Luca Cabibbo

Architettura dei Sistemi Software

Architettura a componenti

dispensa asw470

marzo 2021

The composition
is the organized sum
of the interior functions
of every part of the work

Wassily Kandinsky

- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 27, Architettura a componenti
- [POSA4] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. **Pattern-Oriented Software Architecture (vol. 4)**: A Pattern Language for Distributed Computing. John Wiley & Sons, 2007
- Cheesman, J. and Daniels, J. **UML Components**: A Simple Process for Specifying Component-Based Software. Addison-Wesley, 2000.
- Java Platform, Enterprise Edition
  https://www.oracle.com/java/technologies/java-ee-glance.html
- The Java EE 8 Tutorial
  - https://javaee.github.io/tutorial/

- Obiettivi e argomenti

- Obiettivi
  - introdurre gli aspetti fondamentali relativi ai componenti e all'architettura a componenti
- Argomenti
  - introduzione
  - introduzione ai componenti
  - tecnologie a componenti
  - nozioni legate ai componenti
  - architettura a componenti
  - un esempio di applicazione a componenti
  - contenitori per componenti
  - pattern di deployment di applicazioni a componenti
  - discussione

* Introduzione

- L'*architettura a componenti* organizza un sistema software come la composizione di un insieme di elementi architetturali chiamati "componenti"
  - uno stile architetturale sostenuto dalle tecnologie a componenti
    - queste tecnologie promettono un supporto (trasparente o semplificato) per molte qualità architetturalmente significative – come prestazioni, sicurezza, disponibilità e scalabilità
    - lo sviluppo dei componenti e delle applicazioni si può così focalizzare sulle funzionalità applicative
  - in questo e nei prossimi capitoli ci occupiamo di investigare la veridicità di questa promessa
  - per semplicità, in questo capitolo ignoriamo il supporto fornito dalle moderne tecnologie a componenti per i servizi e l'interoperabilità

* Introduzione ai componenti

- Nel contesto dell'architettura del software, esistono più accezioni per il termine "componente software"
  - un'accezione più astratta e generica
    - componenti come elementi architetturali
  - un'accezione tecnologica
    - *tecnologie a componenti*
  - un'accezione metodologica
    - *sviluppo del software basato su componenti*

- Componenti come elementi architetturali

- L'architettura di un sistema software è la *struttura* ... del sistema, che comprende *elementi software* ...
  - un *componente software* è un elemento software che
    - incapsula un insieme di funzionalità (o di dati)
    - consente l'accesso a queste funzionalità mediante delle interfacce (interfacce fornite)
    - ha delle dipendenze, specificate mediante interfacce (interfacce richieste)
  - un sistema software viene realizzato come la *composizione* di più componenti software
    - si basa sulla connessione di più componenti, mediante le loro interfacce e con l'ausilio di connettori
    - la struttura ottenuta dovrebbe presentare le caratteristiche di qualità desiderate

- Tecnologie a componenti

- Nelle *tecnologie a componenti*
  - un *componente* è un'entità software runtime
    - implementa un insieme di funzionalità
    - offre i suoi servizi mediante delle interfacce (interfacce fornite)
    - può richiedere servizi ad altri componenti, mediante delle interfacce (interfacce richieste)
  - un'applicazione è formata da un insieme di componenti, che vengono *composti* (al momento del rilascio) sulla base delle loro interfacce
  - alcuni esempi sono CORBA, Microsoft .NET, Java EE e il framework Spring

- Sviluppo del sw basato su componenti

- Lo *sviluppo del software basato su componenti* (*CBSD* o *CBSE*)
  - un approccio alla costruzione di sistemi software basato sull'integrazione di componenti software – preesistenti oppure sviluppati appositamente
  - in questo approccio rivestono un ruolo fondamentale
    - la specifica dei componenti – basata sull'uso di interfacce
    - l'integrazione e le interazioni tra i componenti – anche queste sono basate sulle interfacce dei componenti

- Discussione

- È possibile identificare alcuni aspetti comuni tra queste accezioni
  - un componente è un elemento architetturale con responsabilità funzionali
  - la specifica di ogni componente è basata su interfacce fornite e interfacce richieste
  - i componenti sono pensati per essere composti
  - la composizione dei componenti è basata sulla connessione delle loro interfacce
- Facciamo principalmente riferimento all'accezione più astratta e a quella tecnologica dei componenti

