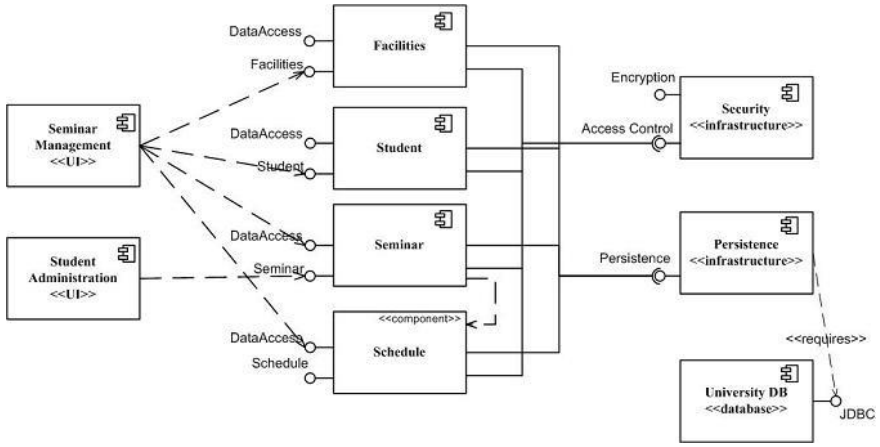
Modellazione dinamica

Interazione + comportamento



Oggi

- *Sono viste strutturali usate per scopi diversi …*
- *Anche se gli dedichiamo poco spazio sono importanti*
- *Come al solito: mix di notazione + guidelines*



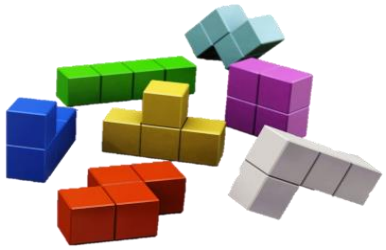
COMPONENT DIAGRAM

Concetto di Componente in UML

- Il termine componente assume un significato (leggermente) diverso a seconda della piattaforma, ambiente, sistema che consideriamo
  - .Net, Java world, …
- In UML un componente è un concetto molto generale:

"Modular unit with well-defined interfaces that is replaceable within its environment"

- Una "**scatola nera**" il cui comportamento esterno è completamente definito dalle sue interfacce
  - Sono **rimpiazzabili** e **componibili**!

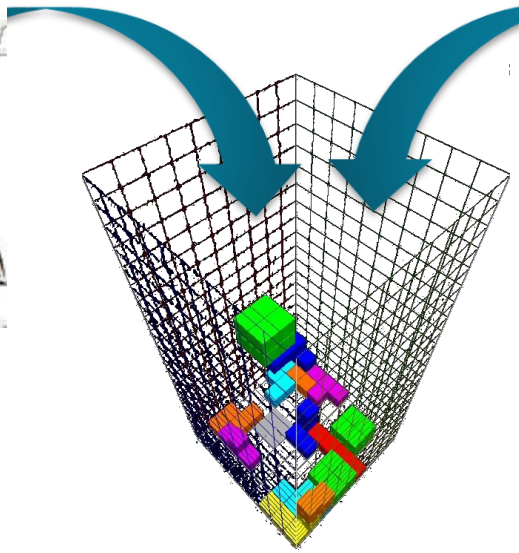


Components = LEGO bricks

Sviluppo basato sui componenti

component marketplace

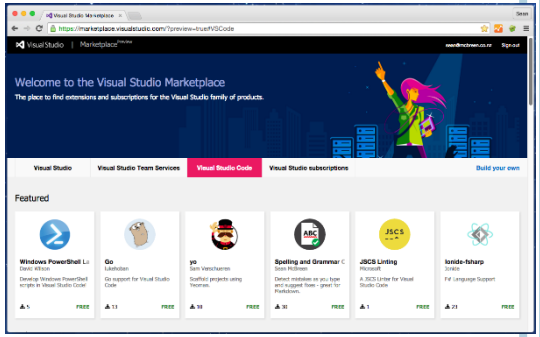
components



glue code

working systems

composition framework ("plumbing" or infrastructure)

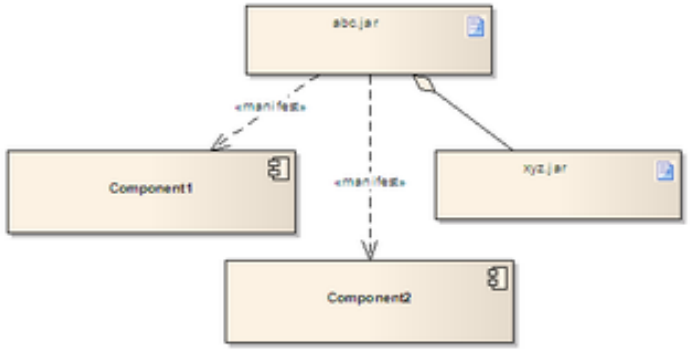


Componenti e Artefatti

- I componenti sono **entità logiche** che sono realizzate da **artefatti** (che sono invece entità fisiche)

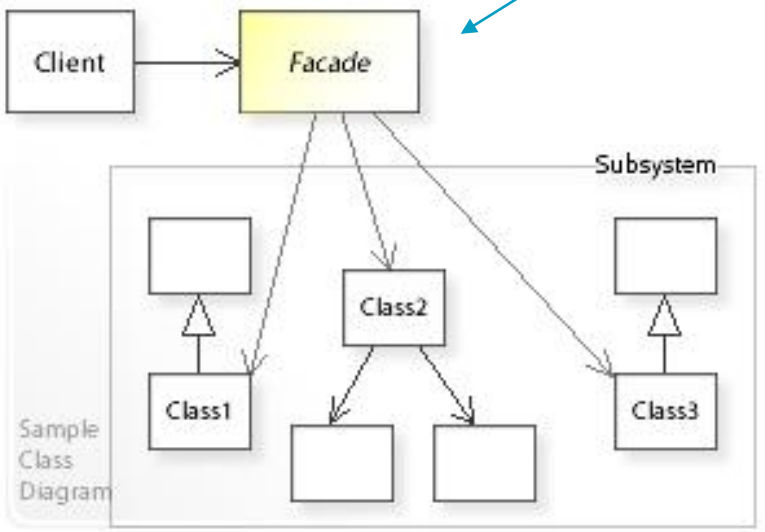
  Es.

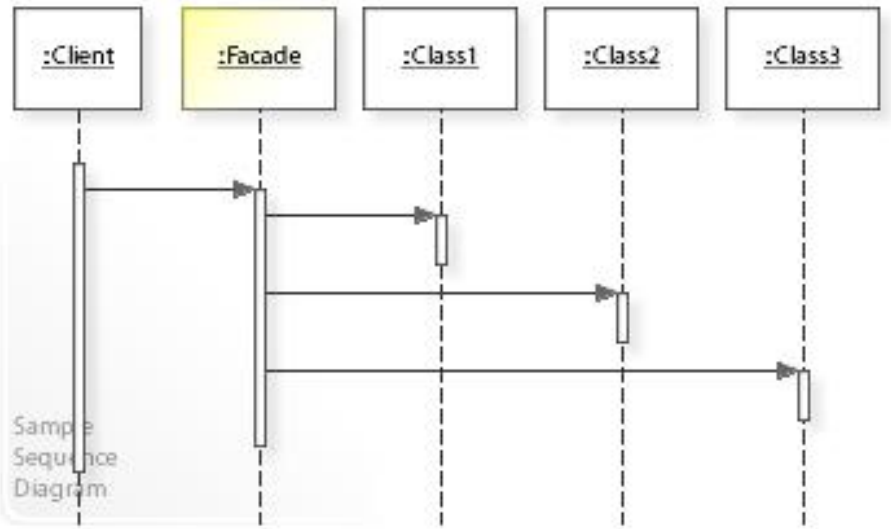
  - **Sottosistema** realizzato tramite un **insieme di classi (file JAR)**
  - **Componente Webratio** realizzato tramite un **file JAR**
  - **Servizio Web REST** realizzato tramite **Jersey / JAX-RS**
  - **Microservice** realizzato con **Spring Boot**
  - **Componente .NET** contenuto in un **.msi package**
    - **Es. EmailVerify**



SOTTOSISTEMA

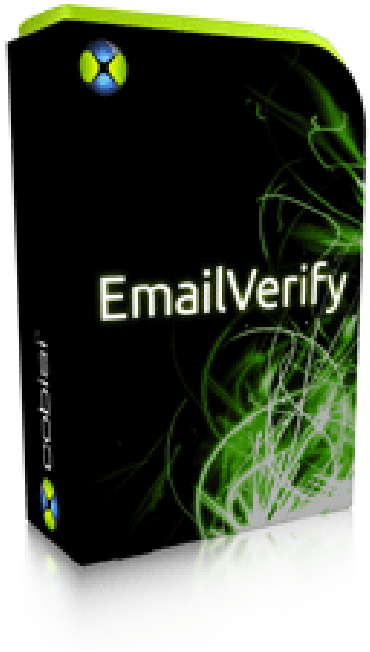
Espone l'interfaccia





Sottoparte di un sistema che può essere eseguita anche a se stante

EmalVerify

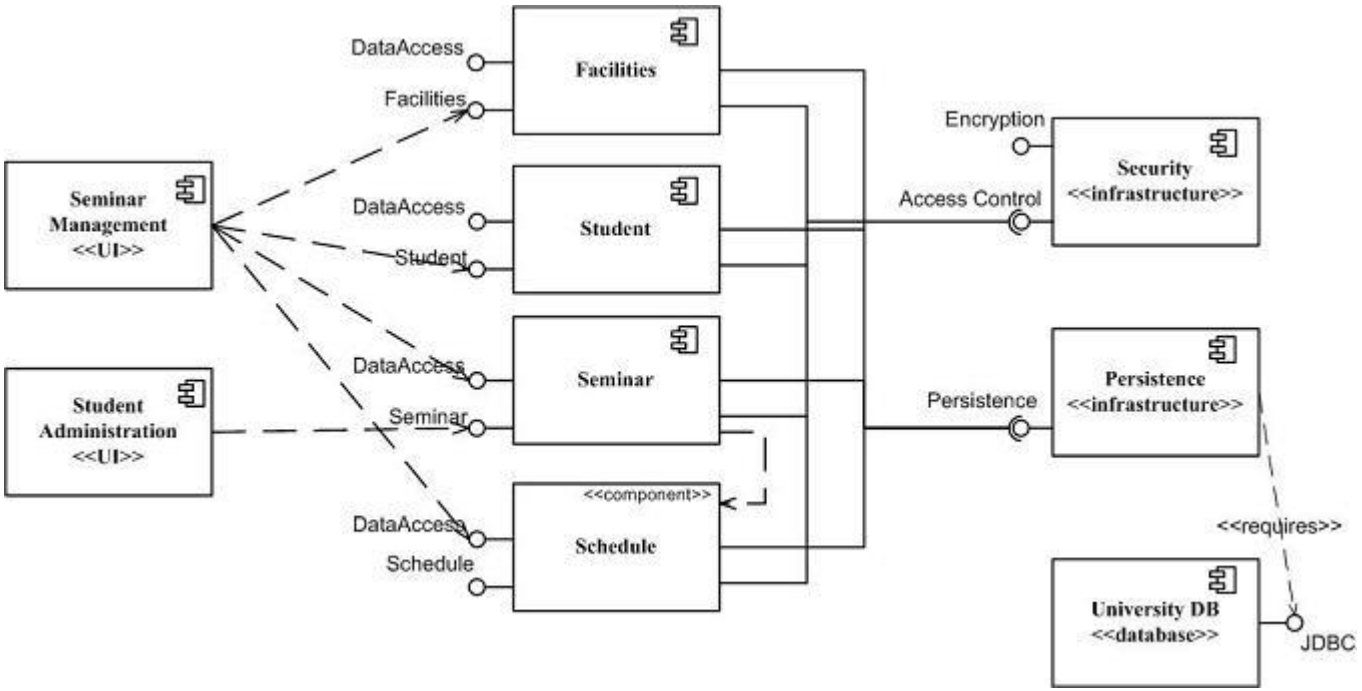



```
var engine = new VerificationEngine();
var result = engine.Run("john@example.com",
                        VerificationLevel.Mailbox).Result;

if (result.LastStatus == VerificationStatus.Success)
{
    // TODO: Show a message box with the great news
}
```

Diagramma delle componenti UML

- Mostrano le componenti che costituiscono il sistema e le dipendenze tra di esse
- Sono utili per **rappresentare l'architettura software** (**High level design**) di un sistema

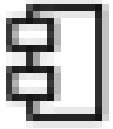


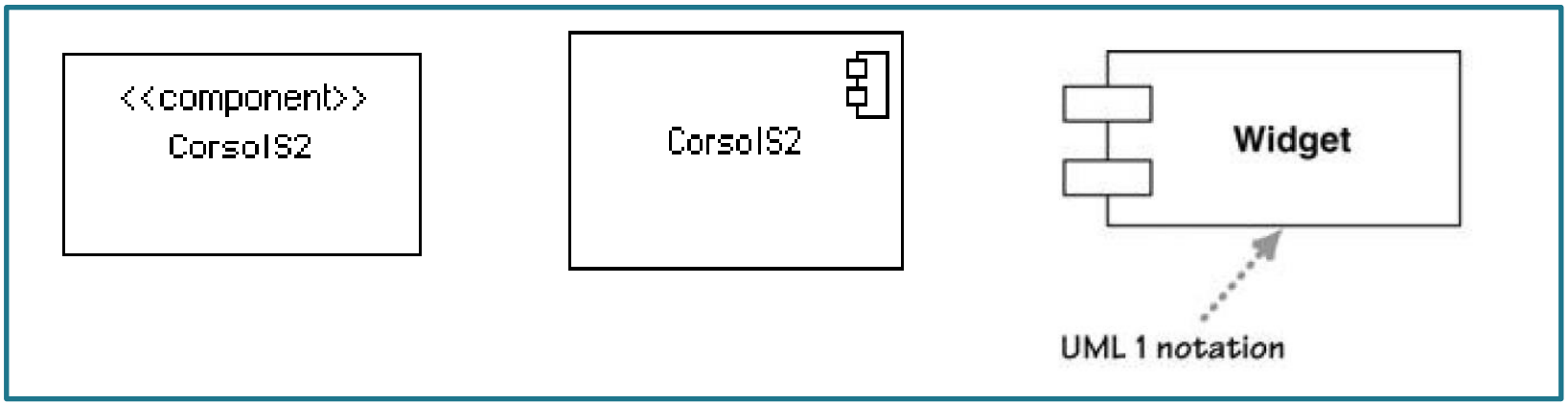
Componenti vs. classi

- Perché si usano le componenti per descrivere l'architettura di un sistema SW?
- Le classi sono componenti di "**grana troppo fine**" per dare una buona panoramica del sistema

Guardando gli alberi si finisce di perdere di vista il bosco!!!

