



Applicazioni Enterprise e Panoramica su J2EE

Alma Mater Studiorum - Università di Bologna
CdS Laurea Magistrale in Ingegneria Informatica
I Ciclo - A.A. 2017/2018
Sistemi Distribuiti M (8 cfu)

01 – Applicazioni Enterprise e Panoramica su J2EE

Docente: Paolo Bellavista

paolo.bellavista@unibo.it

<http://lia.disi.unibo.it/Courses/sd1718-info/>

<http://lia.disi.unibo.it/Staff/PaoloBellavista/>



Cominciamo con delle domande...

- (provocatoria) Che cosa fa un laureato magistrale in **ingegneria informatica** in azienda oggi? Ovviamente oltre a fare data entry e authoring di siti Web ☺...
- Che cos'è un **componente**, in sistemi concentrati?
- Perché vale la pena di utilizzare componenti?
- Che cos'è un **componente**, in sistemi distribuiti?
- Perché vale la pena di utilizzare componenti?



Alcuni Modelli di Base

Importanza dei modelli: per comprendere *linee guida di soluzione e invarianti*, anche oltre le specifiche tecnologie

Ma importanza di **calare i modelli concretamente** in soluzioni tecnologiche allo stato dell'arte (e ce ne sono di ampiamente diffuse in ambito industriale per i sistemi distribuiti di livello enterprise...)

modelli e paradigmi **statici/dinamici**

modelli e strategie **preventivi/reattivi**

modello per **esecuzione nel sistema**

processi/oggetti **replicazione**

modello delle **entità per la allocazione**



Approccio “Filosofico” (1)

Nei **sistemi distribuiti** siamo interessati alla **esecuzione e operatività**

Ci aspettiamo che ci sia sviluppo prima della esecuzione

Non è il nostro focus (lo lasciamo ad altri corsi)

Ci interessa tutto quello che ha impatto durante l'esecuzione e che rimane significativo e vitale durante questa, favorendo e abilitando la distribuzione (e capendo in che modo)

Ad es., ci sono classi che poi diventeranno processi e **componenti attivi e distribuiti per tutta la durata della applicazione**: in realtà sono i processi che ci interessano e che rappresentano una parte della **architettura del sistema runtime**

L'architettura dinamica ci interessa e ci interessa capire come, quanto bene funziona e che **azioni di gestione/sviluppo/deployment compiere per migliorarne l'efficienza**



Approccio “Filosofico” (2)

Nei sistemi distribuiti siamo interessati a **performance e qualità**

Ci aspettiamo che ci siano risorse implicate e casi particolarmente significativi per un'architettura considerata

Ad es., specifici deployment e adozione di specifiche tecnologie per la replicazione hanno un impatto molto forte sul costo e sulla scalabilità del sistema complessivo

Mentre l'uso di strumenti di più basso livello può garantire minore overhead

Durante l'esecuzione ci interessano i colli di bottiglia, ossia i punti critici e le parti che possono determinare un comportamento del sistema poco adatto o carente

Ad es., usare alcune proprietà delle architetture di supporto (supporto alla transazionalità e al deployment dinamico) può introdurre un potenziale bottleneck da considerare e da controllare a posteriori in un progetto

L'architettura va verificata a posteriori

Panoramica su J2EE – Sistemi Distribuiti M

5



Modelli di Esecuzione

Nei sistemi distribuiti siamo interessati a **operatività, performance, reale esecuzione distribuita**

Uso di modelli preventivi/reattivi

Comportamenti preventivi prevengono eventi o situazioni con un **costo fisso** sul sistema (spesso calcolabile)

Comportamenti reattivi permettono di introdurre minore logica (e **limitare il costo**) in caso gli eventi non si verificano

Uso di modelli statici/dinamici

Comportamenti **statici NON** permettono di adeguare il sistema a fronte di variazioni (limitate)

Comportamenti **dinamici** permettono di **fare evolvere il sistema** a fronte di variazioni (limitate) con costi più elevati

Panoramica su J2EE – Sistemi Distribuiti M

6



Supporto al Deployment

Il corso si occuperà attentamente anche delle **problematiche centrali di deployment**

□ **MANUALE**

→ l'utente determina ogni singolo oggetto/componente e lo trasferisce sui nodi appropriati con sequenza appropriata di comandi

□ **APPROCCIO con FILE SCRIPT**

→ si devono eseguire alcuni file di script (qualche linguaggio shell, bash, perl, ecc.) racchiudono la sequenza dei comandi per arrivare alla configurazione *che presenta dipendenze tra oggetti*

□ **APPROCCIO basato su MODELLI o LINGUAGGI DICHIARATIVI**

→ supporto automatico alla configurazione attraverso linguaggi dichiarativi o modelli di funzionamento della configurazione da ottenere (ad es. file di deployment XML e annotazioni Java5)



Modello di Allocazione (1)

Allocazione o Deployment

le entità di una applicazione possono essere

o **statiche** o **dinamiche** (o **miste**)

Allocazione statica: data una specifica configurazione (o deployment), le risorse sono decise **prima dell'esecuzione**

Allocazione dinamica: l'allocazione delle risorse è decisa **durante l'esecuzione** ⇒ **sistemi dinamici**

Risorse statiche (decise in modo statico)

Risorse dinamiche (anche decise in modo statico e...)

