Modelli a Componenti e Enterprise Java Beans

Università di Bologna
CdS Laurea Magistrale in Ingegneria Informatica
I Ciclo - A.A. 2013/2014

02 – Modelli a Componenti e Enterprise Java Beans (base)

Docente: Paolo Bellavista
paolo.bellavista@unibo.it

`http://lia.deis.unibo.it/Courses/sd1314-info/`
`http://lia.deis.unibo.it/Staff/PaoloBellavista/`

Ci occuperemo di…

- Che cosa sono i componenti EJB? Perché utilizzarli?
- Architettura EJB
- Architettura di ***componenti e container***
- Tipi di componenti EJB e ruoli
  - *Session Bean, Message-Driven Bean, Entity Bean* (componenti o oggetti?)
- Modello di ***comunicazione RMI***
- ***Descrittori di deployment*** e packaging di componenti EJB
- ***Modelli EJB2.x e EJB3.x*** a confronto
- Esempi ed esercizi

Definizione di EJB (dalle specifiche)

- "The Enterprise JavaBeans architecture is a ***component architecture*** for the development and deployment of component-based ***distributed business applications***"
- "Applications written using the Enterprise JavaBeans architecture are ***scalable, transactional, and multi-user secure***"
- "These applications may be written once, then ***deployed on any server platform*** that supports the Enterprise JavaBeans specification"

- Una tecnologia per ***componenti server-side***
- Supporto allo sviluppo e al deployment di ***applicazioni distribuite Java enterprise*** che siano:
  - ***multi-tier, transazionali, portabili, scalabili, sicure***, …

EJB: Principi di Design

- Applicazioni EJB e i loro componenti devono essere ***debolmente accoppiati*** *(loosely coupled)*
- Comportamento di EJB definito tramite ***interfacce***
- Applicazioni EJB ***NON*** si occupano della ***gestione delle risorse***
- ***Container*** come supporto al lavoro degli sviluppatori
- ***Applicazioni EJB sono N-tier***
- ***Session tier*** come API verso l'applicazione
- ***Entity tier*** come API verso le sorgenti dati

Scarso Accoppiamento e Interfacce

Scarso accoppiamento (loose coupling)

- Supporto all'integrazione di componenti da vendor differenti
- Componenti EJB riferiscono altri componenti e servizi attraverso ***modi e interfacce predefiniti nelle specifiche***
- Lo sviluppo di componenti EJB ***NON richiede conoscenza e visibilità approfondite dell'ambiente di esecuzione (è sempre auspicabile?)***
- Applicazioni enterprise possono essere assemblate tramite "semplice" composizione di componenti separati

Interazioni di componenti EJB con clienti sono ***specificate completamente in termini di interfacce Java***

- Interfacce ***espongono i metodi*** che i clienti possono invocare, definendo così un "contratto"
- Implementazione nascosta ai clienti
- Supporto a portabilità e modularità

Gestione Risorse e Container

Gestione risorse

- Componenti EJB ***accedono risorse esterne*** (database, sistemi legacy, …) tramite il loro ***container***
  - Nessun bisogno di ***allocazione/deallocazione esplicita*** delle risorse da parte del programmatore
- La ***gestione delle risorse è compito del container***, con obiettivi di ***massima efficienza***
  - Container tipicamente configurato da amministratori di sistema

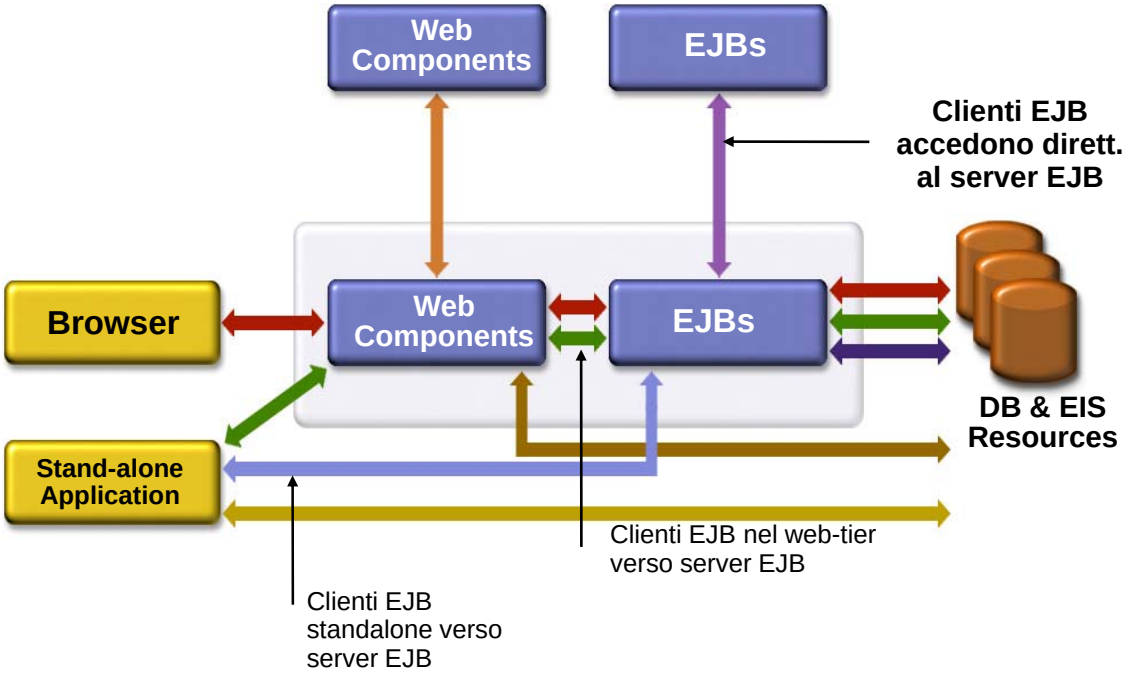
Container fornisce ***servizi di sistema***

- ***Persistenza*** - ***Sicurezza***
- ***Transazionalità*** - ***Connection pooling***
- ***Gestione lifecycle componenti*** - ***Threading***

Gli sviluppatori di applicazioni specificano requisiti in ***modo dichiarativo*** (tramite *deployment descriptor o annotazioni*)

EJB e loro Utilizzo da parte di Clienti Differenti

Componenti EJB possono essere utilizzati in diverse architetture N-tier e ovviamente da più clienti, anche simultanei



Perché EJB?

- Benefici del ***modello a componenti*** per ***lato server***
- ***Separazione*** fra logica di ***business*** e codice di ***sistema***
- Framework di supporto per ***componenti portabili***
  - su differenti server J2EE-compliant e su differenti ambienti
- Supporto a facile configurazione ***a deployment-time***
  - Deployment descriptor e annotazioni
- Necessità di un tier con EJB per sfruttare ***funzionalità di middleware*** offerte dal container
  - Gestione risorse, gestione life-cycle delle istanze, controllo concorrenza e threading
  - Persistenza, transazioni e gestione della sicurezza
  - Scalabilità, affidabilità, disponibilità

Servono sempre?
Sì, se si vogliono realizzare componenti di business ***portabili, scalabili e riutilizzabili***

Servono sempre?
No, se l'obiettivo è una semplice applicazione per leggere tabelle da un database

Classici Problemi "Riconosciuti" di EJB 2.x

- Troppo "pesante" in termini di overhead
- Modello di programmazione complicato, ***non conforme al più classico modello OO***
- Difficoltà di testing

Dunque, in genere, si conviene che EJB 2.x sia (o meglio sia stata) una tecnologia a componenti ***utile per applicazioni distribuite solo in presenza di vincoli importanti di transazionalità e scalabilità***

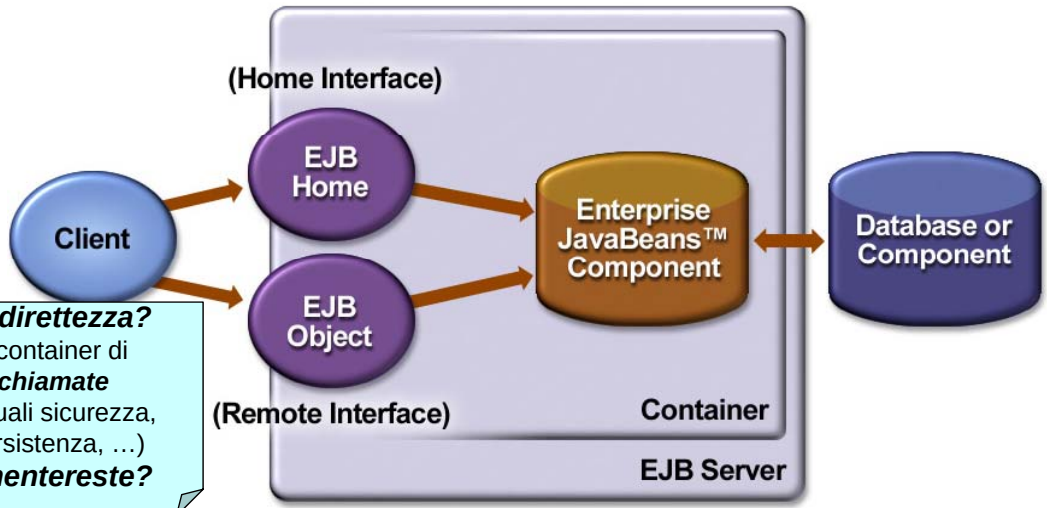
Anche per questo si è passati al modello di componenti EJB 3.x

Architettura EJB 2.x

Interfaccia EJB Home: definisce i metodi che saranno usati dai clienti per ***creare e ritrovare i componenti bean***

Interfaccia EJB Object (o remote): definisce i metodi di business per il componente

Container, a tempo di deployment, creerà due oggetti interni, EJB Home e EJB Object, implementazioni di tali interfacce definite dallo sviluppatore



Perché questa indirettezza?
Per consentire al container di ***intercettare le chiamate*** (servizi di sistema quali sicurezza, transazionalità, persistenza, …)
Come la implementereste?

