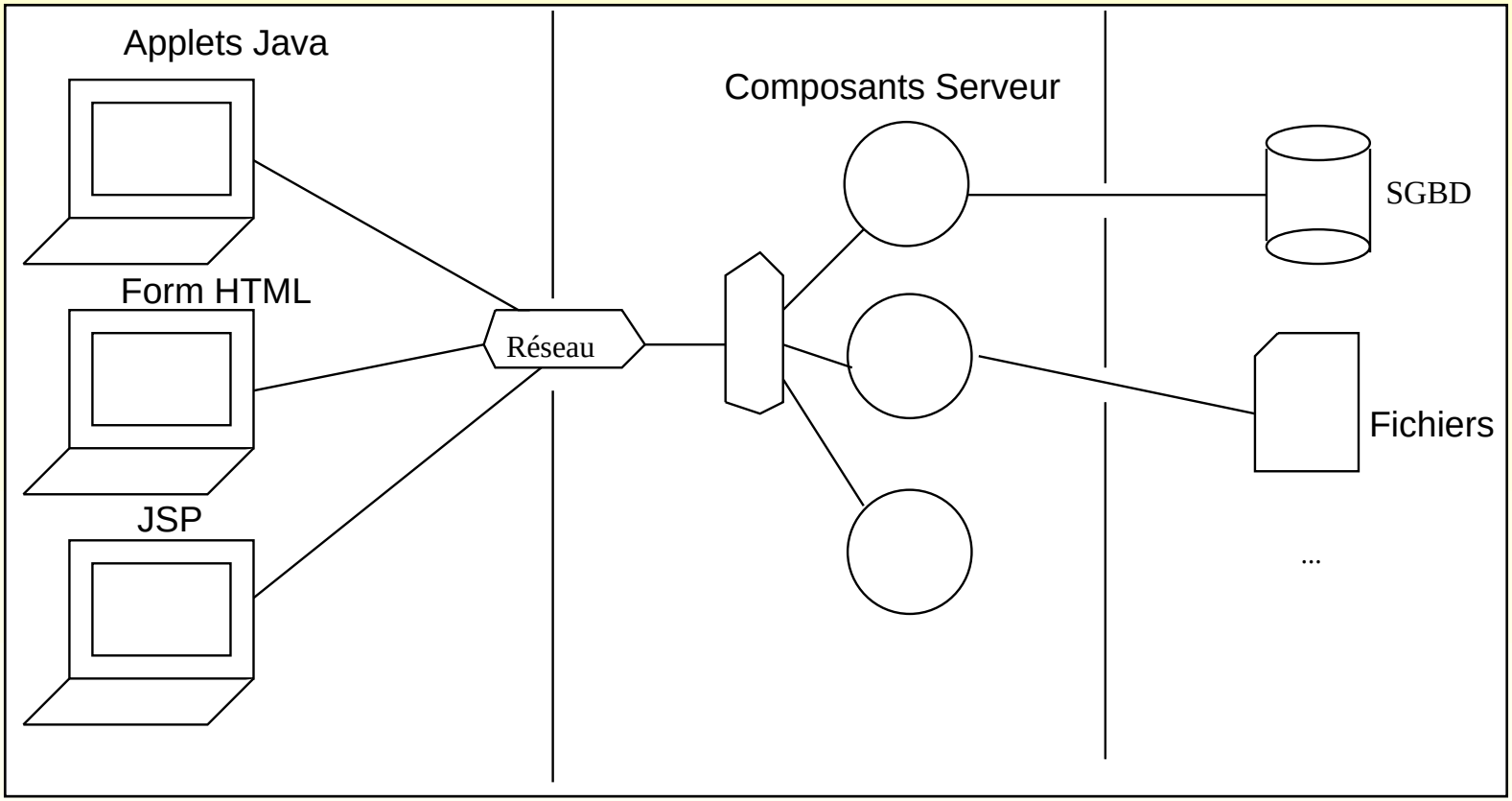
Les architectures 3-tiers
Partie II : les composants d'entreprise

