Luca Cabibbo

Architettura dei Sistemi Software

Contenitori per componenti

dispensa asw480

ottobre 2023

I'm sorry,
but there is no such thing as a hole by itself.
Kurt Tucholsky

- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 28, Contenitori per 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

- Obiettivi e argomenti

- Obiettivi
  - presentare i contenitori, ambienti di esecuzione per componenti
  - presentare il pattern architetturale Container, insieme ad alcuni pattern di supporto
  - fornire alcune intuizioni sulle tattiche implementate dai contenitori per componenti
- Argomenti
  - introduzione ai contenitori per componenti
  - Container (POSA4)
  - un esempio di applicazione a componenti
  - discussione

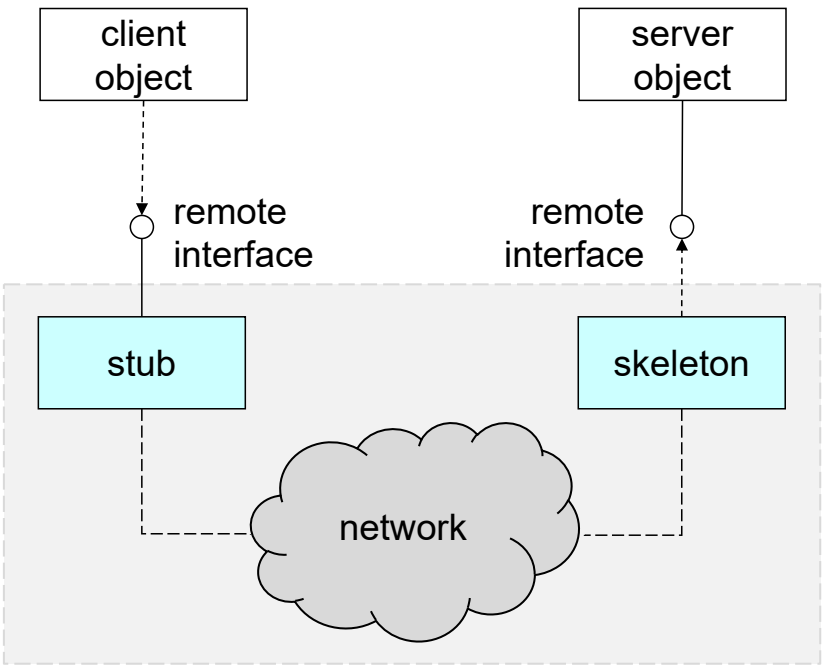
* Introduzione ai contenitori per componenti

- L'architettura a componenti è sostenuta dalle tecnologie a componenti – che sono basate sui contenitori per componenti
  - nelle tecnologie a componenti, i componenti sono infatti pensati per "vivere" dentro dei contenitori
    - "non esiste niente che, di per sé, è un buco" [Kurt Tucholsky]
    - "allo stesso modo, non esiste niente che, di per sé, è un componente – i componenti esistono solo grazie ai contenitori che li definiscono" [Bruce Wallace]
  - queste tecnologie promettono il supporto per diverse qualità architetturalmente significative, come prestazioni, sicurezza, disponibilità, scalabilità, ...
  - discutiamo ora i contenitori per componenti, con riferimento al pattern architetturale Container [POSA4] – discutiamo anche il supporto che forniscono ai loro componenti