Contratti in EJB 2.x (1)

1) Client view contract

- Contratto fra un cliente e il container
- Il cliente di un componente EJB può essere un componente del Web tier (Servlet, JSP), una applicazione Java standalone, una applet, un componente EJB (nello stesso container o in uno diverso), un cliente Web Services (a partire da EJB2.1)
- Un contratto client view include:
  - Home interface
    - (per clienti locali e remoti) contiene i metodi per la creazione e il binding a componenti bean
  - Object Interface (chiamata interfaccia remota o logica)
    - (per clienti locali e remoti) contiene metodi di business
  - Identità dell'oggetto
  - … (ci disinteressiamo del dettaglio; vedremo EJB3.x)

Contratti in EJB 2.x (2)

2) Component contract

- Contratto fra un ***componente Enterprise Bean e il suo container***
- Abilita le invocazioni dei metodi dai clienti
- ***Gestisce il ciclo di vita*** delle istanze dei componenti EJB
- ***Implementa le interfacce Home e Object***
- Supporta la ***persistenza*** per certi tipi di componenti (Entity Bean)
- Fornisce informazioni di ***contesto*** a runtime ai componenti
- Gestisce ***transazioni, sicurezza, eccezioni***, ecc.
- Implementa il meccanismo delle ***callback***

Architettura EJB: i Fondamentali

- La ***vista client*** di un componente EJB è strettamente definita dalle ***interfacce***
- EJB sono isolati e supportati da un container EJB
- Il container EJB fornisce l'illusione di un ***singolo ambiente single-threaded***
- Il container EJB si occupa di gestire le ***transazioni*** con database e le problematiche di ***accesso e sicurezza***
- Operazioni di ***creazione e ritrovamento*** di componenti EJB sono ***standardizzate***
- Possibilità di ***pooling di istanze*** per ragioni di ***efficienza***
- Il container si occupa della ***gestione efficiente delle risorse***
- Processo di ***deployment standardizzato***

EJB sono Isolati e Supportati da Container

- Clienti possono invocare unicamente i metodi esposti dalle interfacce EJB
  - Interfaccia Home (vera e propria factory)
  - Interfaccia Object (con i metodi di business)
- Le chiamate cliente verso i metodi EJB sono ***intercettate dal container*** prima che questo le "deleghi" ai componenti EJB veri e propri
  - Oggetti proxy (oggetti EJBHome e EJBObject) sono ***generati*** dal container
- Come già detto, il container fornisce ***servizi di sistema*** verso i componenti EJB

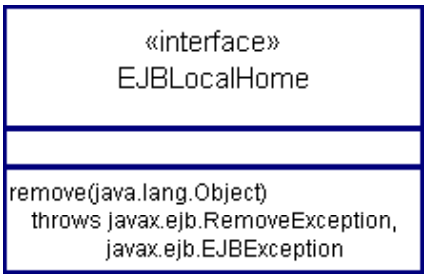
"Illusione" di ambiente single-threaded

- Sviluppatori di componenti non hanno necessità di ***occuparsi del controllo della concorrenza***

Contratto di Interfaccia

Ad esempio, per EJBHome (da estendere da parte dello sviluppatore) due versioni, remota e locale

(vedremo alcuni, pochi, dettagli nel seguito…)



Vi ricordate l'interfaccia **`java.rmi.remote`**, vero?

Ancora su Container EJB

- Il container gestisce ***transazioni sia locali che distribuite*** (vedremo come…)
- Il container supporta ***transazioni le cui proprietà sono definite in modo dichiarativo*** (vedremo che cosa sono…)
- Il container gestisce il ***controllo degli accessi***
  - Quali metodi sono accessibili, e a quali ruoli
  - ***Controllo degli accessi specificato in modo dichiarativo nel descrittore di deployment*** (o annotazioni in EJB3.x)
  - Limitata gestione del controllo degli accessi anche da codice Java
- Il container si occupa anche di ***schemi di autenticazione***
  - Chi sviluppa componenti non si deve mai occupare di fornire codice per le procedure di autenticazione

Creare Componenti EJB e Pooling

Modalità ben definite per i clienti per ***istanziare*** nuovi componenti EJB e/o per ***trovare*** componenti esistenti

- Clienti usano ***JNDI per ottenere un oggetto proxy*** (in realtà il riferimento a uno stub ***dell'oggetto EJBHome***)
- Poi i clienti invocano i metodi ***create() o find() di Home*** per accedere ad un ***altro oggetto proxy*** (in realtà il riferimento a uno stub ***dell'oggetto EJBObject***)
- I clienti interagiscono sempre con oggetti proxy, ***mai in modo diretto con l'istanza del componente EJB***

Container può gestire il ***pooling di istanze di componenti***

- Il container ***"conosce" quando creare o eliminare*** istanze di componenti
- Quando un cliente chiede di creare un bean al container tramite il metodo create(), il container probabilmente ***restituirà una istanza già esistente nel pool***, ***in modo completamente trasparente***

Gestione Risorse Esterne e Deployment

- Le risorse esterne sono ***condivise fra i diversi componenti EJB*** (gestione del pooling di queste risorse da parte del container) e includono:
  - Database
  - Enterprise Information System (EIS)
  - Sistemi di messaging, …
- Le specifiche EJB standardizzano
  - ***Packaging*** di una applicazione EJB
  - ***Descrittore di deployment***
- Ogni piattaforma conforme a J2EE deve essere in grado di fare il ***deployment di qualunque applicazione EJB-compliant***

Quindi, il Container si Occupa di…

- ***Generare*** automaticamente ***classi concrete*** per
  - Interfaccia EJBHome (remota o locale al cliente?)
  - Interfaccia EJBObject (remota o locale al cliente?)
- Effettuare il ***binding dell'oggetto Home*** presso il servizio di naming
  - I clienti possono fare lookup di componenti (oggetti home) usando JNDI
- Creare e ***gestire un pool di istanze*** di "componenti liberi"
- Effettuare il ***caching dei componenti*** acceduti di ***recente***
- Gestire il pool di connessioni JDBC verso database

Un Po' di Terminologia… (1)

- Il termine "EJB" è usato in molte accezioni differenti
  - Technologia, Architettura, Specifica, Implementazione, Prodotto, Server, Container
  - Classe di un componente (Bean)
  - Istanza di un componente (Bean)
  - Modulo EJB
  - Applicazione EJB
  - Oggetto proxy EJB

Noi cercheremo di usare correttamente i termini:

- ***Classe Bean*** (o classe di implementazione)
  - Classe Java che implementa il componente (Bean)
- ***Istanza di Bean***
  - La reale istanza dell'oggetto Bean all'interno di un EJB container

Un Po' di Terminologia… (2)

- ***Modulo EJB*** (o EJB jar file)
  - ejb-jar file
  - Insieme (collezione) di classi di componenti Bean
- ***Applicazione EJB*** (o applicazione J2EE)
  - *.ear file
  - Insieme di moduli EJB e/o file *.war (tipo di archivio usato dalle servlet)
- ***Interfaccia EJBHome***
  - Interfaccia Java che contiene i metodi per creazione/ritrovamento
- ***Oggetto EJBHome*** (implementa l'interfaccia EJBHome)
  - Chiamato anche ***oggetto factory***
- ***Interfaccia EJBObject*** (o interfaccia remote o interfaccia logica)
  - Interfaccia Java che descrive i metodi di business
- ***Oggetto EJBObject*** (implementa l'interfaccia EJBObject)
  - Implementato in modo automatico

Gli oggetti EJBHome e EJBObject sono chiamati anche oggetti proxy

Tipologie di Componenti Bean

- ***Session Bean***
  - ***Stateful session bean***
  - ***Stateless session bean***
- ***Entity Bean***
  - Bean Managed Persistence (BMP)
  - Container Managed Persistence (CMP)
- ***Message Driven Bean***
  - Per Java Message Service (JMS)
  - Per Java API for XML Messaging (JAXM)

Session Bean (1)

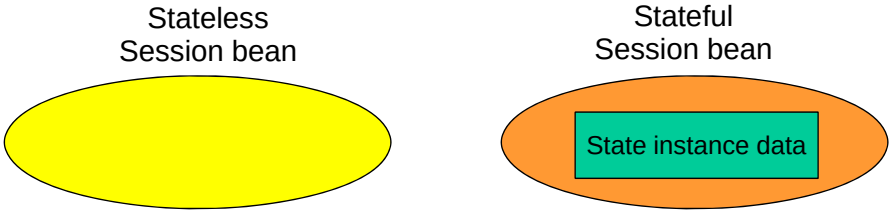
- Lavorano tipicamente per un singolo cliente
- ***Non sono persistenti*** (vita media relativamente breve)
  - Persi in caso di failure di EJB server
- ***Non rappresentano dati in un DB***, anche se possono accedere/modificare questi dati
- La classe Bean corrispondente implementa l'interfaccia javax.ejb.SessionBean

Quando usare i Session Bean?