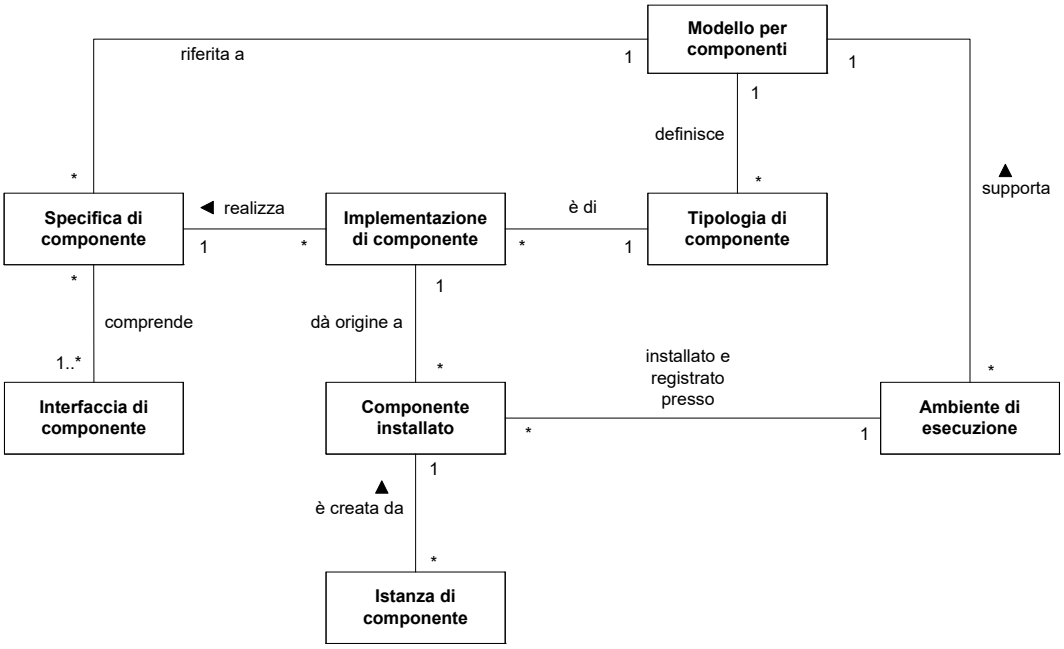
* Tecnologie a componenti

- Lo sviluppo dei sistemi software distribuiti è stato sostenuto da varie tecnologie di middleware – basate su diverse astrazioni
  - le tecnologie a oggetti distribuiti (fine anni '80 e primi anni '90) sono basate sull'utilizzo di un broker – tuttavia, esse presentano alcuni limiti relativi a
    - gestione delle dipendenze tra oggetti
    - deployment e configurazione
    - supporto alle qualità
  - le tecnologie a componenti rappresentano un'evoluzione delle tecnologie a oggetti distribuiti – anche per superare questi limiti

Tecnologie a componenti

- Nelle *tecnologie a componenti* (dalla fine degli anni ‘90)
  - i componenti vivono in ambienti di esecuzione detti contenitori per componenti (container o application server) – questi contenitori per componenti
    - offrono strumenti di deployment e configurazione che supportano la composizione dei componenti in applicazioni, sulla base delle loro interfacce
    - si occupano delle dipendenze tra componenti e della gestione del ciclo di vita dei componenti e delle loro istanze
    - forniscono servizi di supporto alle qualità delle applicazioni
  - queste tecnologie offrono anche una maggior generalità nei meccanismi di comunicazione

* Nozioni legate ai componenti



Modello a componenti

- Un *modello a componenti* è la specifica di uno standard per l'implementazione, la documentazione e il rilascio di componenti
  - alcuni esempi
    - Corba Component Model
    - modello EJB (Java EE)
    - modello .NET (Microsoft)
    - il modello a componenti di Spring
  - i diversi modelli sono differenti tra di loro – ma presentano anche delle caratteristiche simili che li accumunano
  - due categorie principali di modelli a componenti
    - per componenti distribuiti
    - modelli "leggeri" – per componenti eseguiti in un singolo processo

Tipologie di componenti

- Ogni modello a componenti definisce diverse *tipologie di componenti* – con obiettivi specifici e caratteristiche differenziate
  - ad es., nel modello a componenti di Java EE
    - componenti web e componenti EJB (enterprise bean)
    - gli enterprise bean possono essere poi session bean (stateful, stateless e singleton) e message-driven bean
  - ad es., nel modello a componenti di Spring
    - componenti generici (o bean), servizi, repository, controller, controller REST, …
  - ogni tipologia di componenti ha un proprio modello di programmazione, un proprio ciclo di vita e una propria semantica