Polytech'Lille

Les architectures 3-tiers



Les modèles de composants serveur

√ Le modèle de composants J2EE-EJB

- Orienté bases de données, multi-serveur d'applications, Java

Les modèles de composants serveur

- √ Le modèle de composants J2EE-EJB
  - Orienté bases de données, multi-serveur d'applications, Java
- √ Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plateformes
  - Des langages de spécifications de composants (IDL3)
  - Des langages de spécifications de déploiement (CIDL)

Les modèles de composants serveur

- √ Le modèle de composants J2EE-EJB
  - Orienté bases de données, multi-serveur d'applications, Java
- √ Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plateformes
  - Des langages de spécifications de composants (IDL3)
  - Des langages de spécifications de déploiement (CIDL)
- √ Le modèle .Net
  - Multi-langages (C++, VBScript, C#, Java ?)
  - Multi-plateformes ? ?

Des objets aux composants d'entreprise (1/3)

√ Des objets trop complexes

Des objets aux composants d'entreprise (1/3)

√ Des objets trop complexes

```
public class ObjetServeur                                    {
   private int x ;
   public void setX(int value)                              {


     this.x=value ;



   }
   public int getX()                                  {



     return this.x ;
   }
  public ObjetServeur()                                  {   ...  }
}
```

Des objets aux composants d'entreprise (1/3)

√ Des objets trop complexes

```
public class ObjetServeur     extends UnicastRemoteObject       {
   private int x ;
   public void setX(int value)   throws RemoteException     {


     this.x=value ;


                                                                  Gestion du réseau
   }
   public int getX()   throws RemoteException    {




     return this.x ;
   }
  public ObjetServeur()    throws RemoteException     {    ...   }
}
```

Des objets aux composants d'entreprise (1/3)

√ Des objets trop complexes

```
public class ObjetServeur     extends UnicastRemoteObject       {
   private int x ;
   public void setX(int value)    throws RemoteException      {

     if (User.getAuthentification()...
     this.x=value ;



   }
   public int getX()    throws RemoteException     {

     if (User.getAuthentification()...



     return this.x ;
   }
  public ObjetServeur()    throws RemoteException     {    ...   }
}
```

Gestion du réseau

Gestion de la sécurité

Des objets aux composants d'entreprise (1/3)

√ Des objets trop complexes

```
public class ObjetServeur     extends UnicastRemoteObject       {
   private int x ;
   public void setX(int value)    throws RemoteException      {

     if (User.getAuthentification()...
     this.x=value ;
     // code JDBC : stockage de X
     ...

   }
   public int getX()    throws RemoteException     {

     if (User.getAuthentification()...
     //code JDBC : restauration de X
     ...

     return this.x ;
   }
  public ObjetServeur()    throws RemoteException     {    ...   }
}
```

Gestion du réseau

Gestion de la sécurité

Gestion de la persistance

Des objets aux composants d'entreprise (1/3)

√ Des objets trop complexes

```
public class ObjetServeur     extends UnicastRemoteObject       {
   private int x ;
   public void setX(int value)    throws RemoteException      {
      Tx.beginTransaction() ;
     if (User.getAuthentification()...
     this.x=value ;
     // code JDBC : stockage de X
     ...
     Tx.commitTransaction() ;
   }
   public int getX()    throws RemoteException     {
      Tx.beginTransaction() ;
     if (User.getAuthentification()...
     //code JDBC : restauration de X
     ...
     Tx.commitTransaction() ;
     return this.x ;
   }
  public ObjetServeur()    throws RemoteException     {    ...   }
}
```

Gestion du réseau

Gestion de la sécurité

Gestion de la persistance

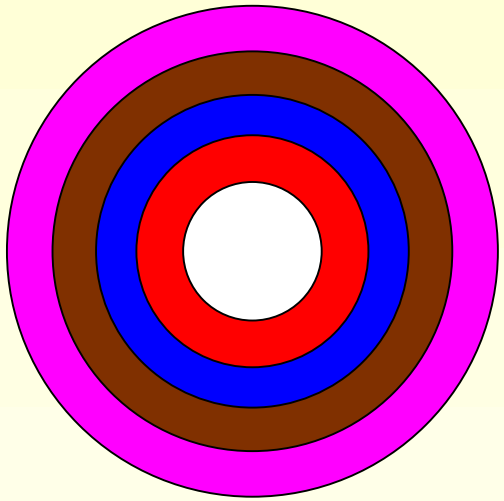
Gestion des transactions

Des objets aux composants d'entreprise (2/3)