Componenti e contenitori

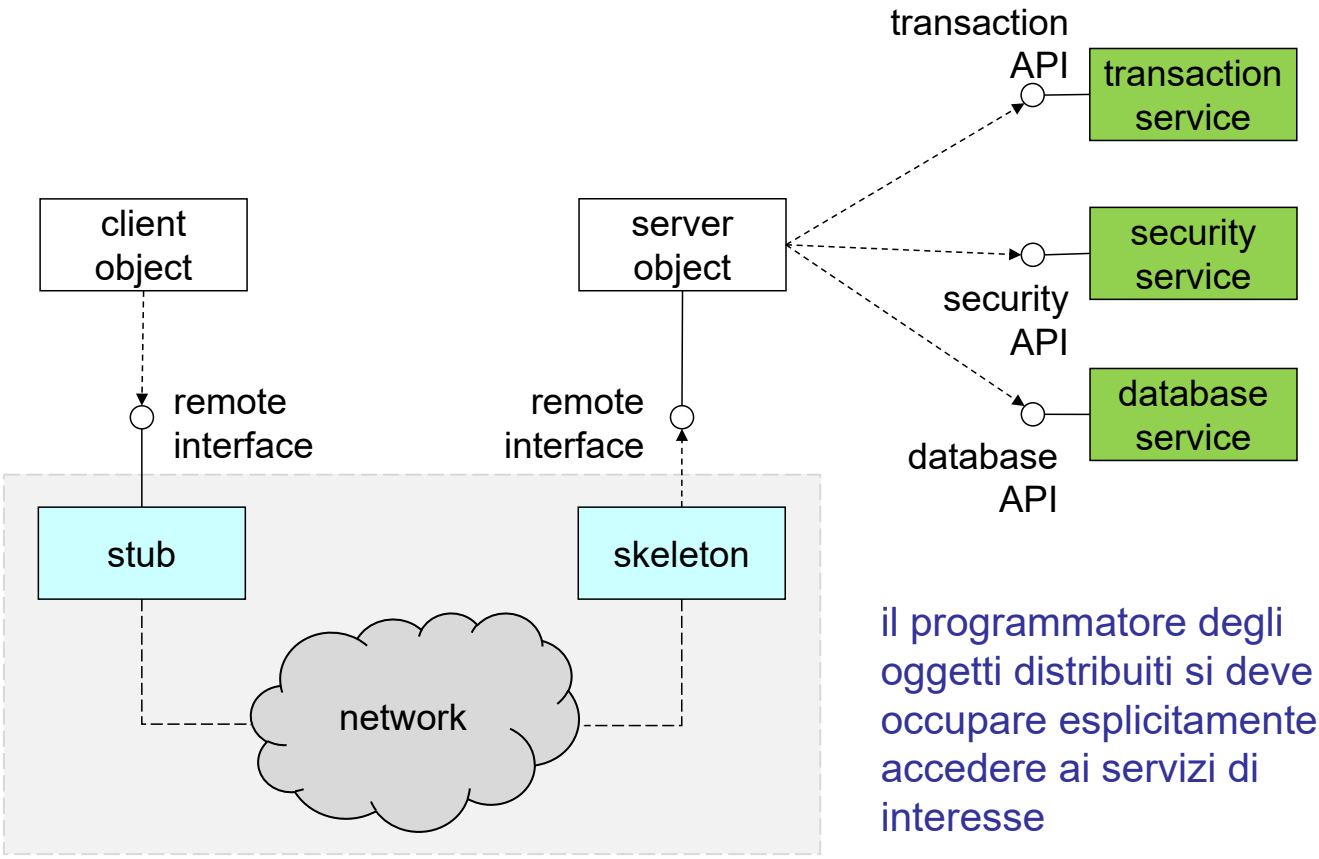
- Per ricordare
  - i componenti sono entità software runtime – che implementano funzionalità e che hanno delle interfacce fornite e richieste
    - un'applicazione a componenti è formata da un insieme di componenti, che vengono composti sulla base delle loro interfacce
  - un contenitore per componenti è un ambiente runtime per la gestione di componenti, in genere lato server – che ha le seguenti responsabilità principali
    - 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