- Per modellare oggetti di processo o di controllo ***specifici per un particolare cliente***
- Per modellare workflow o attività di gestione e per coordinare interazioni fra bean
- Per muovere la ***logica applicativa di business*** dal lato cliente a quello servitore

Session Bean (2)

- ***Stateless***: esegue una richiesta e restituisce risultato ***senza salvare alcuna informazione di stato relativa al cliente***
  - transienti
  - elemento temporaneo di business logic necessario per uno specifico cliente per un intervallo di tempo limitato
- ***Stateful***: può mantenere ***stato specifico per un cliente***



Esempi di Session Bean

- ***Session bean senza stato***

  Per fare alcuni esempi molto semplici: ***consultazione di catalogo di merci o calcolo degli interessi*** su una somma depositata

  - Nessuna necessità di mantenere stato client-specific
  - Anche business logic senza necessità di accesso a database

- ***Session bean con stato***

  Ad esempio, ***carrello della spesa*** (shopping cart)

  - necessità di mantenere stato client-specific

Come gestire il ***pooling*** di questi SB? Fa differenza se si tratta di SB con stato o senza stato?

Entity Bean in EJB2.x (1)

- Forniscono una ***vista ad oggetti*** dei dati mantenuti in un database
  - Tempo di vita ***non connesso alla durata delle interazioni*** con i clienti
  - Componenti permangono nel sistema fino a che i dati esistono nel database - ***long lived***
  - Nella maggior parte dei casi, ***componenti sincronizzati con i relativi database relazionali***
- ***Accesso condiviso*** per clienti differenti
- La classe Bean corrispondente implementa l'interfaccia javax.ejb.EntityBean

perché?

Entity Bean in EJB2.x (2)

- I clienti usualmente ***cercano (look up) entity bean esistenti***
  - Creare un entity bean significa usualmente aggiungere una riga a una tabella di database
  - Trovare un entity bean significa determinare una riga in una tabella di database esistente
  - Rimuovere un entity bean significa eliminare una riga da una tabella di database
- ***Ogni istanza di un entity bean ha un identificatore unico chiamato chiave primaria***
- Esempio di entity bean: ***cliente (e sua profilazione)***
  - I dati relativi al cliente devono persistere e sono mantenuti in un database, fault tolerant rispetto ai guasti del server
  - I dati cliente possono essere condivisi da diverse applicazioni
  - Ogni cliente deve avere un identificatore unico

2 Tipologie di Entity Bean in EJB2.x

Rapidamente, visto che saremo più interessati a utilizzare il nuovo modello di persistenza in EJB3.0…

- ***Container Managed Persistence (CMP)***
  - Persistenza ***gestita completamente dal container***
  - Requisiti di persistenza specificati interamente nel ***descrittore di deployment***
  - Sviluppatori dei bean CMP ***NON*** devono occuparsi in alcun modo della logica di persistenza ***nel codice del componente***
  - Efficienza, performance, facilità di sviluppo e deployment
- ***Bean Managed Persistence (BMP)***
  - Il codice della logica di persistenza è ***responsabilità dello sviluppatore del bean BMP***
  - ***Maggiore controllo*** a livello di programmazione

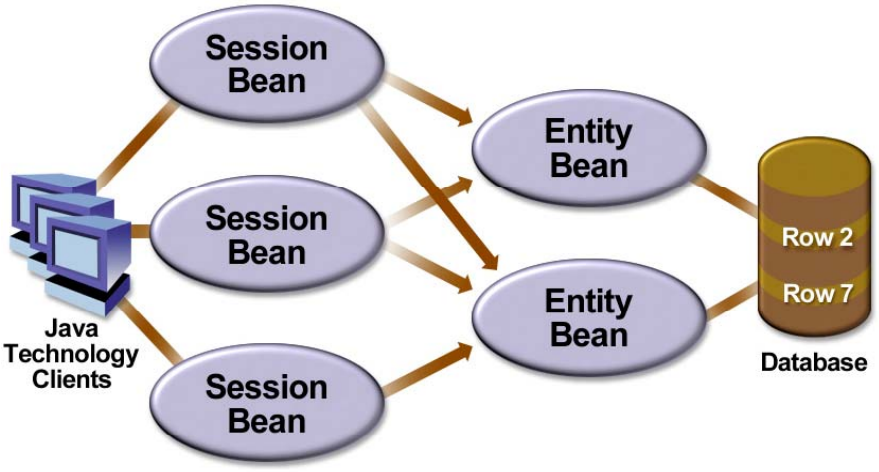
Session ed Entity Bean

Session Bean

- Rappresenta un processo di business
- Una istanza per cliente
- **Short-lived:** vita del bean pari alla vita cliente
- Transient
- Non sopravvive a crash del server
- Può avere proprietà transazionali

Entity Bean

- Rappresenta dati di business
- Istanza condivisa fra clienti multipli
- **Long-lived:** vita del bean pari a quella dei dati nel database
- Persistente
- Sopravvive a crash del server
- Sempre transazionale



Message-Driven Bean (MDB)

- Svolgono il ruolo di ***consumatori di messaggi asincroni***
- ***Non possono essere invocati direttamente dai clienti***
  - Attivati in seguito all'arrivo di un messaggio
  - Non hanno interfacce EJBHome e EJBObject
- I clienti possono interagire con MDB tramite l'invio di messaggi verso le ***code o i topic per i quali questi componenti sono in ascolto*** *(listener)*
- ***Privi di stato***

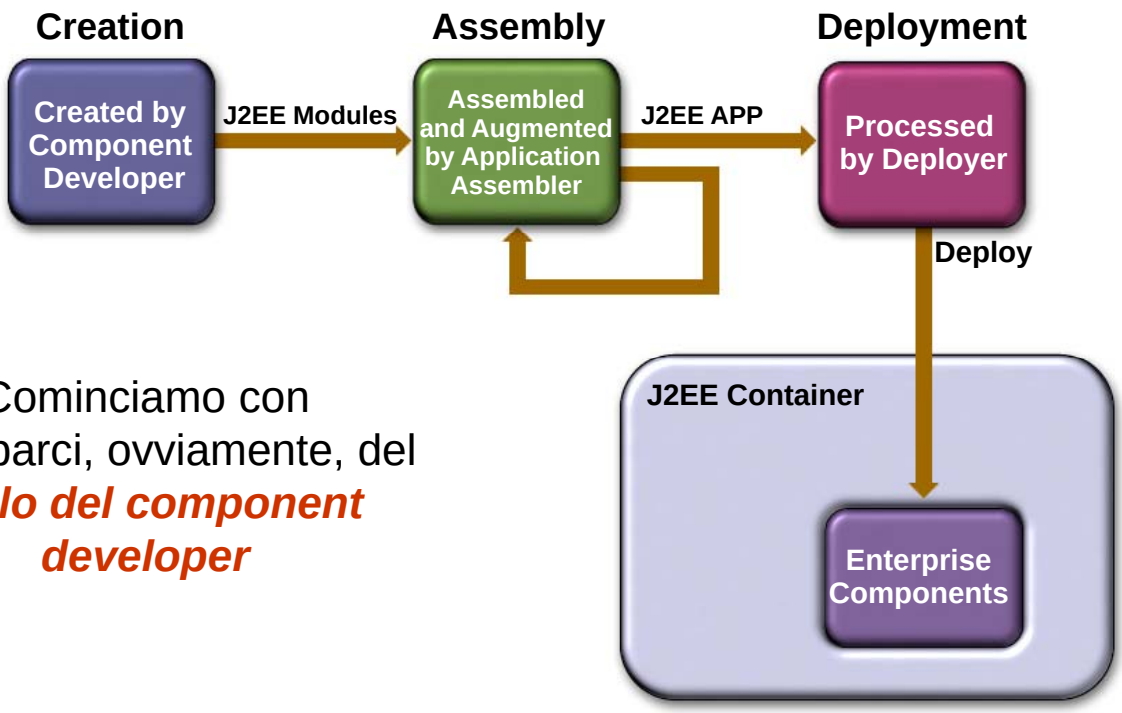
Perché è così importante che siano asincroni?

