Introduction aux Applications d'Entreprise

Plan du cours

- Applications d’Entreprise
- Concepts de base
- Architecture multi-tiers
- Serveur d’entreprise
- Conteneur léger

Applications d'Entreprise

- Réduction des coûts de développement et de maintenance
- Portabilité
- Disponibilité
- Sécurité
- Montée en charge
- Sûreté de fonctionnement
- Extensibilité
- Intégration
- Adaptabilité
- Qualité du code
- Réponse aux besoins exprimés par les utilisateurs

Concepts de base

- Composants
- Frameworks/Cadriciels
- Patrons de conception
- Inversion de Contrôle
- Injection de Dépendance
- Programmation Orientée Aspects

Composant

- Module logiciel autonome pouvant être installé sur différentes plate-formes
- Exporte différents attributs, propriétés ou méthodes
- Peut-être configuré
- Capable de s'auto-décrire
- Briques de base configurables pour permettre la construction d'une application par composition

Composant

- Attention composant est différent d'objet
- Composant : niveau architecture générale du logiciel
- Objet : niveau codage du logiciel
- On peut faire de la programmation orientée composant en C, en ML, en Lisp, en assembleur...

Framework/Cadriciel

- Structure logicielle
- Ensemble de composants logiciels structurels
- Créant les fondations / grandes lignes d'un logiciel
- Générique
- Présentant un cadre de travail qui guide le développeur

Patrons de Conception / Design Patterns

- Arrangement caractéristique de modules reconnu comme bonne pratique
- Solution standard générique ré-utilisable
- Exemples :
  - Factory, Façade, Iterator, Observer, Proxy, Singleton...
  - MVC (combinaison de patterns) : patron d'architecture

Inversion de Contrôle / IoC

- Patron d'architecture pour Framework
- Le code générique/réutilisable contrôle l'exécution du code spécifique

Injection de dépendances / DI

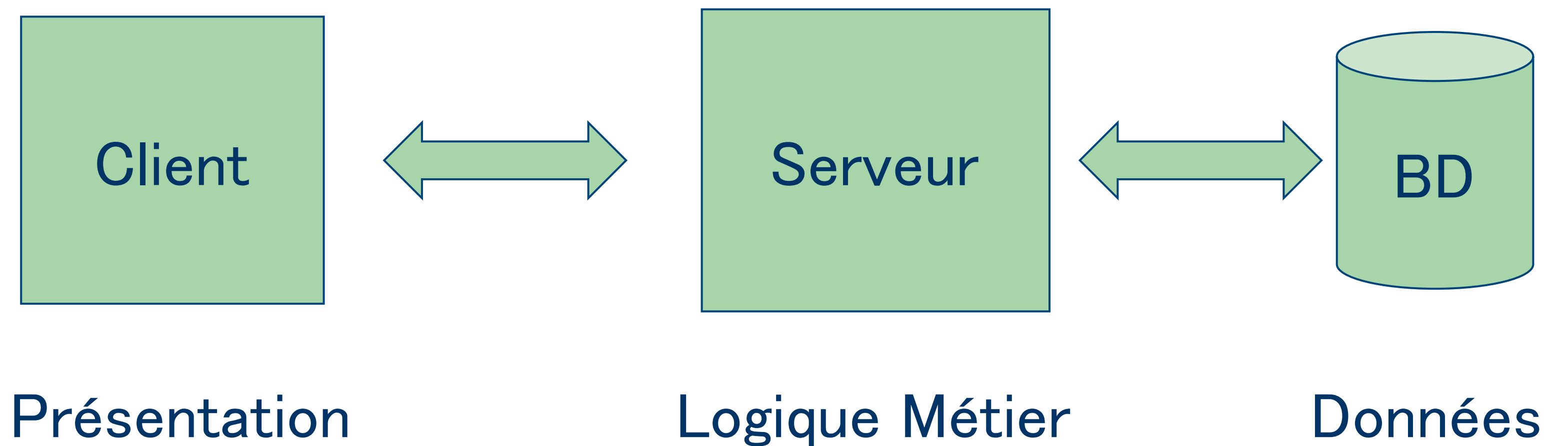
- Patron de conception
- Manière automatique et directe de fournir une dépendance externe dans un composant logiciel
- En java :
  - Annotations ;
  - Fichiers XML ;

Programmation par Aspects / AOP

- Paradigme de programmation
- Augmenter la modularité en améliorant la séparation des préoccupations
- Isoler le code métier du code technique
- Exemples : journalisation, sécurité, transactions…

Architectures multi–tiers

- Trois tiers
  - Présentation / Métier / Données



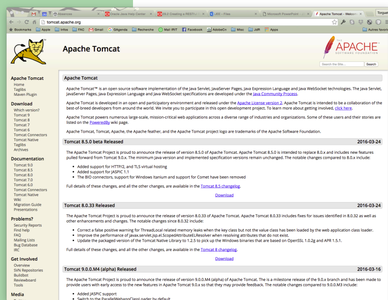
Architectures multi-tiers

- Multi-tiers

  Le tiers données peut lui même être une application orientée service ou ressource, un ERP, un SI d'Entreprise pré-existant ...