Dagli oggetti distribuiti ai componenti



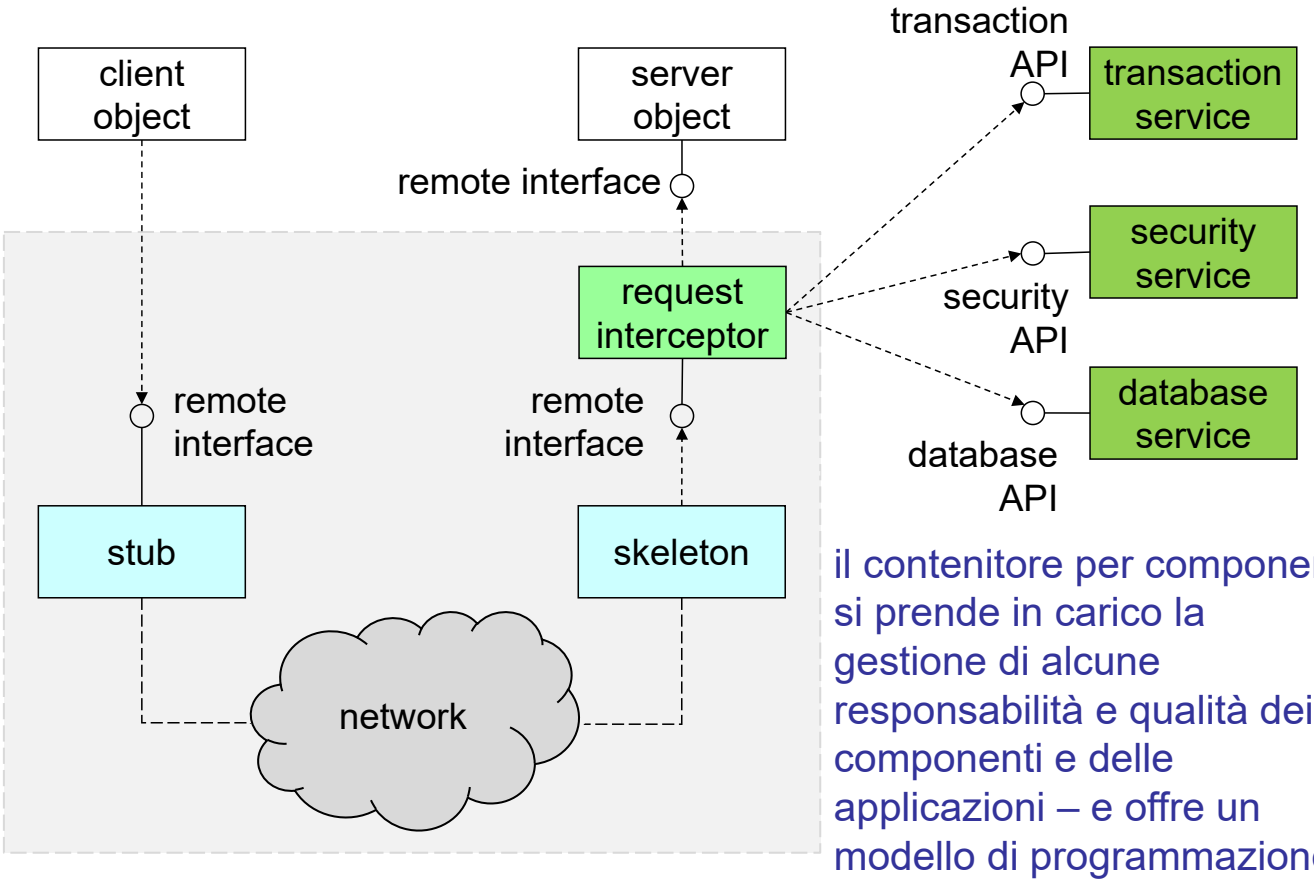
tutte le richieste remote tra gli oggetti distribuiti passano attraverso il middleware (un broker)

Dagli oggetti distribuiti ai componenti



il programmatore degli oggetti distribuiti si deve occupare esplicitamente di accedere ai servizi di interesse

Dagli oggetti distribuiti ai componenti



il contenitore per componenti si prende in carico la gestione di alcune responsabilità e qualità dei componenti e delle applicazioni – e offre un modello di programmazione semplificato

* Container (POSA4)

- I *contenitori per componenti* – chiamati anche *application server* o *component framework*
  - la loro architettura e il loro funzionamento sono descritti dal pattern architetturale **Container** [POSA4]
  - il pattern **Container** viene presentato (insieme ad altri pattern correlati) in un intero capitolo di POSA4 dedicato alla gestione delle risorse
    - per *risorsa* si intende una qualunque risorsa software, tra cui componenti, servizi distribuiti e loro istanze – ma anche connessioni di rete, sessioni nell'accesso a basi di dati, token per la sicurezza, ...
  - la gestione delle risorse e del loro ciclo di vita riveste infatti un ruolo fondamentale nel modo in cui i contenitori per componenti controllano alcune qualità importanti delle applicazioni

Gestione delle risorse

- La gestione delle risorse è una capacità critica nella realizzazione dei sistemi software distribuiti
  - molte proprietà di qualità di un'applicazione – come prestazioni, scalabilità, flessibilità, stabilità, disponibilità, sicurezza – dipendono da come le sue risorse vengono gestite (create, ottenute, accedute, utilizzate, rilasciate o distrutte)
  - gestire le risorse in modo corretto ed efficiente è difficile – soprattutto se si vuole conseguire un buon compromesso tra diverse qualità
    - per gestire le risorse è però possibile adottare diverse tattiche e pattern architetturali provati nel tempo