In sistemi dinamici, si creano risorse dinamiche non previste e si può anche riallocare risorse esistenti (migrazione fisica/logica di task):

le risorse possono muoversi sulla configurazione durante l'esecuzione

Risorse dinamiche anche decise in modo dinamico



Modello di Allocazione (2)

- ❑ **APPROCCIO ESPLICITO**
 - l'utente prevede il **mappaggio per ogni risorsa** potenzialmente da creare **prima dell'esecuzione**
 - **Costo elevato**: l'utente prima dell'esecuzione deve prevedere un mappaggio per ogni risorsa, anche se non sarà utilizzato e necessario
- ❑ **APPROCCIO IMPLICITO** (automatico)
 - il **sistema** si occupa del mappaggio delle risorse dell'applicazione (anche al deployment)
 - **Costo limitato**: il sistema si occupa del mappaggio solo delle parti necessarie → **le risorse statiche e quelle dinamiche su bisogno** (*by need*)
- ❑ **APPROCCIO MISTO**
 - **il sistema adotta una politica di default** applicata sia inizialmente per le risorse statiche sia dinamicamente per l'allocazione delle nuove risorse e la migrazione di quelle già esistenti
 - **eventuali indicazioni dell'utente** sono tenute in conto **per migliorare le prestazioni**
 - **Costo variabile, bilanciabile e adattabile**:
il sistema adotta una politica per ogni risorsa, o statica o dinamica
decisioni statiche possono essere ottimizzate prima del runtime; decisioni dinamiche possono essere a costo diverso, a seconda del carico del sistema



Modello di Allocazione (3)

Scelta di un deployment o di un altro

→ **Può avere un grande impatto durante la specifica esecuzione e deve essere tenuto in conto**

- Pensate a dei componenti che debbano semplicemente comunicare,
- dobbiamo considerare **strumenti di comunicazione internodo** se i componenti potranno essere allocati su nodi diversi
 - dobbiamo scegliere gli **strumenti di interazione fra componenti più adatti per l'allocazione che stiamo determinando** (pensate ad architetture diverse ed eterogenee di supporto), anche in dipendenza dagli specifici requisiti applicativi
 - dobbiamo verificare che il **deployment sia adatto agli strumenti di comunicazione/interazione/integrazione scelti** e non produca problemi (cercando colli di bottiglia e casi critici)



Enterprise Computing: quali sfide?

- ❑ **Portabilità**
- ❑ **Interoperabilità** fra diversi ambienti, anche legacy
- ❑ **Supporto e gestione runtime** (scalabilità, efficienza, tolleranza ai guasti, resilienza, affidabilità, QUALITÀ, ...)
- ❑ **Time-to-market**
- ❑ **Integrazione**
- ❑ **Scalabilità**, ad esempio per Big Data, ma non solo...

Prodotti

- Application Server
- Web Server
- Componenti/Container
- DB, persistenza e oggetti

Sistemi Legacy

- Database (CRM, PLM, ...)
- Trans. Processing (TP) Monitor
- Sistemi Enterprise Info. (EIS)



Evoluzione delle Architetture per Enterprise Application

- ❑ Single tier
- ❑ Two tier
- ❑ Three tier
 - basate su RPC
 - basate su oggetti remoti
- ❑ Three tier (browser HTML e Web server)
- ❑ Application server proprietario (Bea WebLogic, Oracle AS, ...)
- ❑ Application server standard, ad esempio J2EE-compliant (IBM WebSphere, Pramati, Sybase EAServer, Zope, RedHat Jboss/WildFly, Oracle GlassFish, ...)
- ❑ Application server J2EE standard e open source (Jboss/WildFly, GlassFish, Apache Geronimo, JOnAS, ...)

Vi ricordate i pattern corrispondenti e il supporto necessario, vero?!?



Applicazioni Enterprise

Elementi costitutivi di ogni applicazione enterprise:

- **Presentation logic**
- **Business logic**
- **Data access logic** (e modello dei dati)
- **Servizi di "sistema"**

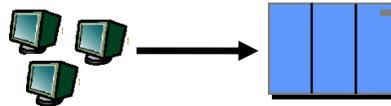
L'evoluzione delle architetture per applicazioni enterprise è sintomo di:

- **flessibilità necessaria** con cui poter approntare modifiche
- **quanto sono rilevanti** e **chi** deve fornire i servizi di sistema?