Ambiente di esecuzione

- Un *ambiente di esecuzione* (o *piattaforma*) è l'implementazione di un modello per componenti – che consente l'esecuzione dei componenti di quel modello
  - alcuni esempi
    - un *application server* Java EE
    - un OS Windows – per il modello .NET
  - l'ambiente di esecuzione funge da *contenitore per i componenti*
  - il modello per componenti definisce anche le interazioni (mutue) previste tra il contenitore per componenti e i suoi componenti

Specifica e interfaccia di componente

- La *specifica di un componente* definisce un *tipo di componente* (non si confonda "tipo" con "tipologia") e il suo comportamento
  - definisce le modalità di interazione del componente con altri componenti – in modo indipendente dalle sue possibili implementazioni
- Gran parte della specifica di un componente consiste nella definizione di un insieme di *interfacce del componente*
  - ad es., un insieme di operazioni con i relativi contratti
  - questo insieme di interfacce è anche chiamato l'*interfaccia del componente*

Implementazione di componente

- Un'*implementazione di un componente* è una realizzazione di una specifica di componente
  - un'implementazione di componente può essere installata in un ambiente di esecuzione
  - per una certa specifica di componente, possono esistere più implementazioni che la realizzano
  - un'implementazione può esistere in diverse "forme"

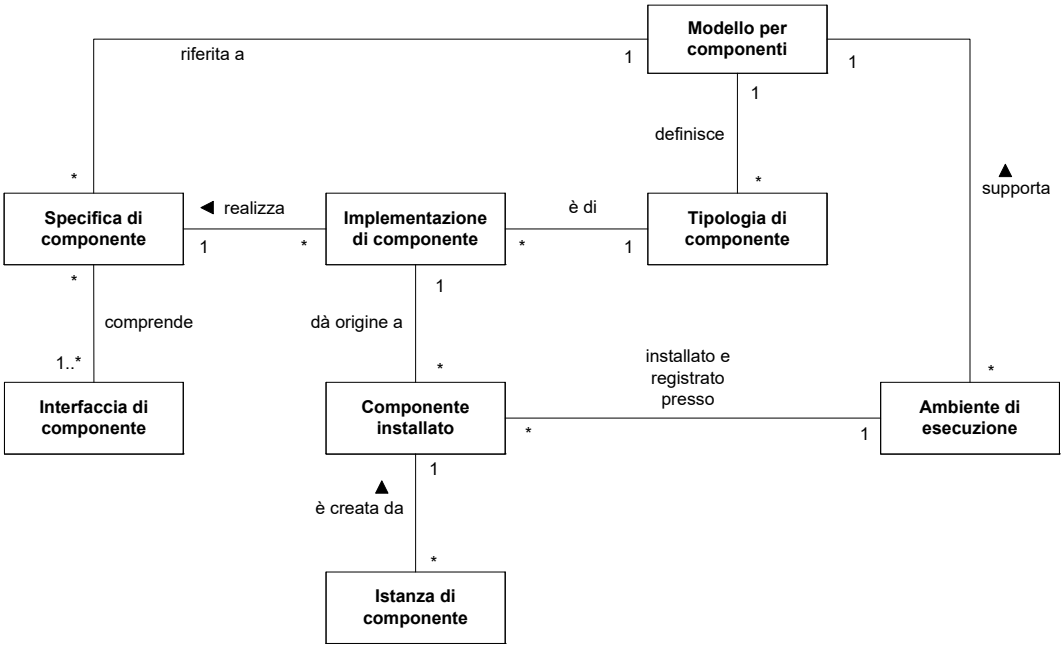
Componente installato

- Un *componente installato* è una copia di un'implementazione di componente che è stata installata su uno specifico ambiente di esecuzione (contenitore per componenti o OS)
  - l'installazione comprende la configurazione e la registrazione del componente in tale ambiente
  - un'implementazione di componente può dare origine a più componenti installati