Gestione delle risorse e qualità

- Alcune considerazioni relative alla gestione delle risorse
  - prestazioni – è importante minimizzare le attività di creazione, inizializzazione, acquisizione, rilascio, distruzione e accesso alle risorse
    - intuizione: il contenitore può gestire un pool di risorse, preparate in anticipo, in modo da minimizzare il costo di accesso a queste risorse
  - affidabilità/disponibilità – per sostenere la disponibilità, è utile replicare alcune risorse – può anche essere utile evitare inconsistenze, per le transazioni che coinvolgono più risorse
    - intuizione: per sostenere la tolleranza ai guasti, il contenitore può occuparsi della replicazione di alcune risorse (come le istanze dei componenti e lo stato delle sessioni)
    - intuizione: il contenitore può occuparsi della gestione delle transazioni

Gestione delle risorse e qualità

- Alcune considerazioni relative alla gestione delle risorse
  - scalabilità – anche per sostenere la scalabilità (orizzontale) è utile applicare delle tattiche per la gestione delle risorse
    - intuizione: il contenitore può essere implementato come un cluster di application server che
      - si occupa di allocare le istanze dei componenti nei nodi del cluster e di distribuire le richieste tra le diverse istanze
      - supporta l'aggiunta di nodi al cluster
  - sicurezza – per sostenere la sicurezza, l'accesso alle risorse può essere basato su autenticazione e autorizzazioni
    - intuizione: il contenitore può occuparsi di autenticare gli utenti delle applicazioni e di gestire le autorizzazioni associate alle risorse – può farlo mentre si occupa di gestire la comunicazione tra componenti

Gestione delle risorse e qualità

- Alcune considerazioni relative alla gestione delle risorse
  - modificabilità/flessibilità – le proprietà di qualità di un'applicazione devono poter essere selezionate al momento del deployment oppure durante l'esecuzione
    - intuizione: il contenitore può occuparsi del deployment e della gestione delle configurazioni dei componenti e delle applicazioni – per fornire a ogni applicazione i servizi di supporto richiesti dalla sua configurazione
  - modificabilità/aggiornamenti – deve essere possibile aggiornare le applicazioni in esecuzione senza interruzioni di servizi
    - intuizione: il contenitore può occuparsi dell'aggiornamento (a caldo) delle applicazioni e dei loro componenti

Gestione delle risorse e qualità

- Alcune considerazioni relative alla gestione delle risorse
  - gestione trasparente del ciclo di vita delle risorse – i client delle risorse dovrebbero poter usare le risorse in modo semplice, senza dover conoscere i dettagli del loro ciclo di vita (creazione, utilizzo, rilascio e distruzione) – infatti una gestione errata del ciclo di vita delle risorse può avere effetti negativi sulle qualità del sistema
    - intuizione: il contenitore si può occupare di gestire il ciclo di vita delle risorse e le dipendenze dei componenti – e può fornire ai client delle risorse un modello di programmazione semplificato per accedere alle risorse

- Pattern per la gestione delle risorse

- L'infrastruttura realizzata da un contenitore per componenti può essere basata su diversi pattern per la gestione delle risorse
  - i pattern principali sono *Container*, *Object Manager* e *Component Configurator* – descrivono l'intera infrastruttura per la gestione delle risorse
  - altri pattern si occupano di diversi aspetti realizzativi per una gestione efficace delle risorse – ad es., *Resource Pool*, *Lifecycle Callback*, *Lookup*, ...
  - altri pattern ancora riguardano strategie per l'acquisizione e il rilascio di risorse – ad es., *Eager Acquisition* e *Lazy Acquisition* e *Automated Garbage Collection*
  - infine, altri pattern si occupano di aspetti relativi alla creazione e distruzione di risorse – ad es., *Factory Method* e *Disposal Method*

- Container (POSA4)

- Il pattern *Container* [POSA4] ha i seguenti obiettivi complessivi
  - definire un ambiente di esecuzione per componenti
  - supportare i servizi infrastrutturali richiesti dai componenti e le loro qualità
  - sostenere e semplificare lo sviluppo di applicazioni software a componenti
    - separare gli aspetti relativi allo sviluppo dei componenti da quelli relativi alla loro composizione e al loro deployment
    - consentire agli sviluppatori di concentrarsi sugli aspetti funzionali e sulla logica applicativa dei componenti
    - consentire di gestire gli aspetti di qualità in sede di deployment e mediante approcci dichiarativi
    - sostenere il riuso di componenti