Single Tier (mainframe-based)

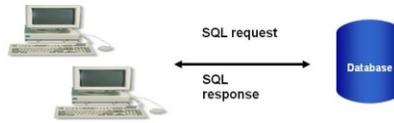
- ❑ **Terminali dumb** connessi direttamente al mainframe
- ❑ **Modello centralizzato**
- ❑ Livelli di presentazione, business logic e accesso ai dati sono **inter-dipendenti** e fusi insieme in un'applicazione monolitica
- ❑ **Pro:**
 - **Nessuna necessità di gestione client-side**
 - **Facilità di ottenere consistenza dei dati**
- ❑ **Contro:**
 - **Funzionalità inter-dipendenti** (livelli di presentazione, modello dei dati e business logic), **difficoltà di aggiornamento, maintenance e riutilizzo di codice**





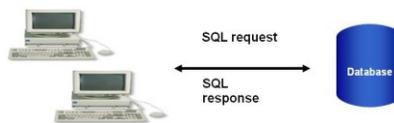
Two Tier (1)

- ❑ **Clienti fat interagiscono con backend DB**
 - Invio query SQL e ricezione dati raw
- ❑ Logica di presentazione, di business e di processamento del modello dei dati nell'applicazione cliente
- ❑ **Pro:**
 - indipendenza dallo specifico prodotto DB (rispetto a single-tier)
- ❑ **Contro:**
 - **funzionalità inter-dipendenti** (livelli di presentazione, modello dei dati e business logic), **difficoltà di aggiornamento, maintenance e riutilizzo di codice**
 - **modello dei dati tightly-coupled per ogni cliente: se cambia DB Schema?**
 - ...



Two Tier (2)

- ❑ **Clienti fat interagiscono con backend DB**
 - Invio query SQL e ricezione dati raw
- ❑ Logica di presentazione, di business e di processamento del modello dei dati nell'applicazione cliente
- ❑ **Contro (continua):**
 - **gli aggiornamenti devono essere distribuiti** (quale deployment?) **verso tutti i clienti**. Come gestire la maintenance del sistema?
 - **connessione al DB per ogni cliente**. Con quale impatto sulla scalabilità?
 - **raw data trasferiti** verso il cliente (responsabile del loro processamento) **produce overhead di rete**





Three Tier (basato su RPC)

- ❑ **Thinner client:** logica di business & modello dati separati dalla logica di presentazione
- ❑ **Middle tier server si occupa dei servizi di sistema**
 - Controllo concorrenza, threading, transazioni, sicurezza, persistenza, multiplexing, ottimizzazione performance, ...
- ❑ **Pro:**
 - Logica di business modificabile in modo più flessibile (maggior parte in middle-tier server)
- ❑ **Contro:**
 - **Complessità middle-tier server**
 - **Accoppiamento stretto fra clienti e middle-tier server** (più stretto che in 3 tier basato su Web)
 - **Scarsa riusabilità del codice** (rispetto a object model based)



Three Tier (basato su Remote Object)

- ❑ Logica di business e modello dati dentro agli oggetti
- **“astrazione” (a livello di linguaggio di interfaccia)**
- ❑ Modelli a oggetti più utilizzati: CORBA, RMI, DCOM. Linguaggi di interfaccia possibili
 - IDL per CORBA; Java interface per RMI
- ❑ **Pro:**
 - Meno strettamente accoppiato del modello di RPC
 - Codice maggiormente riutilizzabile
- ❑ **Contro:**
 - **Ancora troppa complessità nel middle-tier**





Three Tier (Web Server)

- ❑ Browser per livello presentaz
- ❑ Browser interroga Web server via HTTP
- ❑ Logica di business e modello dei dati gestiti tramite tecnologie per "dynamic content generation" (CGI, Servlet/JSP, ASP, ...)
- ❑ Pro:
 - Tipologia di cliente disponibile ovunque
 - Zero client management
 - Supporto a differenti tipi di dispositivi cliente (ad es. telefoni J2ME-enabled)
- ❑ Contro:
 - **Ancora troppa complessità nel middle-tier**



Trend Attuali

- ❑ Transizione da single-tier o two-tier **verso architetture multi-tier**
- ❑ Transizione da modello monolitico **verso modello delle applicazioni object-based**
- ❑ Transizione **verso clienti HTML-based**





Quali problemi ancora aperti?

- ❑ **Complessità del middle tier server**
- ❑ **Duplicazione dei servizi di sistema** per la maggior parte delle applicazioni enterprise
 - **Controllo concorrenza, transazioni**
 - **Load-balancing, sicurezza**
 - **Gestione risorse, connection pooling**
- ❑ Come risolvere il problema?
 - **Container condiviso che gestisce i servizi di sistema**
 - **Proprietario vs. basato su standard aperti?**



Soluzioni a Container: Proprietarie vs. Standard-based

Soluzioni proprietarie:

- ❑ Usano il modello componente-container
 - Componenti per la business logic
 - Container per fornire servizi di sistema
- ❑ Il contratto componenti-container è ben definito, ma in **modo proprietario** (problema di *vendor lock-in*)

Esempi: Tuxedo, .NET

Soluzioni basate su standard aperti

- ❑ Usano il modello componente-container e il container fornisce i servizi di sistema **in modo ben definito in accordo a standard industriali**
- ❑ Ad es. JEE e Java Specification Request (JSR)
(tra l'altro, anche supporto a portabilità di codice perché basato su bytecode Java e API di programmazione basate su standard aperti)



Che cos'è JEE?