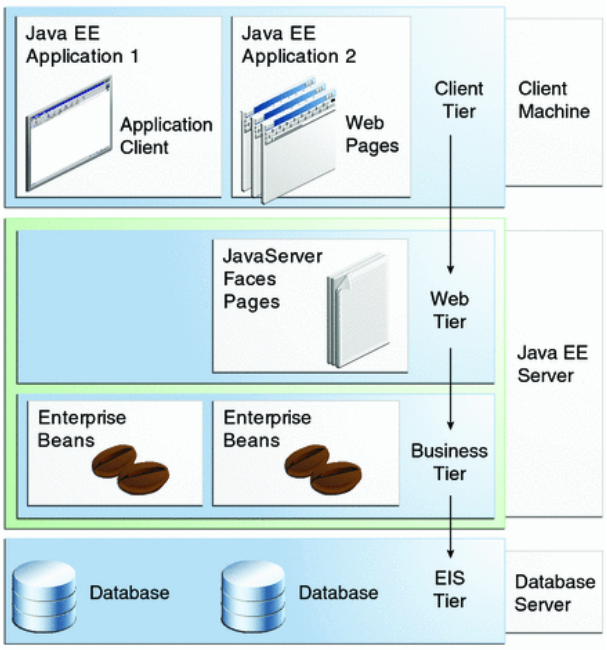
Istanza di componente

- Un'*istanza di componente* (o *componente oggetto*) è un'istanza runtime del componente, creata a partire da un componente installato in uno specifico ambiente di esecuzione
  - le istanze di componenti sono in grado di offrire concretamente i servizi del componente
  - hanno un'identità univoca e un proprio stato
  - da un componente installato possono essere create molte istanze

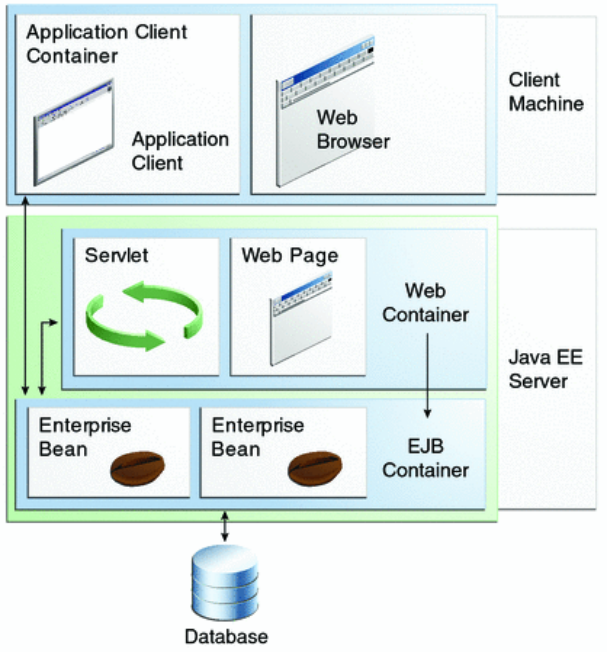
Nozioni legate ai componenti



Esempio: Java EE



Esempio: Java EE



Esempio: un'applicazione web Java EE

- Nel caso di un'applicazione web con Java EE
  - il *modello* per componenti è Java EE
  - l'*ambiente di esecuzione* è un application server (AS) Java EE – ad es., Tomcat (che implementa Java EE per la parte web)
  - l'*implementazione* (unità di distribuzione) è il file war – costruito anche sulla base di un descrittore di deployment
  - per essere usata, l'unità di distribuzione deve essere *installata/deployata* sull'AS
  - richieste HTTP all'applicazione causano/possono causare
    - la creazione di *componenti oggetto* coinvolti – ad es., la creazione di oggetti istanza delle classi servlet (una *tipologia di componente*)
    - la richiesta di esecuzione di operazioni a questi oggetti – queste richieste vengono fatte dall'AS agli oggetti servlet
  - il ciclo di vita dei vari oggetti dell'applicazione è gestito dall'AS

Esempio: MS Word

- MS Word comprende diversi componenti, di cui due abbastanza ovvi
  - il *modello* per componenti è COM
  - l'*ambiente di esecuzione* è un OS Windows, che supporta COM
  - i due *tipi di componenti* "ovvi" sono quelli che rappresentano rispettivamente l'applicazione (WordApp) e un documento (WordDoc)
    - non conosciamo né le loro *specifiche* né le relative *interfacce*
  - il file winword.exe "impacchetta" le *implementazioni* di queste due specifiche di componenti
  - al momento dell'installazione, il file winword.exe viene copiato sul disco, dando origine a due *componenti installati*, che vengono registrati nell'ambiente COM
  - quando viene eseguita l'applicazione, vengono creati due *componenti oggetto*: uno di tipo WordApp per l'applicazione e uno di tipo WordDoc per un nuovo documento
    - ogni nuovo documento aperto sarà rappresentato da un ulteriore *componente oggetto* di tipo WordDoc