Architectures multi-tiers

- Multi-tiers Orienté Web



Technologies : Servlets, JSP, JSF, ASP.Net, PHP…

Technologies : JavaBeans, EJB, Classes PHP, ASP.Net …

Technologies : JDBC, RMI, JMS, JCA, CORBA, COM, WS SOAP, REST …

Navigateur

Présentation

Logique Métier

Intergiciels

Persistance

Java EE

- Spécification
  - Modèle de programmation, assemblage de composants, déploiement
  - Serveur d'entreprise et services
- Implantation de référence opérationnelle
  - Glassfish
- Suite de tests de conformité
  - Certification Oracle/Sun

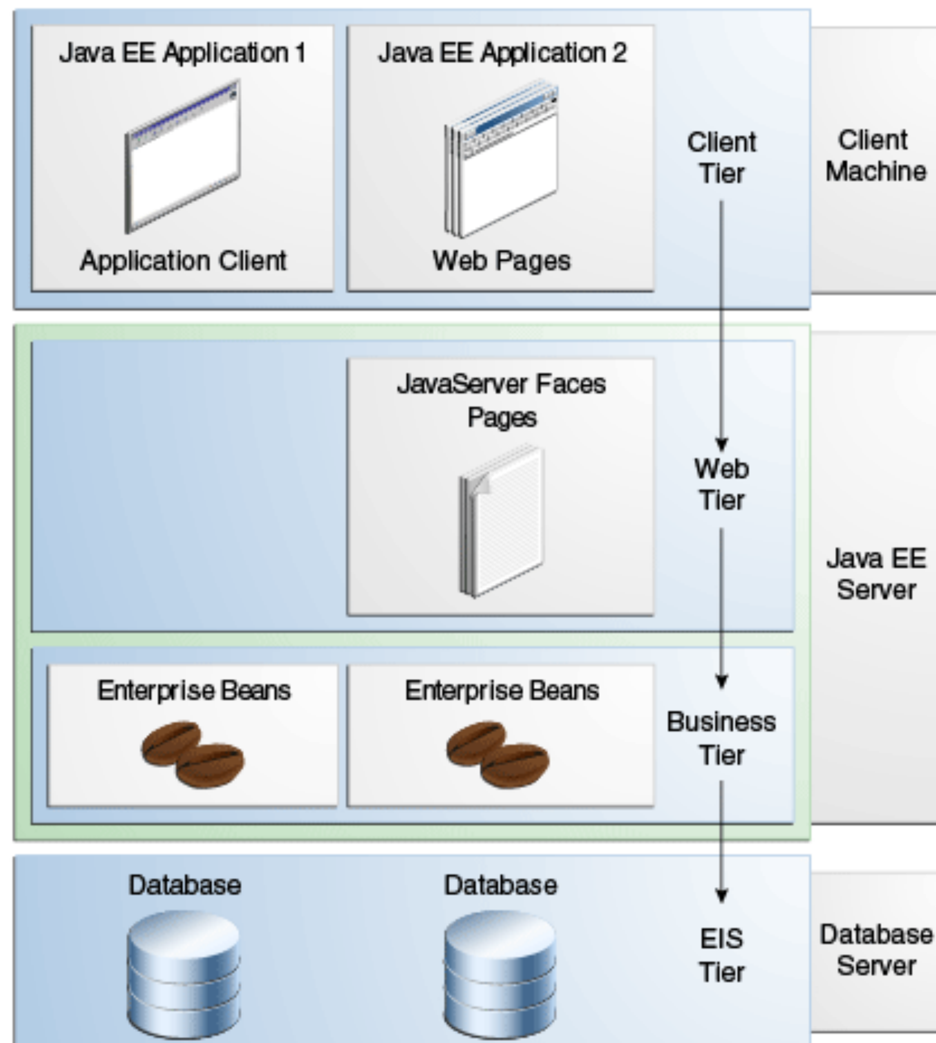
Serveur d'entreprise

- Application générique
- Composée de conteneurs qui gèrent/hébergent les composants de l'application
- Offre un ensemble d'APIs support ou permettant l'accès aux tiers données
- L'application est donc hébergée par le serveur
- Le serveur est administrable