NOTAZIONE (NON PROLISSA)

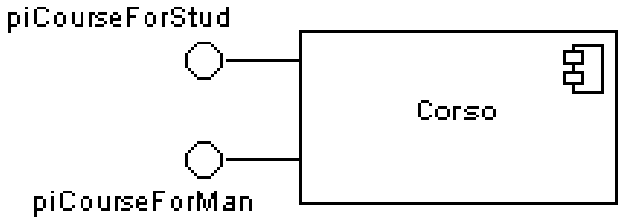
- Un componente è mostrato come un rettangolo con:
  - **Parola chiave <<component>> oppure Icona** 
  - **Nome**



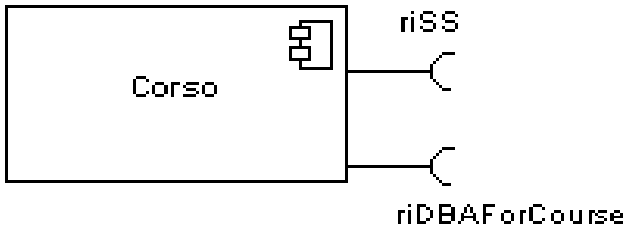
- Esistono parole chiave più specifiche come:
  - <<Web service>>, <<Microservice>>, <<Subsystem>>, ...
  - In più se ne possono sempre aggiungere ....

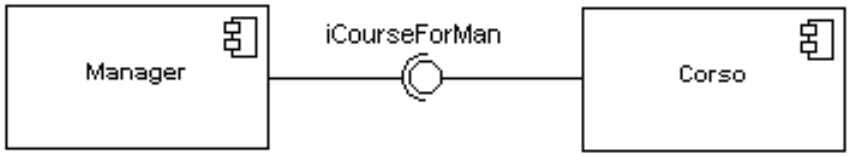
INTERFACCE

- Fornite (**lollipop**)



- Richieste (**socket**)



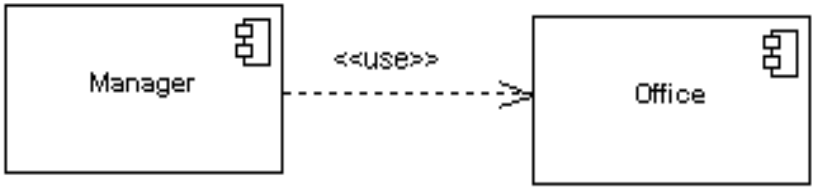


L'interfaccia iCourseForMan **assembla** le due componenti

richiede/usa **fornisce/implementa**

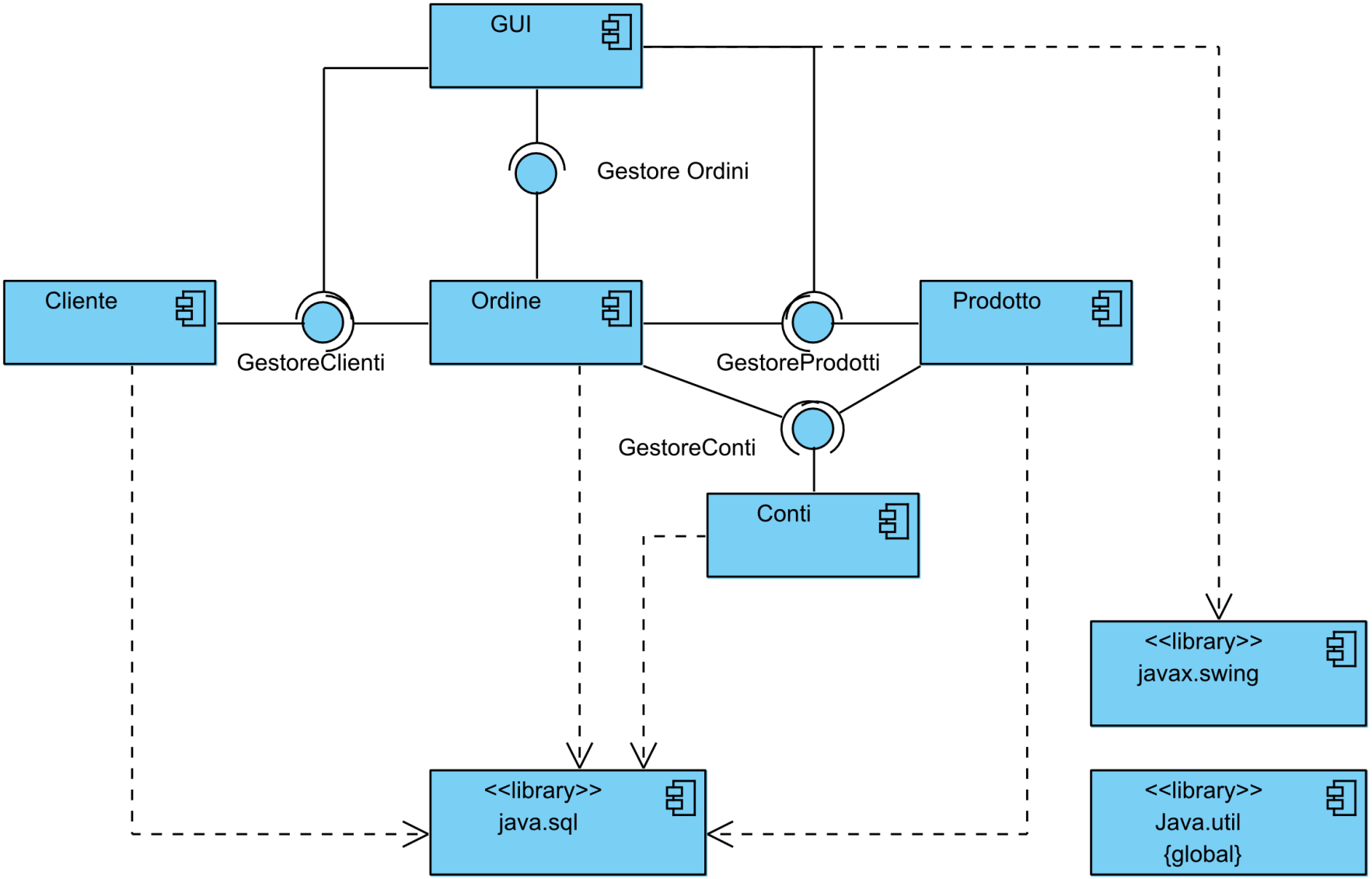
Dipendenza tra componenti

- I componenti possono essere connessi con la relazione di dipendenza

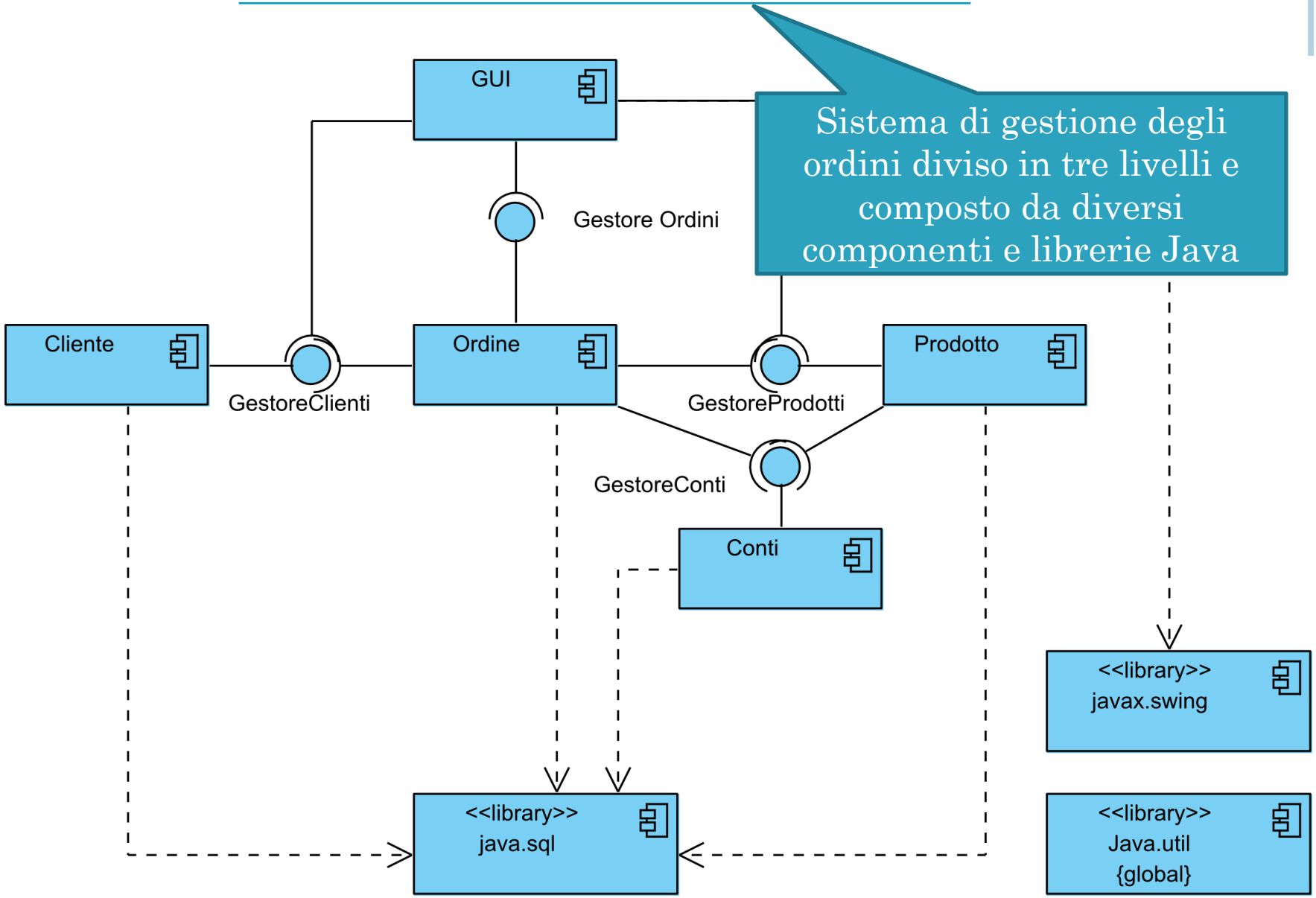


- Solito significato: il componente **Manager** ha bisogno di **Office** per poter funzionare
- Spesso usato quando sono utilizzate più interfacce ma non si vuole specificare quali sono ...
  - Altrimenti si usa **lollipop** e **socket**

