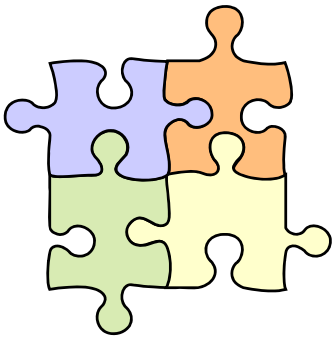
Luca Cabibbo

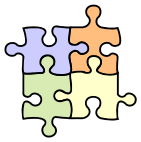


Architetture Software

Introduzione a Java EE

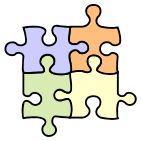
Dispensa MW 5

ottobre 2008



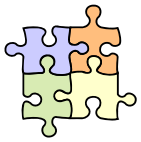
- Fonti

- Java Platform, Enterprise Edition
  http://java.sun.com/javaee/
- The Java EE 5 Tutorial
  http://java.sun.com/javaee/5/docs/tutorial/doc/
- [SAP/2e] Chapter 16, J2EE/EJB



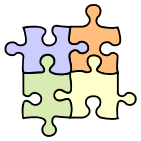
- Obiettivi e argomenti

- Obiettivi
  - descrivere le possibili architetture applicative per la piattaforma Java EE
  - descrivere la tecnologia EJB e il supporto alla loro esecuzione
  - discutere le qualità sostenute dalla piattaforma Java EE
- Argomenti
  - introduzione alla piattaforma Java EE
  - Enterprise Bean
  - Session bean
  - Message-driven bean
  - Java EE e qualità



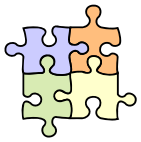
* Introduzione alla piattaforma Java EE

- La piattaforma *Java Enterprise Edition* (*Java EE*)
  - *il problema* – oggi c'è il bisogno di sviluppare applicazioni distribuite, transazionali e portabili, che sfruttino la velocità, sicurezza ed affidabilità delle tecnologie di tipo enterprise – queste applicazioni devono essere progettate, costruite e prodotte con meno soldi, tempo e risorse
  - *la soluzione Java EE* – con Java EE lo sviluppo di applicazioni enterprise con Java non è mai stato così facile o veloce – intento di Java EE 5 è fornire agli sviluppatori un insieme potente di API, per ridurre il tempo di sviluppo, ridurre la complessità delle applicazioni, e migliorare le prestazioni delle applicazioni
  - Java EE 5 introduce un modello di programmazione semplificato ...



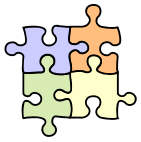
Java EE

- *Java Enterprise Edition* (*Java EE*)
  - modello/piattaforma per la progettazione, lo sviluppo, l'assemblaggio e il deployment di applicazioni distribuite/enterprise
  - supporto per
    - applicazioni distribuite multi-livello
    - componenti riusabili
    - gestione unificata della sicurezza
    - controllo flessibile delle transazioni
    - interazione sincrona/asincrona
    - web service
    - gestione di dati persistenti
    - ...

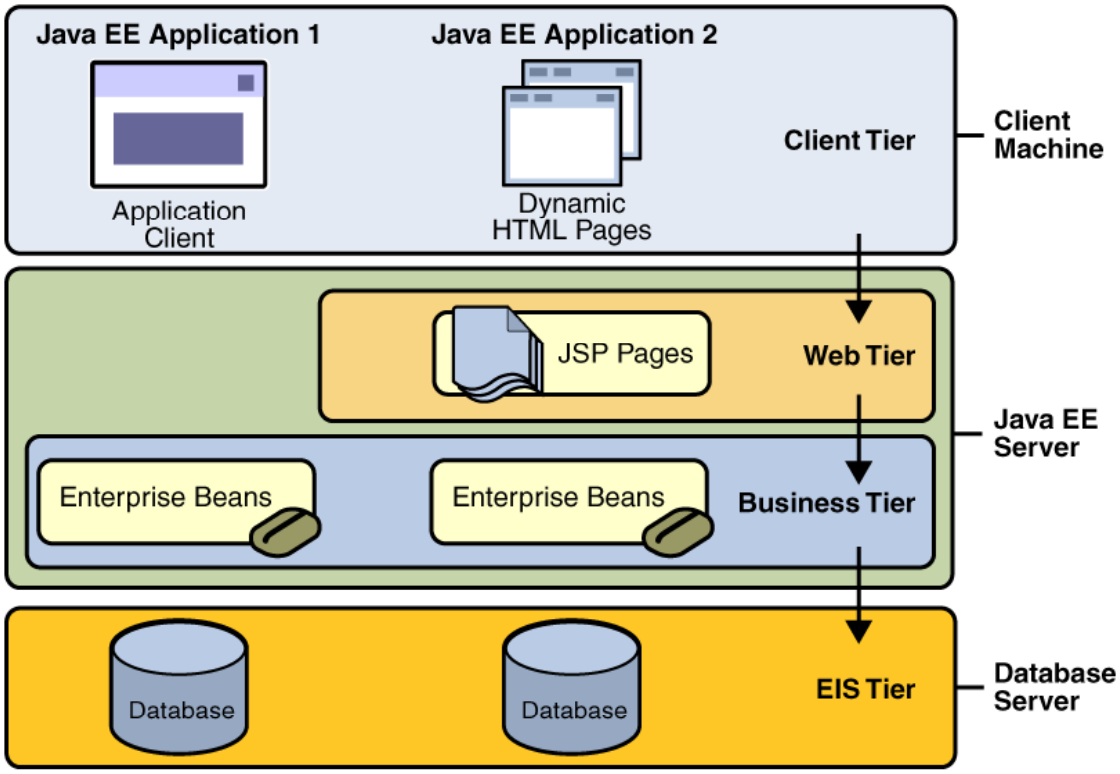


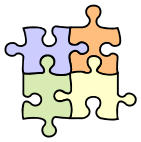
Java EE

- Alcuni obiettivi di Java EE
  - modello di programmazione semplificato
    - ma richiede una comprensione degli aspetti architetturali e di altri aspetti resi trasparenti
  - supporto per qualità architetturalmente significative
    - portabilità, disponibilità e scalabilità, sicurezza, apertura, ... – ma come sono ottenute?
  - supporto per l'integrazione di sistemi esistenti



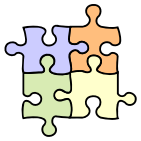
Modello applicativo di Java EE





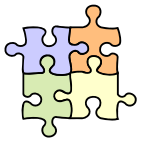
Modello applicativo di Java EE

- Modello applicativo di Java EE
  - Java EE fornisce un modello per lo sviluppo di applicazioni distribuite multi-livello
  - client tier, middle tier e back-end tier
    - client tier – supporto per diversi tipi di client
    - web tier – componenti web in esecuzione sul server Java EE
    - business tier – componenti di logica applicativa in esecuzione sul server Java EE
    - back-end tier – EIS tier – accessibile tramite API standard