Offre existante

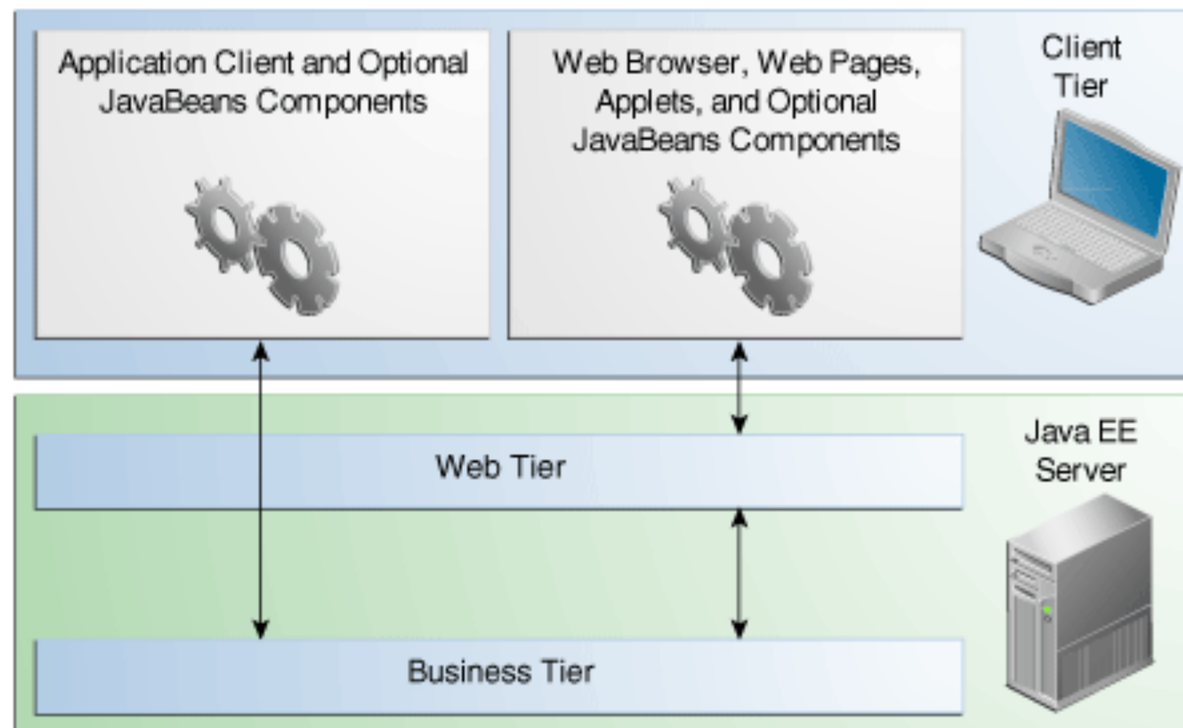
- Commerciale
  - IBM WebSphere
  - BEA WebLogic
  - Oracle Application Server
  - SAP Web Application Server
- Libre
  - Jboss (RedHat)
  - Jonas (ObjectWeb)
  - GlassFish (Oracle/Sun)