√ 1ère étape :une meilleure structuration objet

Des objets aux composants d'entreprise (2/3)

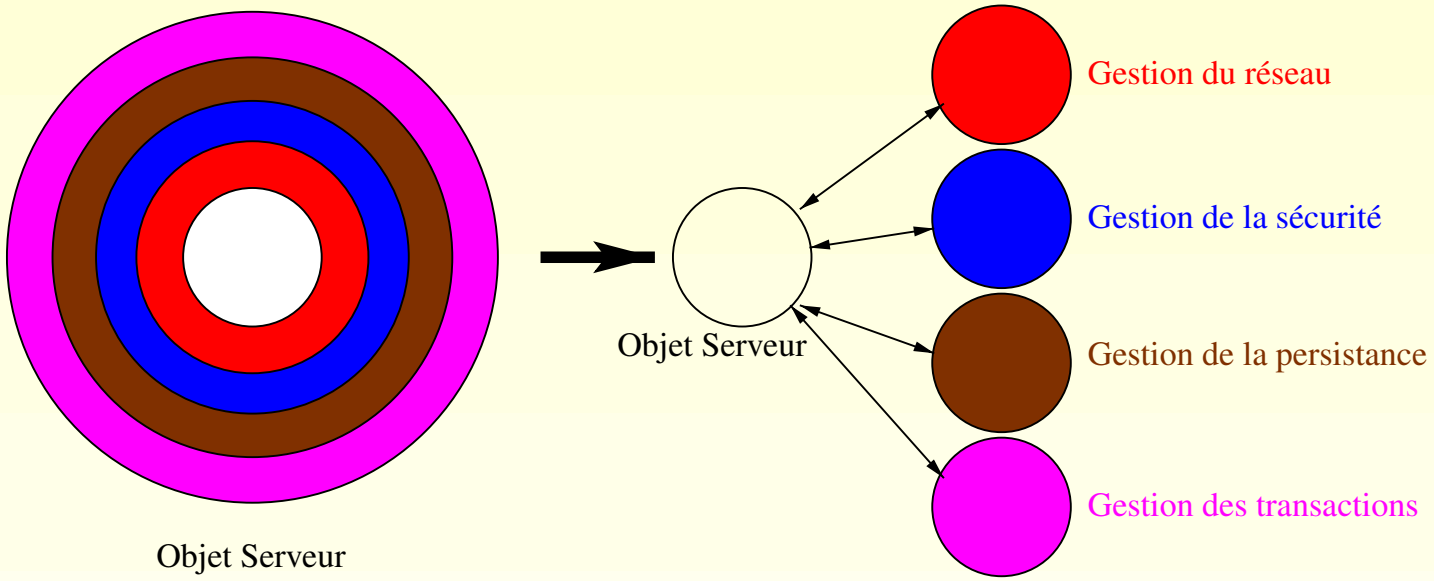
√ 1ère étape :une meilleure structuration objet