MDB e JMS

- Nel caso di ***utilizzo di JMS***
  - Il bean MDB corrispondente deve implementare l'interfaccia javax.jms.MessageListener interface
  - L'implementazione del metodo onMessage() deve contenere la business logic
  - Il bean viene configurato come ***listener per una queue o un topic JMS***

- Comunque, le API JMS per l'invio di messaggi sono disponibili per qualunque tipo di componenti EJB
  - ***Utilizzo in unicast***, ad es. per Reliable Queuing
  - ***Utilizzo in modalità pub/sub***

Ciclo di Vita e Ruoli



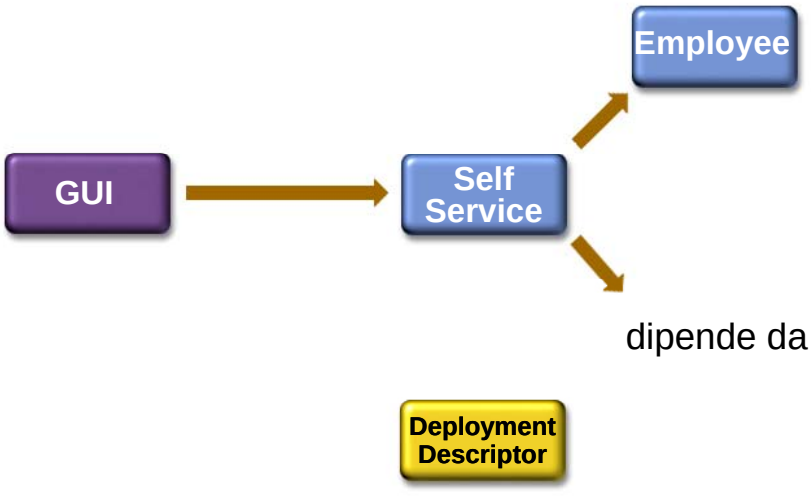
Cominciamo con l'occuparci, ovviamente, del ***ruolo del component developer***

Esempio di Semplice Scenario: Passo 1

Produttore di Software A crea un **EJB Payroll (busta paga)**

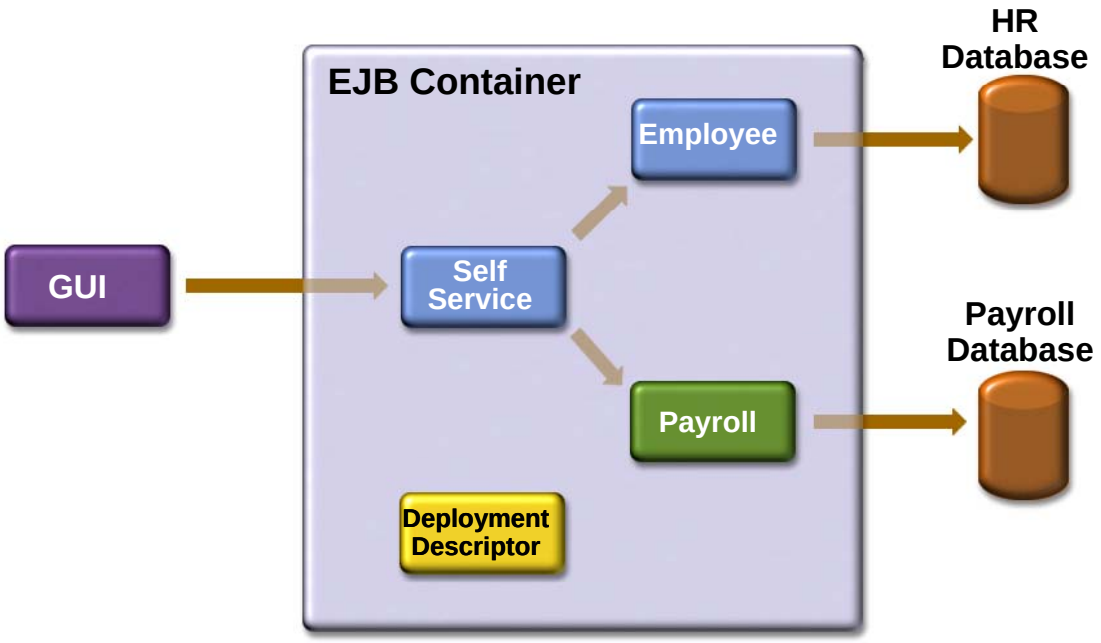
Esempio di Semplice Scenario: Passo 2

Venditore B: **Sviluppatore di EJB e Assemblatore di Applicazioni**



Esempio di Semplice Scenario: Passo 3

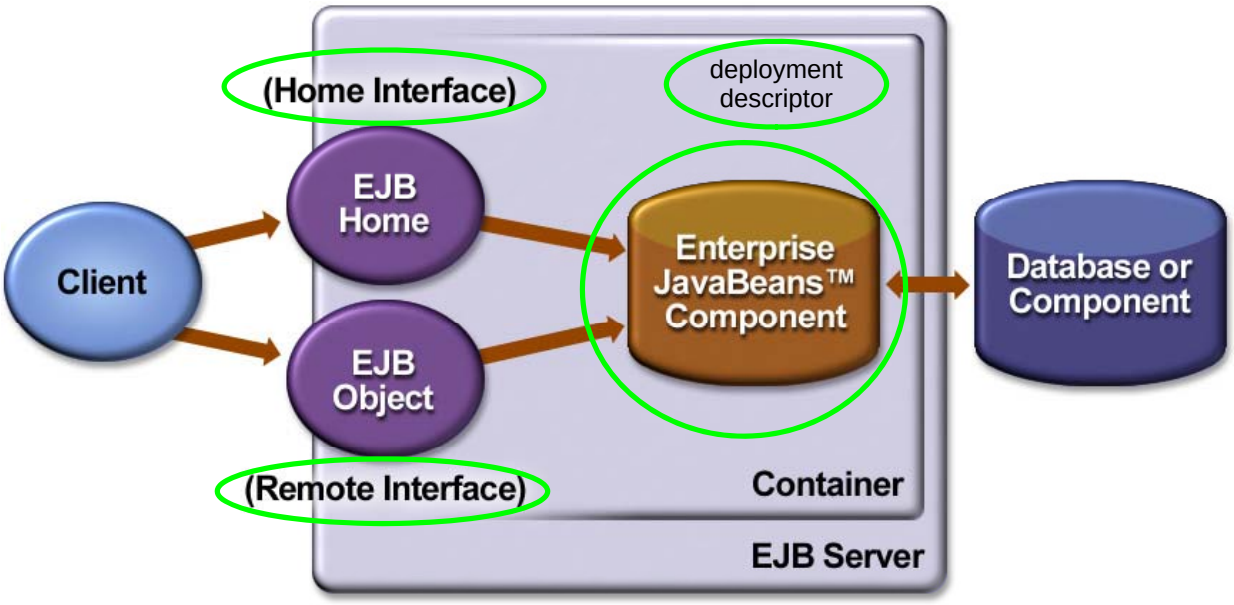
Un terzo ruolo può essere quello di chi si occupa del solo ***deployment in uno specifico container target***



Scendiamo in Esempi Pratici: Contenuti di un Modulo EJB

- Lo sviluppatore di EJB ***crea moduli EJB (file EJB-JAR)***
- Un modulo EJB contiene
  - ***Classi di interfaccia*** (ci devono essere)
    - Interfaccia EJBHome
    - Interfaccia EJBObject (o remote)
  - ***Classi per il componente*** (ci devono essere)
  - ***Descrittore di deployment*** (ci deve essere)
  - Classi Helper (presenti solo quando necessarie per la specifica classe del bean)

Scendiamo in Esempi Pratici: Contenuti di un Modulo EJB



Scendiamo in Esempi Pratici: Interfaccia EJBHome

Interfaccia EJBHome

- Dichiara i metodi per la ***creazione, il ritrovamento e la distruzione*** di bean
  - Svolge il ruolo di ***interfaccia factory***
- ***Implementata dal container***
  - L'implementazione è l'oggetto EJBHome, che viene sviluppato in modo automatico (strumenti di supporto)
- Il cliente ottiene il ***riferimento all'oggetto stub dell'oggetto EJBHome*** tramite JNDI
- Può essere ***remota e/o locale***

Esempio: Interfaccia EJBHome (remota)

```
package com.ejb_book.interest;
import javax.ejb.*;
import java.rmi.*;

// Interfaccia Home (remota) per un EJB di nome Interest

public interface InterestHome extends EJBHome{

   //Crea una istanza di EJB
   public Interest create()
          throws CreateException, RemoteException;
}
```


Esempio: Interfaccia EJBHome (locale)

```
package com.ejb_book.interest;
import javax.ejb.*;
import java.rmi.*;


// Interfaccia Home (locale) per un EJB di nome Interest
public interface InterestLocalHome extends EJBLocalHome {

   // Crea un'istanza di EJB
   public InterestLocal create() throws CreateException;
}
```

Definizione diversa dell'interfaccia a seconda di locale e/o remota

Scendiamo in Esempi Pratici: Interfaccia EJBObject

Interfaccia EJBObject

- Dichiara i metodi della logica applicativa (metodi di business)
- ***Implementata dal container***
  - ***Oggetto EJB***
- Il cliente ottiene il riferimento all'oggetto stub di EJBObject attraverso i metodi create() o find() ***dell'interfaccia EJB Home***
- Può essere ***remota o locale***

Perché non può essere direttamente l'oggetto con la logica applicativa prodotto dallo sviluppatore finale?