Container

- La soluzione proposta da *Container*
  - definire un *container* (*contenitore per componenti*) che fornisce un ambiente di esecuzione opportuno ai componenti
  - il contenitore realizza l'infrastruttura necessaria per comporre i componenti in applicazioni
  - in particolare, il contenitore deve
    - fornire un mezzo per registrare e configurare dinamicamente i componenti nel contenitore, sulla base di una specifica dichiarativa
    - inizializzare e fornire il contesto runtime per l'esecuzione dei propri componenti
    - consentire le interazioni tra componenti
    - fornire opportuni servizi di supporto ai componenti – notifica di eventi, replicazione, bilanciamento del carico, sicurezza, persistenza, transazioni, …

Discussione

- *Container* è solo il "punto di ingresso" ai pattern per i contenitori per componenti
  - gli altri pattern consentono di affrontare in modo coerente numerosi aspetti di interesse per un contenitore
  - in particolare, il contenitore è responsabile dei componenti registrati (*Component Configurator*) nonché della gestione di questi componenti (*Object Manager*)
  - la gestione dei componenti richiede che i componenti offrano un'interfaccia per la loro amministrazione (*Lifecycle Callback*)
  - nell'ambito della gestione del ciclo di vita dei componenti, il contenitore deve controllare gli attributi di qualità richiesti
    - per queste importanti responsabilità, si vedano i successivi pattern – ma anche le "intuizioni" descritte in precedenza

- Component Configurator (POSA4)

- Il pattern *Component Configurator* si occupa di consentire la composizione e la configurazione dei componenti in modo flessibile – nella realizzazione di un'applicazione a componenti
  - le applicazioni dovrebbero poter essere configurate al momento del loro deployment (e non prima) – e talvolta anche dopo il loro deployment iniziale
    - ad es., un'applicazione deve potersi avvantaggiare dall'uso di componenti più nuovi o migliori
  - il cambiamento della configurazione di un'applicazione in esecuzione deve avere un impatto basso sul sistema

Component Configurator

- La soluzione proposta da *Component Configurator*
  - separa le interfacce dei componenti dalle loro implementazioni
  - organizza i componenti in unità di deployment dinamiche
  - fornisci un meccanismo (*component configurator*) per configurare i componenti in un'applicazione in modo dinamico, senza dover interrompere e riavviare l'applicazione
  - consenti la configurazione dei componenti e la loro composizione sulla base di specifiche dichiarative (*declarative component configuration*)
  - il contenitore utilizzerà queste informazioni di configurazione nella gestione delle dipendenze tra componenti, delle interazioni tra componenti e nel sostenere gli attributi di qualità dei componenti

- Object Manager (POSA4)

- Il pattern *Object Manager* si occupa di gestire le istanze (oggetti) dei diversi tipi di componenti, di gestire il loro ciclo di vita, e di gestire le loro risorse, relazioni e dipendenze – ed inoltre di fornire un accesso controllato alle istanze dei componenti
  - alcuni oggetti di un'applicazione (come le istanze dei componenti, ma anche thread e connessioni) richiedono un'opportuna gestione del loro ciclo di vita e un controllo accurato dell'accesso
  - queste funzionalità non vanno implementate né negli oggetti stessi né nei loro client – perché altrimenti ne aumenterebbe le responsabilità e la complessità, e ne renderebbe difficile l'uso e l'evoluzione

Object Manager

- La soluzione proposta dal pattern *Object Manager*
  - separa l'utilizzo di un oggetto dal controllo del suo ciclo di vita e del suo accesso
  - introduci un gestore degli oggetti (*object manager*) per gestire gli oggetti di interesse e il loro ciclo di vita
  - i client ottengono gli oggetti da questo gestore degli oggetti – e fruiscono in modo trasparente dei servizi che fornisce
  - il gestore degli oggetti, mentre gestisce il ciclo di vita dei componenti e le loro interazioni, può applicare pattern e tattiche per controllare le qualità desiderate per i componenti e per l'intera applicazione (come specificato in sede di deployment)