Objet Serveur

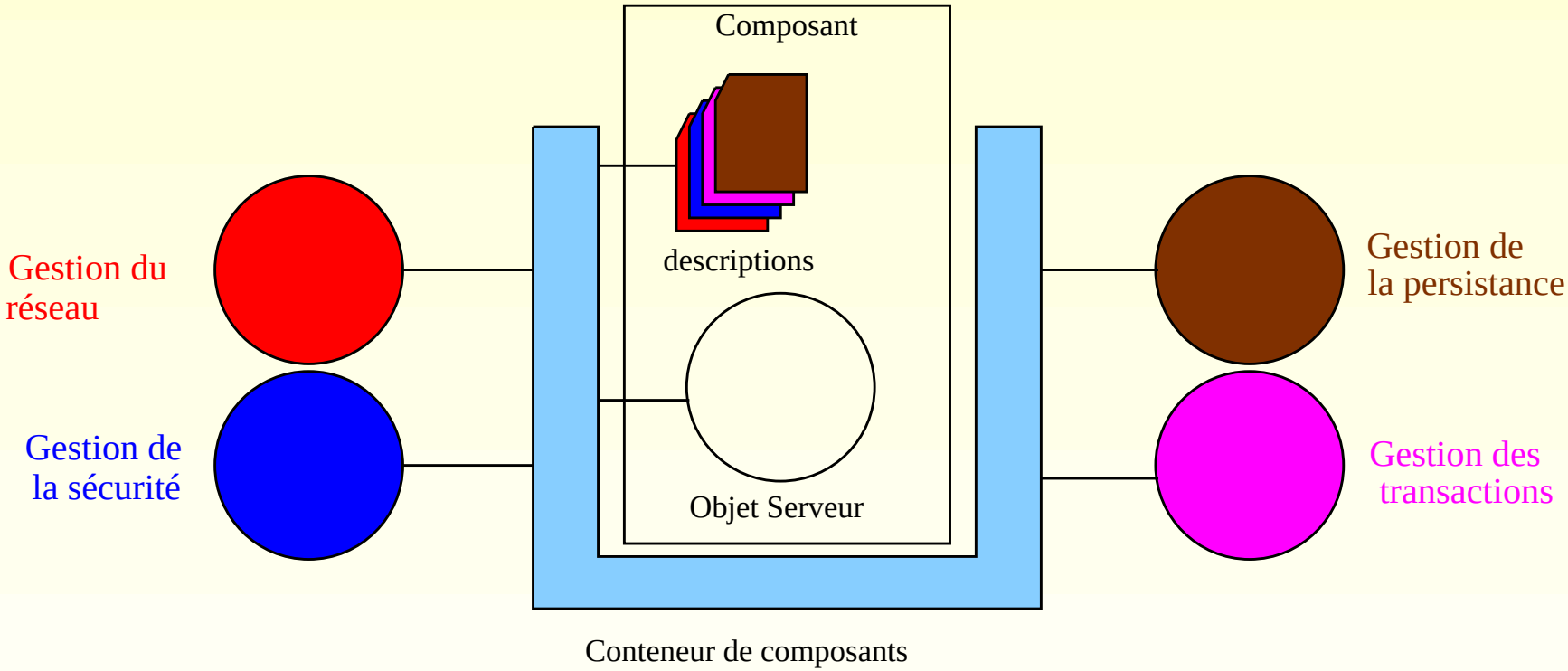
Des objets aux composants d'entreprise (2/3)

√ 1ère étape :une meilleure structuration objet



Des objets aux composants d'entreprise (3/3)

√ 2nde étape : complète séparation, notion de **conteneur**



Les composants d'entreprise : en résumé

- √ Applications cibles trop complexes :
  - ▶ Gérer le réseau, la sécurité, les transactions, les bases de données, le déploiement,. . .
- √ La solution : fournir un modèle de composants qui autorise un découpage aspects fonctionnels et aspects techniques :
  - ▶ aspects fonctionnels (le code métier) écrits dans un langage de programmation suivant un cadre de programmation
  - ▶ aspects techniques décrits séparément (utilisation XML possible)
- √ Avantage : réutilisez une application avec aspects techniques différents sans toucher au code !

Objectifs de la norme J2EE-EJB
Enterprise Java Beans

√ Conçue par Sun Microsystems (1998)

Objectifs de la norme J2EE-EJB
Enterprise Java Beans

- √ Conçue par Sun Microsystems (1998)
- √ Modèle différent des Java Beans !