Java Enterprise Edition (JEE), ampiamente indicata e nota come J2EE prima di Java 7

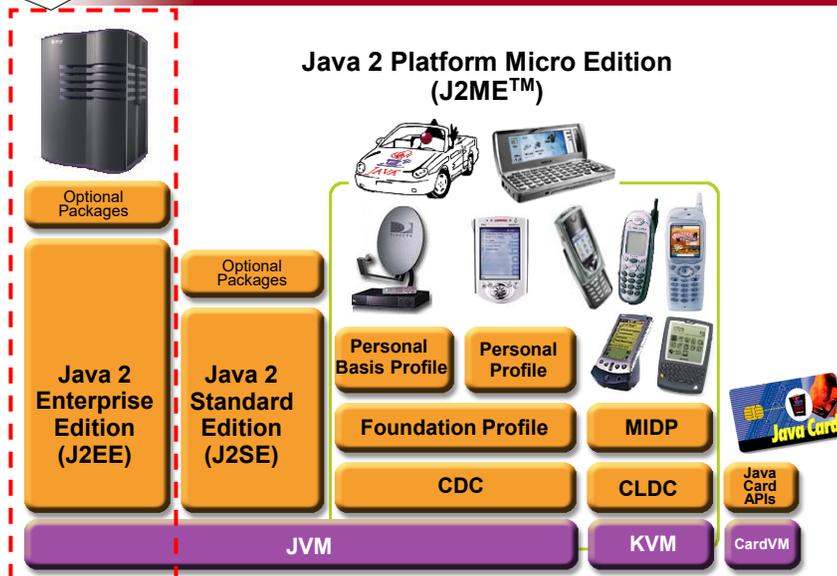
Piattaforma **open e standard** per lo sviluppo, il deployment e la **gestione di applicazioni enterprise n-tier, Web-enabled, server-centric** e basate su **componenti**



23

In questo corso ci occuperemo prevalentemente di questa parte...

Varie "Edizioni" della Piattaforma Java

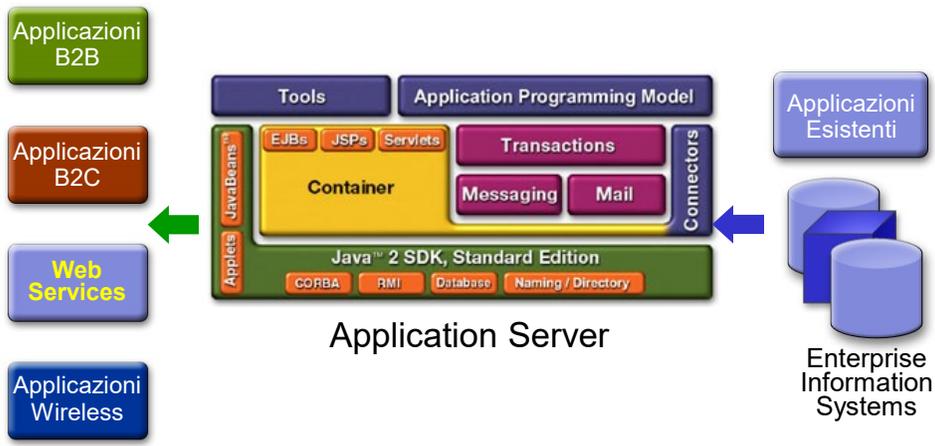


Panoramica su J2EE - Sistemi Distribuiti M

24

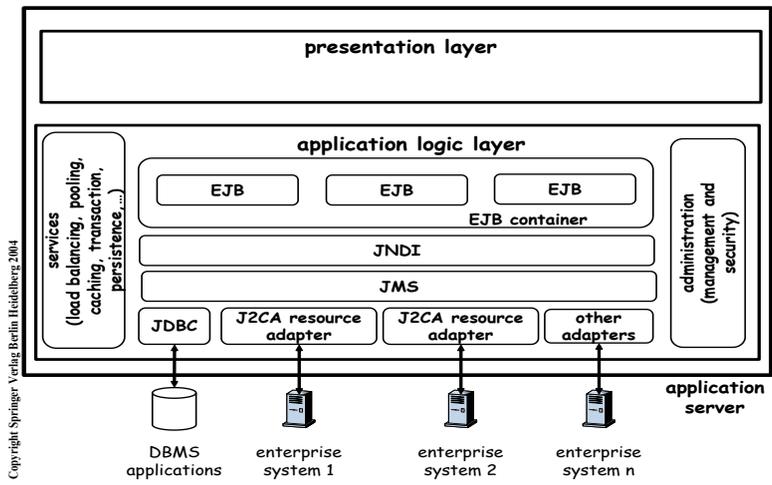


Architettura J2EE per Application Server



Architettura J2EE (chi ha già visto qualcosa in Reti M?)