Serveur d'entreprise Java EE

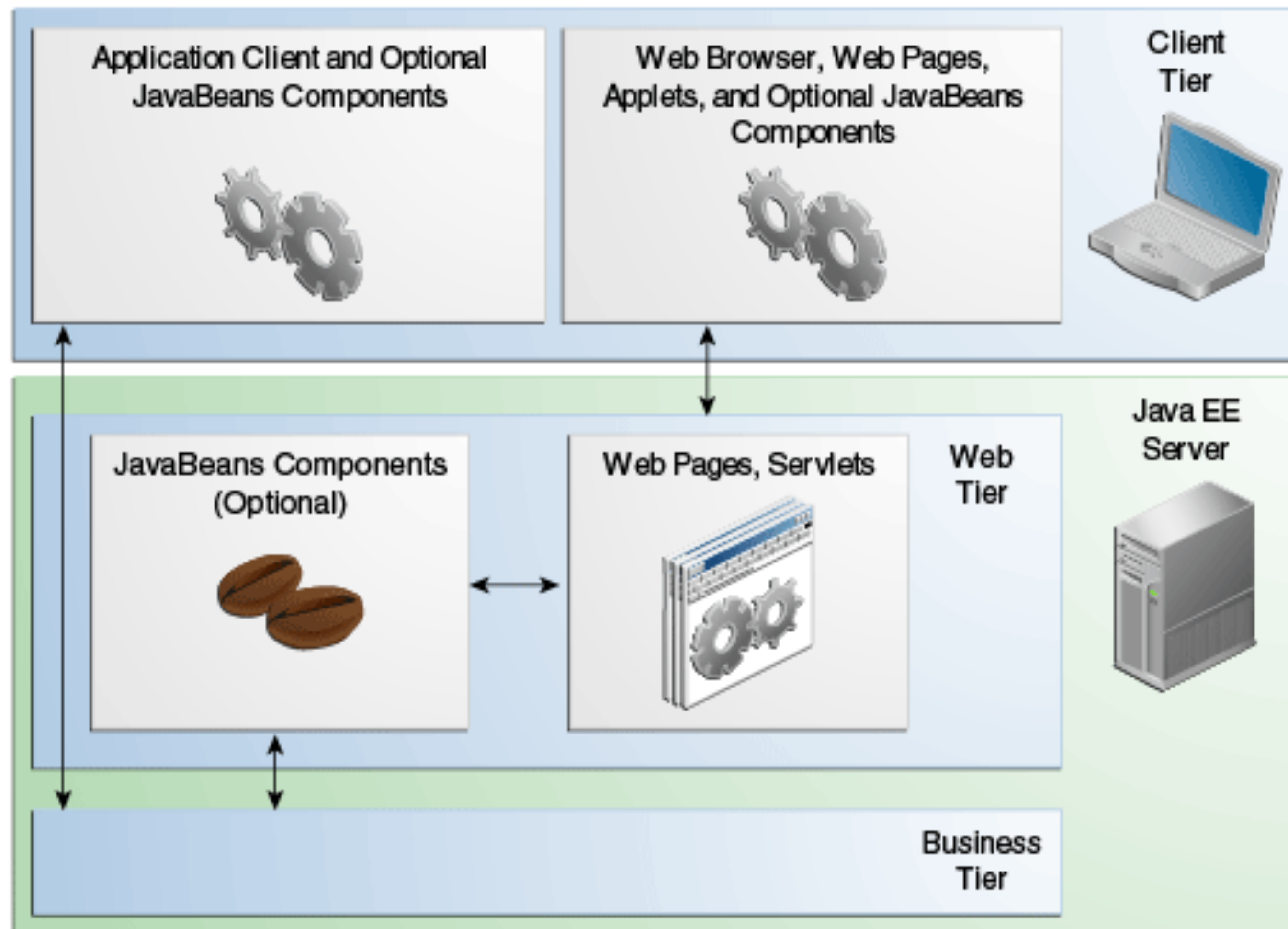


Communications C/S

- Directe : RMI/IIOP
- Indirecte : HTTP



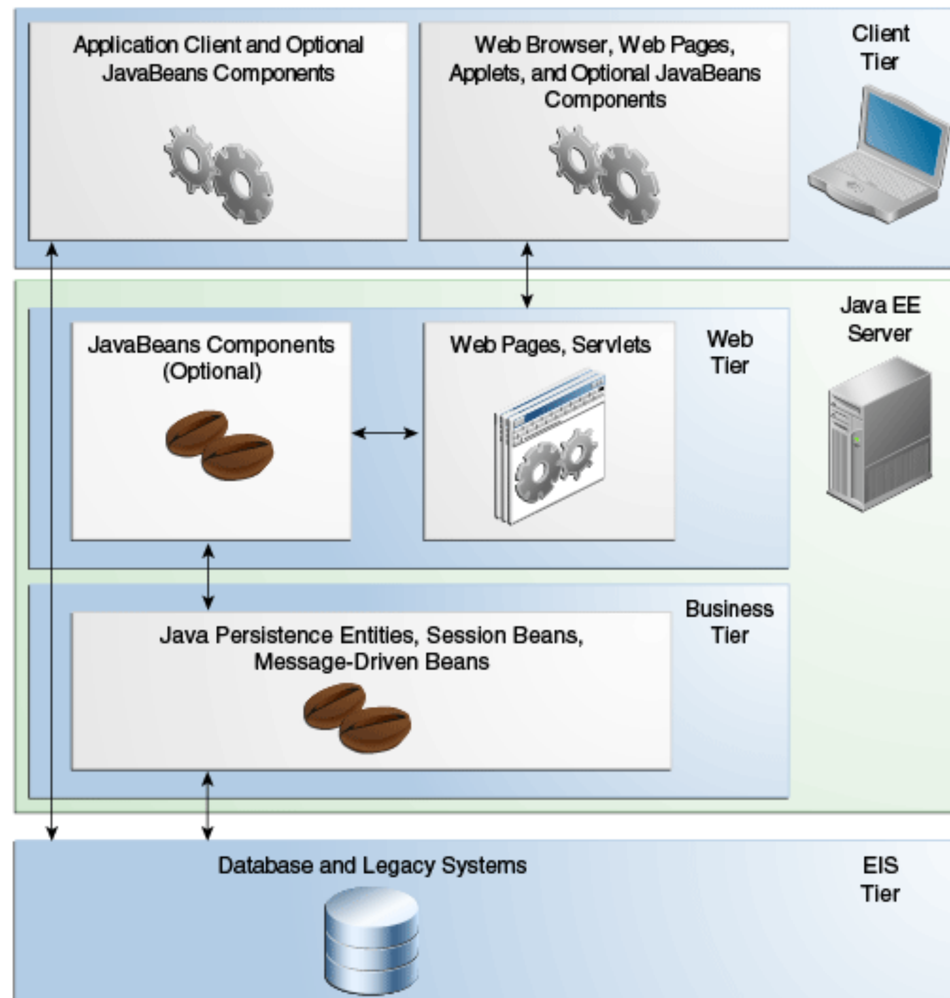
Composants Web



Composants Web Java EE

- Servlets : code java exécuté par le serveur
- JSP : mélange Java et HTML
- JavaServer Faces : Framework MVC – séparation code HTML et code Java
- WebServices : application orientée services – services vus comme des méthodes appelées à distance
- REST : application orientée ressources – CRUD sur des ressources offertes par l'application