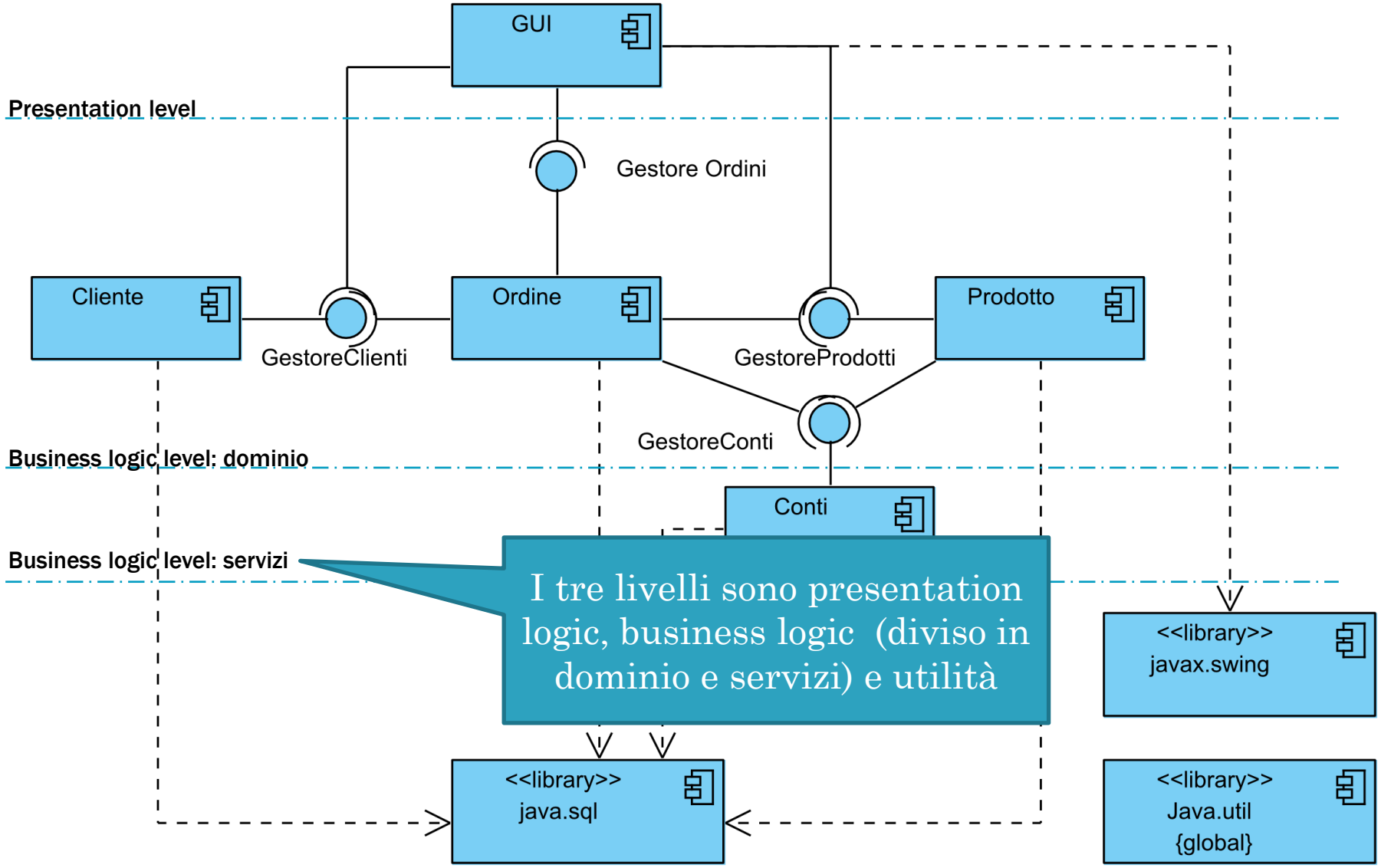
Esempio: sistema gestione ordini



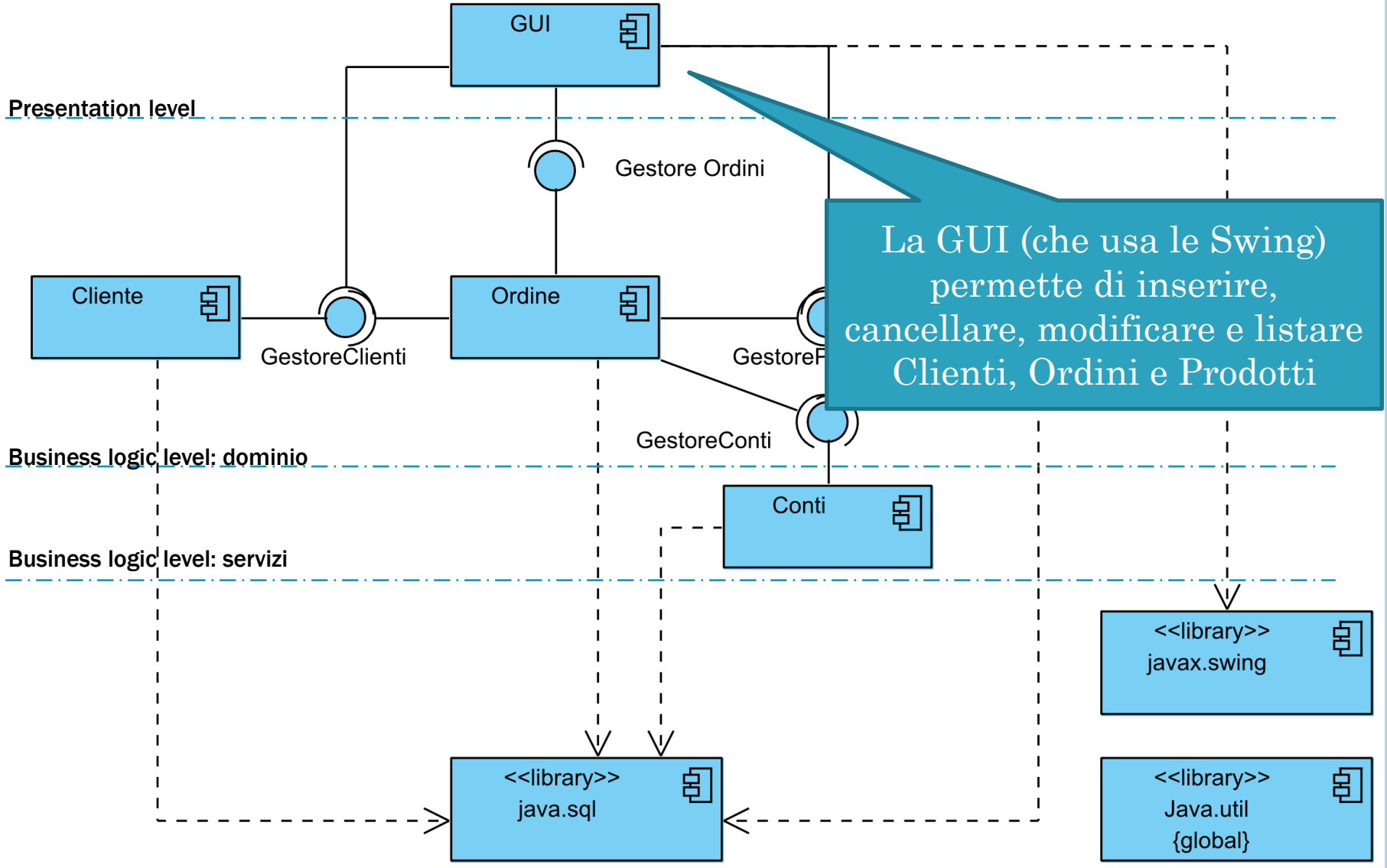
ESEMPIO: SISTEMA GESTIONE ORDINI



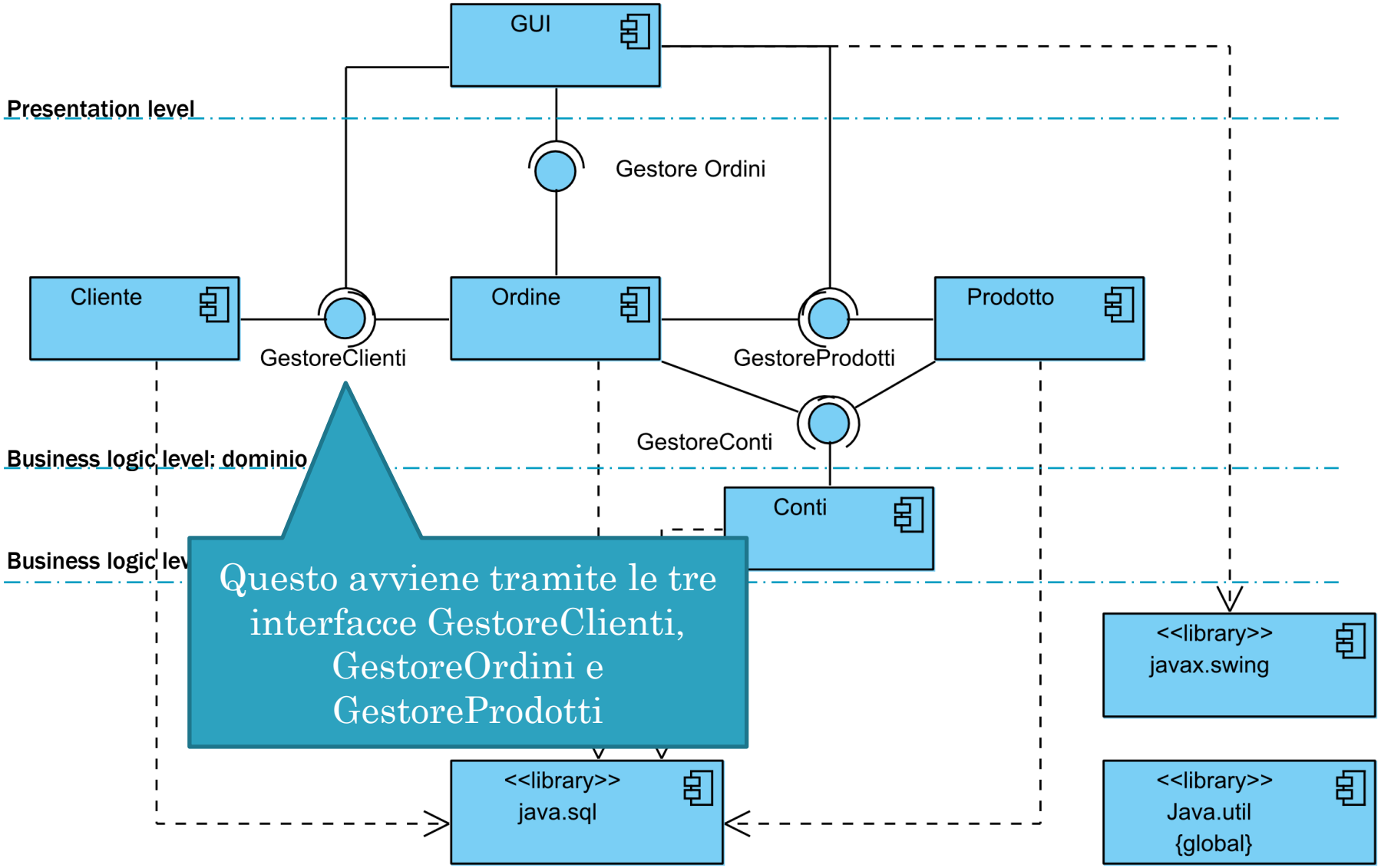
ESEMPIO: SISTEMA GESTIONE ORDINI



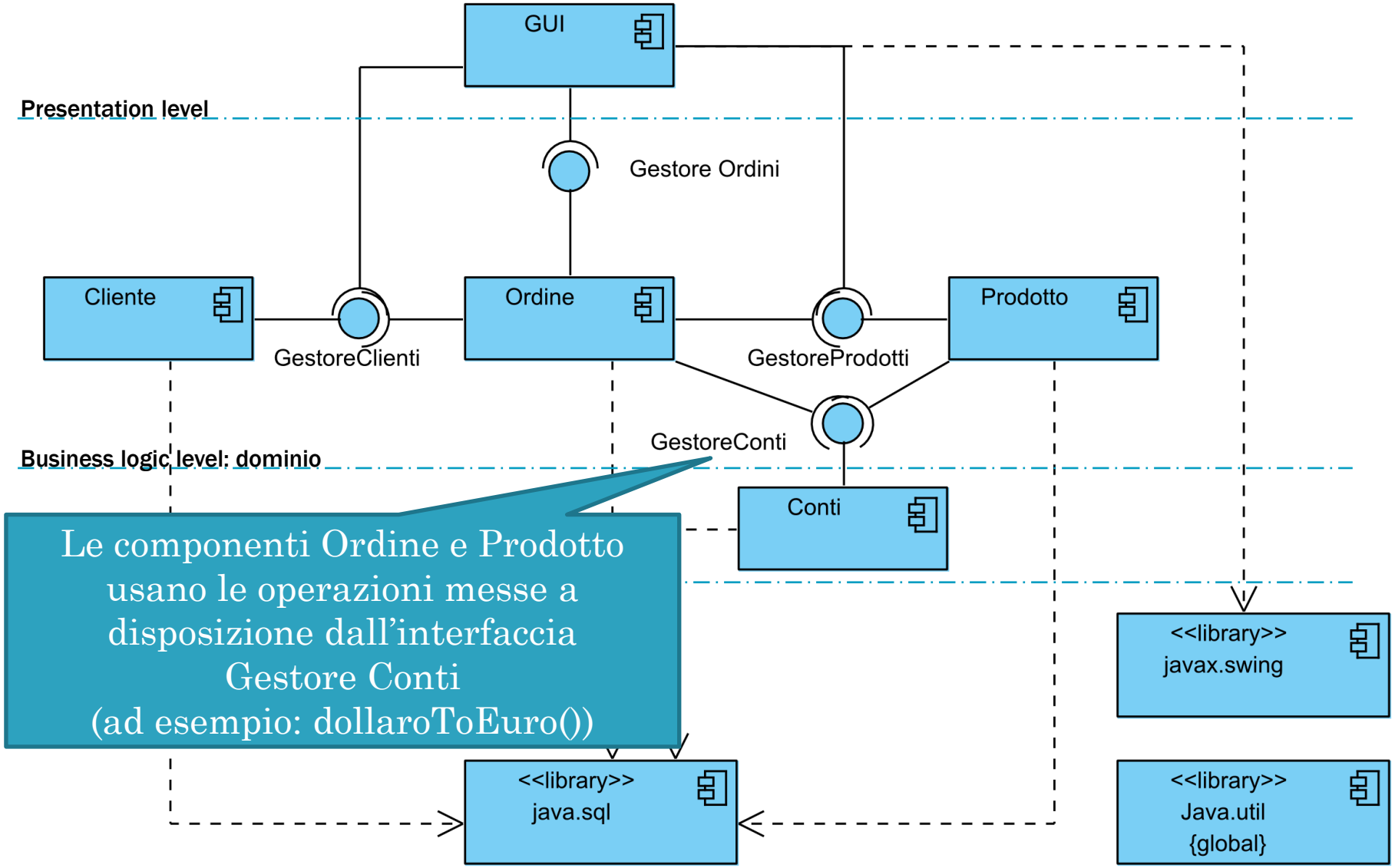
ESEMPIO: SISTEMA GESTIONE ORDINI



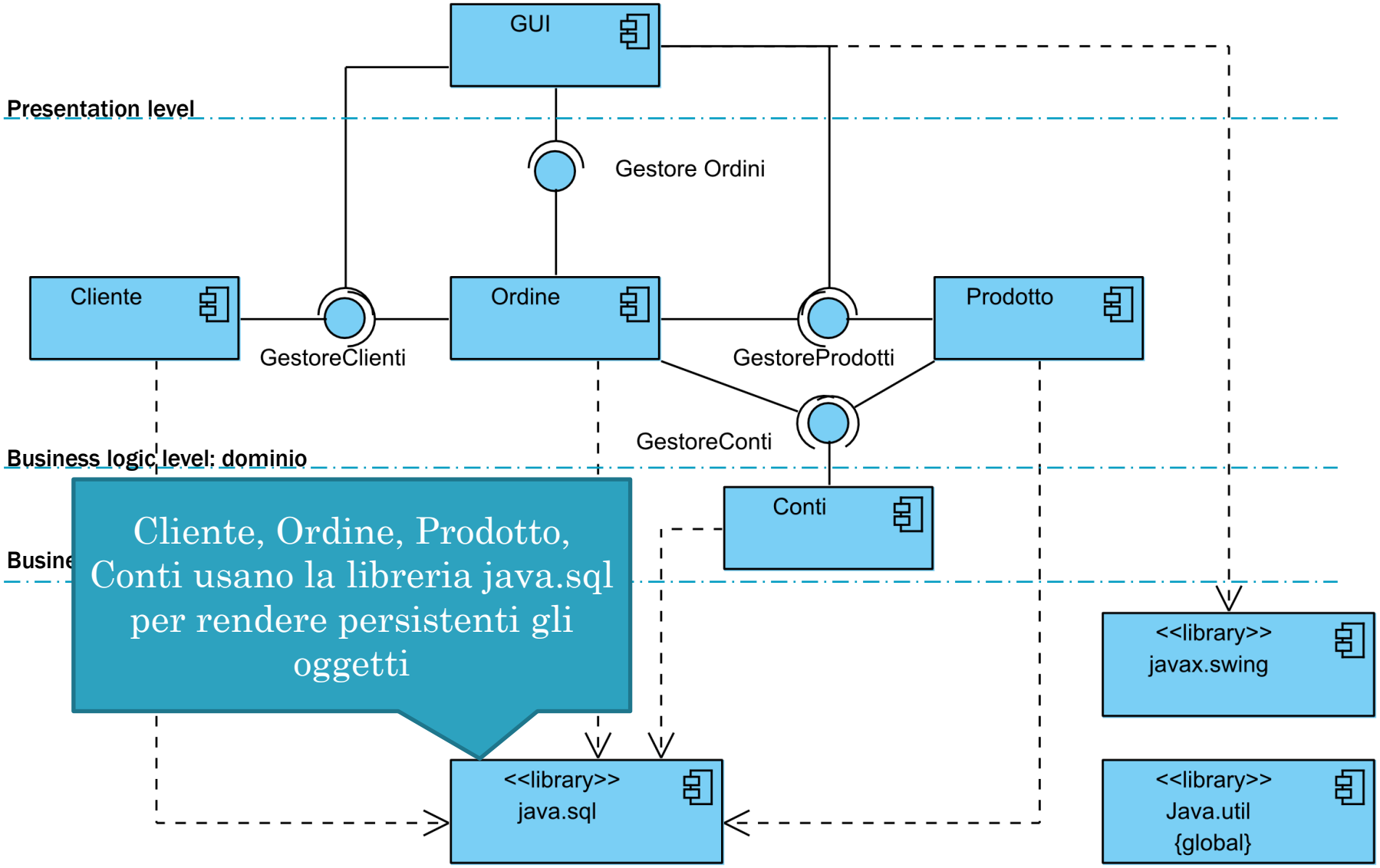
ESEMPIO: SISTEMA GESTIONE ORDINI



ESEMPIO: SISTEMA GESTIONE ORDINI

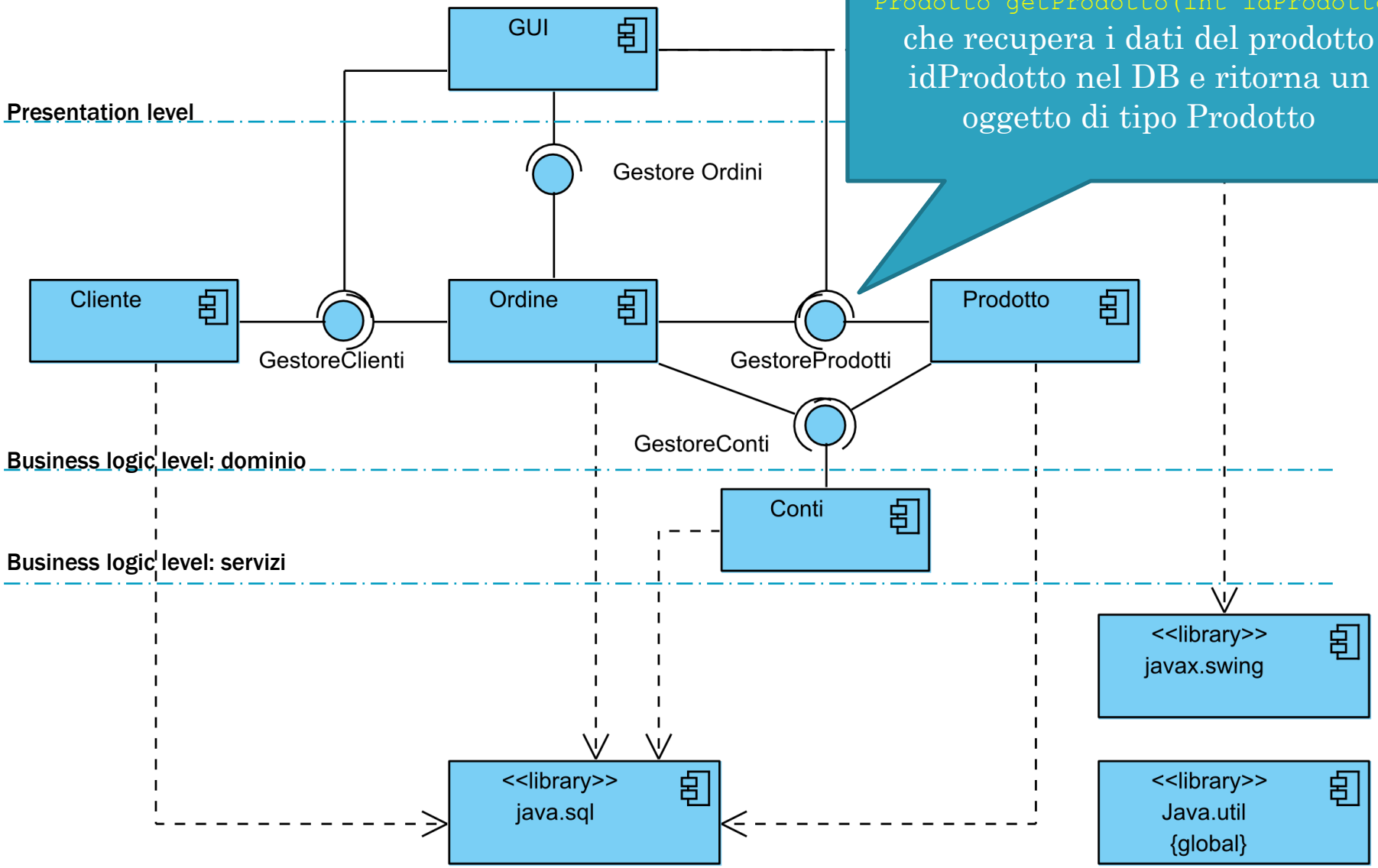


ESEMPIO: SISTEMA GESTIONE ORDINI

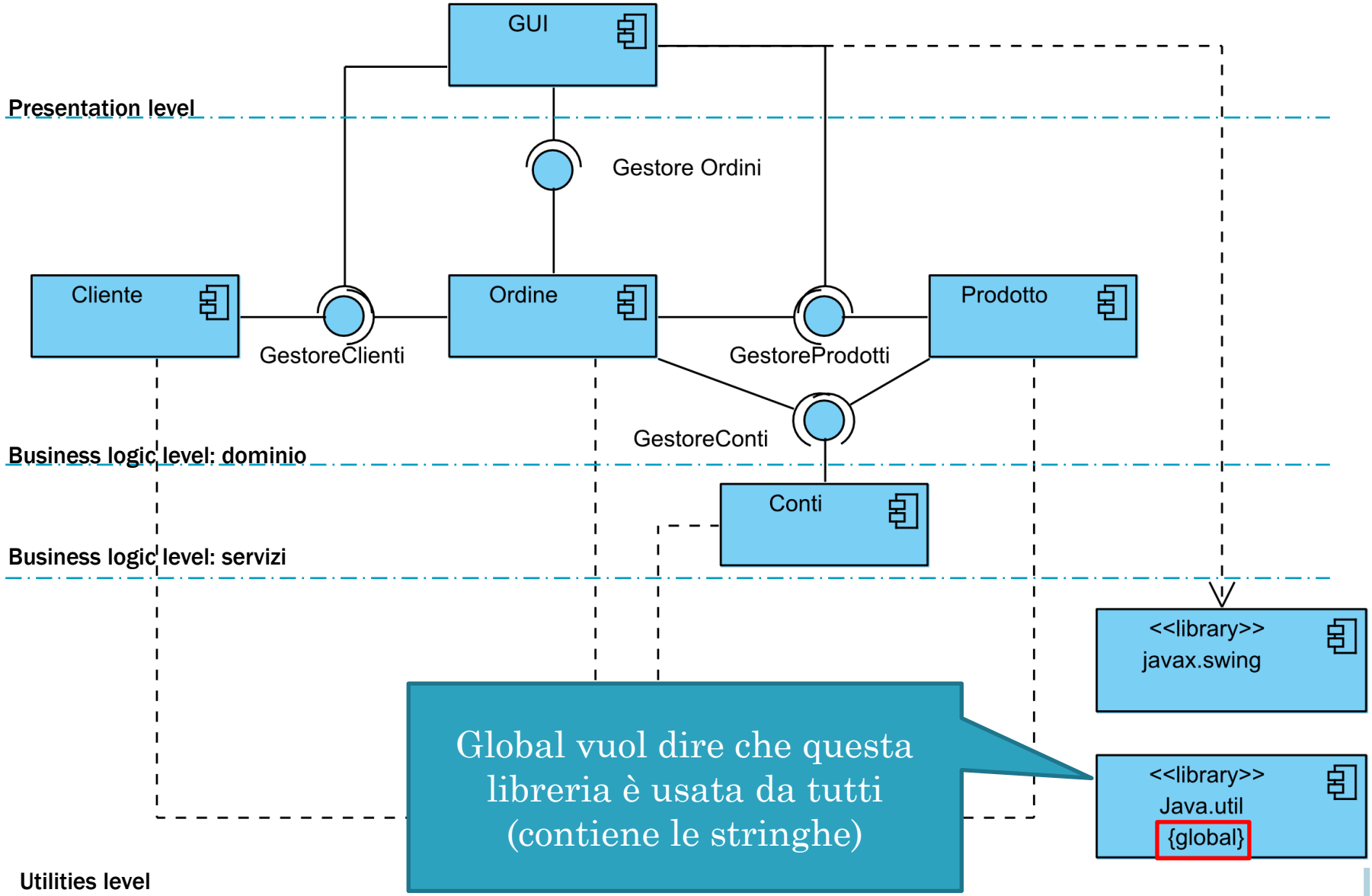


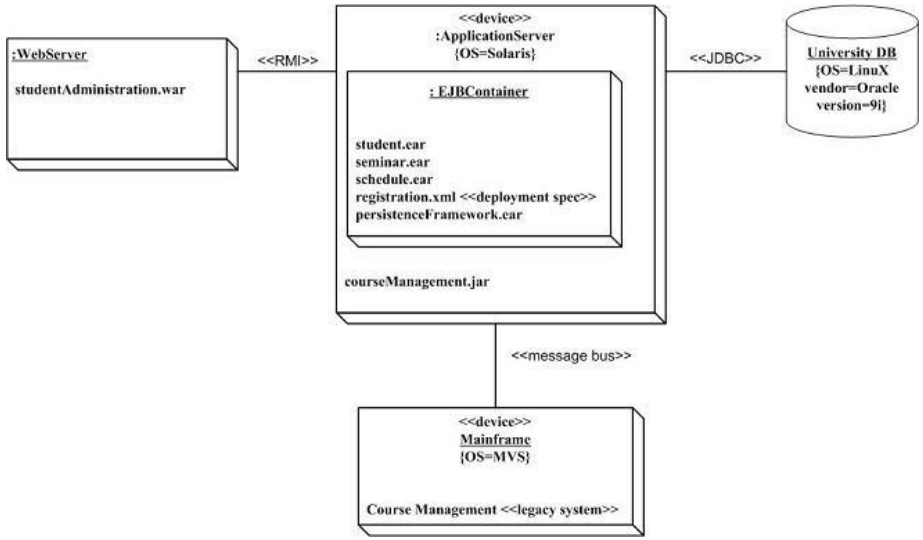
ESEMPIO: SISTEMA GESTIONE

Ad esempio GestoreProdotti fornisce un operazione
`Prodotto getProdotto(int idProdotto)`
che recupera i dati del prodotto idProdotto nel DB e ritorna un oggetto di tipo Prodotto