Objectifs de la norme J2EE-EJB
Enterprise Java Beans

- √ Conçue par Sun Microsystems (1998)
- √ Modèle différent des Java Beans !
- √ Objectifs :
  - ▶ Fournir aux applications d'entreprise une architecture normalisée de composants logiciels répartis

Objectifs de la norme J2EE-EJB
Enterprise Java Beans

- √ Conçue par Sun Microsystems (1998)
- √ Modèle différent des Java Beans !
- √ Objectifs :
  - ▶ Fournir aux applications d'entreprise une architecture normalisée de composants logiciels répartis
  - ▶ Etre multi-plateforme (pas de recompilation Java)

Objectifs de la norme J2EE-EJB
Enterprise Java Beans

- √ Conçue par Sun Microsystems (1998)
- √ Modèle différent des Java Beans !
- √ Objectifs :
  - ▶ Fournir aux applications d'entreprise une architecture normalisée de composants logiciels répartis
  - ▶ Etre multi-plateforme (pas de recompilation Java)
  - ▶ Prise en compte de 3 aspects techniques : persistance (couplage bases de données), sécurité et transactions

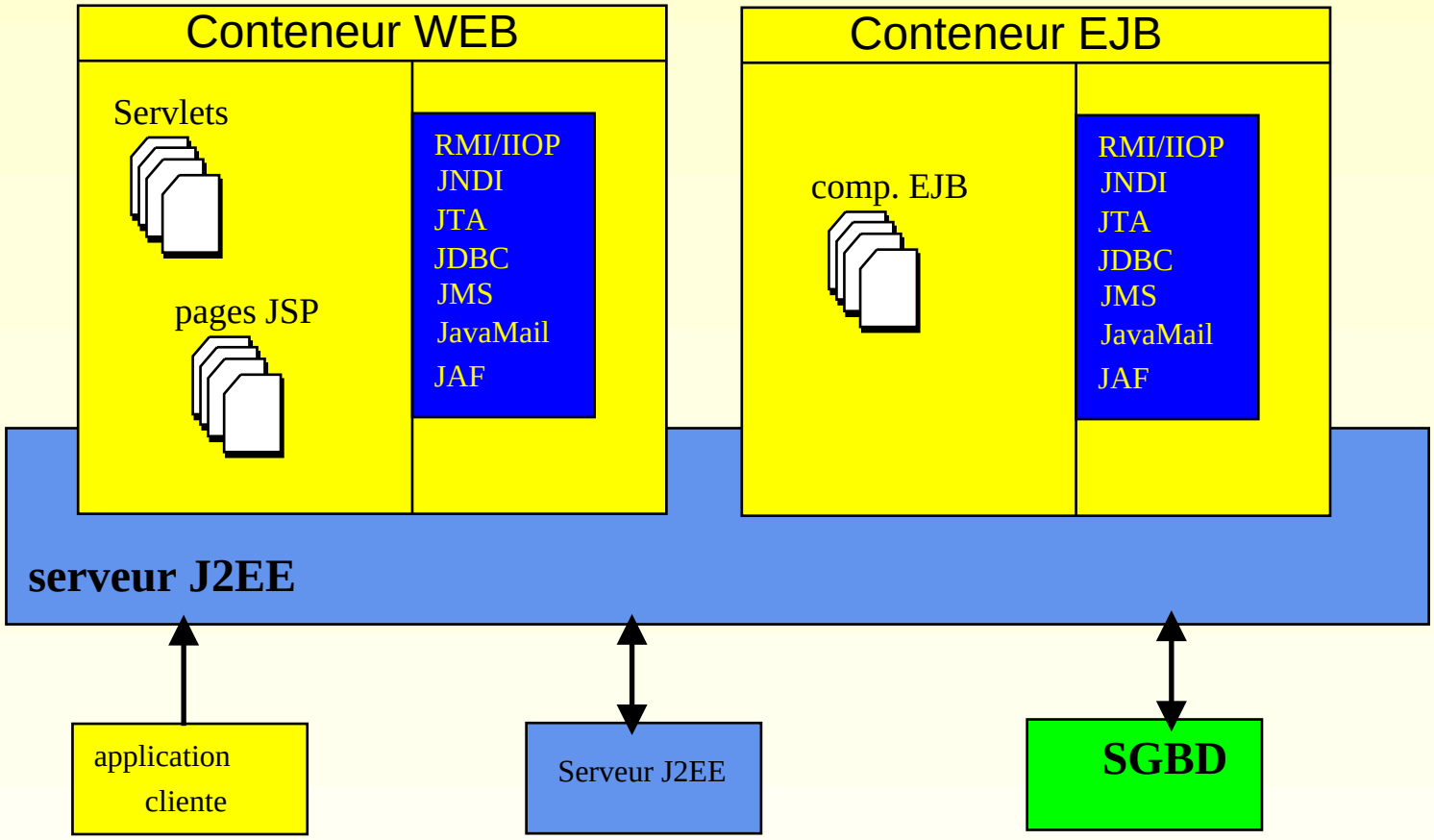
Objectifs de la norme J2EE-EJB
Enterprise Java Beans