Esempio: Interfaccia EJBObject Interest (remota)

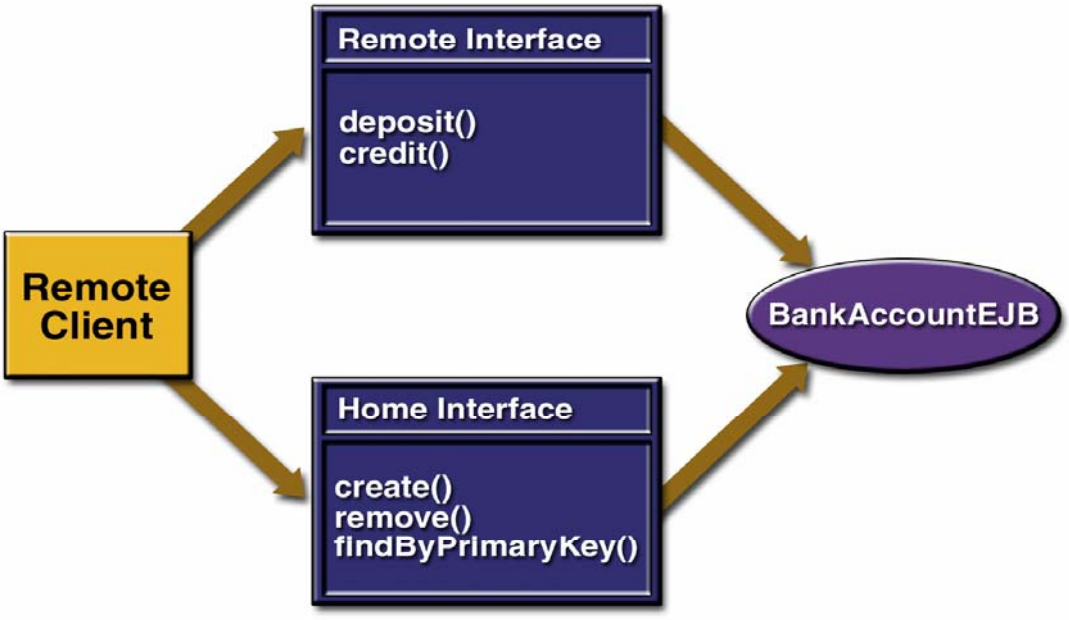
```
package com.ejb_book.interest;
import javax.ejb.*; import java.rmi.*;

// Interfaccia remota del componente Interest
public interface Interest extends EJBObject{
   // Calcola l'interesse da pagarsi ad un dato proprietario, ad uno
   // specifico tasso di interesse (percentuale per term)
   public double getInterestOnPrincipal
        (double principal, double interestPerTerm, int terms)
                                          throws RemoteException;

   // Calcola l'ammontare totale da pagarsi ad un dato propr., ad uno
   // specifico tasso di interesse (percentuale per term)
   public double getTotalRepayment
        (double principal, double interestPerTerm, int terms)
                                            throws RemoteException;

}
```

Esempio: Interfaccia EJBObject Interest (locale)

```
package com.ejb_book.interest;
import javax.ejb.*; import java.rmi.*;

// Interfaccia locale del componente EJB Interest
public interface InterestLocal extends EJBLocalObject {

   public double getInterestOnPrincipal
        (double principal, double interestPerTerm, int terms);

   public double getTotalRepayment
       (double principal, double interestPerTerm, int terms);
}
```


Interfacce EJBHome e EJBObject

In modo analogo nel caso di cliente locale…



Scendiamo in Esempi Pratici: Cliente

Per interagire con il componente EJB il cliente deve:

1. ***Ottenere l'oggetto EJBHome*** (in realtà un oggetto stub per l'oggetto EJBHome) via JNDI
   - ***Ottenere contesto iniziale***
   - ***Effettuare lookup***
   - ***Effettuare narrowing***
2. Dall'oggetto EJBHome, ottenere ***l'accesso all'oggetto EJB desiderato*** (in realtà un oggetto stub per l'oggetto EJBObject)
3. Invocare i metodi di business tramite l'oggetto EJB
4. Effettuare il ***clean up finale***

Esempio: Cliente (1)

```
package com.ejb_book.interest;
import javax.ejb.*; import javax.naming.*;
import javax.rmi.*; import java.rmi.*;

public class InterestTestClient {
     public static void main (String[] args) throws Exception {
          Interest interest = getInterest();
          double principal=10000.0; double rate=10.0; int terms=10;
          System.out.println ("Principal = $" + principal);
          System.out.println ("Rate(%) = " + rate);
          System.out.println ("Terms = " + terms);
          // Passo 3: invocazione metodi di business
          System.out.println ("Interest = $" +
           interest.getInterestOnPrincipal(principal, rate, terms));
          System.out.println ("Total = $" +
                interest.getTotalRepayment(principal, rate, terms));
          // Passo 4: clean up
          interest.remove();        }
```

Si occupa dei passi 1 e 2

Esempio: Cliente (2)

```
// Ottiene una istanza dell'EJB Interest. Si noti che il codice
// EJB-specific è incluso tutto in questo metodo, cosicché
// la logica nella slide precedente può essere "plain Java"
public static Interest getInterest()
        throws CreateException, RemoteException, NamingException {
 // Passo 1: ottenere un'istanza di EJBHome (in realtà un oggetto
 // stub per l'oggetto EJBHome) via JNDI
        InitialContext initialContext = new InitialContext();
        Object o = initialContext.lookup ("Interest");
        InterestHome home = (InterestHome)
             PortableRemoteObject.narrow (o, InterestHome.class);

 // Passo 2: creare un oggetto EJBObject remoto (in realtà uno
 // stub all'oggetto EJBObject remoto
        return home.create();
    }
}
```

Quali aspetti rendono la chiamata non trasparente (rispetto a invocazione locale)?

Narrowing vs. Casting (1)

Piccola parentesi con tematica di tipo "linguaggistico" (in quanto tale, di minore interesse per questo corso…)

Comunque, perché talora è ***indispensabile fare narrowing e non casting***? In quali casi?

Tutti voi vi ricordate che cos'è e come funziona ***casting***…

È indispensabile che ***sui due lati*** di un'interazione client/server sia disponibile ***un sistema di tipi*** che permetta di effettuare e capire il casting desiderato

E perché potrebbe non essere disponibile?

Ambienti multi-linguaggio in cui vogliamo integrare componenti (o pezzi di software), anche legacy, che non possono accedere/comprendere un determinato sistema di tipi

Narrowing vs. Casting (2)

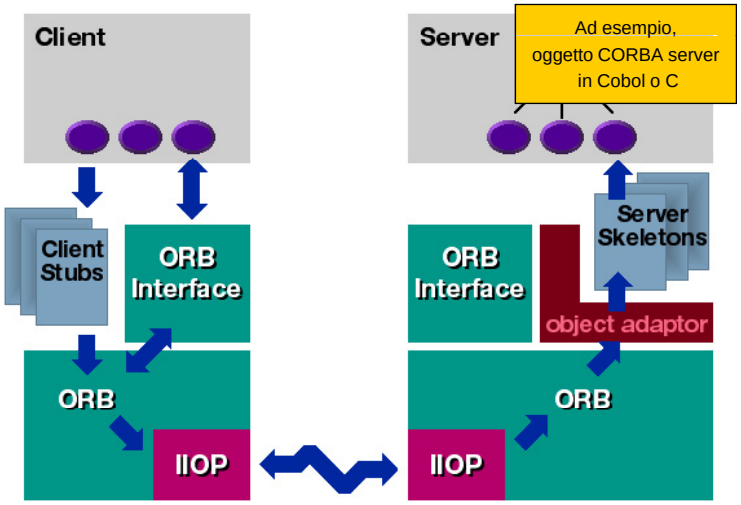
Ad esempio chi si ricorda come funzionano le cose in CORBA? ***Casting non può essere nativo in CORBA*** perché CORBA integra anche componenti in linguaggi che non hanno casting…

E in DCOM esiste il problema?

In EJB, uniformità di Java sui due lati dell'interazione C/S.

Ma è sempre vero?