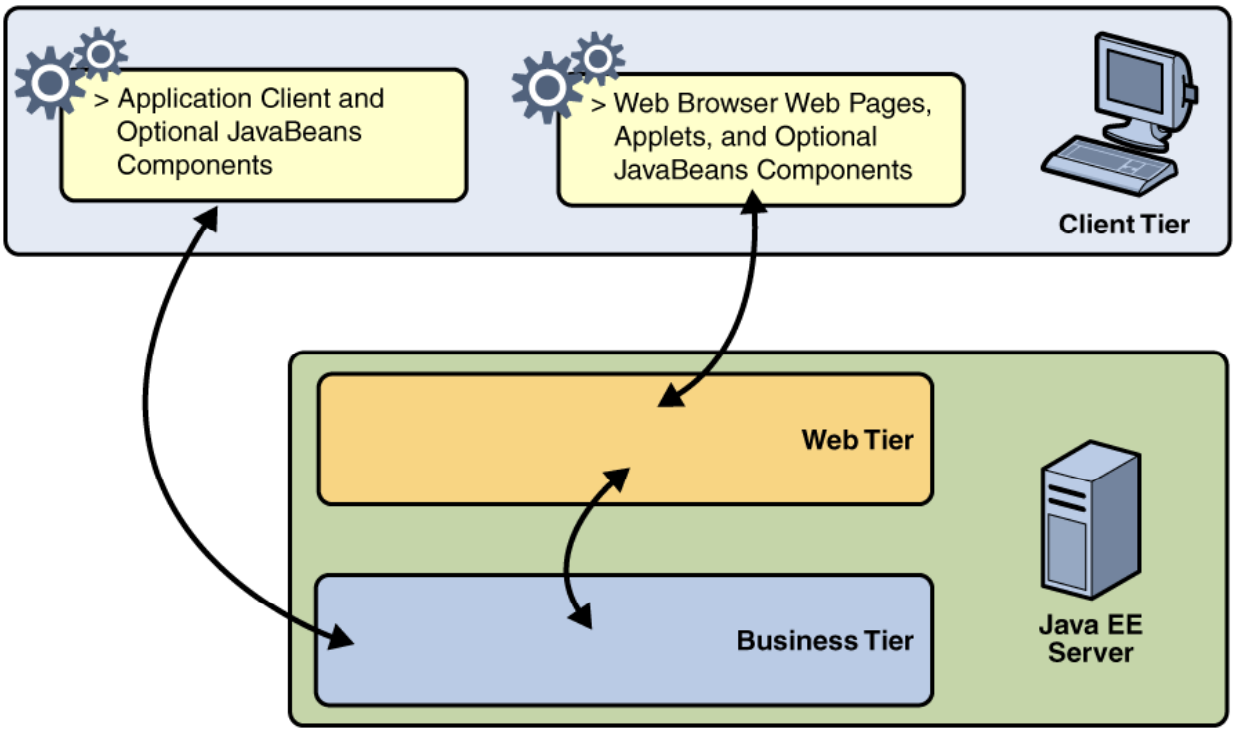


Componenti Java EE

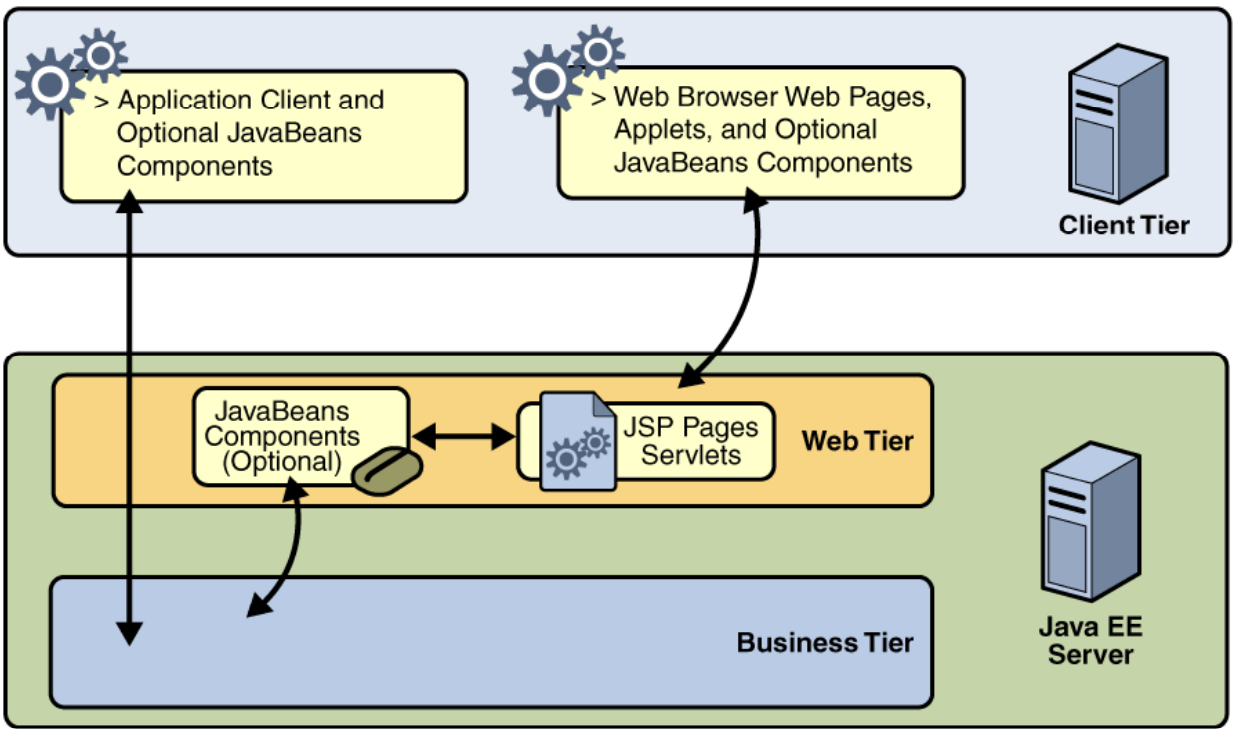
- Le applicazioni Java EE sono fatte di *componenti* – ce ne sono di diverse tipologie
  - client applicativi e applet – componenti di tipo client
  - componenti web – Java Servlet, JSP, JSF – gestiti dal server Java EE
  - Enterprise Java Beans (EJB) o semplicemente Enterprise Bean – gestiti dal server Java EE
- I componenti sono programmi Java, il cui ciclo di vita è diverso da quello dei programmi Java più semplici
  - compilazione
  - assemblamento
  - deploy
  - esecuzione e gestione a cura di un application server Java EE