- √ Conçue par Sun Microsystems (1998)
- √ Modèle différent des Java Beans !
- √ Objectifs :
  - ▶ Fournir aux applications d'entreprise une architecture normalisée de composants logiciels répartis
  - ▶ Etre multi-plateforme (pas de recompilation Java)
  - ▶ Prise en compte de 3 aspects techniques : persistance (couplage bases de données), sécurité et transactions
  - ▶ Support réseau assuré par protocole RMI/IIOP

Objectifs de la norme J2EE-EJB
Enterprise Java Beans

- √ Conçue par Sun Microsystems (1998)
- √ Modèle différent des Java Beans !
- √ Objectifs :
  - ▶ Fournir aux applications d'entreprise une architecture normalisée de composants logiciels répartis
  - ▶ Etre multi-plateforme (pas de recompilation Java)
  - ▶ Prise en compte de 3 aspects techniques : persistance (couplage bases de données), sécurité et transactions
  - ▶ Support réseau assuré par protocole RMI/IIOP
  - ▶ Adaptée aux architectures 3-tiers

L'architecture J2EE



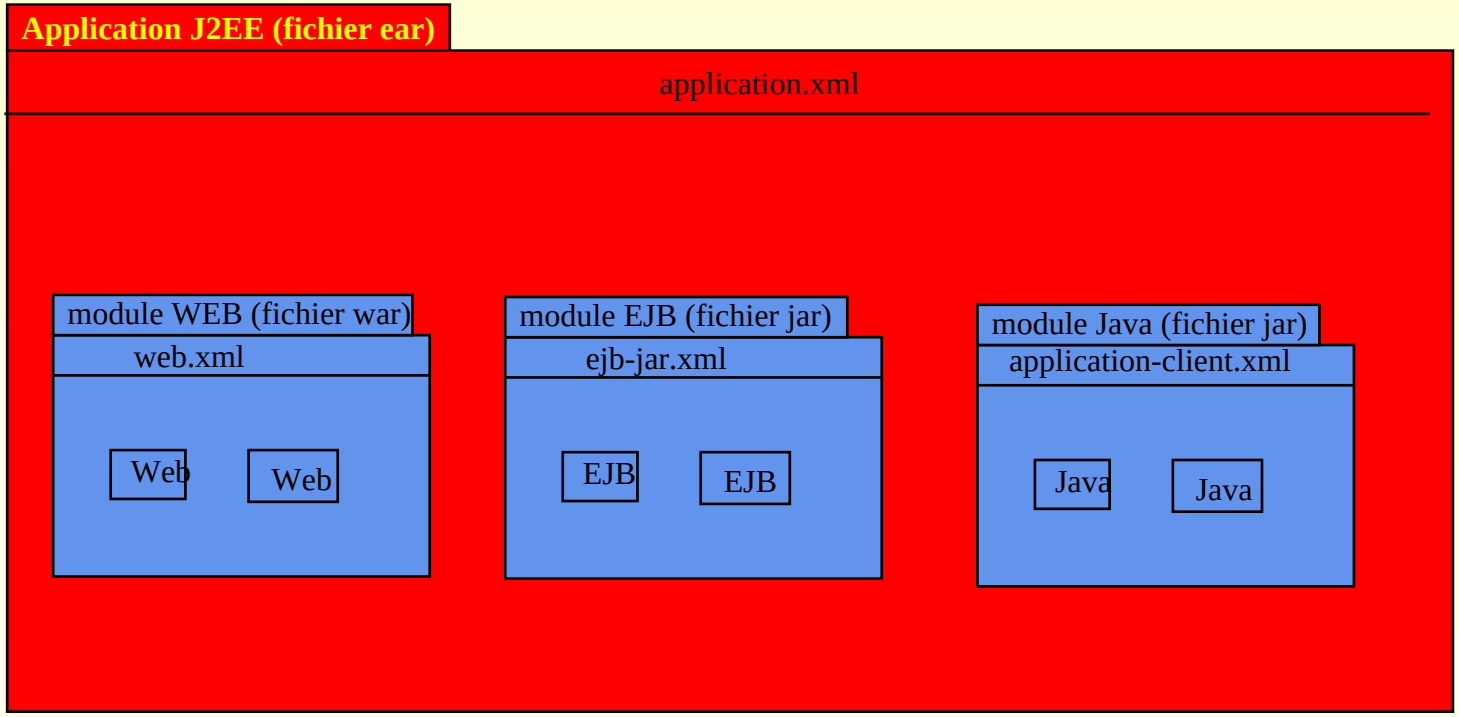
Une multitude de serveur d'applications EJB