* Architettura a componenti

- L'**architettura a componenti**
  - organizza un sistema software (un'*applicazione a componenti*) come la composizione di un insieme di componenti
  - ciascun componente
    - ha responsabilità funzionali
    - è caratterizzato da un insieme di interfacce
    - è di una specifica tipologia
  - la composizione dei componenti
    - avviene connettendo le interfacce dei componenti
    - avviene in sede di deployment

- Componenti e interfacce

- La specifica dei componenti e delle loro interfacce è fondamentale nell'architettura a componenti
  - per "interfaccia" si intendono le assunzioni che i componenti possono fare circa gli altri componenti
    - ad es., nomi delle operazioni e parametri, contratti delle operazioni, pre- e post-condizioni, effetti collaterali, consumo di risorse globali, vincoli di coordinamento, service level agreement, ...
  - le interfacce descrivono le responsabilità di un componente, i servizi offerti e il suo protocollo d'uso (e di composizione)
  - le dipendenze tra componenti sono espresse in termini di interfacce – i componenti interagiscono tra di loro solo sulla base della loro interfacce

Componenti e interfacce

- Due tipi di interfacce per i componenti
  - *interfaccia fornita*
    - i servizi forniti da un componente ad altri componenti
  - *interfaccia richiesta*
    - i servizi che devono essere resi disponibili a un componente, affinché esso possa funzionare come specificato
- Un componente può avere molte interfacce fornite e molte interfacce richieste

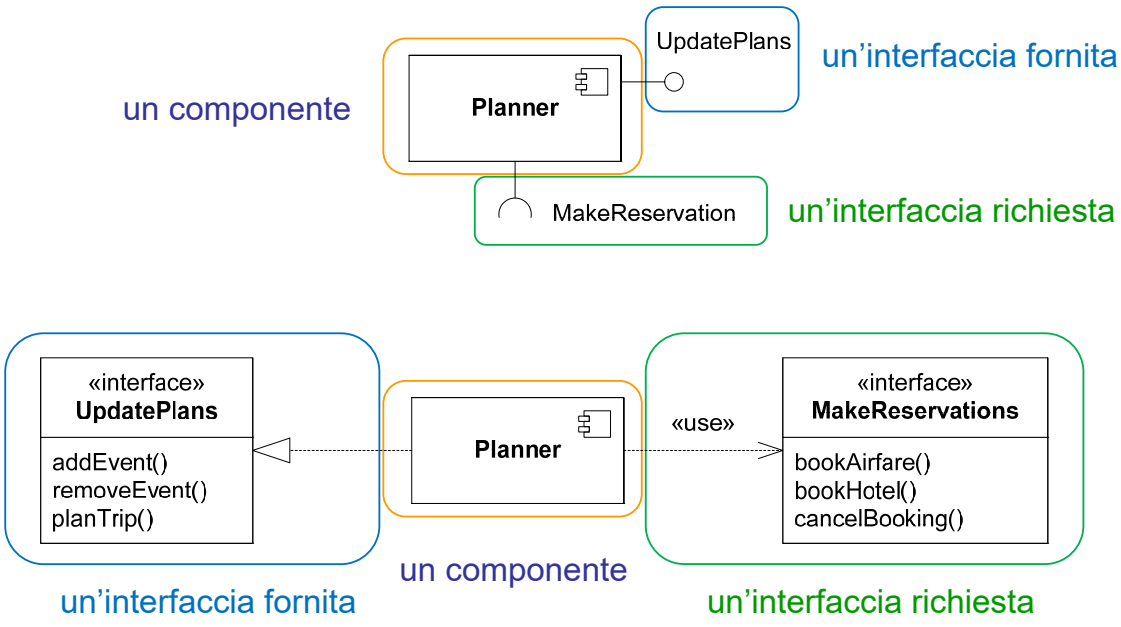
Stili per le interfacce

- Due stili principali per le interfacce
  - *stile procedurale*
    - per l'invocazione remota
    - un'interfaccia è basata su un insieme di operazioni e di tipi
  - *stile orientato ai messaggi*
    - per la comunicazione asincrona e la notifica di eventi
    - un'interfaccia è basata su un insieme di tipi di messaggi ed eventi e di canali
- Un componente può avere sia interfacce nello stile procedurale che interfacce nello stile orientato ai messaggi

Notazione UML per i componenti

- In UML, un *componente* è una parte modulare di un progetto di un sistema che nasconde la sua implementazione dietro un insieme di interfacce esterne