Composants Métiers Java EE



Composants Métiers

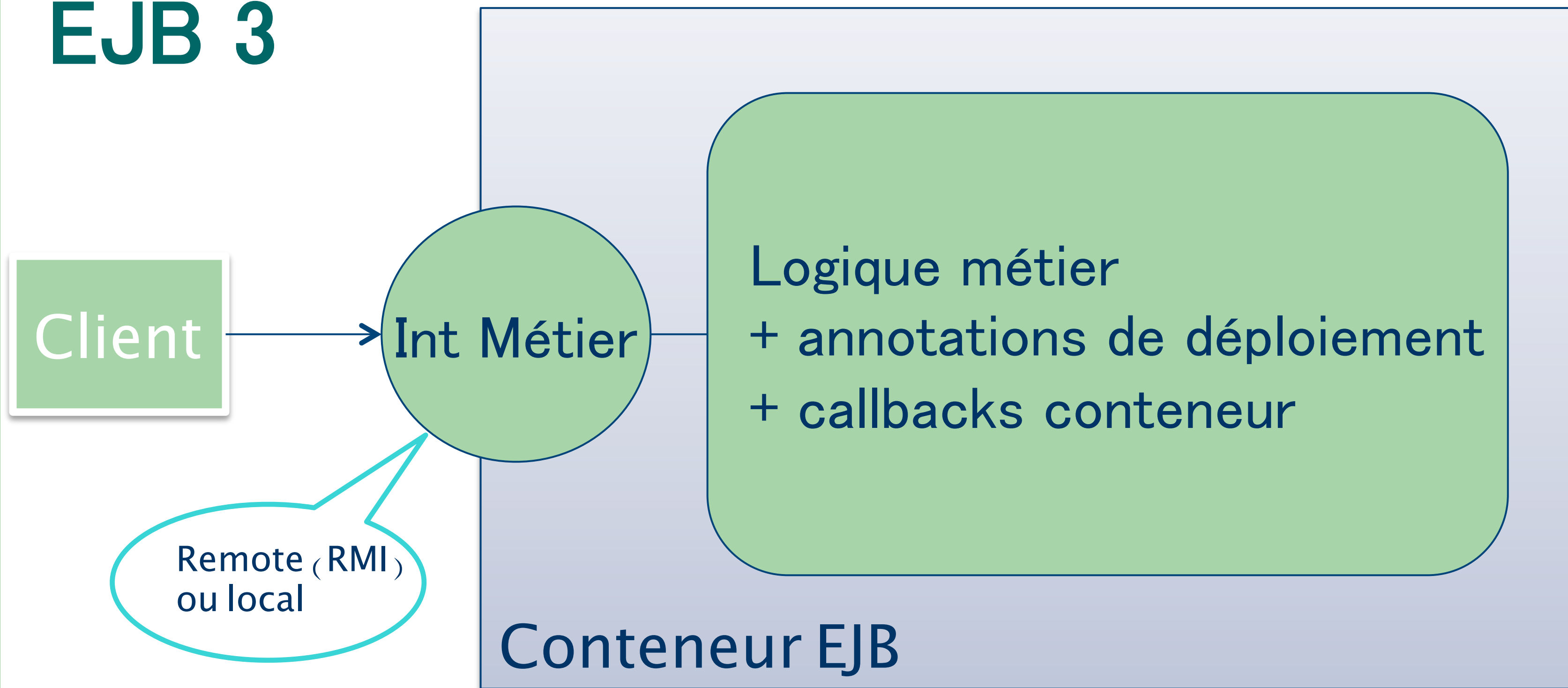
- EJB : Entreprise Java Beans
- Session Beans : code métier synchrone avec ou sans état (appels distants via RMI)
- Message Driven Beans : code métier asynchrone (appels distants via JMS)
- Entity Beans : encapsulation des données (ORM)

EJB : gamme de services implicites

- Gestion du cycle de vie
- Gestion de l'état
- Sécurité
- Transactions
- Persistance
- Localisation transparente
- Répartition de charge

=> Le développeur se focalise sur les aspects "métier"

EJB 3



Simplification :

- 1 interface et 1 classe ;
- Descripteur de déploiement (XML) facultatif – remplacé par annotations mais prioritaire sur annotations

Interface métier

- Remote (RMI/IIOP) ou Local
- Vue cliente de l'EJB
- Déclare les méthodes "métier"
- Implémentée par les outils du conteneur EJB au moment du déploiement
- Pas de dépendance directe

Exemple d'interface métier

```
@Remote
public interface BanqueSessionBeanRemote {
    public long creerCompte(String nom, String prenom, float solde);
    public long chercherCompte(String nom, String prenom);
    public void ajouter(long id, float somme);
    public void retirer(long id, float somme);
    public Position position(long id);
}
```

EJB : implémentation

- Implémente les méthodes de l'interface métier
- Peut hériter d'un autre EJB ou d'un POJO
- Spécifie les caractéristiques de déploiement par annotations
  - Type de bean
  - Comportement : transactions, sécurité, persistance...
  - Callbacks conteneur

Exemple : implémentation

```
@Stateless
public class BanqueSessionBean implements BanqueSessionBeanRemote {
   @PersistenceContext
   private EntityManager entityManager = null;

   @Override
   public long creerCompte(String nom, String prenom, float solde) {
      Compte c = new Compte();
      c.setNom(nom);
      c.setPrenom(prenom);
      c.setSolde(solde);
      c.setDateDerniereOperation(new Date());
      entityManager.persist(c);
      Logger.getLogger(BanqueMessageBean.class.getName()).log(Level.INFO, "Compte cree");
      return chercherCompte(nom, prenom);
   }
...
```

Utilisation

- Via JNDI

```
System.out.println("Initial context");
InitialContext ic = new InitialContext();
System.out.println("Lookup");
BanqueSessionBeanRemote b =
        (BanqueSessionBeanRemote) ic.lookup("org.torguet.BanqueSessionBeanRemote");
if(b != null) {
   System.out.println("chercherCompte");
   long id = b.chercherCompte("Torguet", "Patrice");
   …
```

Utilisation et Stateful

- Bean Stateful : code métier + données de session
  - Exemple : caddie
- Chaque lookup retourne une nouvelle instance
- Exemple (servlet) :

```
CartBean c = (CartBean)httpSession.getAttribute(« caddie »);
if(c == null) {
    c = initialContext.lookup(« ShoppingCart »);
    httpSession.setAttribute(« caddie », c);
}
```