- √ Sun J2EE SDK
- √ Gemstone/J (SGBDOO)
- √ Oracle (SGBDR, modèle objet-relationnel)
- √ Inprise, JBoss, JONAS,. . .

Une multitude de serveur d'applications EJB

- √ Sun J2EE SDK
- √ Gemstone/J (SGBDOO)
- √ Oracle (SGBDR, modèle objet-relationnel)
- √ Inprise, JBoss, JONAS,. . .
- √ L'écriture d'une application EJB est indépendante d'un serveur EJB

Une application J2EE (1/3)



Une application J2EE (2/3)

```
monAppli/
  META-INF/
    application.xml
  mesComposantsWeb.war
  mesComposantsEJB.jar

jar cvf monAppli.ear
    monAppli/*
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC '-//Sun Microsystems,
Inc.//DTD J2EE Application 1.3//EN'
'http://java.sun.com/dtd/application_1_3.dtd'>
<application>
  <display-name>monAppli</display-name>
  <description>un exemple</description>
  <module>
    <web>
      <web-uri>mesComposantsWeb.war</web-uri>
      <context-root>repertoireRacine</context-root>
    </web>
  </module>
  <module>  <ejb>mesComposantsEJB.jar</ejb>
  </module>
</application>
```

Une application J2EE (3/3)

√ ne peut être déployée que sur un seul serveur

Une application J2EE (3/3)

- √ ne peut être déployée que sur un seul serveur
- √ Une application peut communiquer avec d'autres applications situées sur le même serveur ou pas

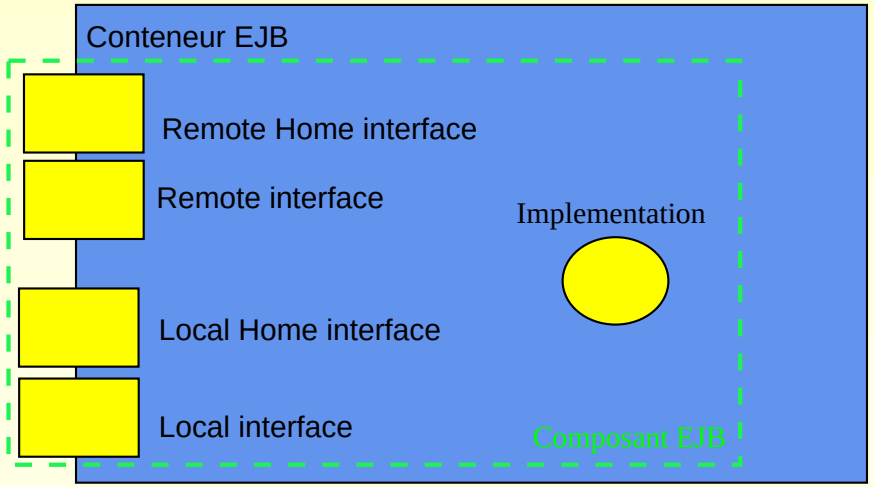
Une application J2EE (3/3)