Visione Java-oriented **J2EE**: Java Naming & Directory Interface, Java Message Service, J2EE Connector Architecture

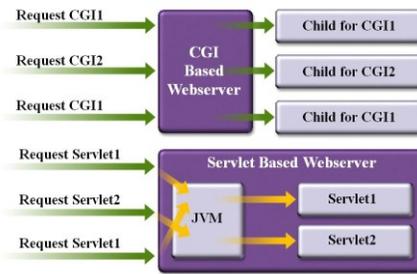


Copyright Springer Verlag Berlin Heidelberg 2004



Siete già massimi esperti mondiali di Java servlet...

- ❑ Oggetti Java che estendono le funzionalità di un server HTTP
- ❑ **Generazione dinamica di contenuti**
- ❑ Alternativa più efficace a CGI, NSAPI, ISAPI, ...
 - Efficienza
 - Indipendenza da piattaforma e server HTTP
 - Gestione della sessione
 - Java-based



Panoramica su J2EE – Sistemi Distribuiti M

27

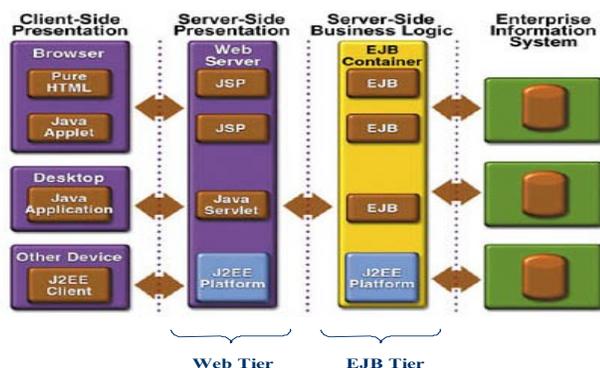


Architettura J2EE per Applicazioni N-tier

Vi siete già ampiamente occupati di **Web tier** in **Tecnologie Web T**

Questo corso si focalizzerà principalmente sul tier intermedio non-Web (EJB)

- **Componenti EJB**
- **Container**
- **Servizi di Sistema**



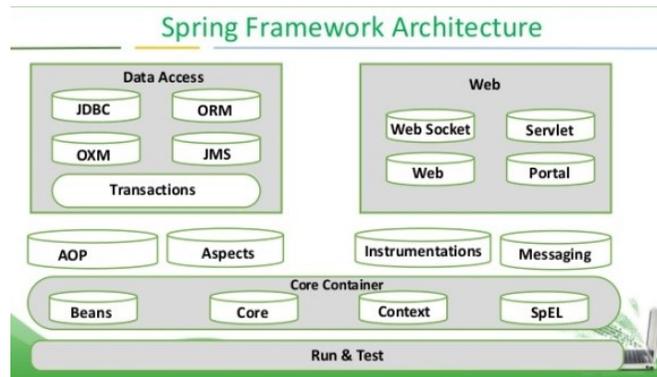
Panoramica su J2EE – Sistemi Distribuiti M

28

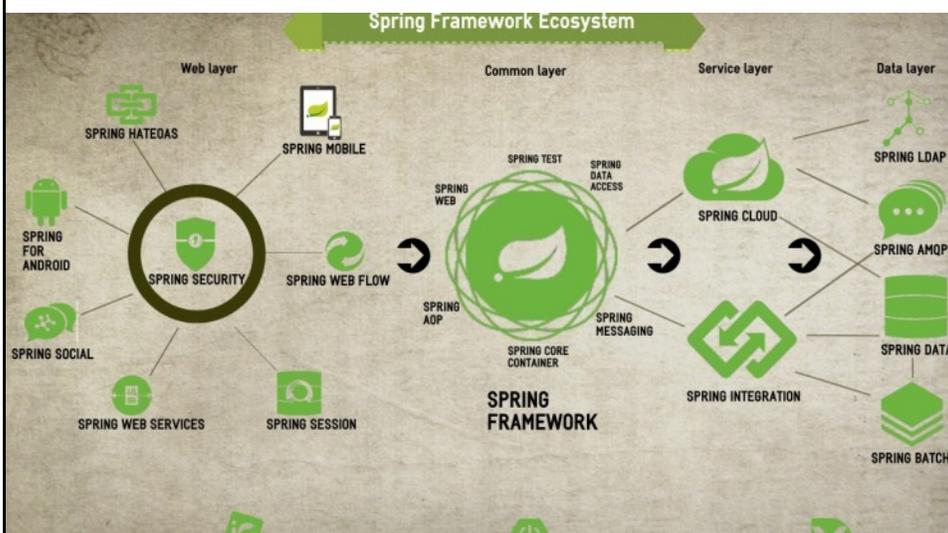


Solo J2EE?

- **Spring come framework** di grande interesse industriale, più adottato correntemente per enterprise Java
- **Modello container leggero**
- Progetto open-source partito nel 2003



Solo J2EE? tante altre tecnologie a componenti...