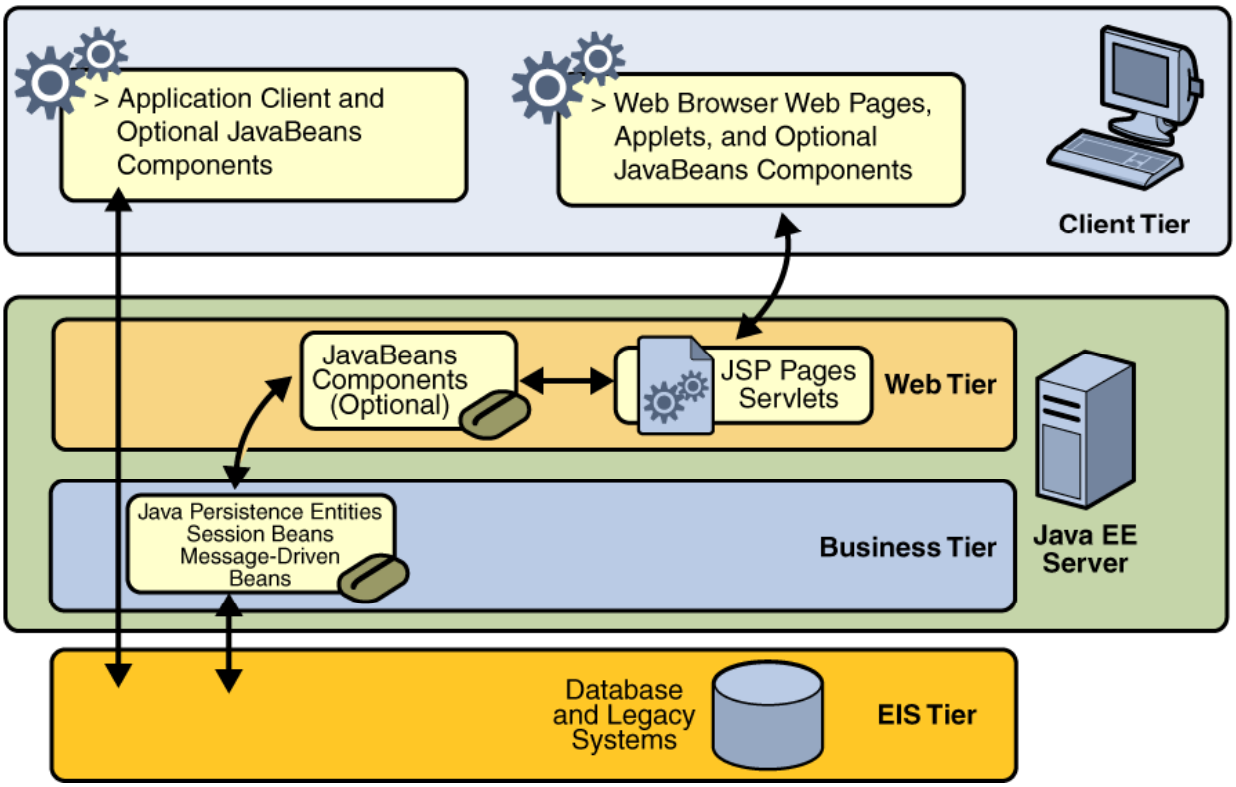
Livello client e comunicazione col server

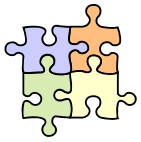


Livello web



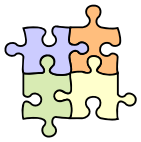
Livello business e EIS



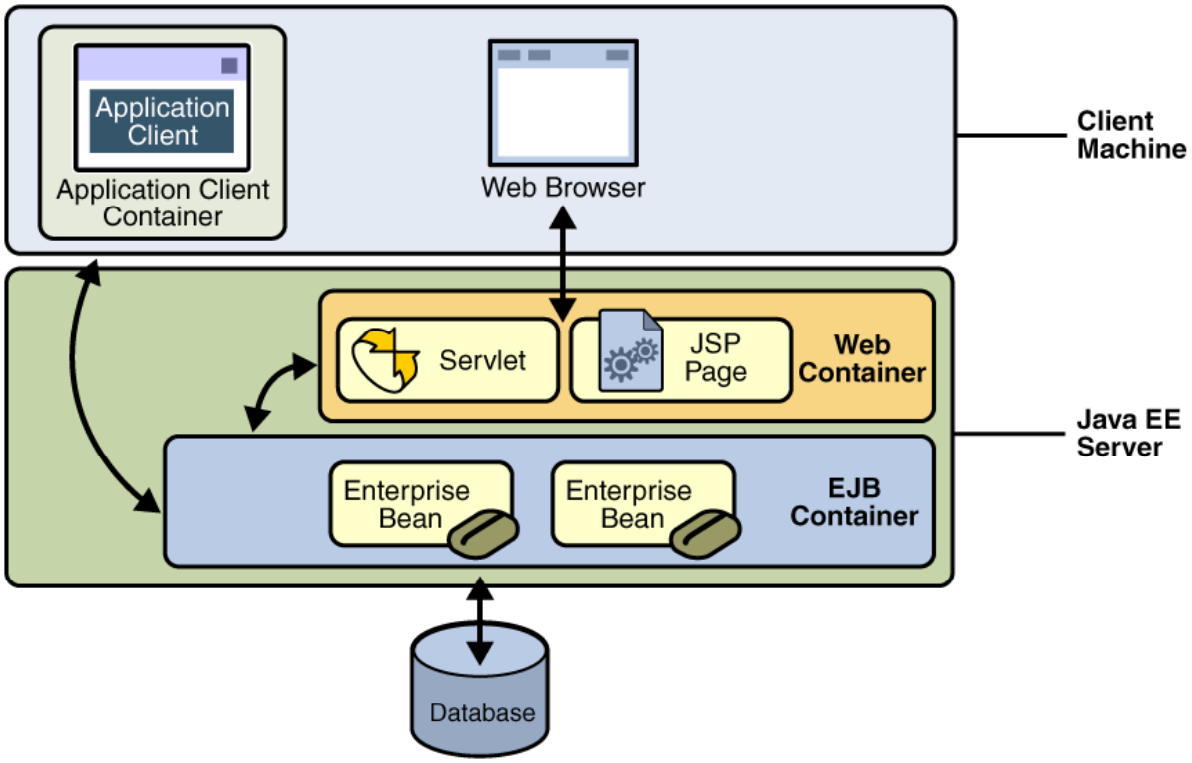


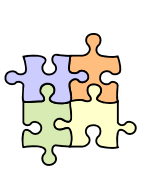
Contenitori Java EE

- La gestione dei componenti è basata su contenitori
  - i componenti devono essere assemblati e rilasciati (deployed) in un contenitore
  - i *contenitori* (*container*) sono ambienti runtime standardizzati – che forniscono servizi specifici ai componenti
    - ad es., sicurezza, gestione delle transazioni, persistenza, ...
    - i componenti si aspettano che questi servizi siano offerti da tutte le implementazioni della piattaforma Java EE
  - l'assemblaggio prevede la specifica di parametri per personalizzare il tipo di supporto che deve essere fornito dal contenitore
  - l'erogazione di questi servizi è basato su meccanismi di intercettazione delle richieste e delegazione