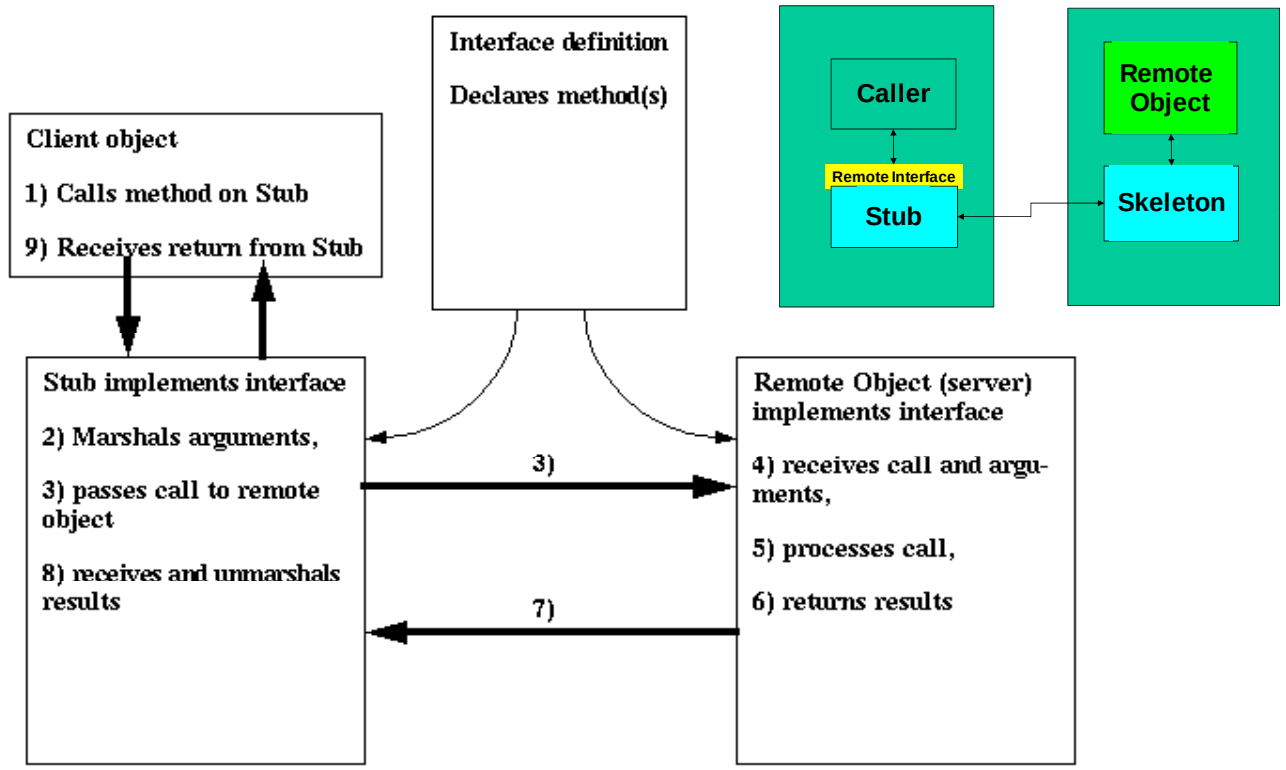
E comunque si è deciso di utilizzare ***RMI over IIOP***…



I Componenti sono Distribuiti!

- Gli oggetti che cooperano ovviamente eseguono in ***JVM differenti*** in tutti i casi di vero interesse
  - ***Gli oggetti lato cliente invocano metodi di oggetti lato server***
- Necessariamente ci sono (gli usuali) meccanismi per
  - ***Condividere la signature*** dei metodi dal cliente al servitore
  - Fare il ***marshalling dei parametri*** da cliente a servitore
  - Fare ***l'unmarsalling dei parametri*** ricevuti lato server
  - Fare il ***marshalling dei valori di ritorno*** dal servitore al cliente
  - Fare ***l'unmarshalling dei risultati*** ricevuti lato cliente

RMI
e dunque "Usuale" Modello

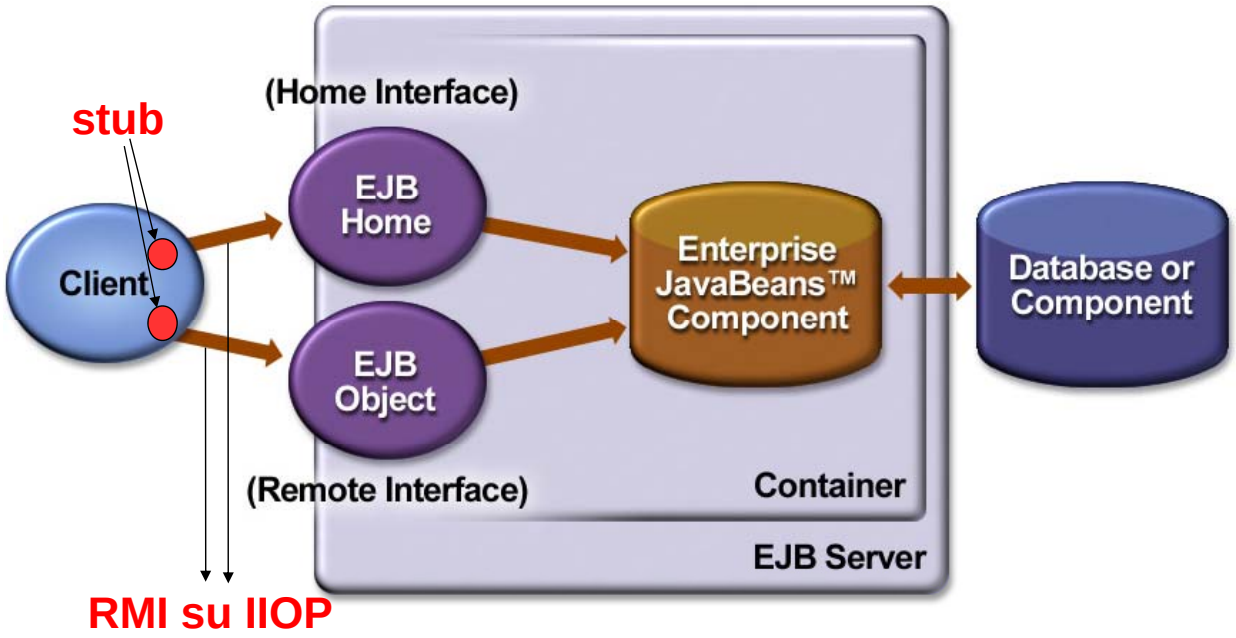


RMI
e dunque "Solito" Modello

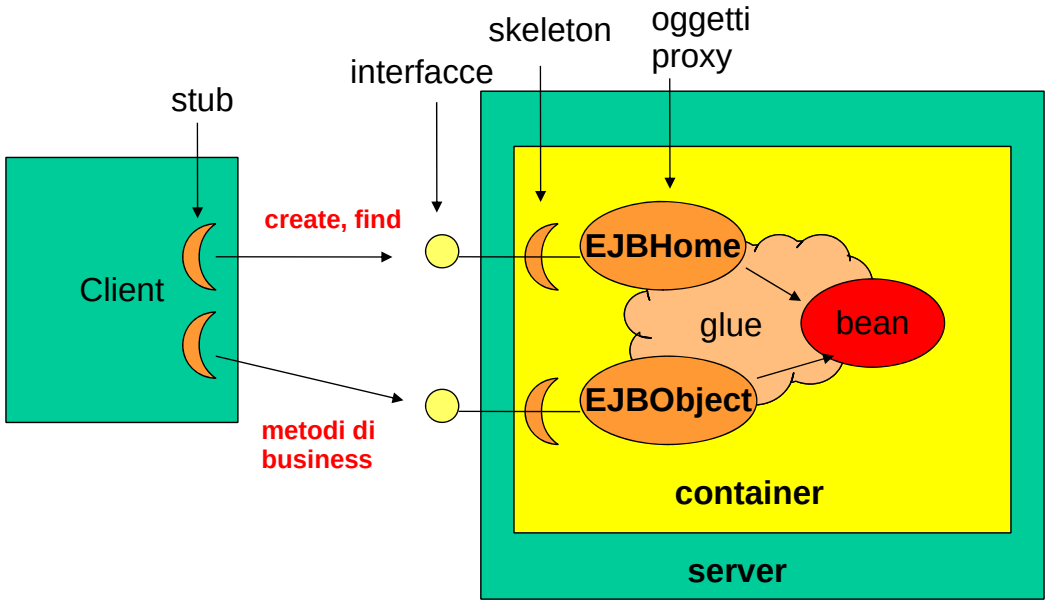
- ***Chiamante (cliente)***
  1. Invoca un metodo dell'oggetto remoto
- ***Lo stub dell'oggetto remoto***
  1. Intercetta l'invocazione di metodo
  2. Effettua il marshalling dei parametri
  3. Effettua la chiamata vera e propria all'oggetto remoto
- ***Oggetto remoto***
  1. Riceve l'invocazione tramite il suo skeleton
  2. Effettua l'unmarshalling dei parametri
  3. Esegue l'invocazione localmente
  4. Effettua il marshalling dei risultati e li invia al cliente
- ***Lo stub dell'oggetto remoto***
  1. Riceve i risultati, effettua unmarshalling e li restituisce al cliente

Oggetti di Supporto Presenti a Runtime (1)