Modelli a Contenimento

CONTENIMENTO

Spesso molte funzionalità possono essere non controllate direttamente ma lasciate come responsabilità ad una **entità delegata supervisore (contenitore)** che se ne occupa

- spesso introducendo politiche di default
- evitando che si verifichino errori
- controllando eventuali eventi

I **contenitori** (entità dette anche **CONTAINER, ENGINE, MIDDLEWARE, ...**) possono occuparsi di azioni automatiche da cui viene sgravato l'utilizzatore che deve specificare solo la parte contenuta tipicamente

*di alto livello,
non ripetitiva,
fortemente dipendente* dalla logica applicativa



Modelli a Contenimento

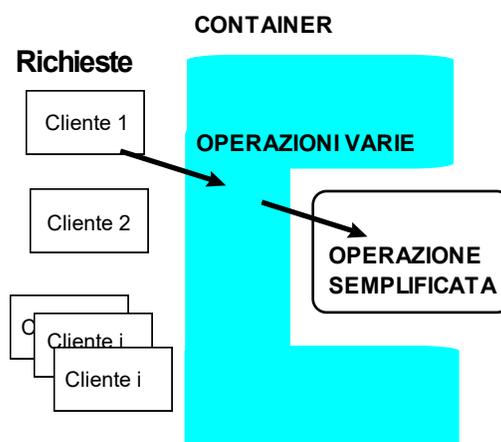
CONTAINER

Un **servizio utente** potrebbe essere integrato in un ambiente (**middleware**) che si occupa in modo autonomo di molti aspetti diversi

Vedi

CORBA tutti aspetti C/S
Engine per framework a GUI
Container per servlet
Supporto per componenti

Container possono ospitare **componenti più trasportabili e mobili**





Delega al Container

Il **container** può fornire “*automaticamente*” molte delle funzioni per supportare il **servizio applicativo verso l’utente**

❑ **Supporto al ciclo di vita**

Attivazione/deattivazione del servitore

Mantenimento dello **stato** (durata della sessione?)

Persistenza trasparente e recupero delle informazioni (interfaccia DB)

❑ **Supporto al sistema dei nomi**

Discovery del servitore/servizio

Federazione con altri container

❑ **Supporto alla qualità del servizio**

Tolleranza ai **guasti**, selezione tra possibili **deployment**

Controllo della **QoS** richiesta e ottenuta

❑ **Sicurezza**

❑ ...



Perché J2EE (1)?

Elementi costitutivi di J2EE:

- ❑ **Specifiche aperte** di API e tecnologie
- ❑ Piattaforma per lo sviluppo e il deployment
- ❑ Implementazione “standard” di riferimento e production-quality
- ❑ Compatibility Test Suite (CTS)
- ❑ J2EE brand

Motivazioni della scelta:

- ❑ Possibilità di uso di **qualsunque implementazione J2EE**
 - Sia implementazione aperta, “standard” e production-quality a utilizzo gratuito per sviluppo/deployment
 - **Sia prodotti commerciali J2EE-compliant** per alta scalabilità e tolleranza ai guasti
- ❑ Ampia disponibilità di **risorse di community** per J2EE (libri, articoli, tutorial, esempi di codice, linee guida per best practice, ...)



Perché J2EE (2)?

- ❑ ...
- ❑ Possibilità di utilizzo di componenti di business offerti da terze parti
- ❑ Vendor che **lavorano insieme sulle specifiche** e poi **competono sulle implementazioni**
 - Nelle aree specifiche di **scalabilità, performance, affidabilità, disponibilità**, strumenti di supporto a gestione e sviluppo, ...
- ❑ **Spazio per innovare** pur mantenendo portabilità delle applicazioni
- ❑ Nessuna necessità di creare/mantenere API proprietarie
- ❑ **Portabilità delle applicazioni**
- ❑ Diverse scelte implementative sono possibili in dipendenza da svariati requisiti
 - prezzo, scalabilità (da singola CPU a cluster), reliability, performance, strumenti, ...
- ❑ Ampia comunità di sviluppatori



EJB in una slide...

- ❑ Tecnologia per **componenti server-side**
- ❑ Sviluppo e deployment semplificato di applicazioni Java:
 - Distribuite, con supporto alle transazioni, multi-tier, portabili, scalabili, sicure, ...
- ❑ Porta e amplifica i **benefici del modello a componenti** sul lato server
- ❑ **Separazione** fra logica di business e codice di sistema
 - **Container** per la fornitura dei servizi di sistema
- ❑ Fornisce un framework per **componenti portabili**
 - Su differenti server J2EE-compliant
 - Su differenti ambienti di esecuzione
- ❑ Rende possibile (e semplice) la **configurazione a deployment-time**
 - **Deployment descriptor**