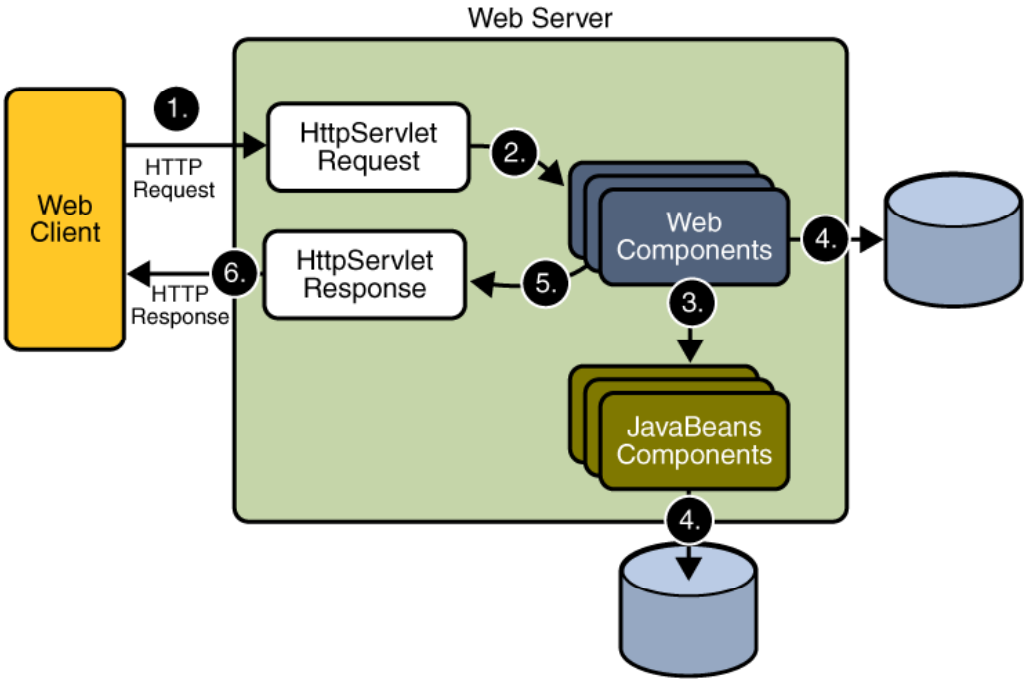


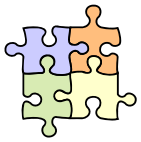
Contenitori





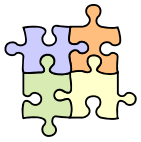
Un esempio noto: applicazioni web





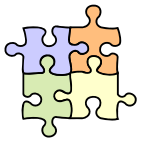
Un esempio noto: applicazioni web

- Una *servlet* (HTTP) è una classe usata per estendere le capacità del server che ospita l'applicazione sulla base di un modello di programmazione richiesta-risposta
- Ciclo di vita di una servlet – controllato dal server
  - quando il server riceve una richiesta per una servlet
    - se un'istanza non è stata creata, la crea e la inizializza
    - invoca il metodo di servizio – ad es., doPost – passandogli gli oggetti richiesta e risposta
  - se il server ha bisogno di rimuovere la servlet, la finalizza e la rimuove
- In linea di principio, un oggetto servlet può essere usato per gestire zero o più richieste – in zero o più sessioni



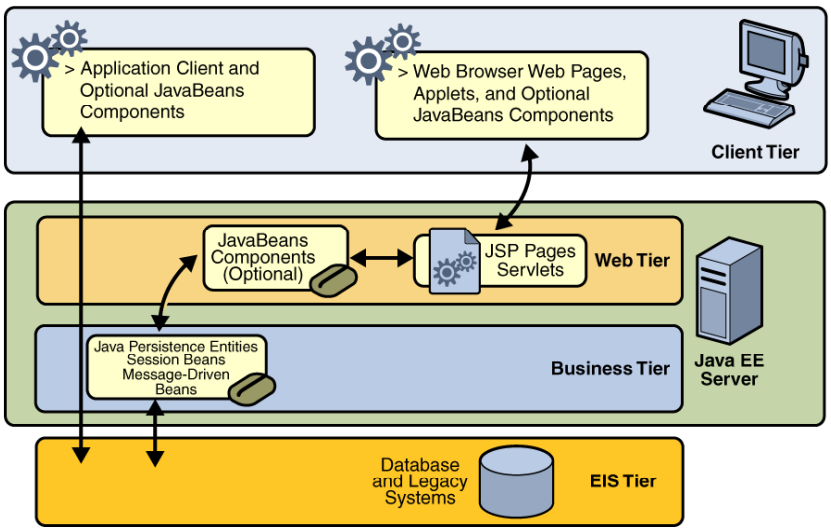
Un esempio noto: applicazioni web

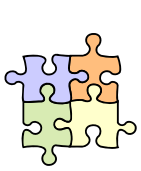
- Condivisione di informazioni tra componenti web
  - non è opportuno che informazioni da condividere siano gestite dall'oggetto servlet
  - sono usati di alcuni oggetti di supporto – gestiti dal server
    - contesto web – per gestire lo stato dell'applicazione
    - sessione – per gestire lo stato della sessione
    - richiesta
    - pagina
- Come fa il contenitore ad offrire questi servizi ai suoi componenti?
  - la servlet può chiedere il suo contesto al contenitore
  - il contenitore passa sempre la sessione alla servlet – nella richiesta – possibile perché il server intercetta la richieste
  - contestualmente, il contenitore può offrire ulteriori servizi ai suoi componenti