Ciclo di vita dei componenti

- Il ciclo di vita di un componente è definito in termini di un insieme di stati, di transizioni tra stati, e di quello che avviene in ciascuna transizione di stato
  - ad es., i componenti EJB più semplici hanno gli stati *Does not exist* (*DNE*) e *Ready*
    - quando un componente viene creato (*DNE* → *Ready*), il contenitore si occupa di soddisfare le sue dipendenze e di assegnargli le risorse di cui ha bisogno – e poi invoca i metodi annotati **@PostConstruct** (per ottenere risorse non gestite direttamente dal contenitore)
    - quando un componente è nello stato *Ready*, può eseguire le proprie operazioni di "business"
    - quando un componente deve essere distrutto (*Ready* → *DNE*), il contenitore si occupa prima di liberare le sue risorse – e invoca anche i metodi annotati **@PreDestroy** (per liberare risorse non gestite dal contenitore)

- Altri pattern per la gestione delle risorse

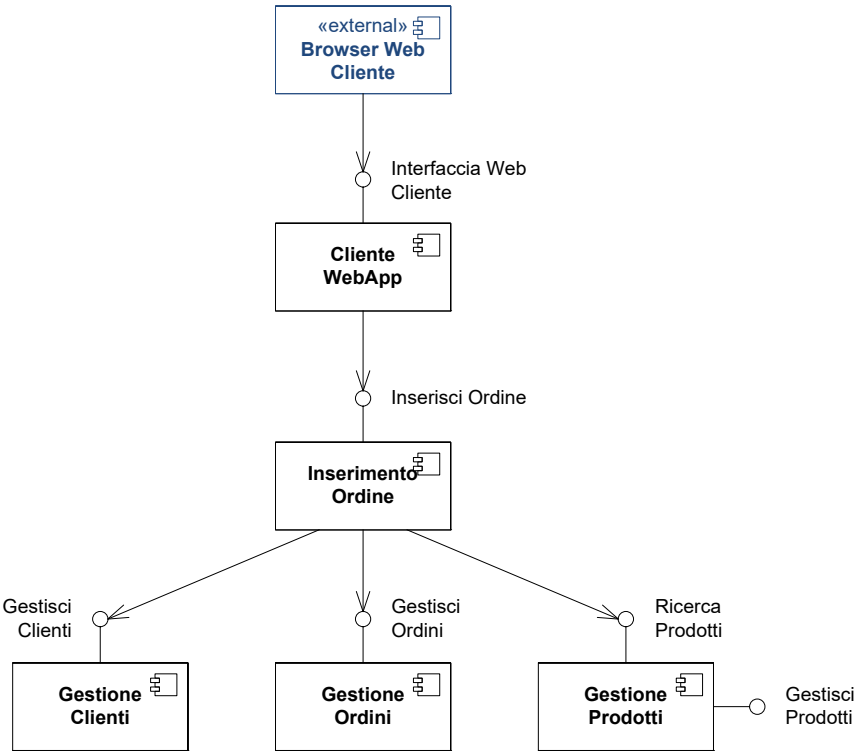
- *Lookup*
  - un servizio di registry, per registrare i riferimenti ai servizi/componenti (quando divengono disponibili) e per de-registrarli (quando non sono più disponibili)
- *Virtual Proxy*
  - un proxy per un oggetto, anche se esso non esiste ancora in memoria
- *Resource Pool*
  - mantiene un certo numero di istanze di una risorsa in un pool di risorse in memoria

Altri pattern per la gestione delle risorse

- *Activator*
  - minimizza il consumo di risorse attivando i servizi su richiesta
- *Evictor*
  - uno “sfrattatore” per monitorare l’uso delle risorse e controllare la loro vita
- *Lifecycle Callback*
  - gli eventi fondamentali del ciclo di vita degli oggetti (o componenti) che l’object manager deve gestire sono definiti come metodi di callback di un’interfaccia
  - questi metodi di callback sono implementati dagli oggetti (o componenti) – e chiamati dall’object manager quando necessario
  - gli oggetti possono così partecipare al sostegno delle qualità desiderate dell’applicazione