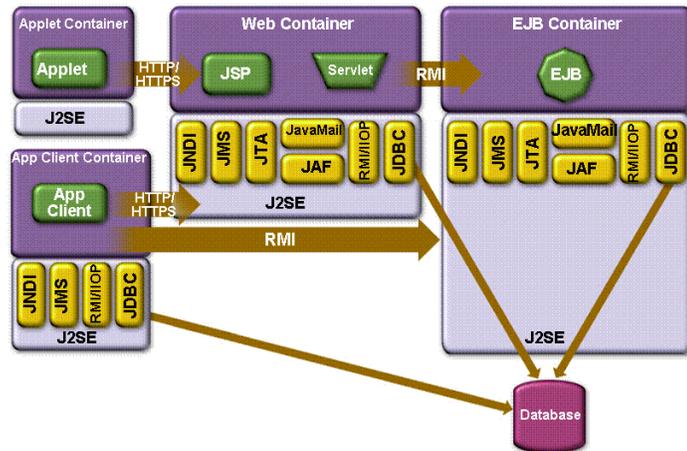


J2EE:

Componenti e Container

Diversi tipi di container (Web, EJB, Applet, ...) ma stessa metodologia, tipologia e "filosofia" di approccio

"Contratto" basato su interfacce



J2EE per Applicazioni N-tier

❑ Modello 4-tier e applicazioni J2EE

- Cliente HTML, JSP/Servlet, EJB, JDBC/Connector

❑ Modello 3-tier e applicazioni J2EE

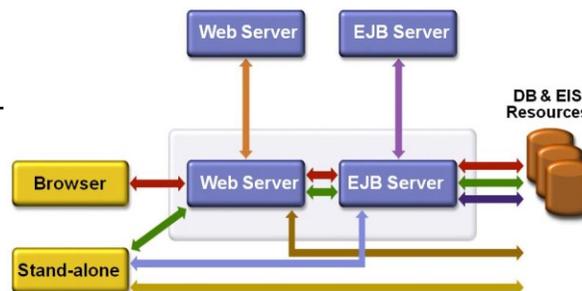
- Cliente HTML, JSP/Servlet, JDBC

❑ Modello 3-tier e applicazioni J2EE

- Applicazioni standalone EJB client-side, EJB, JDBC/Connector

❑ Applicazioni enterprise B2B

- Interazioni tra piattaforme J2EE tramite messaggi JMS o XML-based

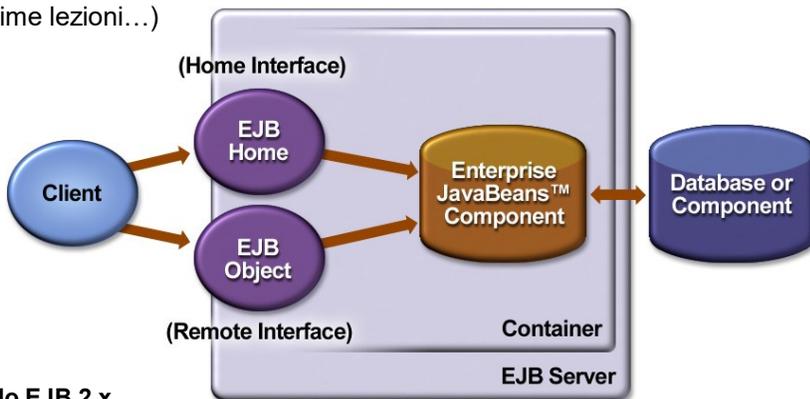




Architettura EJB

Idea di base: **container attivo all'interno di un EJB Server**
(Application Server)

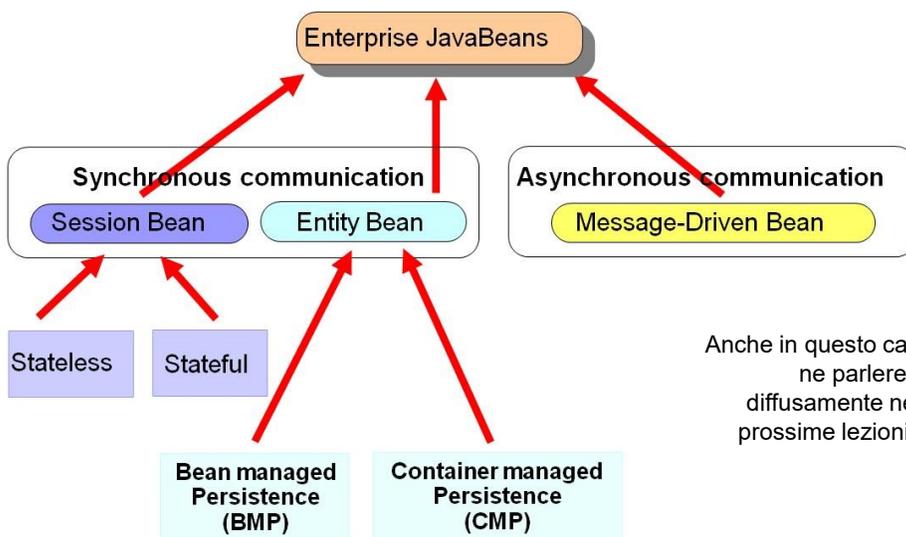
Cliente può interagire **remotamente** con componente EJB
tramite **interfacce ben definite** (ne parleremo a lungo nelle
prossime lezioni...)



Modello EJB 2.x



Principali Componenti EJB



Anche in questo caso,
ne parleremo
diffusamente nelle
prossime lezioni....