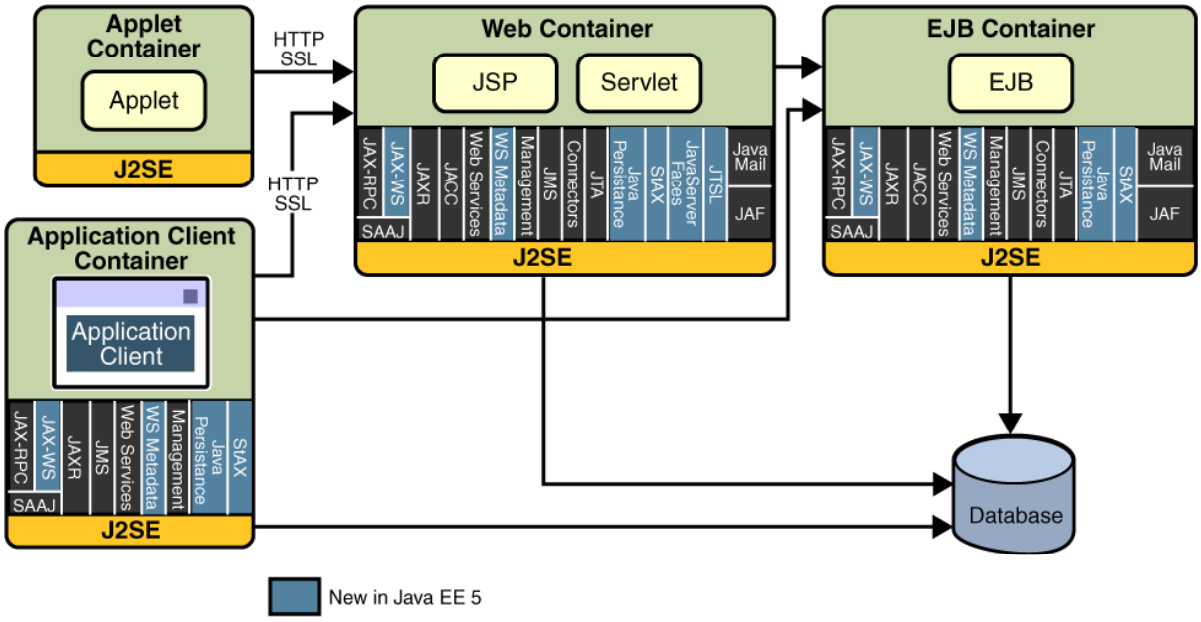
Java EE: scenari applicativi

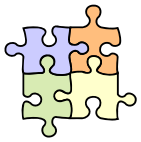
- Con Java EE sono possibili numerosi scenari applicativi, a due o più livelli
  - i vari livelli sono infatti opzionali – tranne forse il livello client





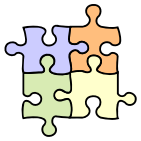
API di Java EE





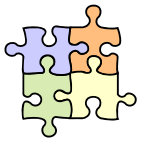
* Enterprise Bean

- Gli **Enterprise Bean** sono componenti Java EE che implementano la tecnologia **EJB**
  - componenti server-side
    - la specifica EJB definisce un “modello (standard) per componenti”
    - si tratta dunque di un’architettura basata su componenti
  - in applicazioni multi-livello
  - per implementare logica applicativa
  - per usufruire di servizi forniti dal contenitore EJB come gestione delle transazioni, controllo della concorrenza, affidabilità, sicurezza, scalabilità
    - poiché questi servizi sono disponibili, lo sviluppatore può concentrarsi sulla soluzione di problemi di business
  - componenti portabili, e che possono essere riusati in applicazioni diverse



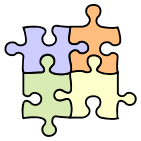
Tipi di enterprise Bean

- Due tipi di enterprise bean
  - *session bean*
    - esegue un compito per un client
  - *message-driven bean*
    - agisce come un listener per un certo tipo di messaggi



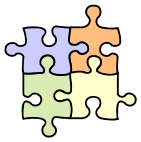
Session bean

- Session bean
  - un **session bean** esegue un compito per un cliente – rappresenta un singolo client all'interno dell'AS
    - serve ad implementare (incapsulare) logica applicativa specifica per il cliente
  - il suo stato può riflettere (o meno) la sua interazione con un client particolare
    - in linea di principio non è condiviso tra più client



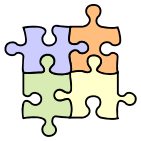
Session bean

- Due tipi di session bean
  - *stateless*
    - fornisce/rappresenta del comportamento lato server – può anche implementare un web service
  - *stateful*
    - gestisce anche lo stato della conversazione con un client – per tutta la durata di una sessione
- Si noti che la presenza o l'assenza di stato
  - non ha niente a che vedere con il fatto che il session bean abbia un proprio stato
  - riguarda il fatto che il session bean debba conservare – o meno – lo stato della sessione/conversazione con il suo client



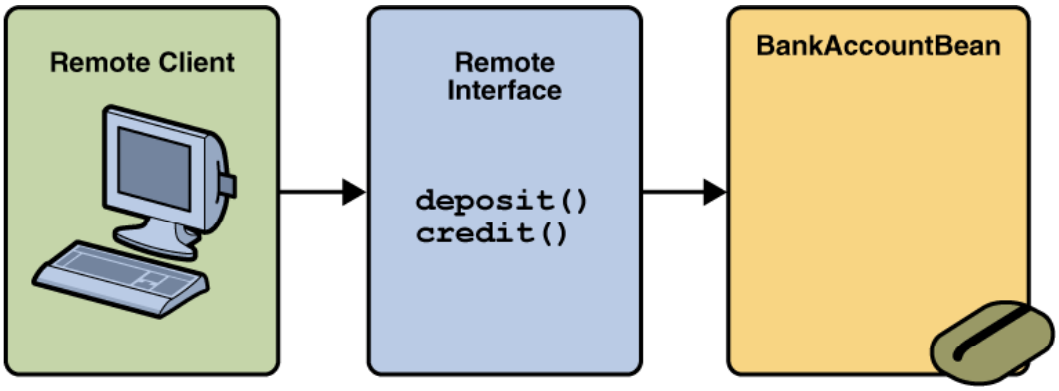
Message-driven bean

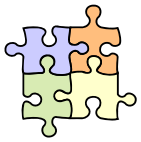
- Message-driven bean
  - un **message-driven bean** consente di elaborare messaggi in modo asincrono – agisce come un listener di messaggi JMS o altri tipi di messaggi
    - definisce un metodo onMessage – invocato dal contenitore quando è disponibile un messaggio – così come doPost di una servlet è invocato dal contenitore quando viene ricevuta una richiesta http
    - la destinazione da cui riceve messaggi è specificata dal suo descrittore di deployment
  - ad es., per la gestione di workflow
- Nota
  - i session bean possono essere usati per inviare e ricevere messaggi JMS in modo sincrono – ma non in modo asincrono



* Session bean

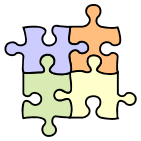
- Consideriamo il modello di programmazione degli enterprise bean di tipo session
  - un session bean implementa ed espone un'*interfaccia remota*
  - i suoi client accedono ai servizi offerti mediante l'interfaccia remota





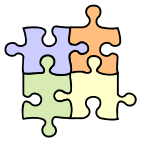
Modello di programmazione

- Accesso locale
  - oltre all'interfaccia remota, un session bean può offrire servizi localmente, mediante un'*interfaccia locale*
  - i servizi locali possono essere acceduti solo da componenti client che risiedono nello stesso contenitore
  - per brevità, ignoriamo questo aspetto