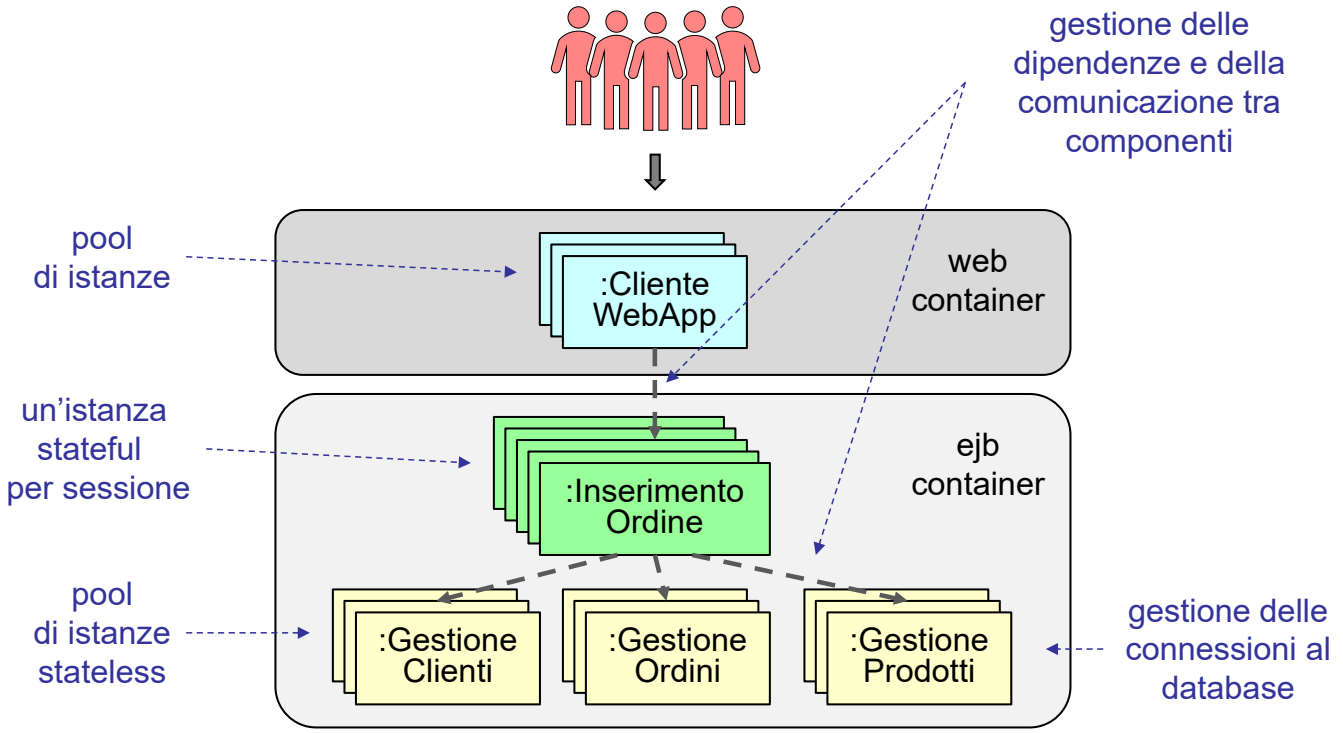
* Un esempio di applicazione a componenti

- Consideriamo di nuovo l’esempio di un’applicazione a componenti per la gestione di ordini, fatti da clienti relativamente a prodotti