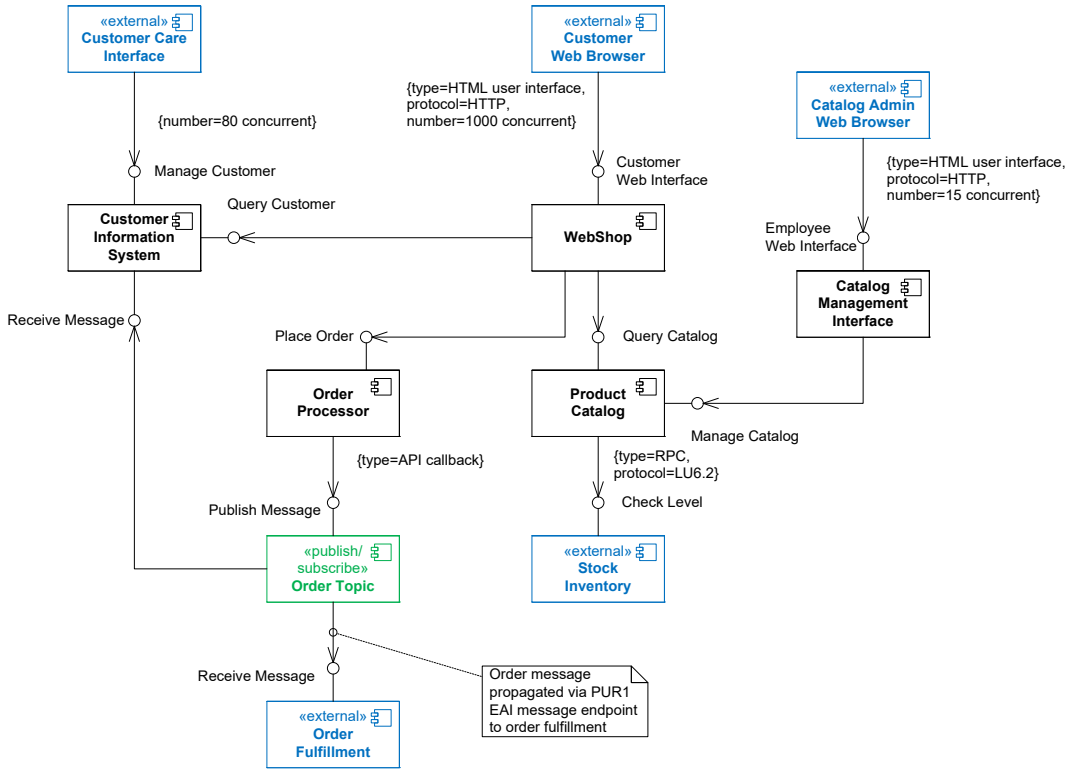
Notazione UML per le interfacce



- Composizione

- La composizione di un'applicazione a componenti avviene sulla base di un certo numero di componenti – connettendo le loro interfacce richieste e fornite, per soddisfare le dipendenze di ogni componente
  - in sede di *specifica*, avviene connettendo le interfacce fornite e richieste dei componenti
    - quali componenti servono (ciascuno con la sua specifica) e come vanno collegati – ma senza fare riferimento alle loro implementazioni
  - in sede di *deployment*, bisogna specificare anche l'implementazione scelta per ciascun componente
    - la composizione sulla base delle interfacce consente infatti la sostituzione di componenti

Esempio



Composizione

- È necessaria anche una “composizione a runtime” tra le istanze dei componenti
  - alcuni problemi da considerare
    - quante e quali istanze di componenti creare? quando creare tali istanze?
    - come soddisfare le dipendenze di ciascuna istanza di componente? quali istanze di componenti collegare? come e quando collegare tali istanze?
  - la risposte dipendono da molti fattori, tra cui
    - l’implementazione di ciascun tipo di componente
    - la tipologia di ciascun tipo di componente
    - la strategia di creazione di istanze per ciascun tipo di componente
    - l’uso di proxy e la possibilità di collegare dinamicamente le istanze di componenti

- Strategie di creazione dei componenti

- Lo *scope* (“portata” o “scopo”) definisce la strategia per la creazione delle istanze dei componenti
  - definisce quante istanze di componenti creare e quando farlo
  - viene usato dal contenitore per componenti nella gestione delle istanze dei componenti e del loro ciclo di vita
  - lo scope può dipendere dalla tipologia del componente e dalle dipendenze tra i componenti
  - alcuni tipi comuni di scope
    - *singleton*, *prototype* e *pool*
  - alcuni tipi di scope per i componenti web
    - *request*, *session* e *application*