ESEMPIO: SISTEMA GESTIONE ORDINI

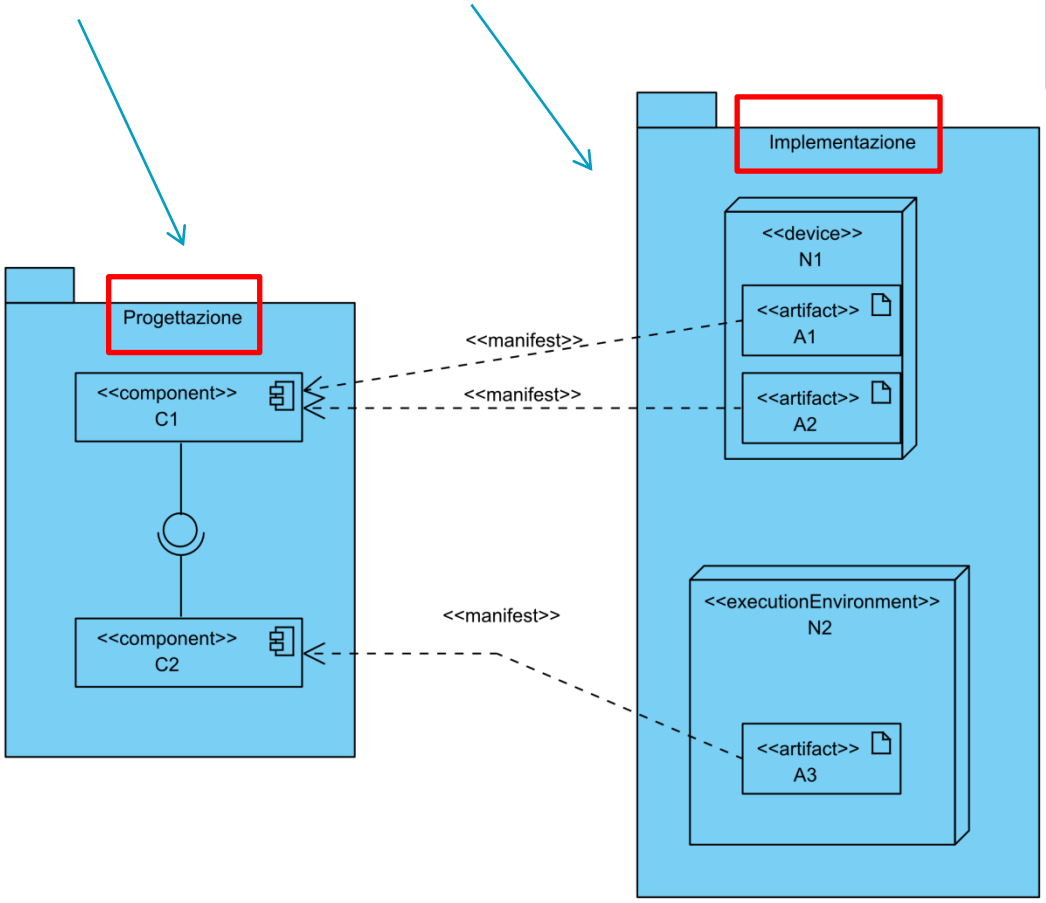




DEPLOYMENT DIAGRAM

Deployment diagram

- Vista **molto implementativa**
- Mostra la **relazione tra hardware e software** in un sistema
- Forte link tra **component** e **deployment diagram**

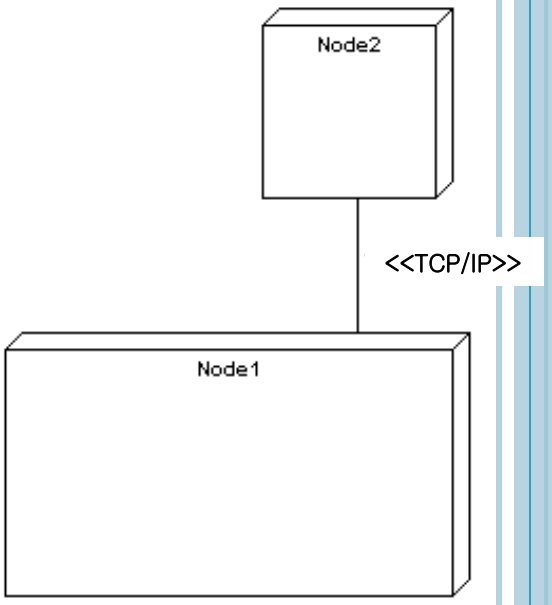


Ogni nodo rappresenta "qualcosa" che può "ospitare" del software

DEPLOYMENT DIAGRAM (NODI E CONNESSIONI)

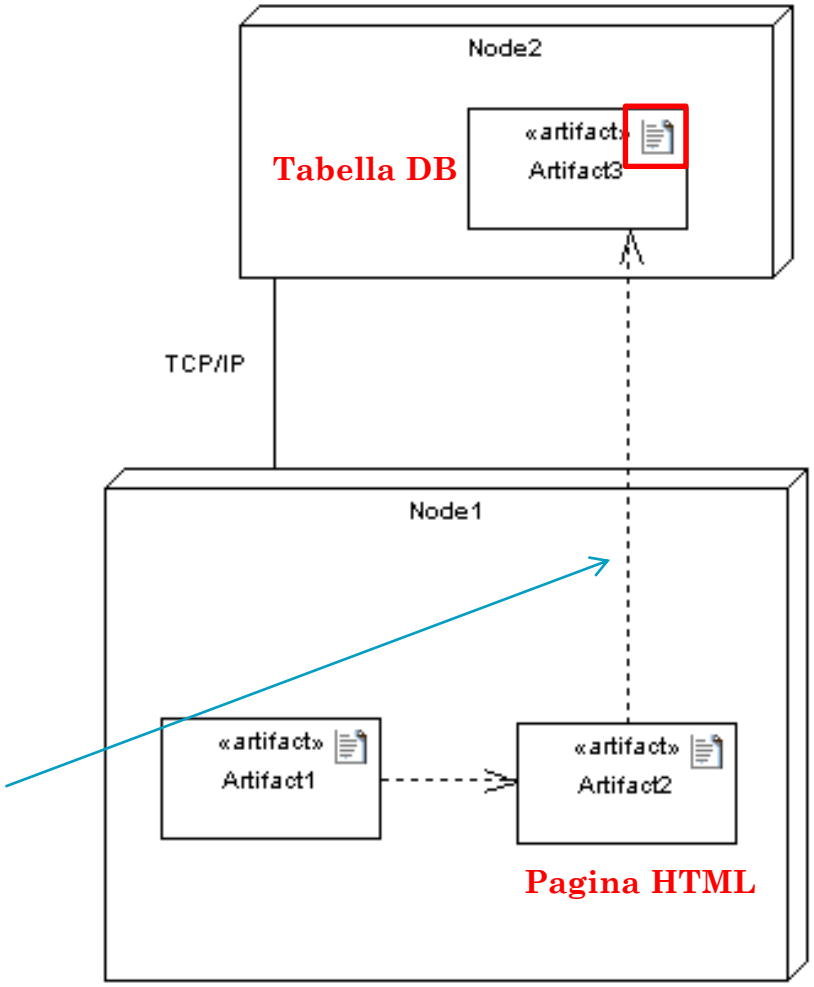
- Contiene **nodi** e **connessioni**
- Un nodo rappresenta un tipo di **risorsa computazionale**
  - Una periferica fisica (**<<device>>**)
    - per esempio un PC o un server
  - Un'**ambiente software di esecuzione**
    - per esempio un browser, una macchina virtuale, o un Docker container ...