Optimisation par le conteneur

- Stateless Session Bean : sans état
  - Pool d'instances
  - Le serveur peut y accéder via un pool de threads
- Stateful Session et Entity beans
  - Activation / Passivation (appel par le conteneur des callbacks @PostActivate et @PrePassivate)
- Pour tous les Session Beans
  - Callbacks @PostConstruct et @PreDestroy
  - Gestion possible par un listener @CallbackListener

Injection de dépendances

- Variable d'instance initialisée par le conteneur
  - Alternative au lookup JNDI
  - Interne au conteneur (pas client distant !)
- Egalement utilisé par JPA pour le conteneur de persistance
- Implicite ou avec spécification du nom (name="nom JNDI")

```
@EJB BanqueSessionBeanRemote beanBanque; // Injection d'EJB (local)
@Resource javax.sql.DataSource ds; // Injection de ressource
```

EJB Entité

- Utilisation de JPA
  - Mapping Objet / Relationnel
  - Utilise un framework de persistance : Hibernate, EclipseLink…
  - Génération automatique à partir du code, de la BD ou d'un fichier XML
- Persistance gérée par le conteneur
  - Pas d'accès BD dans le code
  - Gestion déclarative (annotations)
    - Classe persistante (@Entity)
    - Champs persistants : variables d'instance ou propriétés avec get/set
  - Présence obligatoire d'un constructeur sans argument et implémentation de Serializable si transmission côté client
  - Relations entre instances d'entités (cardinalités 1-1, 1-N, N-P, uni ou bi-
- EJB-QL
  -
  -

Exemple : le bean Entity

```
@Entity
public class Facture implements java.io.Serializable {
        private int numfact;
        private double montant;
        private Client client;

        public Facture() { } // Constructeur par défaut (obligatoire pour Entity bean)
        public Facture(String numfact) { this.numfact = numfact; }

        @Id
        public int getNumfact() { return numfact; }
        public void setNumfact(int nf) { numfact = nf; }

        public void setMontant(double montant) { this.montant = montant; }
        public double getMontant( ) { return montant; }

        @ManyToOne
        @JoinColumn (name = “refclient”)
        public Client getClient( ) { return client; }
        public void setClient(Client client) { this.client = client; }
}
```

Exemple : utilisation

```
@Stateless
@Remote(FacturationRemote.class)
public class FacturationBean implements FacturationRemote {

        @PersistenceContext
        private EntityManager entityManager = null;

        public void creerFacture(String numfact, double montant) {
                Facture fact = new Facture(numfact);
                fact.setMontant(montant);
                entityManager.persist(fact);
        }
        public Facture getFacture(String numfact) {
                return entityManager.find(Facture.class, numfact);
        }
}
```

Relations entre tables

- Part de la classe courante
  - Exemple : Client, lien OneToMany avec Facture
- Le ″propriétaire″ de la relation correspond à la table qui définit la clé étrangère (ne s'applique pas à ManyToMany)
  - Lien Client/Factures : la facture est "propriétaire" de la relation
  - Attribut "mappedBy" côté Client et "joinCollumn" côté Facture
- OneToOne (exemple : Fournisseur et Adresse)
- OneToMany (exemple : Client et Facture)
- ManyToOne (exemple : Facture et Client)
- ManyToMany (exemple : Produit et Fournisseur)

Relations et matérialisation

- Par défaut : "lazy loading" : récupération des
-

  données

  Annotation sur la relation (défaut : LAZY)
  - @OneToMany(fetch=FetchType.EAGER)

EntityManager

- persist(e) : sauvegarde
- merge(e) : sauvegarde après rattachement au contexte de persistance (retourne l'entité rattachée)
- remove(e) : suppression
- find(e.class, clé_primaire) : recherche par clé
- refresh(e) : annulation des modifications non sauvegardées
- flush() : sauvegarde immédiate des modifications (cache)
- createQuery(ejbQL)
- createNativeQuery(SQL)

EJB-QL

- Dialecte proche de SQL
  - "select ... from ... where ..." sur des objets.
  - Requêtes de modification : update, delete
  - Requêtes plus complexes : join, group by...
  - Possibilité de requêtes paramétrées
  - Utilisation directe ou requêtes nommées (@NamedQuery)
- Exemple

```
public List<Facture> listFactures( ) {
        Query qry = entityManager.createQuery(
                ″select f from Facture f ″);
        return qry.getResultList();
}
```

Callbacks cycle de vie

- Appelés par le ″persistence manager″
  - Annotations @PrePersis, @PostPersist…
  - Gestion possible par une classe dédiée

    @EntityListener(MyListener.class)
- Exemple

```
@PreRemove
void preRemove() {
    logger.log(Level.INFO, "Avant suppression");
}
```

Transactions

- Appicable aux 3 profils de composants
  - Session, Entité, Message driven
- Gestion explicite (utilisant JTA)
  - Begin, commit, rollback
- Gestion déclarative (annotations)
  - Au niveau des méthodes

  Required (défaut), Supports, NotSupported, RequiresNew, Mandatory, Never

Sécurité

- Domaine de protection (ou de confiance)
  - Le conteneur (ou un ensemble de conteneurs)
  - A l'intérieur, pas de nouvelle authentification
    - Le contexte de sécurité est propagé
    - L'objet invoqué fait confiance à l'appelant
  - Mécanisme de changement d'identité (@RunAs("identité"))
- Côté client : mapping avec les méthodes classiques (Basic, Form, Client-Cert)
  - Authentification par le conteneur Web et transmission du contexte au conteneur EJB