Modello di programmazione

- Implementazione di un session bean
  - un session bean è costituito da vari elementi
    - l'interfaccia remota – e l'eventuale interfaccia locale
    - la classe che implementa il session bean
    - informazioni circa la sua configurazione

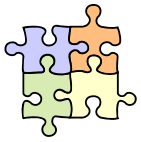


Remote interface per Hello

```
package hello;

import javax.ejb.Remote;

@Remote
public interface HelloRemote {

    /* dichiara le funzionalità offerte dall'enterprise bean */
    public String hello(String name);

}
```

session bean definito nell'ambito di un progetto EJB

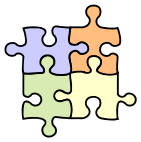
da compilare e fare il deploy su un AS



Implementazione per Hello

```
package hello;

import javax.ejb.Stateless;

@Stateless(mappedName="ejb/Hello")
public class Hello implements HelloRemote {

    /* implementa i metodi dell'interfaccia remota */
    public String hello(String name) {
        return "Hello, " + name + "!";
    }

}
```

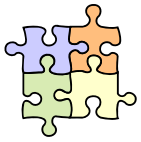
sono possibili diverse convenzioni con i nomi
- HelloRemote per l'interfaccia
- HelloBean, HelloImpl o Hello per l'implementazione



Client per Hello (1)

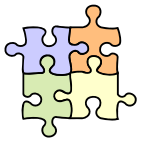
```
package hello.client;

import javax.ejb.EJB;
import hello.HelloRemote;

public class HelloClient {
   @EJB(mappedName="ejb/Hello")
   private static HelloRemote hello;

   public HelloClient() { }

   ...
}
```

da compilare ed eseguire come application client



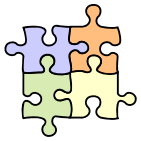
Client per Hello (2)

```
public static void main(String[] args) {
    HelloClient client = new HelloClient();
    client.doConversation();
}

public void doConversation() {
    String answer = hello.hello("Luca");
    System.out.println(answer);
}
```

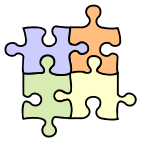
EJB ed annotazioni

- Nel codice mostrato, le annotazioni rivestono un ruolo importante
  - @Remote, @Local, @Stateless e @Stateful dichiarano il ruolo svolto dalle diverse unità di programmazione
    - nell'esempio, quando verrà fatto il deploy dell'applicazione, il session bean sarà registrato sul server JNDI
      - con un nome di default o con il nome indicato esplicitamente – ejb/Hello
    - hanno a che fare con le "interfacce fornite"
  - @EJB indica una dependency injection
    - prima di iniziare l'esecuzione dell'application client, verrà effettuata una ricerca sul server JNDI di un EJB di tipo HelloRemote (ejb/Hello) – il riferimento remoto al bean memorizzato nella variabile hello
      - ricerca con un nome indicato esplicitamente o di default
    - si tratta di un' "interfaccia richiesta"



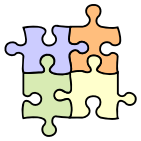
EJB e composizione

- Il binding (collegamento) tra i diversi componenti runtime di un'applicazione Java EE (in questo caso, l'EJB e l'application client) viene dunque effettuato sulla base delle informazioni di deployment
  - nell'esempio, sono immerse nel codice tramite annotazioni
    - in pratica, potrebbero essere separate dal codice in opportuni file di configurazione
  - si noti il tipo di dipendenze tra componenti
    - l'EJB non sa nulla dei suoi client
    - l'application client deve conoscere interfaccia e nome simbolico dell'EJB – ma non deve sapere nulla circa la sua particolare implementazione



Modello di programmazione

- Si noti anche che
  - l'enterprise bean non viene mai creato esplicitamente dal programmatore
    - come una servlet, viene creato dal contenitore, se e quando serve
  - le chiamate al bean sono in realtà chiamate remote
    - come in Java RMI
  - come per le servlet, ci aspettiamo che il bean possa definire dei metodi che verranno chiamati dal contenitore – e non dai suoi client remoti
    - in momenti specifici della vita del bean



Il session bean Counter

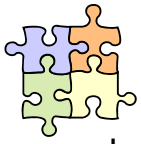
- Consideriamo un altro breve esempio
  - un enterprise bean di tipo session/stateful
- Interfaccia remota

```
package counter;

import javax.ejb.Remote;

@Remote
public interface CounterRemote {
        /* incrementa il contatore e restituisce il suo valore */
        public int count();
}
```


Implementazione del bean Counter

```
package counter;

import javax.ejb.Stateful;
import javax.annotation.*;

@Stateful(mappedName="ejb/Counter")
public class CounterBean implements CounterRemote {
        private int val;

        /* assegna un valore iniziale al contatore */
        @PostConstruct
        public void initialize() {
                this.val = 0;
        }
        /* incrementa il contatore e restituisce il suo valore */
        public int count() {
                val++; return val;
        }
}
```

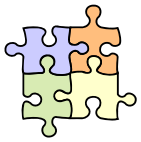
un metodo annotato @PostConstruct viene eseguito subito dopo la costruzione dell'istanza del bean



Client per Counter (1)

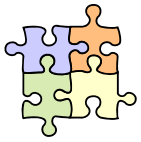
```
package counter.client;

import javax.ejb.EJB;
import counter.CounterRemote;

public class CounterClient {
   @EJB(mappedName="ejb/Counter")
   private static CounterRemote counter;

   public CounterClient() { }

   public static void main(String[] args) {
      CounterClient client = new CounterClient();
      client.doConversation();
   }

   ... segue ...
}
```

da compilare ed eseguire come application client



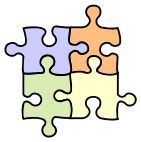
Client per Counter (2)

```
public void doConversation() {
   for (int i=1; i<=50; i++) {
      System.out.println( counter.count() );
      /* stamperà 1, 2, ..., 50
       * anche se vengono mandati in esecuzione
       * più client concorrenti */
   }
}
```

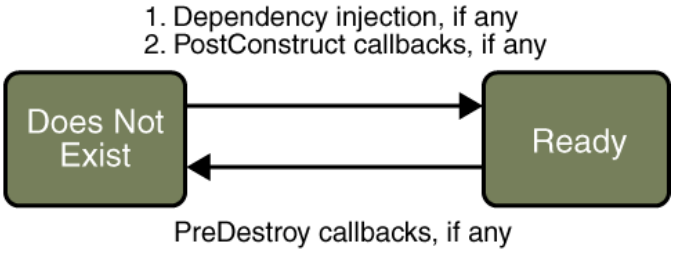


Ciclo di vita dei session bean

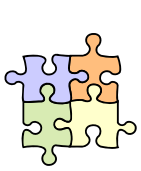
- La programmazione degli enterprise bean richiede la conoscenza del loro ciclo di vita
  - durante alcuni eventi significativi della vita di un enterprise bean, vengono infatti invocati eventuali metodi etichettati con delle opportune annotazioni