* Un esempio di applicazione a componenti

- Un'applicazione a componenti è costituita da un insieme di componenti, ciascuno di un certo tipo, con le sue interfacce, e di una certa tipologia – questi componenti sono composti tra di loro
  - ma quali e quanti componenti utilizzare in un'applicazione? e di che tipo? e con quali interfacce? e come interagiranno tra di loro?
  - la progettazione dei componenti in un'architettura a componenti è un problema ampio, che qui viene solo accennato, con riferimento a un semplice esempio
    - facciamo riferimento al modello a componenti di Java EE

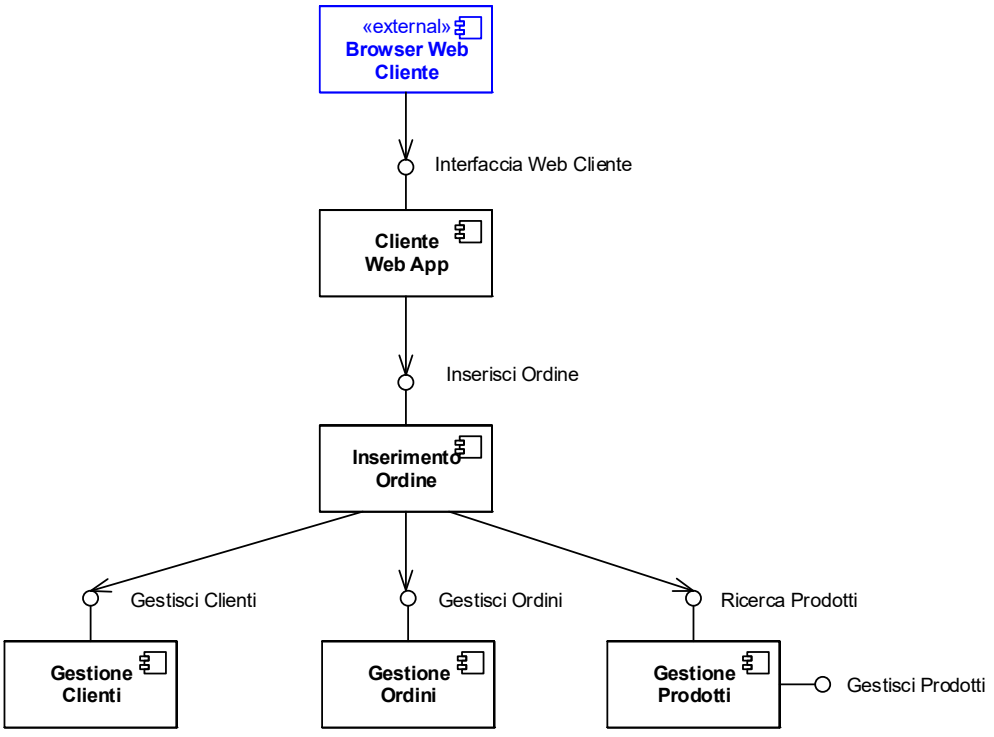
Un piccolo esempio

- Consideriamo un sistema di commercio elettronico, per la gestione di ordini, fatti da clienti relativamente a prodotti
  - come può essere realizzata con la piattaforma Java EE?
- Livello client
  - il sistema viene acceduto mediante un client (browser) web
- Livello web
  - un'applicazione (componente) web per ciascun attore primario del sistema
    - ad es., un'applicazione web **ClienteWebApp**

Un piccolo esempio

- Livello EJB – logica applicativa – session bean stateful
  - un session bean stateful per ciascun caso d'uso del sistema
    - ad es., un session bean **InserimentoOrdine**
  - ognuno ha la responsabilità delle operazioni di sistema di un caso d'uso e dello stato delle relative sessioni
- Livello EJB – logica di dominio – session bean stateless
  - un session bean stateless per ciascun tipo di dati "principale"
    - ad es., i session bean **GestioneClienti**, **GestioneProdotti** e **GestioneOrdini**
  - ognuno ha la responsabilità delle operazioni per accedere e manipolare il tipo di dati di interesse – e di implementare l'accesso ai relativi dati persistenti

Un piccolo esempio



Discussione

- In questo esempio è possibile vedere l'applicazione di alcuni pattern architetturali fondamentali
  - i pattern architetturali Layers e client-server a più livelli
  - Domain Model
    - una modellazione dei casi d'uso
    - una modellazione delle informazioni