JCA

- Java Connector Architecture
- Intégration avec les SI d'Entreprise (EIS)
  - Applications (ERP, SupplyChain...)
  - Intergiciels (gestionnaire de transaction...)
- Connecteur composé de
  - Contrats système (définition de la connectivité – connexions, transactions, sécurité...)
  - Adaptateur de Ressources (jars + dll/dso)

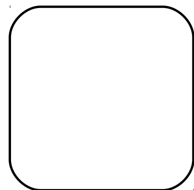
Rôles définis par la spec EJB

- Fournisseur de beans
  - Développeur qui crée les EJB
- Assembleur d'applications
  - Crée l'application par assemblage d'EJB
- Administrateur
  - Déploiement, sécurité, exploitation, montée en charge...
  - Analogue au rôle de DBA pour les BD

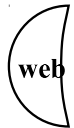
Packaging

- Application JavaEE (agrégation de différents tiers)
  - Fichier ".ear" + descripteur "application.xml"
- Tiers client
  - Web : fichier ".war" + descripteur "web.xml"
  - Client lourd : fichier ".jar" + descripteur "application-client.xml"
- Tiers EJB
  - Fichier ".jar" + descripteur "ejb-jar.xml"
- Tiers "EIS"
  - Fichier ".rar" + descripteur "rar.xml"

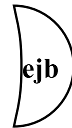
Répartition de charge : notations



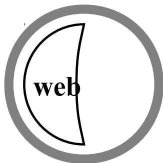
Un noeud (machine) qui héberge un ou plusieurs serveurs



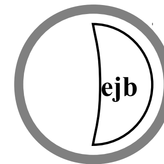
Un conteneur Web



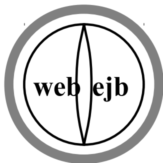
Un conteneur EJB



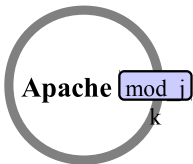
Un serveur qui héberge un conteneur Web



Un serveur qui héberge un conteneur EJB



Un serveur qui héberge un conteneur Web et un conteneur EJB



Un serveur Apache avec le module mod_jk

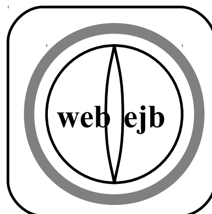
Répartition de charge

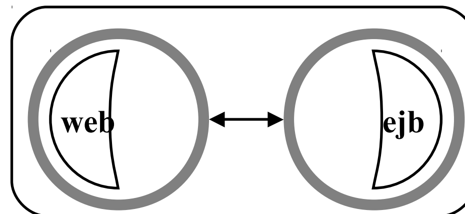
Répartition du serveur JavaEE

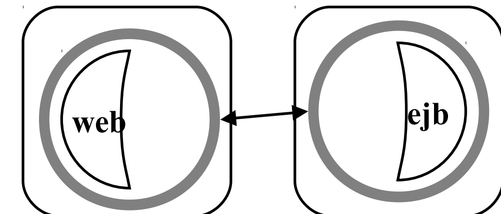
Compact

Réparti
(au sein d 'un même nœud)