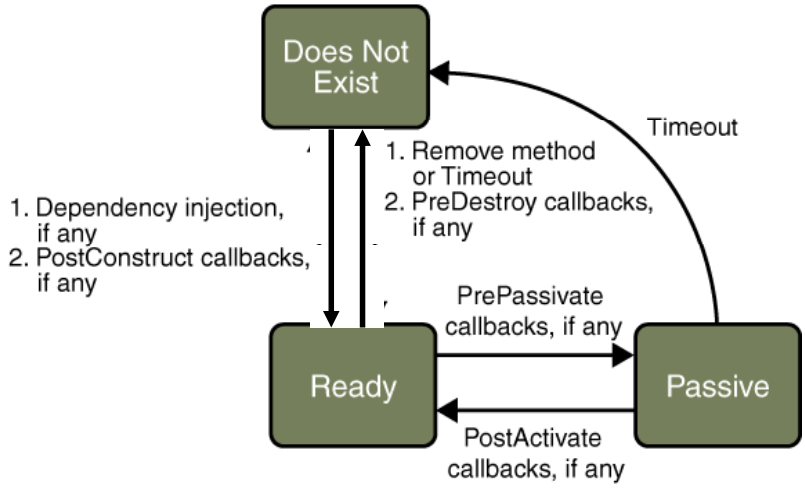
Ciclo di vita dei session bean stateless



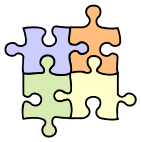
- Ad esempio, si consideri un session bean (stateless) che fornisce un metodo findProduct(int id) per effettuare la ricerca di un Product in una base di dati
  - findProduct potrebbe ottenere una connessione alla base di dati, fare l'interrogazione, e chiudere la connessione
  - la connessione jdbc potrebbe essere riutilizzata in più esecuzioni di findProduct – aprendola in un metodo annotato @PostConstruct, e chiudendola in un metodo annotato @PreDestroy



Ciclo di vita dei session bean stateful

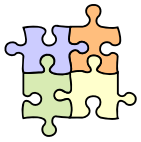


- Il ciclo di vita dei session bean stateful è più complesso
  - un session bean stateful può essere "passivato" – lo stato della sessione può essere serializzato e salvato automaticamente in memoria secondaria – per essere poi "attivato" (ricaricato) in memoria in un secondo tempo



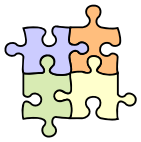
Ciclo di vita dei session bean stateful (1/2)

- I metodi di callback del ciclo di vita devono essere pubblici, senza parametri, void – ed annotati come segue
  - @PostConstruct
    - invocato dal contenitore su istanze del bean appena costruite – dopo che tutte le iniezioni di dipendenze sono state completate – prima della prima invocazione dei metodi di business
  - @PreDestroy
    - invocato dopo l'esecuzione di metodi annotati @Remove – prima che l'istanza del bean sia rimossa dal contenitore
  - @Remove
    - invocato su richiesta dal client per indicare la fine della sessione – ad es., @Remove public void endSession() {}



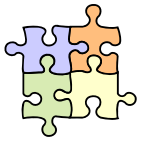
Ciclo di vita dei session bean stateful (2/2)

- @PrePassivate
  - invocato prima che il contenitore passivi il bean (serializza e salva lo stato) – prima che il container rimuovi il bean temporaneamente dalla sua memoria
- @PostActivate
  - invocato dal contenitore dopo aver attivato (caricato e deserializzato lo stato) il bean



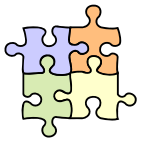
Attivazione e passivazione

- Il contenitore EJB gestisce un pool di oggetti EJB
  - assegna le richieste ad oggetti in questo pool
  - può decidere di riassegnare un oggetto, inizialmente assegnato ad un client, ad un client diverso
  - il client diverso potrebbe non aver ancora completato la sua sessione
- Il container gestisce lo swapping di oggetti EJB mediante
  - *passivazione* – lo stato del bean serializzato e salvato in memoria – il metodo @PrePassivate specifica come rilasciare risorse non serializzabili (ad es., connessioni)
  - *attivazione* – lo stato del bean ripristinato dai dati salvati in memoria – il metodo @PostActivate specifica come riprendere risorse



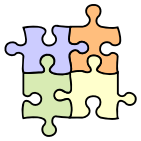
* Message-driven bean

- Message-driven bean
  - un message-driven bean è un enterprise bean che consente di elaborare messaggi in modo asincrono
    - funge normalmente da ascoltatore di messaggi JMS
    - questi messaggi JMS possono essere inviati da altri componenti Java EE e non



Similitudini e differenze

- I message-driven bean sono simili ai session bean stateless
  - le istanze sono equivalenti, poiché non gestiscono stato conversazionale
  - un message-driven bean può elaborare messaggi da più clienti
- I message-driven bean sono diversi dai session bean
  - poiché non vengono acceduti mediante un'interfaccia – ma tramite una destinazione JMS

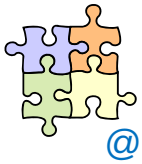


Produttore di messaggi

- Come produttore, può essere utilizzato l'application client SimpleProducer mostrato nel contesto di JMS
  - che genera ed invia un certo numero di messaggi di testo alla destinazione jms/MyQueue

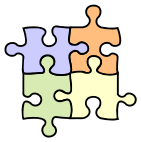
Esempio – SimpleMessageConsumerEJB (1)

```
package simplemessageconsumer;

import javax.ejb.*;
import javax.jms.*;
import java.util.logging.*;

@MessageDriven(
    activationConfig = {
        @ActionConfigProperty(
            propertyName = "destinationType",
            propertyValue = "javax.jms.Queue"
        ) },
    mappedName = “jms/MyQueue")
public class SimpleMessageConsumer implements MessageListener {

    ...

}
```



Esempio – SimpleMessageConsumerEJB (2)

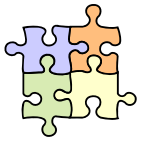
```
@MessageDriven(...)
public class SimpleMessageConsumer implements MessageListener {
    public SimpleMessageConsumer() {  }

    public void onMessage(Message message) {
        ...
    }
}
```

Esempio – SimpleMessageConsumer (3)