Componenti e Container

- ❑ Container svolgono il loro lavoro “dietro le quinte”
 - No API complicate
 - **Controllo tramite interposizione**
- ❑ Container implementano J2EE
 - Per certi versi il loro sviluppo non è drasticamente diverso da quello dei componenti
 - I vendor possono competere sui container con grande margine di innovazione

Container gestiscono

- Concorrenza
- Sicurezza
- Disponibilità
- Scalabilità
- Persistenza
- Transazionalità
- Life-cycle management
- Management

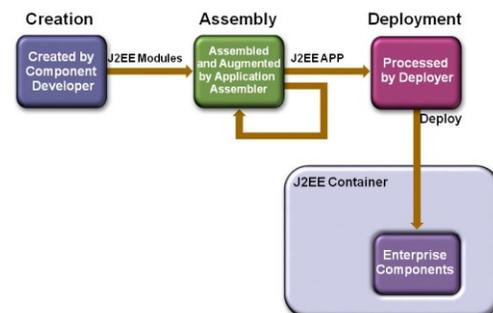
Componenti gestiscono

- Livello di presentazione
- Business Logic



Ciclo di Vita delle Applicazioni J2EE

- ❑ Sviluppo e compilazione del codice dei componenti
 - Servlet, JSP, EJB
- ❑ Scrittura di **deployment descriptor** per i componenti
 - Da JavaEE5, possibilità di utilizzo delle **annotazioni**
- ❑ Assemblaggio di componenti in package pronti per il deployment
- ❑ Deployment del package sul server





Descrittori di Deployment

- ❑ Forniscono **istruzioni al container** su come gestire e controllare il comportamento (anche runtime) di componenti J2EE
 - Transazioni
 - Sicurezza
 - Persistenza
 - ...
- ❑ Permettono la **personalizzazione tramite specifica dichiarativa** (NO personalizzazione tramite programmazione)
 - file XML oppure annotazioni (a partire da Java5)
- ❑ Semplificano **portabilità** del codice



Ruoli e Opportunità per Sviluppatori J2EE

- ❑ Realizzatori di **componenti**
 - Sviluppatori di componenti EJB
- ❑ **Assemblatori** di applicazioni a partire dai componenti
- ❑ Responsabili di **deployment**
- ❑ Realizzatori di **piattaforme di supporto**
 - **Sviluppatori di container**
- ❑ Realizzatori di **strumenti di supporto**
- ❑ **Amministratori** di sistemi e servizi



Varie Risorse Legate al Mondo J2EE (1)

- ❑ **Strumenti di sviluppo e supporto**
 - IDE: Borland JBuilder Enterprise, WebGain Visual Cafe', IBM Visual Age for Java™, Forte™ for Java™, Oracle JDeveloper, Macromedia Kawa, **Eclipse, NetBeans**, ...
 - Modeling, Performance, Testing, ...
- ❑ **Enterprise Integration**: Connector, Java Message Service (JMS) API, XML, Java Business Integration (JBI)
- ❑ **Componenti**
- ❑ **Framework J2EE** (**Apache TomEE, Apache Geronimo, RedHat JBoss/WildFly, Oracle Glassfish** (reference), ma anche Beas Systems, Borland, CA, Fujitsu, Hitachi, HP, IBM, IONA, iPlanet, Macromedia, NEC, Pramati, SilverStream, Sybase, Talarian, ...)
- ❑ **Applicazioni**



Varie Risorse Legate al Mondo J2EE (2)

- ❑ Per verificare la portabilità delle applicazioni J2EE:
 - J2EE RI 1.3.1 (e successive) e **J2EE Application Verification Kit (J2EE AVK)**
 - **Verifica statica/dinamica** della portabilità (da parte degli sviluppatori di applicazioni)
- ❑ **IDE per lo sviluppo J2EE** utilizzabili nel corso (quelli che preferite, ma provate quelli open-source)
 - **Eclipse IDE for Java EE Developers** (disponibile per Win/MacOS/Linux, sia 32 che 64 bit, <http://eclipse.org>)
 - **NetBeans IDE8.0** (e successivi, <http://netbeans.org>)



Varie Risorse Legate al Mondo J2EE (3)

Tutorial e buoni articoli su argomenti specifici di interesse per il corso

- J2EE home page <http://www.oracle.com/technetwork/java/javaee/overview/index.html>
- J2EE SDK download <http://www.oracle.com/technetwork/java/javaee/overview/index.html>
- J2EE 1.5 Tutorial <http://download.oracle.com/javaee/5/tutorial/doc/>
- J2EE 1.6 Tutorial <http://download.oracle.com/javaee/6/tutorial/doc/>
- J2EE Blueprints java.sun.com/blueprints/enterprise/index.html
- www.theserverside.com
- <http://www.javabeat.net/tutorials/>
- <http://www.ibm.com/developerworks/java/tutorials/j-gsejb/>

Quindi, non ci sono scuse accettabili ☺ relative a mancanza di materiale per uno studio molto molto approfondito, con esperienza diretta di sviluppo di codice di esempio...