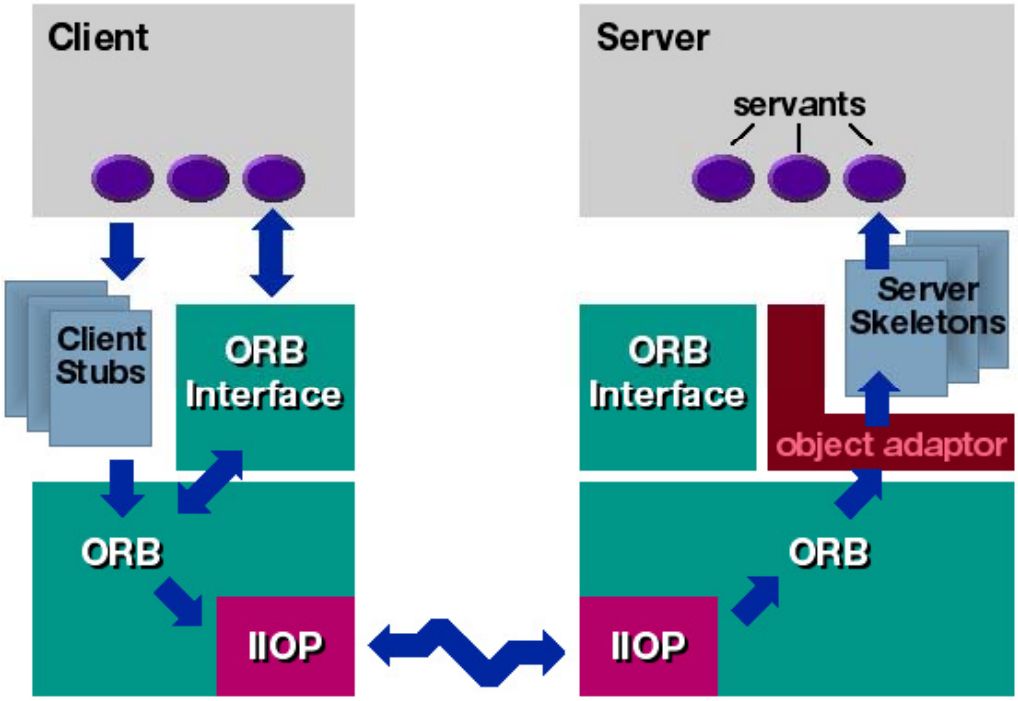
Quindi, quali problematiche di ***efficienza*** derivano ***dall'utilizzo di RMI su IIOP*** (riguardare contenuti di Reti di Calcolatori…)?



Oggetti di Supporto Presenti a Runtime (2)



RMI su IIOP (1)



RMI su IIOP (2)

- ***RMI è utilizzato per la comunicazione fra cliente e server EJB***
  - Prima di EJB 2.0 (J2EE 1.3), RMI su IIOP doveva essere usato anche se cliente e server eseguivano sulla stessa JVM, con conseguente overhead non necessario
  - Oggi alcuni vendor forniscono prodotti J2EE in cui la comunicazione viene ottimizzata se cliente e servitore EJB sono sulla stessa JVM
- ***Le operazioni RMI sono costose*** *(vi ricordate perché?)*
  - Ragione per cui "local interface" è stata introdotta a partire da EJB2.0

Vista Cliente in caso di EJB Locali

- ***Interfacce locali, da usarsi quando il cliente esegue nella stessa JVM del componente EJB di interesse (e del suo container)***
- Nessun ***overhead*** dovuto alla comunicazione RMI su IIOP
- Possibilità introdotta a partire da EJB2.0 (J2EE1.3)
- Ovviamente, in tal caso i metodi non devono produrre ***RemoteException***
- Possibilità di ***chiamata per riferimento*** *("call by reference")*
- Possibilità utilizzata tipicamente per rendere più ***efficiente il funzionamento di session bean*** che svolgono il ruolo di ***clienti locali verso i loro entity bean***

Perché altrimenti come avverrebbe la chiamata? Con quali costi?

Interfacciamento Locale

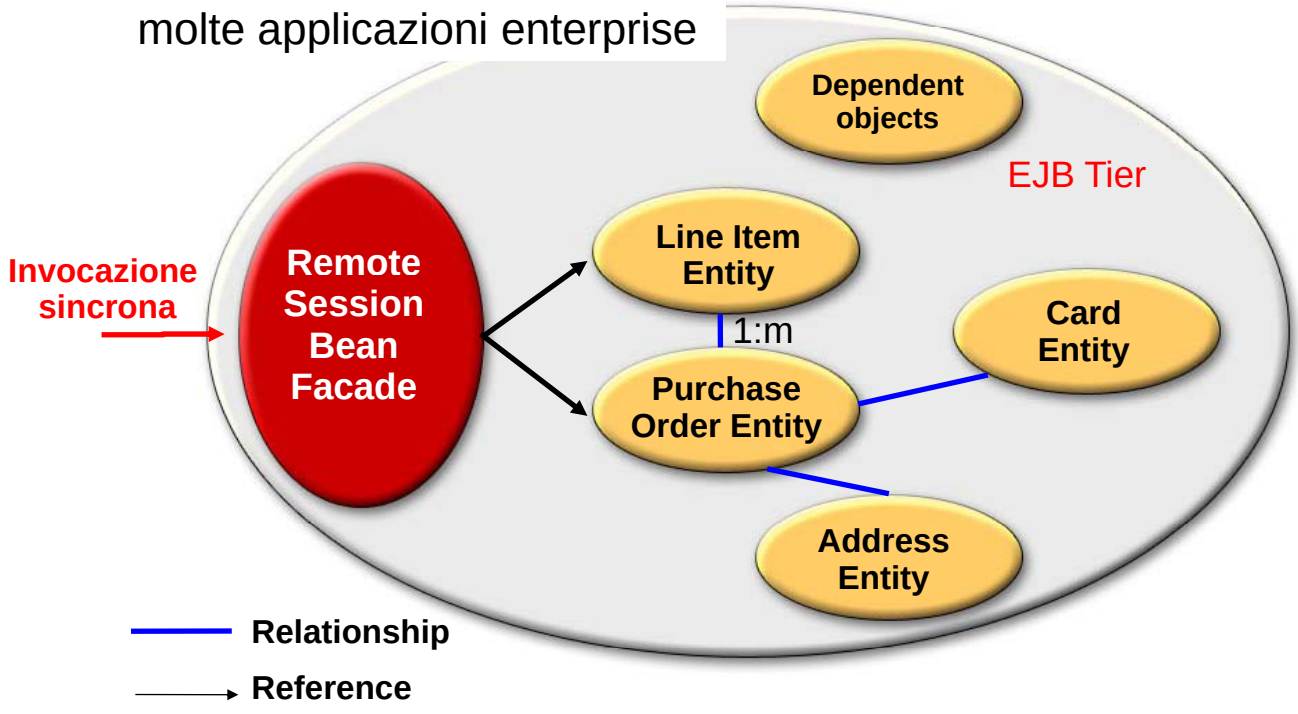
- Vantaggi
  - Accesso più ***efficiente dovuto alla co-locazione*** (no RMI su IIOP)
  - Possibilità di ***condividere dati fra clienti e bean*** tramite ***call by reference***
- Svantaggi
  - ***Accoppiamento*** stretto fra cliente e bean
  - ***Minore flessibilità di distribuzione e deployment***
- Utilizzo di bean con interfacce locali (in contrapposizione a quelli remoti) per ***operazioni fine-grained***
  - Un singolo bean può supportare ***interfacce locali e remote***
  - Un cliente EJB ***NON*** può usarle entrambe
    - ***Decisione compile-time, non runtime***