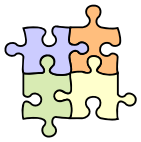
```
public void onMessage(Message message) {
    Logger logger = Logger.getLogger("SimpleMessageConsumer");

    try {
        if (message instanceof TextMessage) {
            TextMessage msg = (TextMessage) message;
            logger.info("SimpleMessageConsumer: Message received: " +
                        msg.getText());
        } else {
            logger.warning("Message of wrong type: " +
                           message.getClass().getName());
        }
    } catch (Exception e) {
        System.err.println("Exception in SimpleMessageConsumer");
        e.printStackTrace();
    }
}
```

da compilare e fare il deploy su un AS



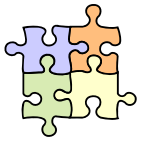
* Java EE e qualità

- La piattaforma Java EE si propone di sostenere/realizzare un certo numero di qualità
  - alcune tipiche per applicazioni web e applicazioni distribuite
  - altre più strategiche
- Per alcuni attributi di qualità, discutiamo brevemente
  - obiettivi di progetto
  - come raggiunti
  - tattiche usate



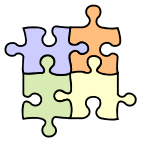
Soluzioni Java EE – in sintesi

- Soluzioni di Java EE
  - componenti EJB
  - JSP
  - Java servlet
  - JMS – Java Messaging Service
  - JNDI – Java NAming and Directory Interface
  - JTS – Java Transaction Service
  - JCA – J2EE Connector Architecture
  - Client Access Services COM Bridge
  - RMI over IIOP
  - JDBC
  - WS
  - ...



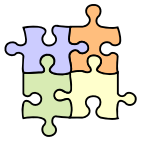
Portabilità

- Portabilità – “write once, run everywhere”
  - obiettivi
    - portabilità su piattaforme/AS diversi – con uno sforzo minimo
    - non garantito dal solo linguaggio Java – i componenti lato server richiedono servizi addizionali come transazioni e sicurezza
  - come raggiunta
    - la specifica impone che i contenitori per componenti forniscano servizi addizionali definiti contrattualmente
    - contenitori definiti per le principali piattaforme
  - tattiche usate
    - maintain existing interfaces, generalize modules, abstract common services



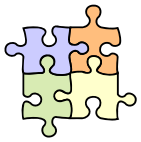
Costruibilità

- Costruibilità
  - obiettivi
    - alcuni servizi comuni devono poter essere gestiti in modo semplice – ad es., transazioni, sicurezza, servizi di naming, ...
  - come raggiunta
    - la specifica impone che i contenitori implementino diversi servizi comuni pronti da usare
    - servizi specificati spesso a livello di rilascio – i descrittori di deployment consentono di selezionare e configurare i servizi da usare
  - tattiche usate
    - abstract common services, maintain interfaces, hide information



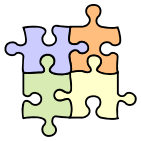
Specificità bilanciata

- Specificità bilanciata
  - obiettivi
    - standard per componenti dettagliato abbastanza – per definire una semantica di riferimento
    - ma allo stesso tempo, abbastanza generale per consentire ai venditori di offrire funzionalità/caratteristiche specifiche/aggiuntive
  - come raggiunta
    - servizi per EJB specificati in termini di API – dettagliato – ma generale per consentire al costruttore del contenitore di implementare caratteristiche specifiche e ottimizzazioni
  - tattiche usate
    - anticipate expected changes, abstract common services, hide information



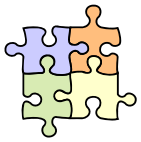
Trasparenza nell'implementazione

- Trasparenza nell'implementazione
  - obiettivi
    - fornire una trasparenza completa in modo che programmi client possano essere indipendenti dai dettagli implementativi – ad es., piattaforma hw/sw, posizione dei componenti, ...
  - come raggiunta
    - l'uso di interfacce (remote e locali) incoraggia il disaccoppiamento tra interfaccia e implementazione
    - le decisioni relative alle implementazioni degli enterprise bean sono quindi trasparenti ai suoi client
  - tattiche usate
    - maintain existing interfaces, semantic coherence



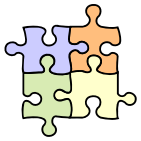
Interoperabilità

- Interoperabilità
  - obiettivi
    - supportare l'interoperabilità di componenti lato server realizzati su contenitori di fornitori diversi
    - interoperabilità con CORBA e componenti Microsoft
  - come raggiunta
    - supporto per l'interoperabilità tra componenti lato server definito contrattualmente a livello della piattaforma
    - sostiene anche la definizione di bridge verso altre piattaforme (CORBA e COM)
    - supporto per Web services
  - tattiche usate
    - adherence to defined protocols



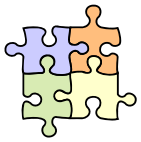
Evolvibilità/estendibilità

- Evolvibilità/estendibilità
  - obiettivi
    - possibilità di adottare diverse tecnologie in modo incrementale
    - possibilità di incorporare nuove tecnologie quando introdotte
  - come raggiunte
    - specifiche partizionate in categorie (ad es., web tier, EJB tier) – che possono essere adottate ed evolvere separatamente
    - specifica basata su componenti consente estensioni future – alcune tecnologie attuali erano assenti in versioni precedenti
  - tattiche usate
    - semantic coherence, hide information
    - anticipate expected changes



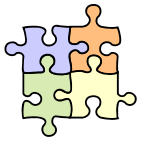
Disponibilità/affidabilità

- Disponibilità/affidabilità
  - obiettivi
    - sistema disponibile 24/7 – con fuori servizio di breve durata
  - come raggiunta
    - contenitori implementano servizi per la gestione di transazioni e meccanismi di recovery
  - tattiche usate
    - heartbeat, transactions, passive redundancy



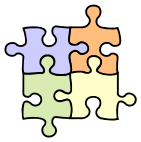
Prestazioni

- Prestazioni
  - obiettivi
    - il sistema deve rispondere rapidamente agli utenti
  - come raggiunta
    - l'approccio a componenti distribuiti rende possibile la ridistribuzione del carico e il tuning
  - tattiche usate
    - configuration files, load balancing, maintain multiple copies



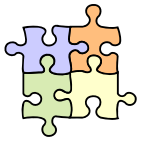
Scalabilità

- Scalabilità
  - obiettivi
    - il sistema deve sopportare variazioni del carico – senza intervento umano
  - come raggiunta
    - architettura client/server multi-livello
    - la piattaforma prevede la possibilità di espandere il numero di server e fare bilanciamento del carico
  - tattiche usate
    - load balancing



Sicurezza

- Sicurezza
  - obiettivi
    - autenticazione e autorizzazioni
  - come raggiunta
    - la specifica definisce diversi meccanismi per la sicurezza – dichiarativi, basati su ruoli e programmabili
  - tattiche usate
    - authentication, authorization, data confidentiality



Usabilità

- Usabilità
  - obiettivi
    - utenti diversi devono poter accedere contenuti diversi in forme diverse
  - come raggiunta
    - la piattaforma prevede tecnologie specifiche (ad es., JSP e servlet) per fornire contenuti in forme diverse
  - tattiche usate
    - separate user interface