- Una connessione tra nodi rappresenta un **canale di comunicazione** attraverso cui possono passare delle informazioni
  - Usualmente si indica il tipo di connessione
    - Es. TCP/IP



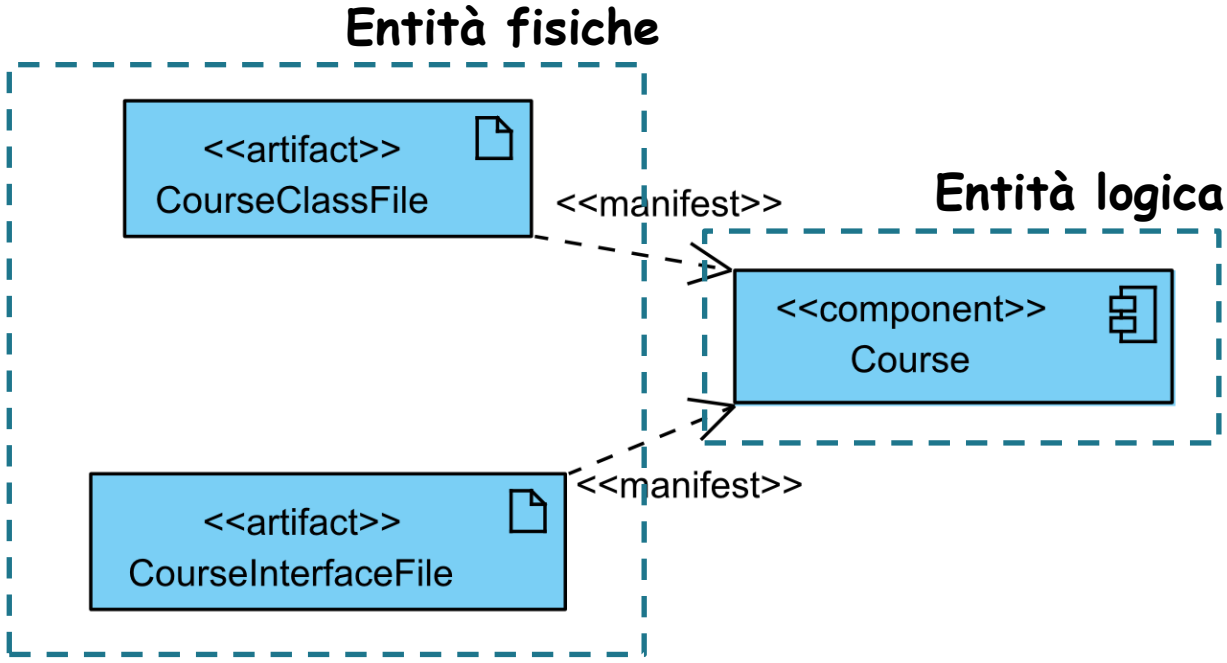
Artefatti

- Sono **entità concrete** del mondo reale
- Per esempio:
  - **file di codice sorgente, file eseguibili, script, tabelle in un database, pagine HTML ...**
- Possono essere **dislocati** ('deployati') sui nodi
- Esiste la relazione di dipendenza tra artefatti

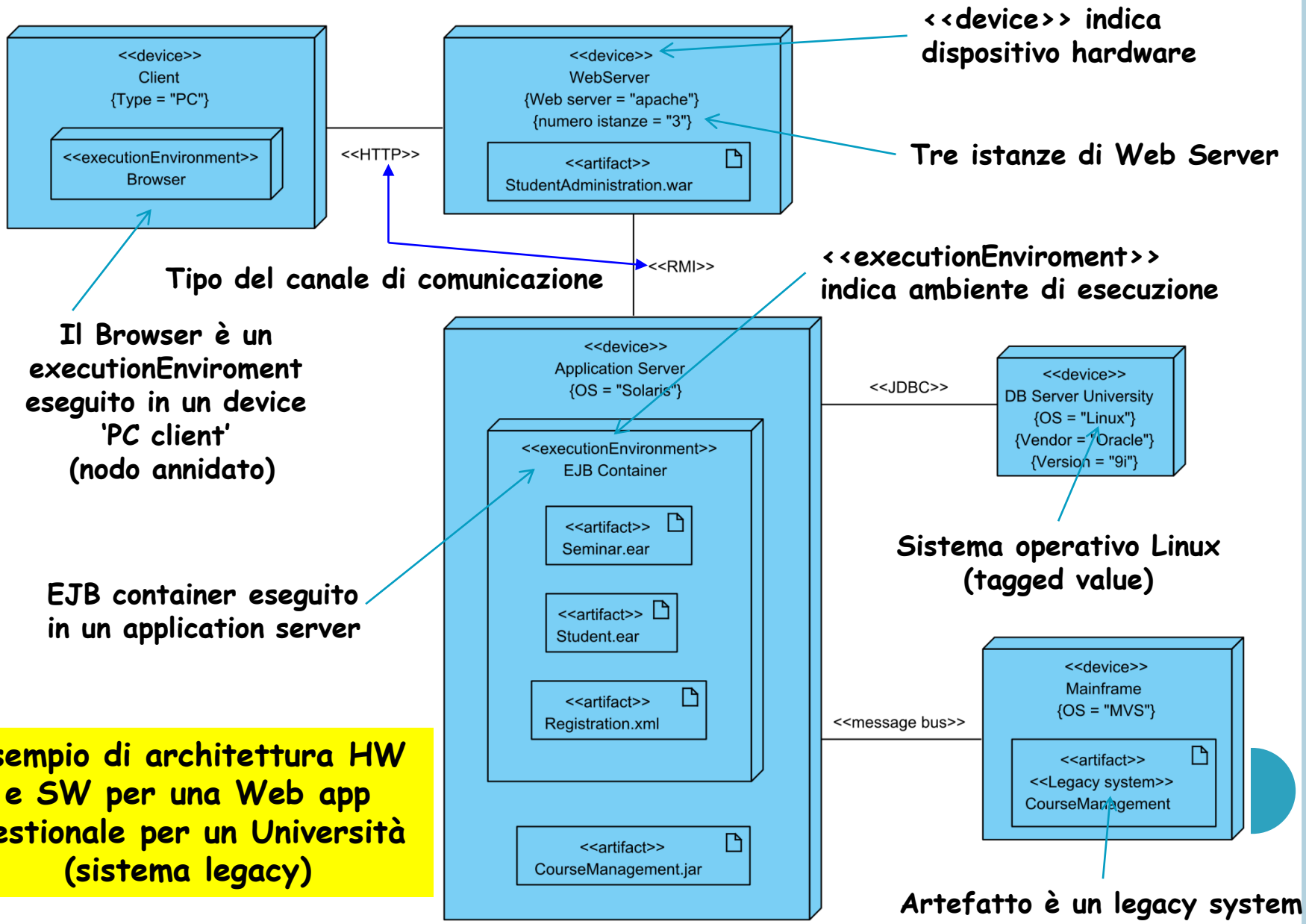


MANIFEST

- La relazione "**manifest**" indica che gli artefatti sono **rappresentazioni/manifestazioni fisiche** dei componenti
  - Per esempio: un componente può includere una classe e un interfaccia, realizzate da due file che contengono il codice sorgente



NODI ANNIDATI E TAGGED VALUES



ESEMPIO/ESERCIZIO: OFFICINA MECCANICA

- Si supponga di dover decidere l'architettura software (cioè le **componenti**) e hardware (cioè i **nodi**) per un software che gestisce un **officina meccanica**
- Due tipi di utilizzatori:
  - **Meccanici** che devono richiedere i pezzi di ricambio per ogni intervento
    - Operazione: **richiesta pezzi**
  - **Magazzinieri** che devono fornire i pezzi
    - Ma devono anche tenere il magazzino sempre con le dovute scorte facendo le richieste ai fornitori quando i pezzi scarseggiano
    - Operazione: **carico e scarico pezzi dal magazzino**

Esercizio 1 dispense IS Ambriola, Montangero, Semini

COME DOVREBBE FUNZIONARE IL SOFTWARE?

In un officina ogni meccanico ha associato una lista di interventi.

Dopo avere scelto un intervento tramite la GUI, il meccanico analizza il veicolo e identifica le operazioni da effettuare e i pezzi di ricambio necessari per l'intervento.

Inserisce (tramite GUI) nel sistema la richiesta pezzi e la spedisce al magazziniere

Il magazziniere periodicamente controlla la lista di richieste pezzi dalla sua GUI.

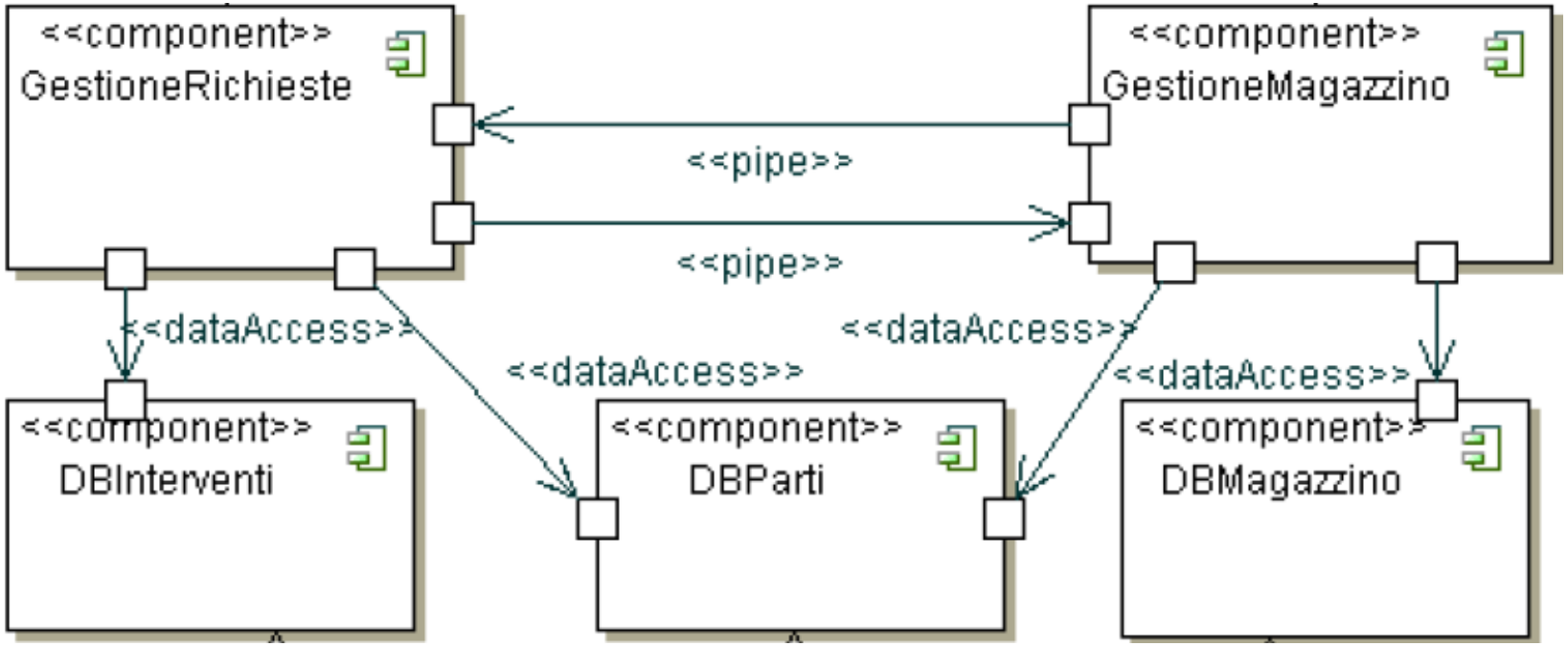
Seleziona la prima richiesta, verifica la disponibilità dei pezzi e se ci sono: prepara i ricambi, li scarica dal magazzino e avverte il meccanico.

Se le scorte sono basse o mancano i pezzi esegue le richieste ai fornitori

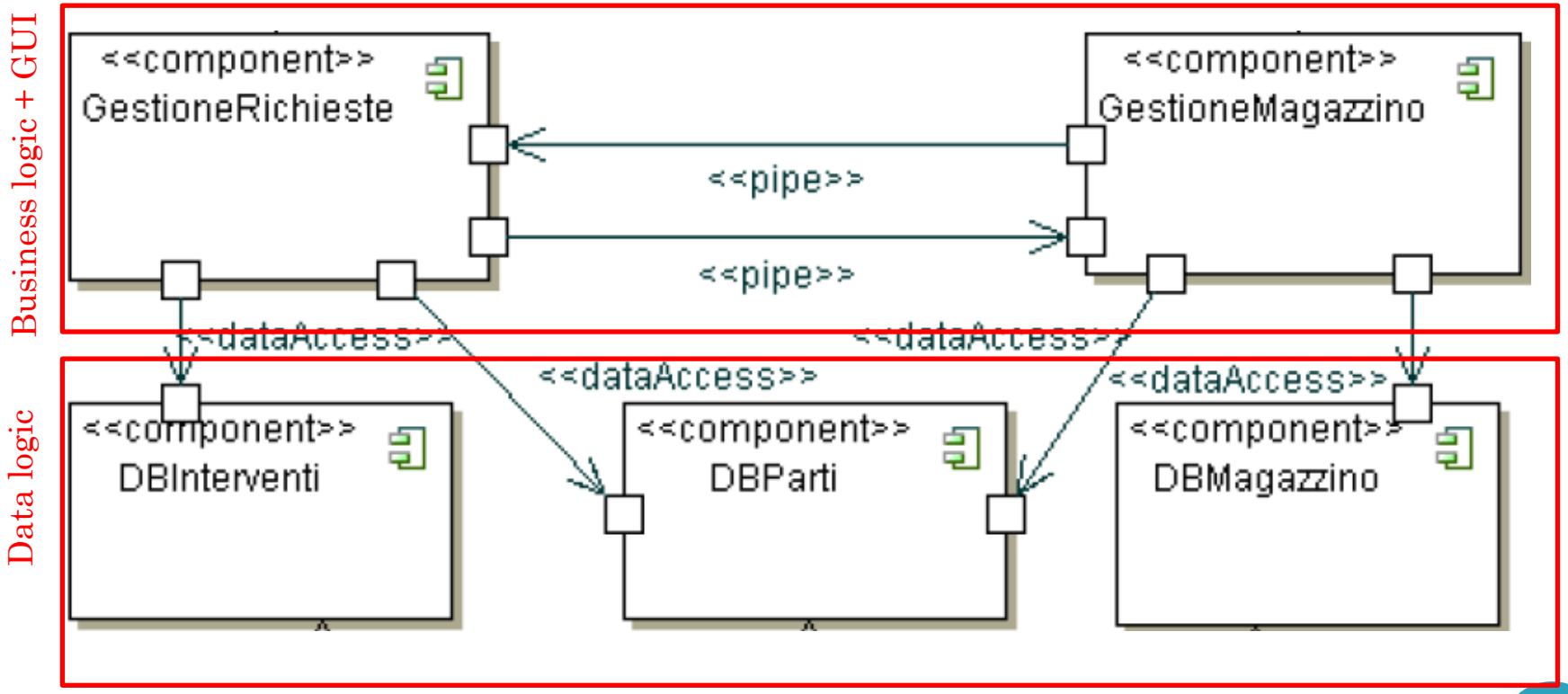
Possibile Architettura hardware (deployment Diagram)

'Tre Personal Computer in rete …'

Possibile Architettura Software (Component Diagram)



Possibile Architettura Software (Component Diagram)