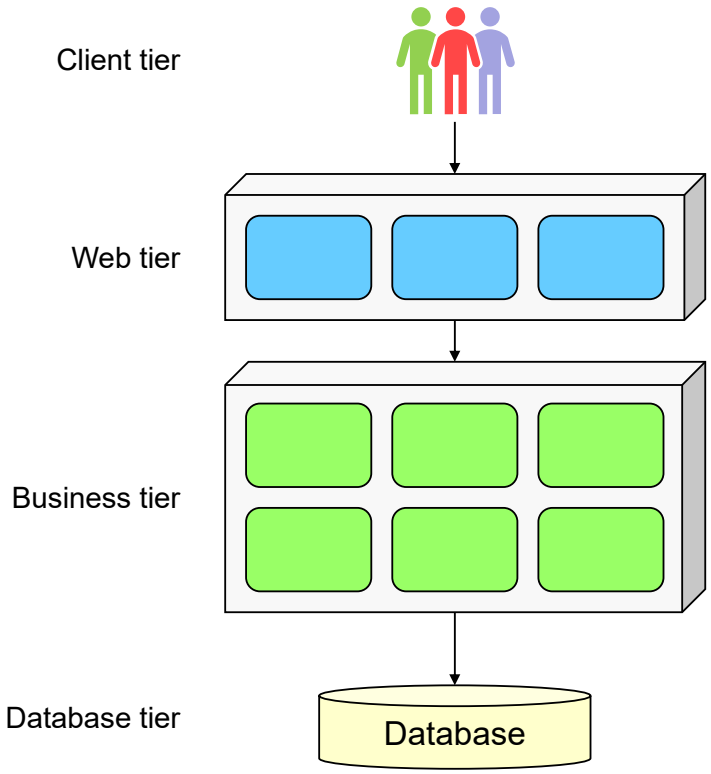
* Contenitori per componenti

- Le applicazioni a componenti e i loro componenti vanno rilasciati ed eseguiti in ambienti di esecuzioni specializzati
  - un *contenitore per componenti* è un ambiente runtime per la gestione di componenti, in genere lato server
    - i componenti possono essere eseguiti solo dopo essere stati configurati, assemblati e rilasciati in un contenitore
  - responsabilità principali di un contenitore per componenti
    - consentire la composizione di componenti e il deployment e la configurazione di applicazioni a componenti
    - gestire il ciclo di vita dei componenti
    - consentire la comunicazione tra componenti
    - fornire ai componenti servizi per sostenere diverse qualità
    - in questo modo, lo sviluppo dei componenti può essere focalizzato sull'implementazione delle funzionalità

* Pattern di deployment di applicazioni a componenti

- Le applicazioni a componenti vengono rilasciate utilizzando un pattern di deployment su più livelli – ad es., le applicazioni web
  - *livello client*
    - gli utenti interagiscono di solito con l'applicazione mediante un browser web standard – ma a volte è richiesto un contenitore per componenti lato client
  - *livello web*
    - componenti web che gestiscono la presentazione – eseguiti in un contenitore per componenti web
  - *livello di business*
    - componenti che realizzano la logica di business – in un contenitore per componenti di business
  - *livello dei dati*
    - i dati persistenti dell'applicazione sono di solito gestiti mediante un DBMS, esterno ai contenitori per componenti

Pattern di deployment di applicazioni a componenti



* Discussione

- Abbiamo introdotto alcuni aspetti di base dell'architettura a componenti – sono tuttavia utili degli approfondimenti, almeno in due direzioni
  - l'architettura a componenti è sostenuta dalle tecnologie a componenti e dai contenitori per componenti
    - discusso in un capitolo successivo
  - un uso efficace delle tecnologie a componenti richiede anche delle considerazioni di natura metodologica, per guidare la progettazione di un'architettura a componenti
    - discusso in una successiva dispensa sulla progettazione di software a componenti

Discussione

- Uno dei limiti iniziali delle tecnologie a componenti è che sono essenzialmente "autarchiche" e "autoreferenziali"
  - ogni tecnologia a componenti è autosufficiente e si basa esclusivamente su se stessa
  - l'architettura a componenti è "mono-tecnologica"
    - questo consente delle ottimizzazioni altrimenti impossibili – sono tecnologie eccellenti per costruire singole applicazioni
  - tuttavia, l'interoperabilità non è supportata – e nemmeno lo sviluppo di applicazioni distribuite su larga scala
  - questo limite (iniziale) è stato poi affrontato dalle tecnologie a servizi
    - oggi, le moderne tecnologie a componenti offrono anche un ampio supporto per i servizi e l'interoperabilità
    - ne parleremo nel contesto dell'architettura a servizi