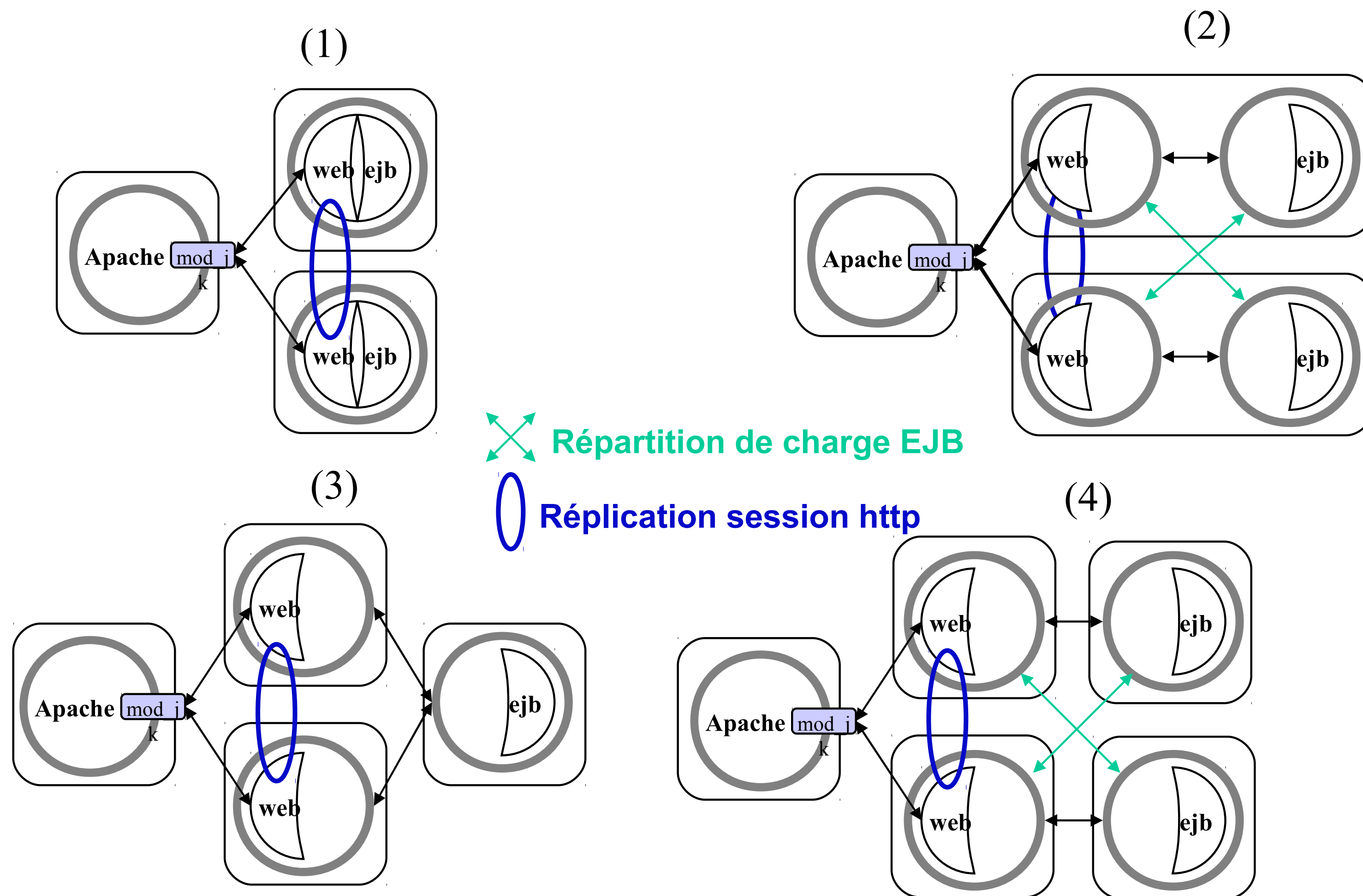
Réparti







Répartition de charge



Autres APIs JavaEE

- JPA : Java Persistence API – Mapping Objet Relationnel – Utilisé par les EJB Entité
- JTA : Java Transaction API – Gestion des transactions
- CDI : Context and Dependency Injection for Java – Injection de dépendances basées sur les annotations

Autres APIs JavaEE

- JCA : Java EE Connector Architecture – connexion avec des EIS existants, ERPs par exemple
- JavaMail API : notifications par e-mail
- Java API for WebSocket : permet des communications par WebSocket avec les clients web typiquement
- JSON-P : JSON Processing – permet la création et l'analyse de JSON

Autres APIs JavaEE

- JMS : Java Message Service – Envoi/réception de messages complexes transitant par des files ou des sujets de discussion – utilisé par MDB
- RMI : Remote Method Invocation – appel de méthodes à distance
- JDBC : Java DataBase Connectivity API – connexion à des SGBD relationnels

Autres APIs JavaEE

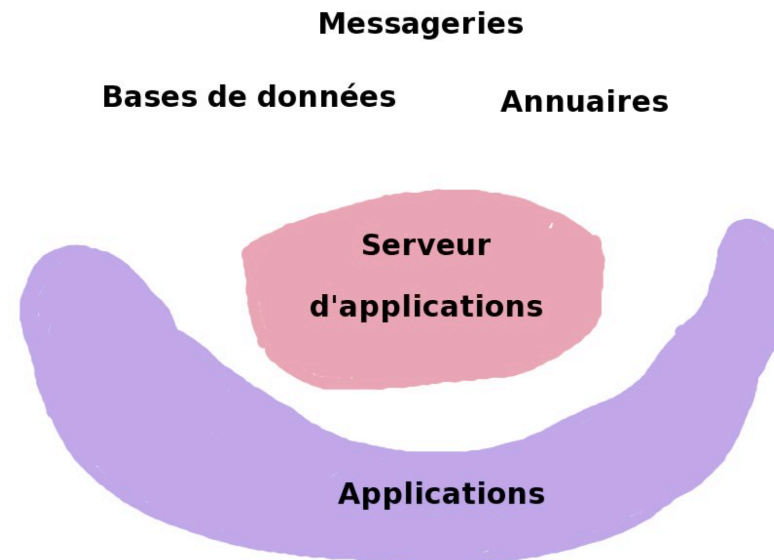
- JNDI : Java Naming and Directory Interface – Accès aux annuaires – permet de trouver les beans et les ressources (DB, outils de comm JMS…)
- JMX : Java Management eXtensions – Administration du serveur et de ses composants
- JAAS : Java Authentication and Autorization Service – gestion de la sécurité – rôles des utilisateurs

Inconvénients des serveurs d'entreprise

- Problème de qualification du serveur
- Peut prendre du temps
- Retarde l'adoption de nouvelles APIs et de nouveaux paradigmes
- Pourquoi ne pas inverser la relation entre le conteneur et l'application ?

Conteneurs légers

- L'application met en place un conteneur léger qui héberge les composants métiers
- Exemples :
  - Spring
  - Pico
  - Avalon
  - HiveMind



Références

- Tutoriaux Java EE Oracle
- Cours de Gautier Loyauté (Univ Marne la Vallée)
- Cours de Pierre-Yves Gibello (gibello.com)
- Cours de Fabienne Boyer (Univ Grenoble)