Suggerimenti

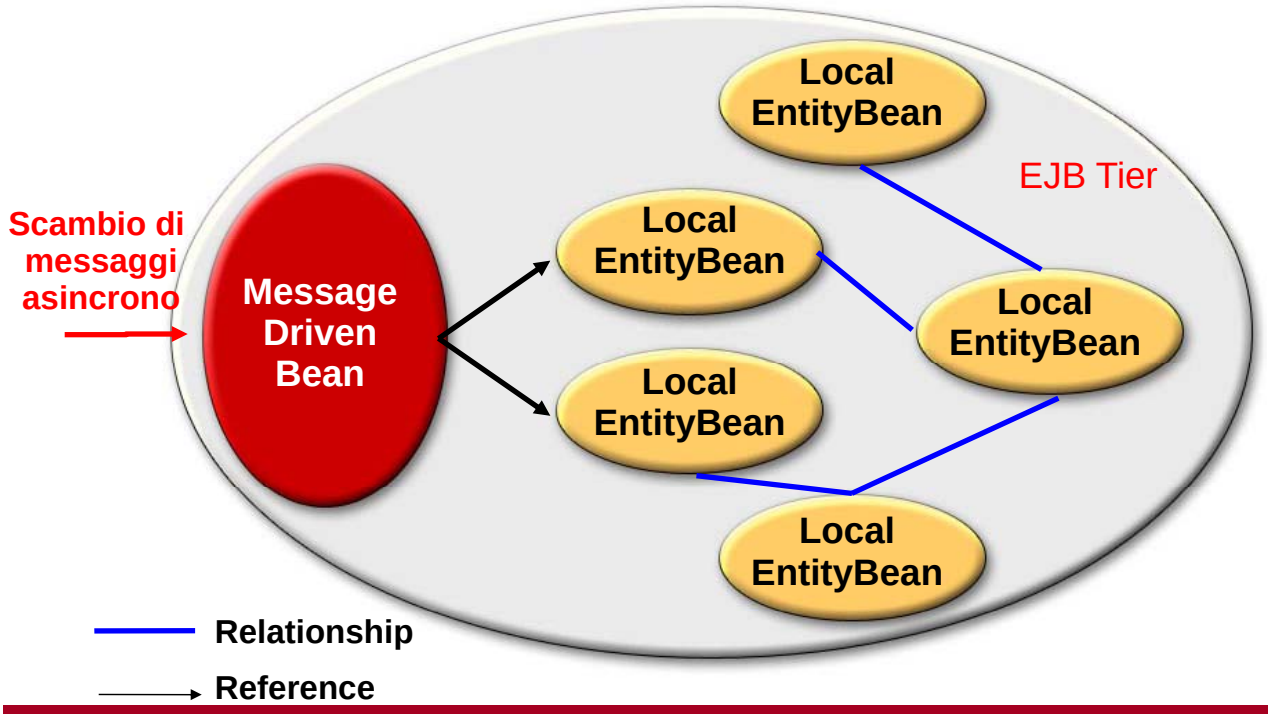
- Utilizzare ***interfacce locali*** ogniqualvolta è possibile
  - ***Creare isole di componenti locali*** (entity bean locali e i loro oggetti dipendenti)
- ***Utilizzare il pattern facade in cui un session bean con interfaccia remota (per operazioni sincrone) o un MDB (per chiamate asincrone) invocano entity bean a loro locali***
- Utilizzare ***interfacce remote per ridurre l'accoppiamento***

Remote Session Bean Facade con un'Isola di Entity Bean Locali

Caso estremamente comune in molte applicazioni enterprise



MDB Facade con un'Isola di Entity Bean Locali



Descrittore di Deployment EJB2.x in EJB-JAR

- Serve a dare ***istruzioni al container su come gestire il componente EJB***
- Supporta ***personalizzazione di tipo dichiarativo*** (*declarative customization*)
- ***Può controllare comportamenti*** per:
  - Transazionalità
  - Ciclo di vita
  - Persistenza
  - Sicurezza
  - Gestione dello stato
  - …
- Definisce il contratto fra produttore e consumatore del file ejb-jar
- È un ***documento XML*** che deve essere ***ben formato*** (in senso XML) e deve essere ***valido in relazione alla DTD*** descritta nella specifica EJB
- Deve essere memorizzato con nome META-INF/ejb-jar.xml nel file ejb-jar

Esempio: Descrittore di Deployment (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD
   Enterprise JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-
   jar_2_0.dtd'>

<ejb-jar>
  <display-name>Interest_ejb</display-name>
  <enterprise-beans>
    <session>
      <display-name>InterestBean</display-name>
      <ejb-name>InterestBean</ejb-name>
      <home>com.ejb_book.interest.InterestHome</home>
      <remote>com.ejb_book.interest.Interest</remote>
      <local-home>com. ejb_book.interest.InterestLocalHome
          </local-home>
```


Esempio: Descrittore di Deployment (2)

```
      <local>com.ejb_book.interest.InterestLocal</local>
      <ejb-class>com.ejb_book.interest.InterestBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
      <security-identity>
         <description></description>
         <use-caller-identity></use-caller-identity>
       </security-identity>
    </session>
  </enterprise-beans>
```

File Contenuti all'Interno di un Package EJB

- Applicazione J2EE (chiamata anche applicazione EJB)
  - file *.EAR *(Enterprise ARchive)*
  - Può contenere moduli Web tier (file *.WAR - *Web ARchive*) e file EJB-JAR
- EJB-JAR (modulo EJB)
  - file *.jar
  - Alcune implementazioni di container permettono il deployment diretto di file EJB-JAR
- Jar per il cliente EJB

File *.EAR

- Contiene sia i moduli Web-tier che quelli EJB (file EJB-JAR)
  - Può contenere file EJB-JAR multipli
- Ha un suo proprio descrittore di deployment
  - application.xml
- Per effettuare il deployment di una applicazione EJB, è sempre necessario creare un file *.EAR anche se l'applicazione prevede un solo file EJB-JAR e nessun modulo Web
  - Alcuni container permettono il deployment diretto del file EJB-JAR

File EJB-JAR

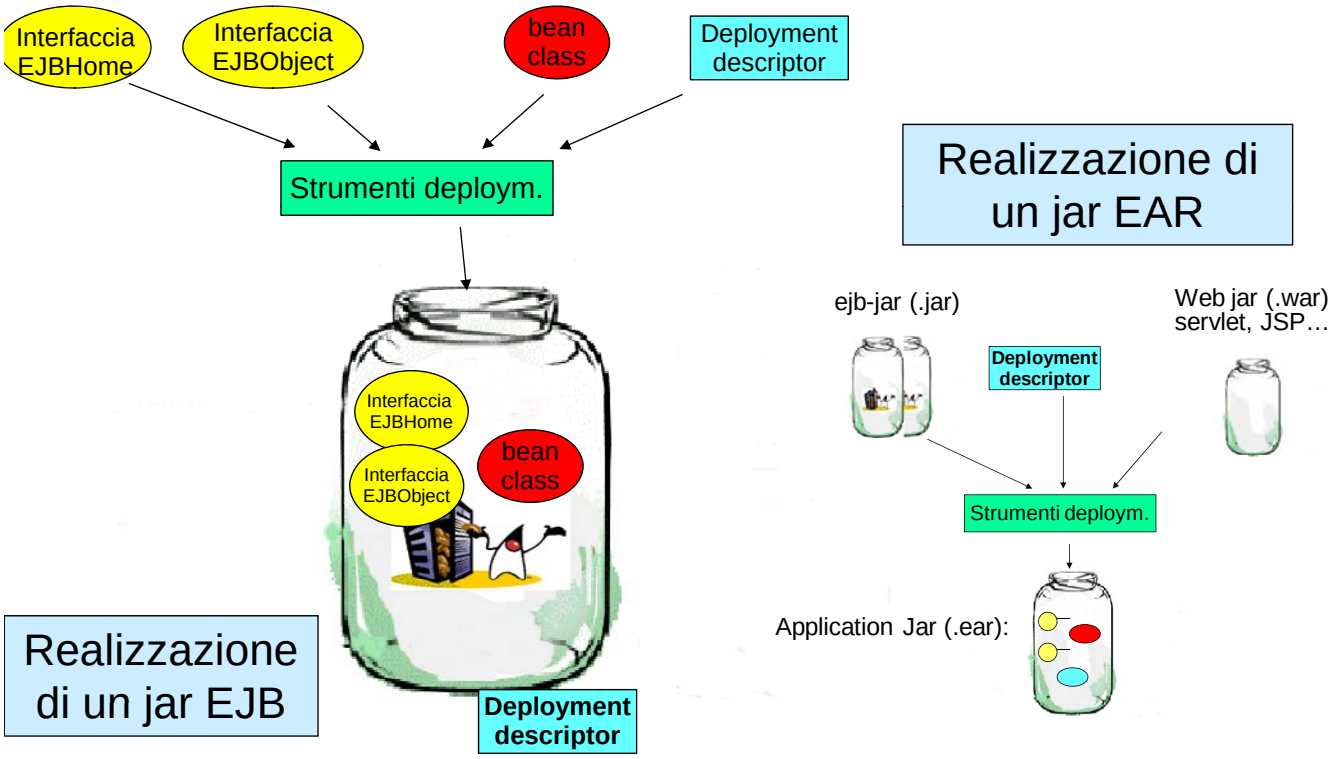
- ***Formato standard per il packaging di componenti EJB***
- Utilizzato per raggruppare in un package componenti EJB, sia assemblati che separati
- ***Deve necessariamente contenere un descrittore di deployment***
- Per ogni componente EJB, il file ejb-jar deve contenere i seguenti file di classe:
  - Classe dell'enterprise bean
  - Interfacce EJBHome e EJBObject
  - Classe che funge da chiave primaria nel caso di entity bean

File jar per il Cliente EJB

- Il produttore del file ejb-jar può creare anche il file jar per il cliente
- ***Questo file consiste di tutte le classi necessarie per il programma cliente per utilizzare la vista cliente*** dei componenti EJB contenuti nel file ejb-jar
- Può essere specificato nel descrittore di deployment del file ejb-jar
- Chi si occupa del deployment dell'applicazione deve ***assicurare*** che questo file jar cliente ***sia accessibile al class loader dell'applicazione cliente***

Eventualmente anche da remoto?

Ruolo dello Sviluppatore di Bean



Ruolo di Chi si Occupa di Deployment