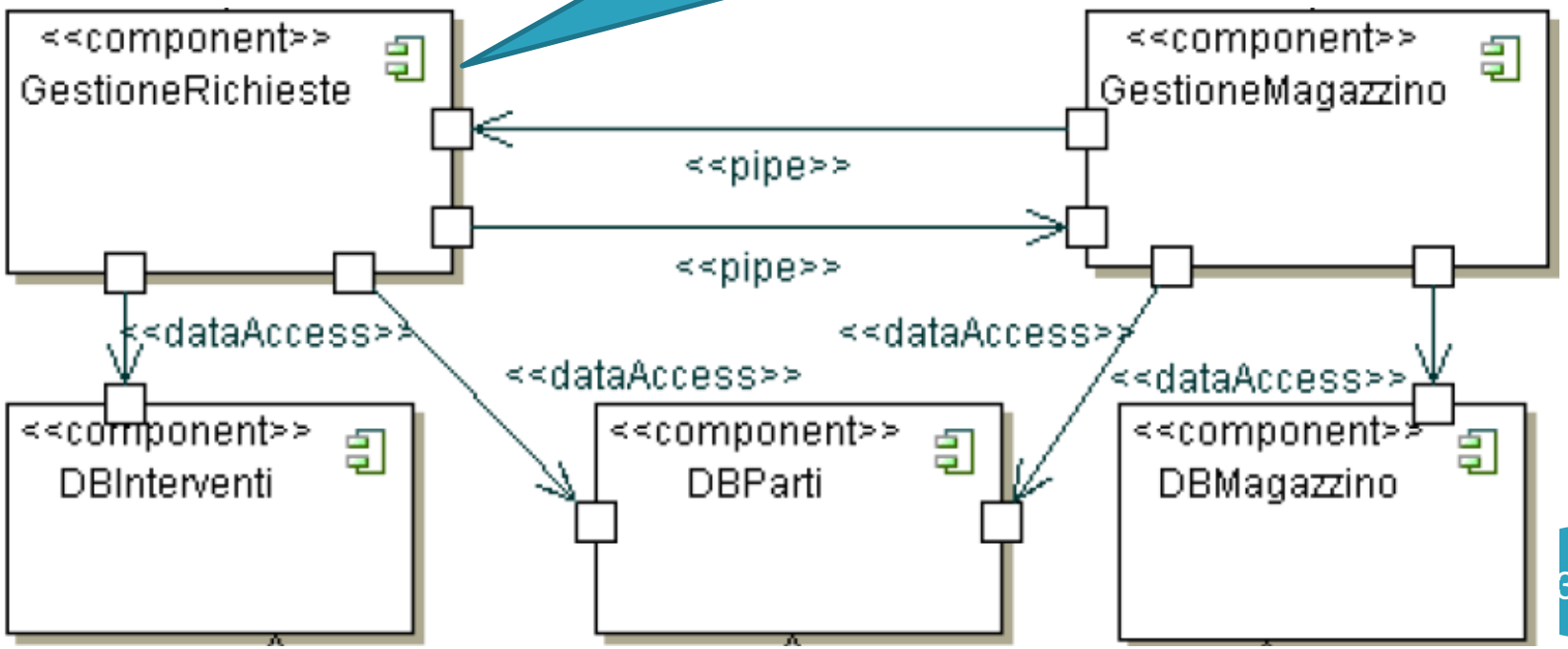


Gestore dati

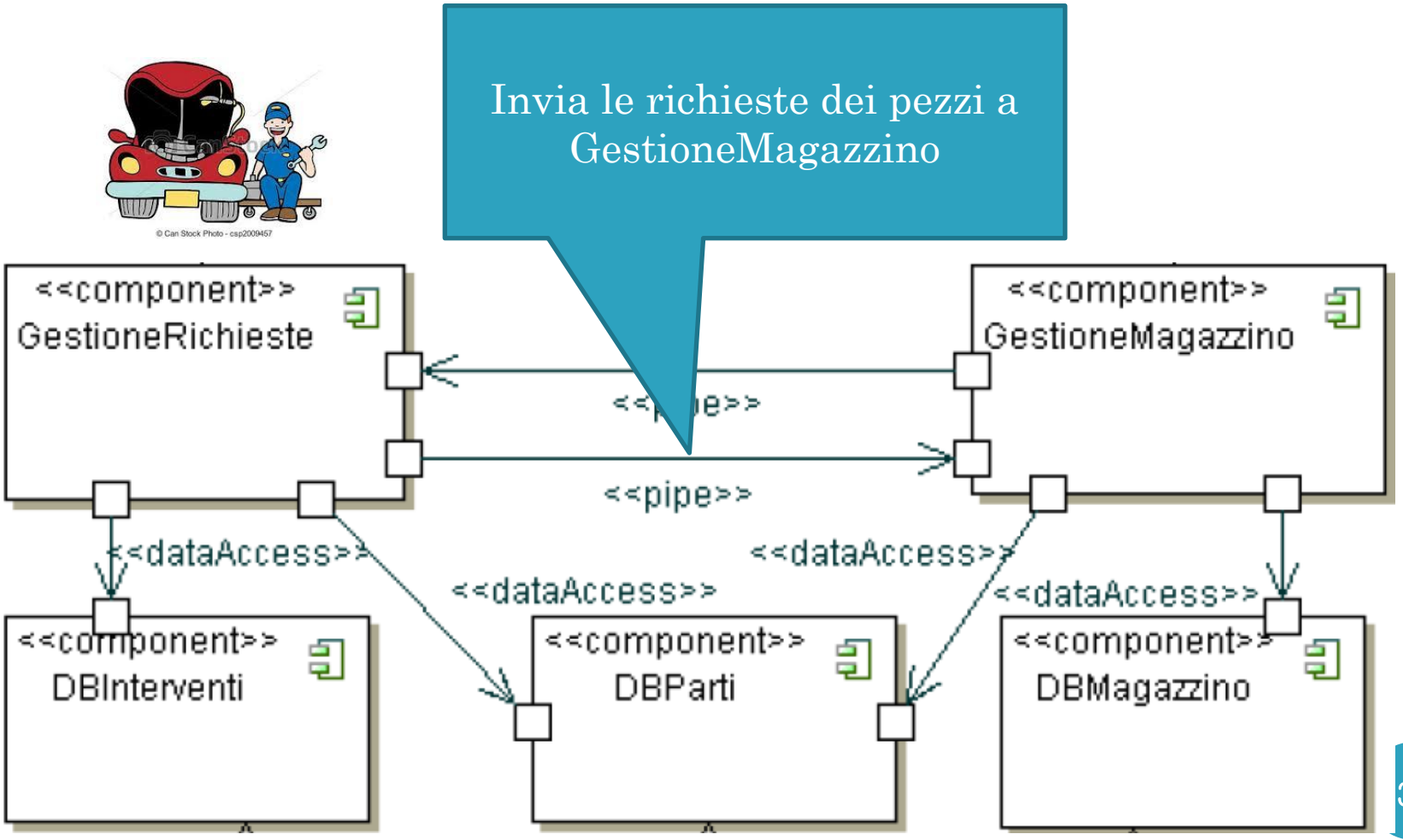
Sarebbe stato meglio una suddivisione su tre livelli (GUI ,Business Logic e Data Logic)

Possibile Architettura Software (Component Diagram)

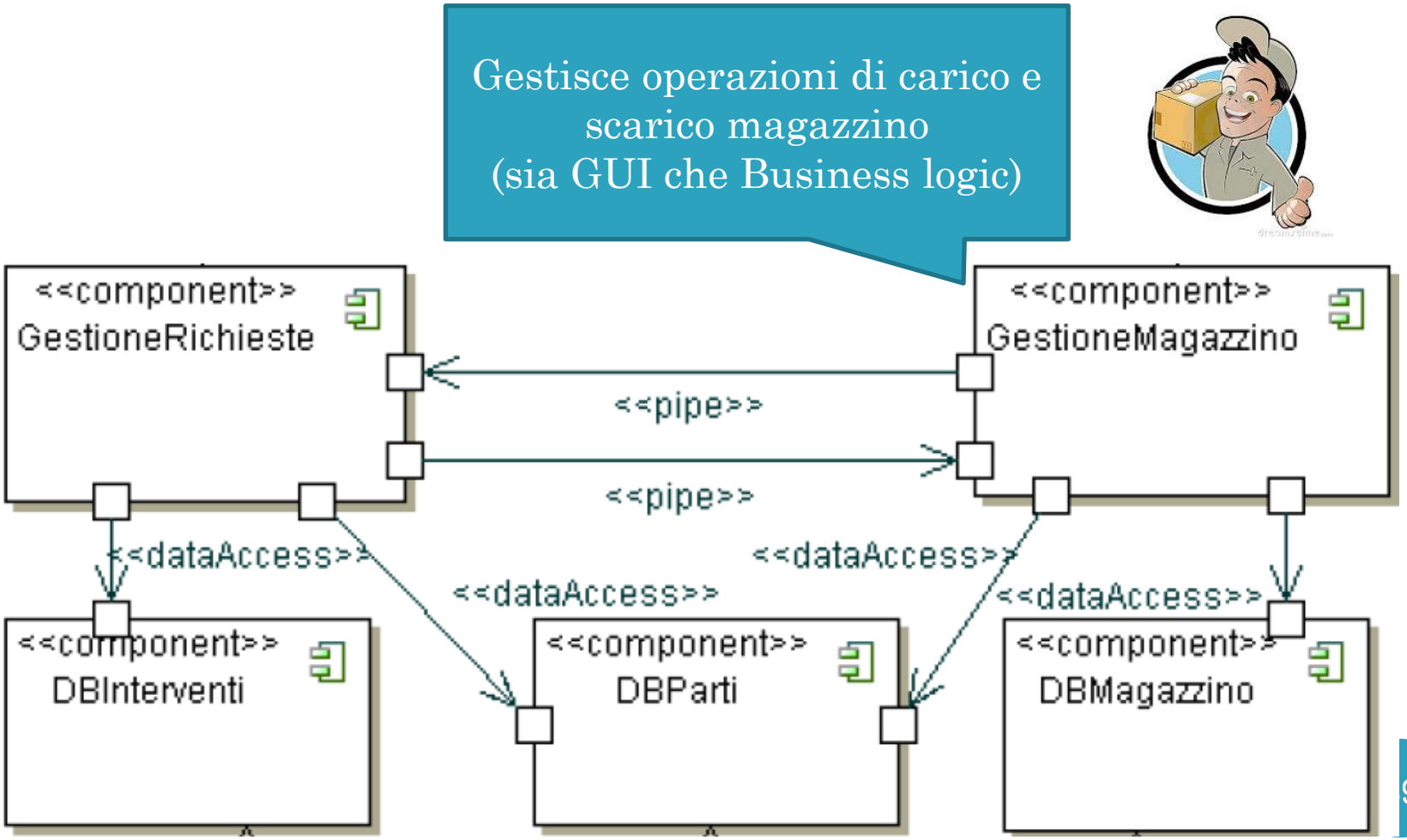
Gestisce le richieste dei pezzi da parte dei meccanici (sia GUI che Business logic)



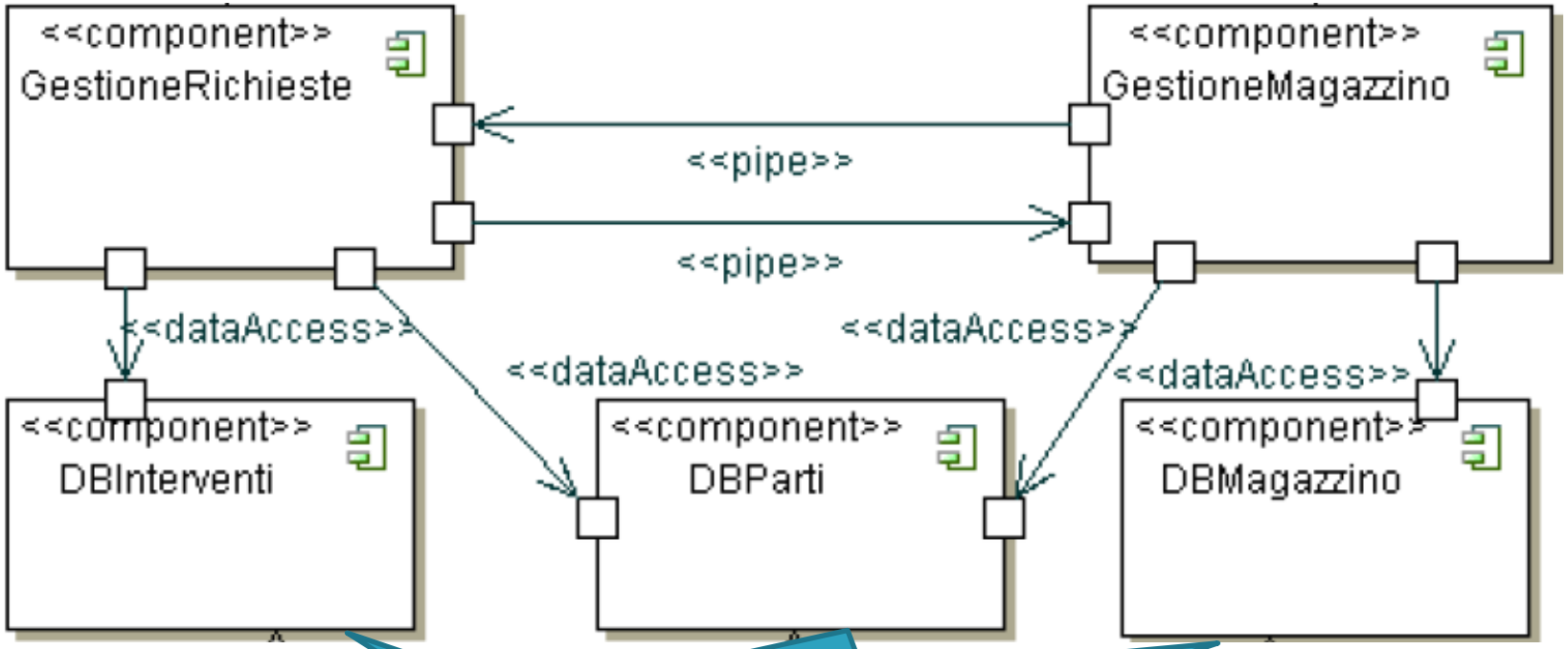
Possibile Architettura Software (Component Diagram)



Possibile Architettura Software (Component Diagram)