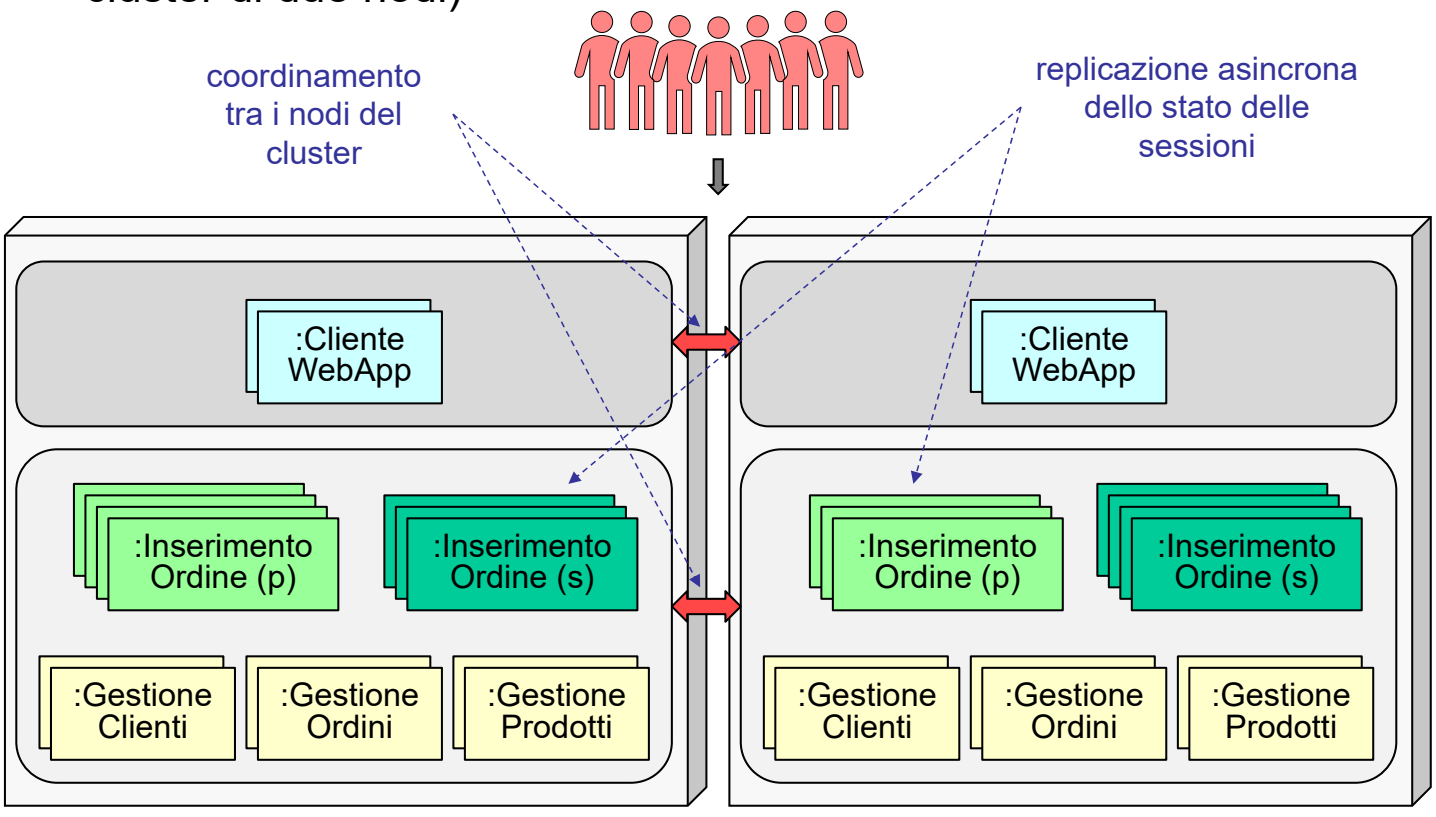
Un piccolo esempio

- Gestione dei componenti e delle istanze dei componenti



Un piccolo esempio

- Supporto per la scalabilità e la disponibilità (esecuzione in un cluster di due nodi)



- Discussione

- Discutiamo brevemente il supporto fornito da alcuni application server (AS) alla scalabilità orizzontale
  - nota: gli AS commerciali moderni potrebbero fornire un supporto alla scalabilità migliore di quanto qui discusso
  - scalabilità lungo l'asse Y
    - supporto alla decomposizione in componenti dell'applicazione (nell'ambito di ciascun contenitore)
    - supporto all'esecuzione dei diversi contenitori dell'AS (web, ejb, …) in nodi differenti
  - scalabilità lungo l'asse X
    - supporto alla replicazione delle istanze dei componenti, diversificata per tipo di componente (nell'ambito di ciascun contenitore) – e alla replicazione dello stato delle sessioni
    - supporto alla scalabilità di un'intera applicazione su più nodi – ma in genere su un numero di nodi limitato

* Discussione

- Un contenitore per componenti è un ambiente runtime per l'esecuzione di componenti e di applicazioni a componenti
  - realizza l'infrastruttura per la gestione del ciclo di vita dei componenti e della loro comunicazione
  - fornisce ai componenti i servizi richiesti per una loro corretta esecuzione e per supportare le loro qualità
  - alcune di queste qualità possono essere raggiunte con un'implementazione distribuita del contenitore

Discussione

- I contenitori per componenti sono essenzialmente delle soluzioni auto-contenute e mono-tecnologiche
  - questo è certamente una limitazione delle tecnologie a componenti
  - ma è proprio grazie a questa “limitazione” che i contenitori riescono a fornire un servizio di supporto efficace ai componenti e ad effettuare delle buone ottimizzazioni