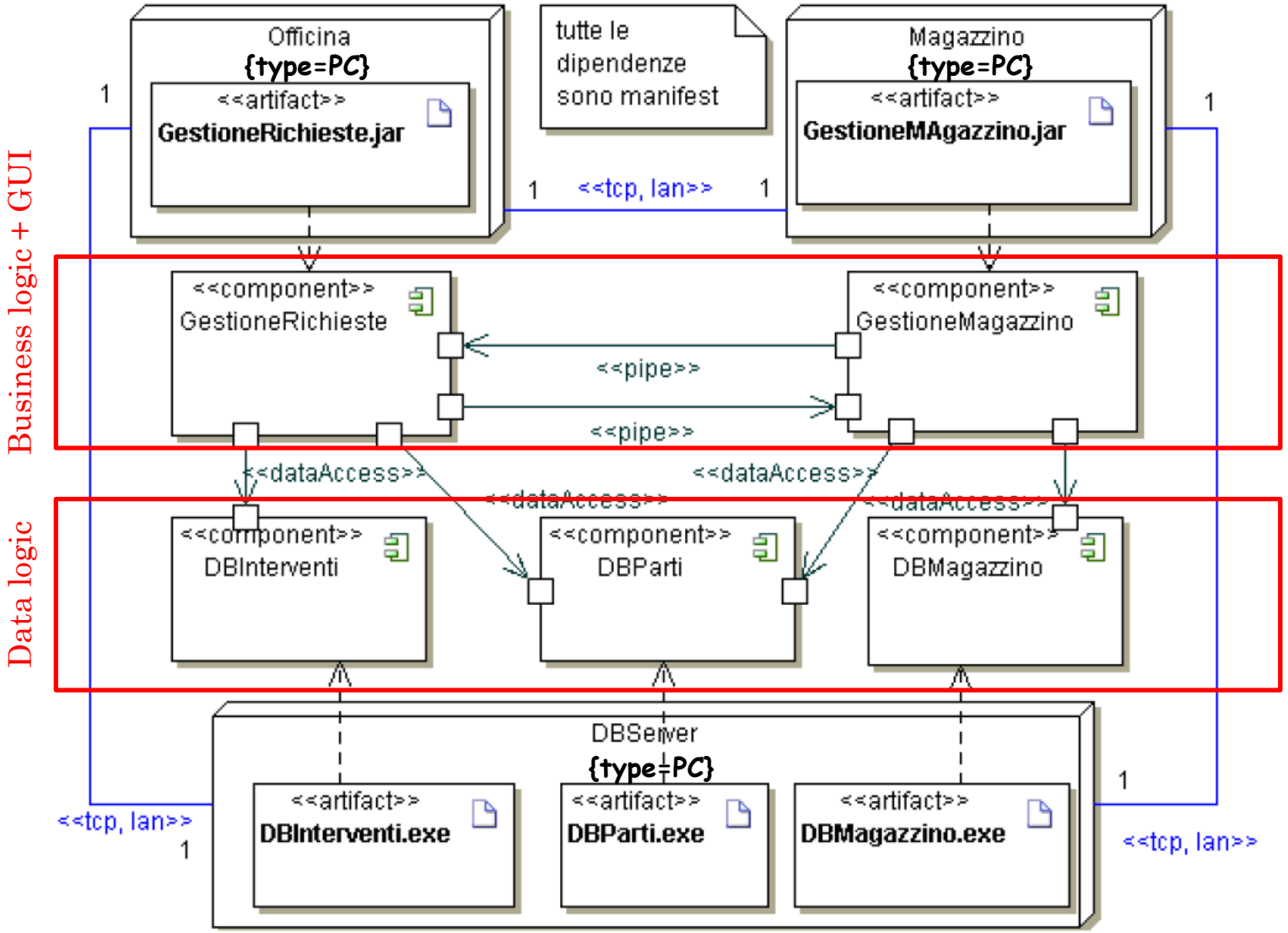
Possibile Architettura Software (Component Diagram)

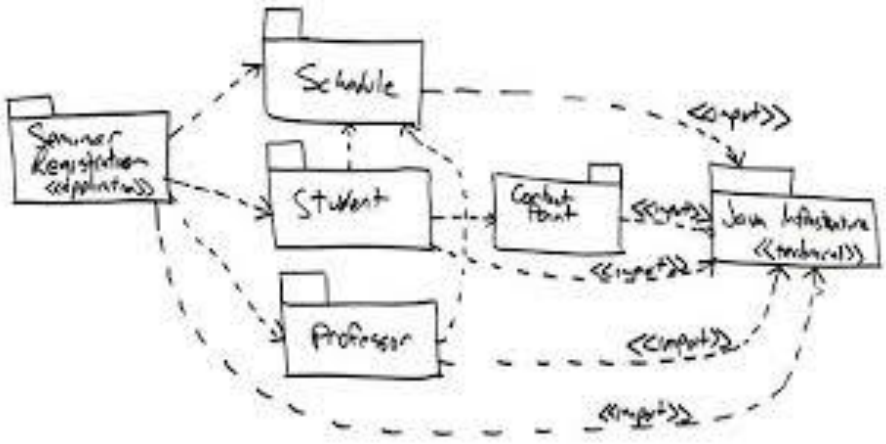


Gestiscono la persistenza (e il recupero) degli interventi, pezzi di ricambio e quantità in magazzino

COMPONENT + DEPLOYMENT DIAGRAM

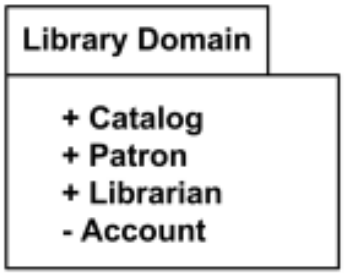
Spesso può essere utile integrare i due diagrammi …





PACKAGE DIAGRAM

PACKAGE



- Un package (in UML) è un **costrutto** che permette di prendere un numero arbitrario di elementi UML e **raggrupparli** insieme
  - Elementi UML = classi, casi d'uso, modelli UML, …
    - Noi principalmente li vedremo come contenitori di classi
- Un package può contenere sia elementi (es. classi) che sotto-package
  - Gerarchie di package
- Concetto molto simile ai package nei linguaggi di programmazione (es. Java)
  - In UML: **"concetto + generale"**

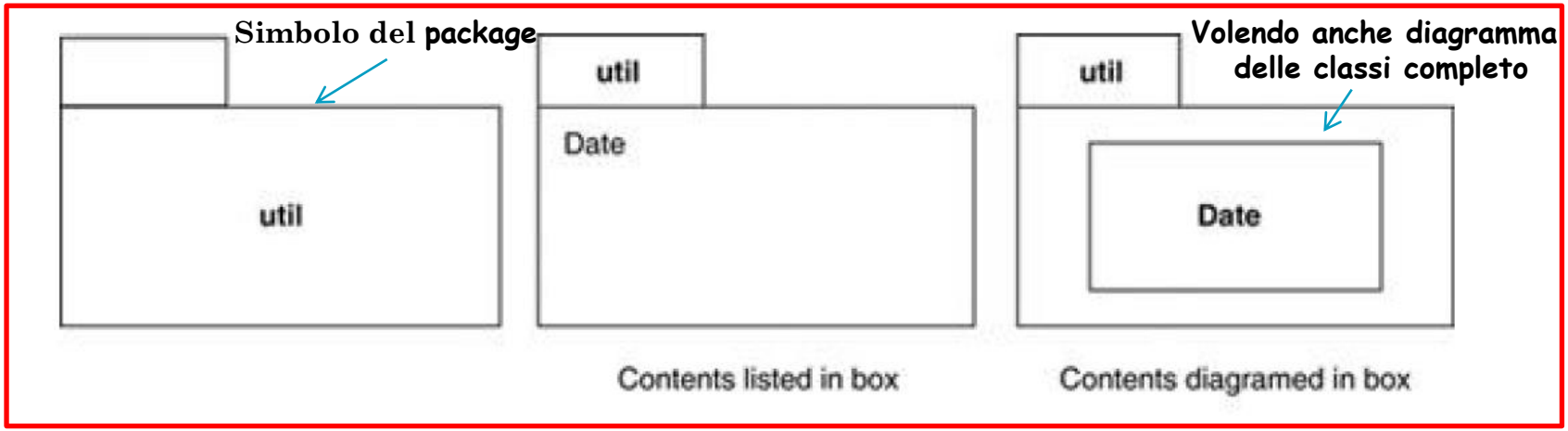
NAMESPACE

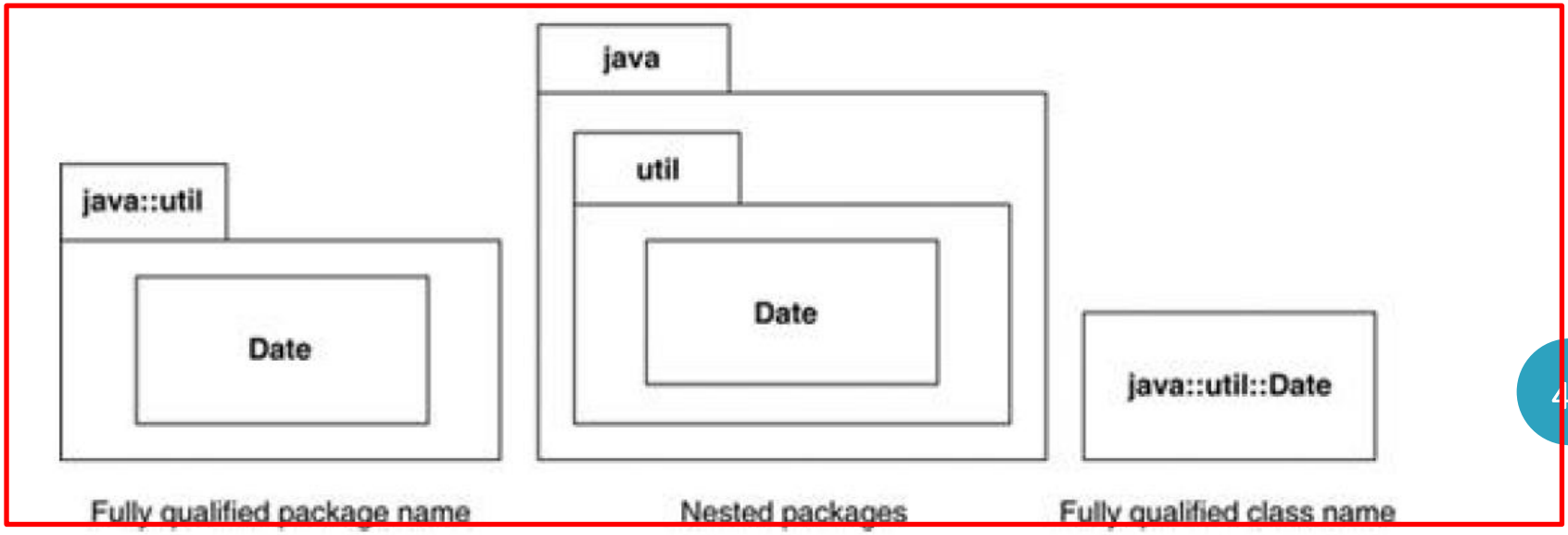
- Ogni package definisce un **namespace**
  - **Regione** all'interno del quale tutti i nomi devono essere univoci
- Lo scopo dei namespace è quello di evitare confusione ed equivoci nel caso siano necessarie molte entità con nomi simili
- Per indicare la classe a cui mi sto riferendo occorre usare un **nome completamente qualificato**

System::Data

MartinFowler::Util::Data

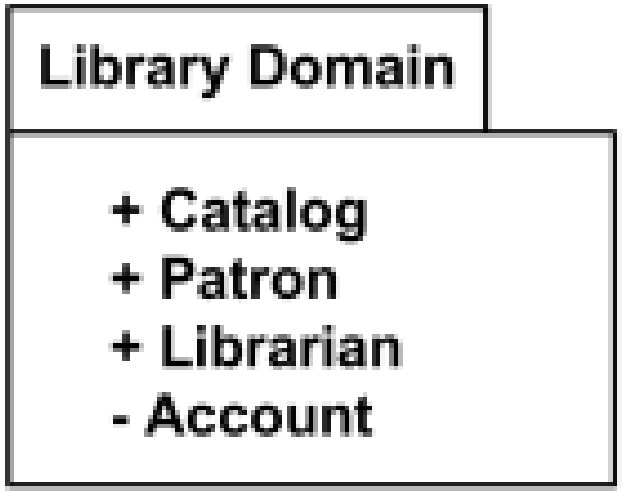
Modi di rappresentare i Package





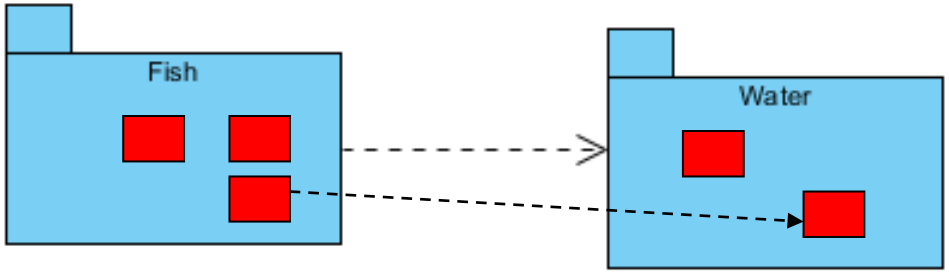
VISIBILITÀ

- Gli elementi contenuti in un package possono avere **una visibilità che indica se gli elementi sono visibili o meno ai clienti del package**
  - + = pubblica
  - - = privata
- Gli elementi visibili formano l'interfaccia del package



Package Diagram (dipendenze)

- I **Package diagram** descrivono i package e le dipendenze

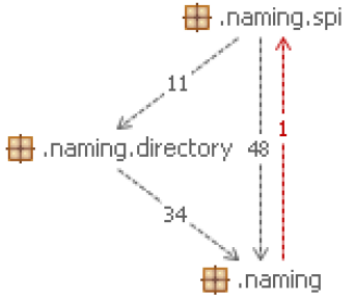


- **Fish dipende da Water**: vuol dire che nel package Fish esiste <u>almeno</u> una classe che dipende da una classe che è nel package Water …
  - La classe in Water deve essere visibile!
- Dipendenze tra package riassumono quelle tra gli elementi contenuti

Come suddividere le classi in package?

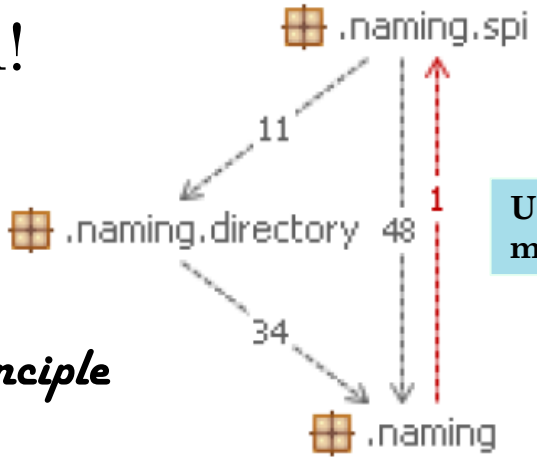
- Molto difficile!
  - Occorre molta esperienza
  - E' un compromesso!

- Seguire i **principi di buona progettazione**:
  - **High cohesion and low coupling**
  - **Common Reuse Principle**: le classi in un package dovrebbero essere sempre riusate assieme
  - **Acyclic Dependency Principle**: no cicli!
  - …



Cicli e dipendenze entranti

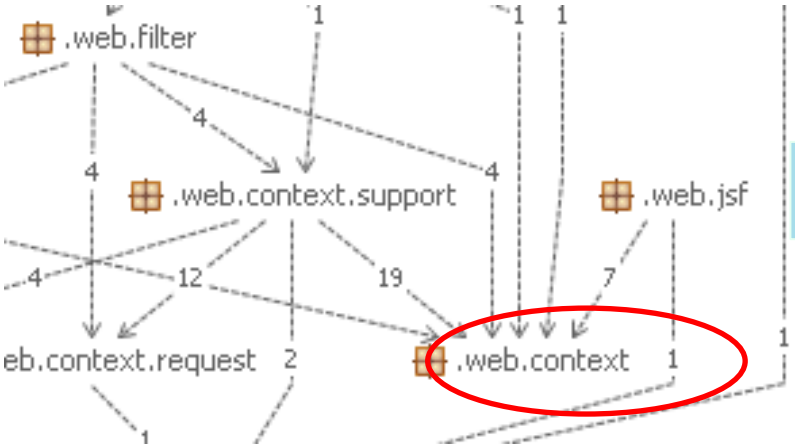
- Attenzione ai cicli!



Una modifica potrebbe comportare modifiche a cascata …

Acyclic Dependency principle

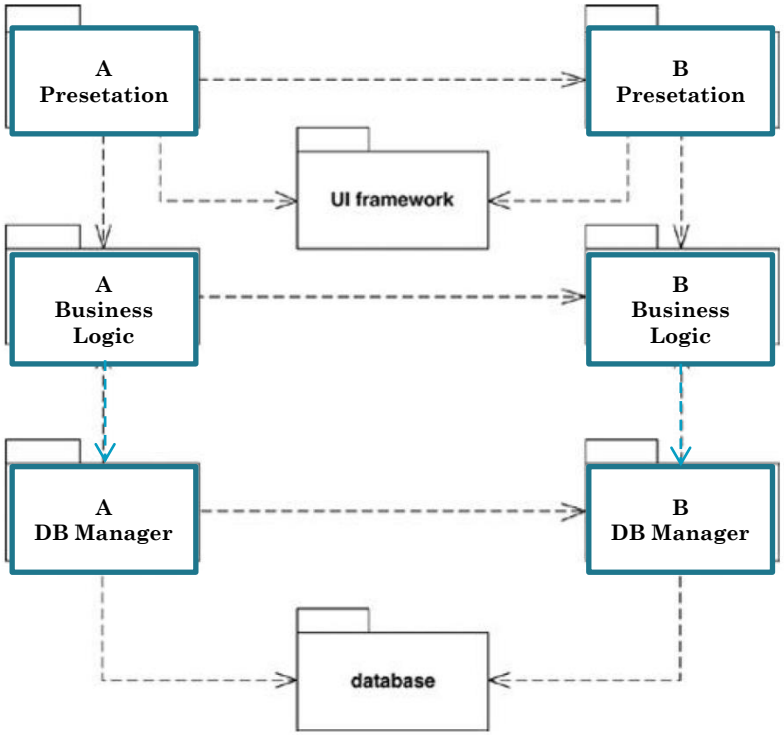
- Attenzione ai package con molte dipendenze entranti!

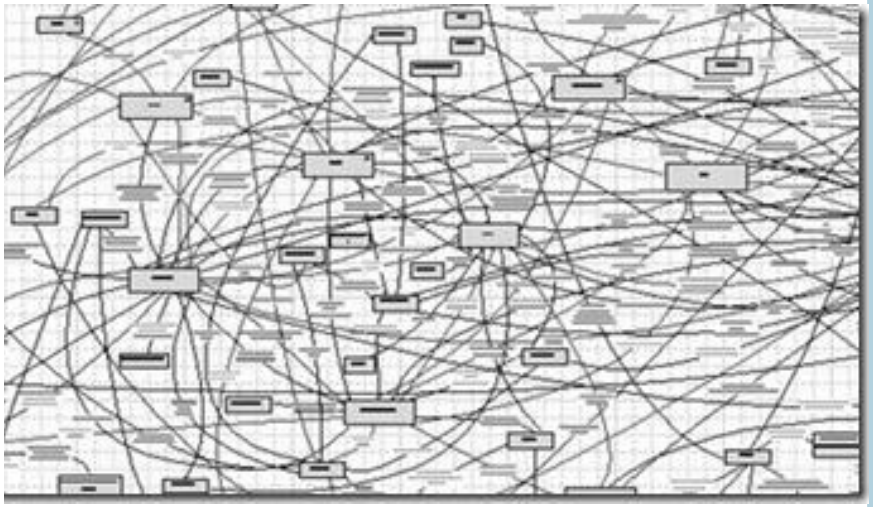


Una modifica a web.content potrebbe impattare molti altri package!

Package diagram

- *"In un sistema medio-grande, un diagramma dei package è una delle **cose più utili** per tenere sotto controllo la complessità strutturale del codice"*



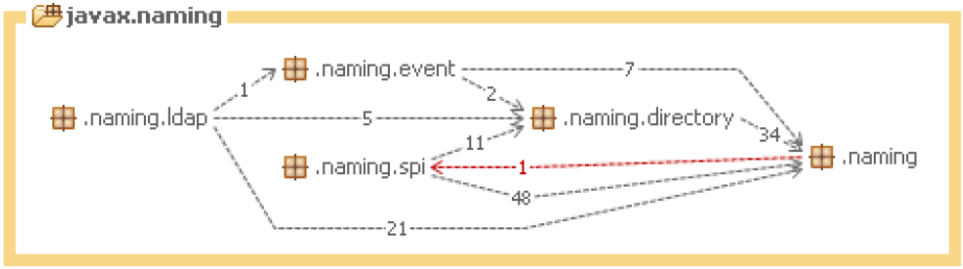


COME FARE IN PRATICA?

- Progettare l'applicazione cercando di seguire il più possibile i **principi di buona progettazione**
- Generare il **diagramma dei package** a partire dal codice stesso usando tool specifici (**reverse engineering**)
  - Es. Stan4J
- Identificare cicli, package poco coesi e package con tante dipendenze entranti
  - Ristrutturare il sistema

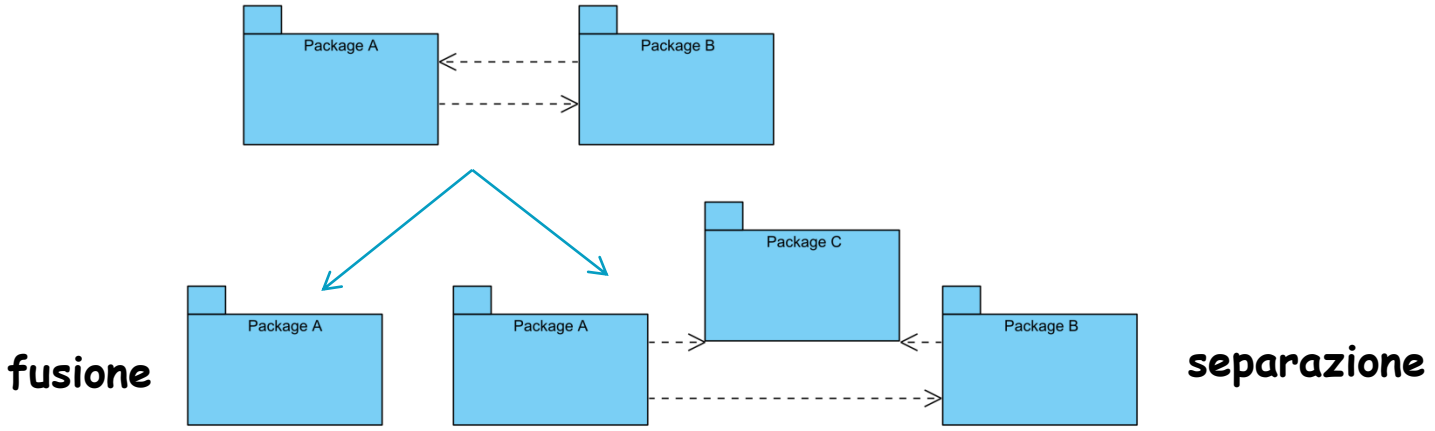
Risoluzione dei cicli

- **Eliminare** una o più dipendenze (se possibile)



Stan4J ci indica la strada più breve

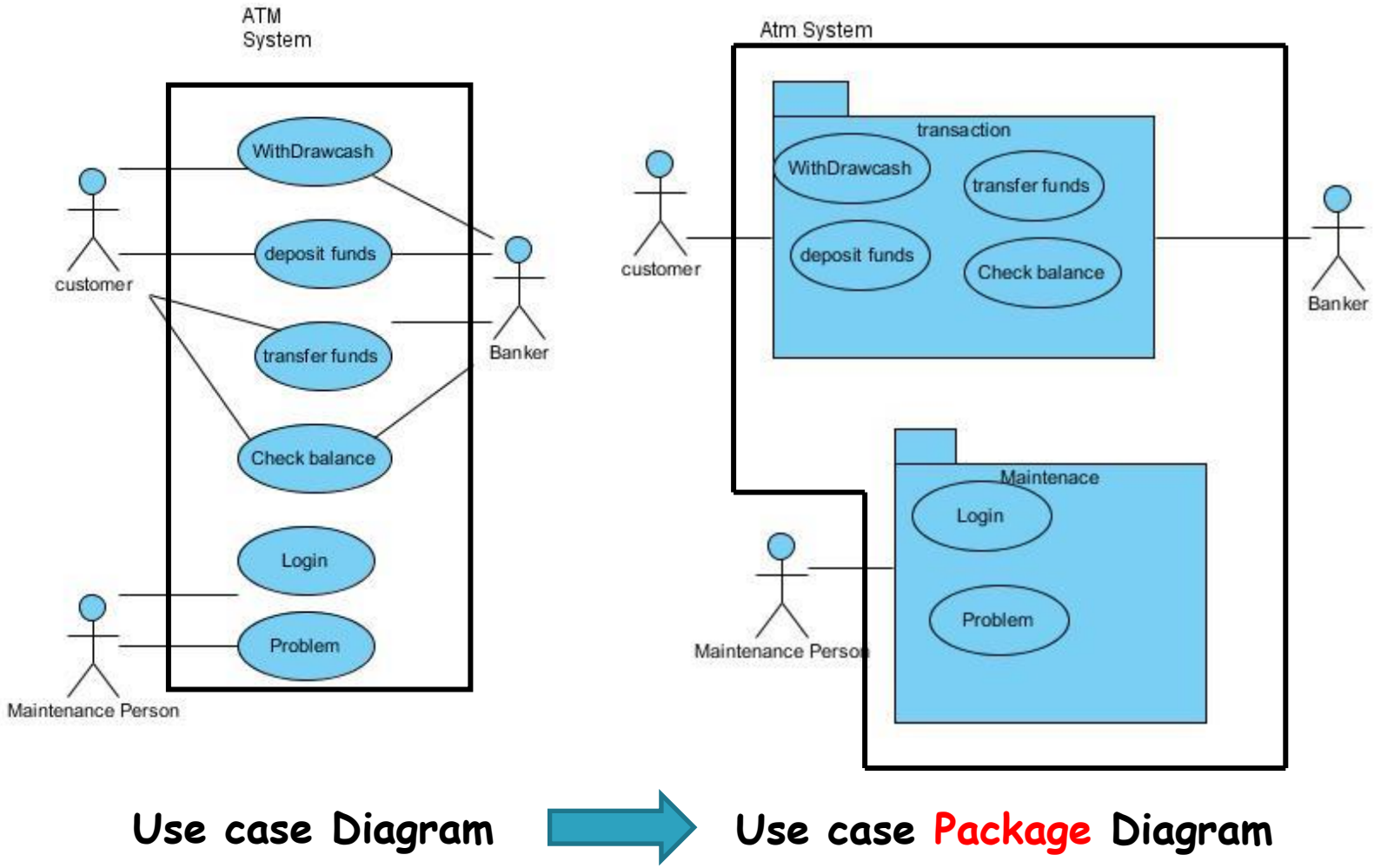
- **Fusione**
- **Separazione**. Si tenta di estrarre le entità che creano il ciclo e si spostano in un altro package



fusione

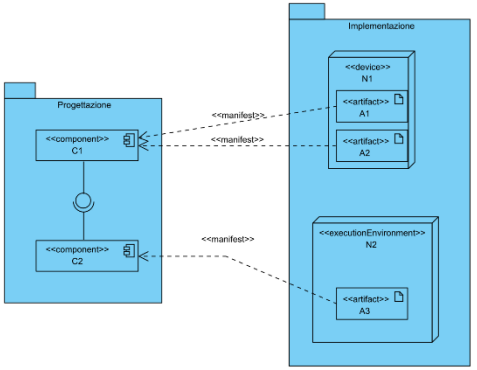
separazione

NON SOLO CLASSI: ESEMPIO USE CASE DIAGRAM

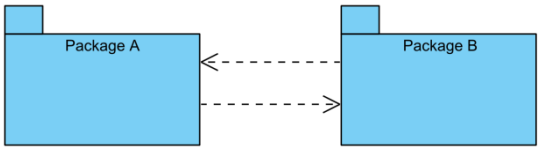


Use case Diagram → **Use case Package Diagram**

RIASSUMENDO



- Visti tre tipi di diagrammi strutturali:
- **Usati per scopi diversi**
  - Component diagram → descrive l'architettura software
  - Deployment diagram → descrive la relazione tra HW e SW
  - Package diagram → "raggruppa elementi UML"
- **Usati in momenti diversi dello sviluppo**
  - Component diagram → design dell'architettura
  - Deployment diagram → implementazione/deployment
  - Package diagram → dipende da cosa raggruppiamo ...



Materiale e riferimenti

- Per realizzare la seguente presentazione sono stati utilizzati:
  - UML distilled M. Fowler
  - UML 2 e Unified Process

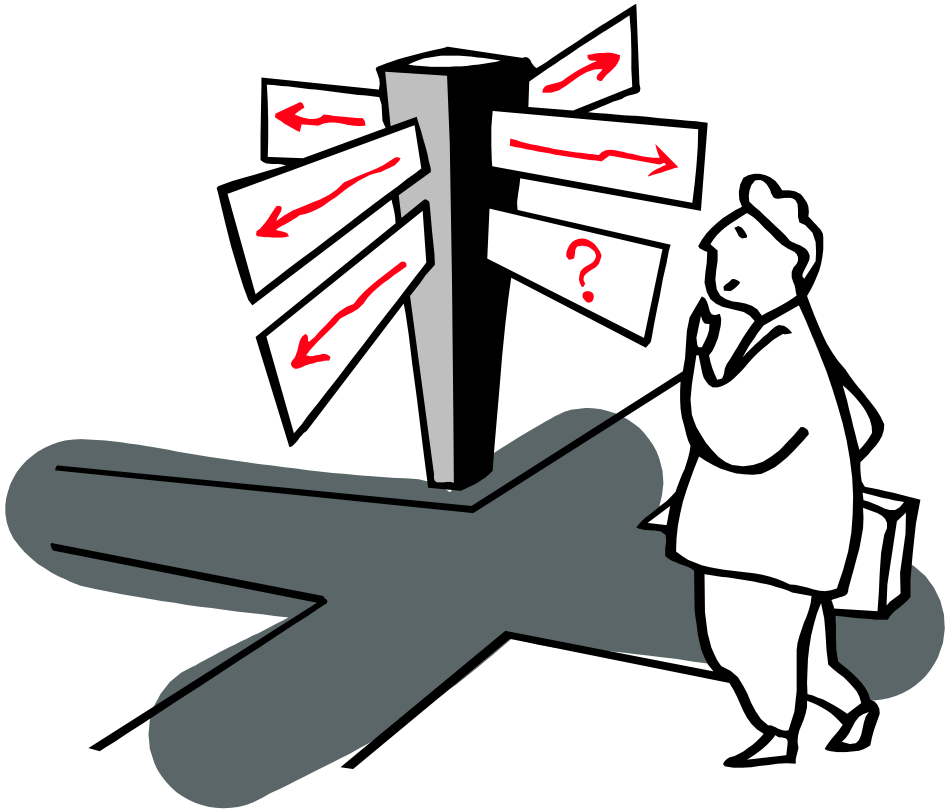
    Jim Arlow e Ila Neustadt
  - Slide di Orazio Tomarchio

    (Università di Catania)
  - Slide del Prof. J. Guo

    Package Diagram
  - Slide di Veronica Carrega

    Component Diagram in UML 2.0

The end …



Domande?