- √ ne peut être déployée que sur un seul serveur
- √ Une application peut communiquer avec d'autres applications situées sur le même serveur ou pas
- √ On distingue (pour des raisons de performances) :
  - ▶ Les accès locaux : composants EJB ou Web situés dans une même application

Une application J2EE (3/3)

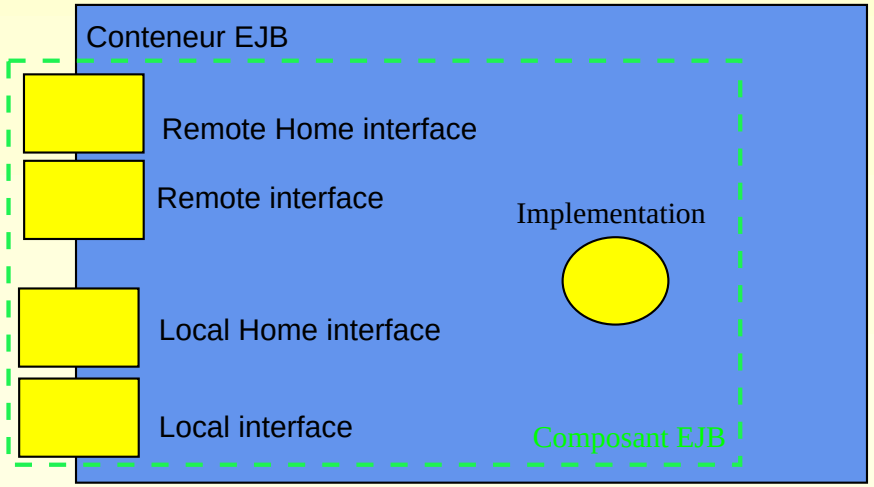
- √ ne peut être déployée que sur un seul serveur
- √ Une application peut communiquer avec d'autres applications situées sur le même serveur ou pas
- √ On distingue (pour des raisons de performances) :
  - ▶ Les accès locaux : composants EJB ou Web situés dans une même application
  - ▶ Les accès distants : composants entre plusieurs applications (protocole RMI/IIOP)

Le point de vue du client



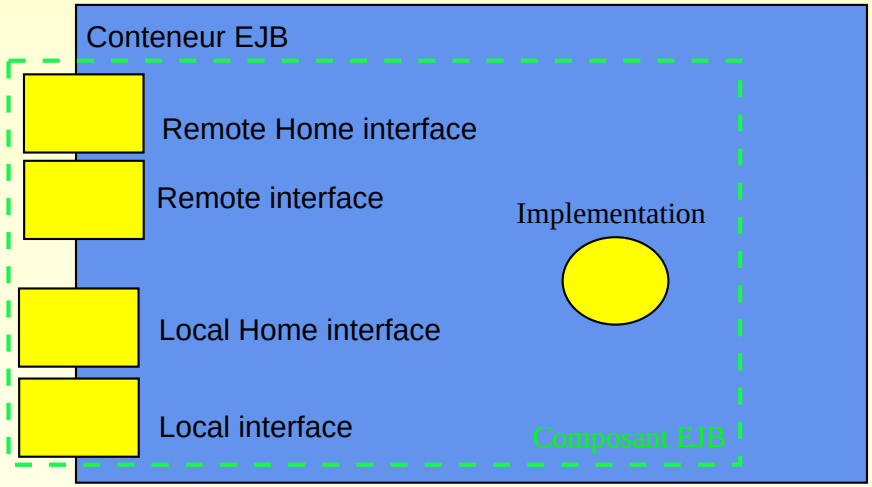
√ Seules les fabriques et interfaces sont connues par le ou les clients

Le point de vue du client



- √ Seules les fabriques et interfaces sont connues par le ou les clients
- √ Selon les cas, un seul des modes d'accès est requis (local ou distant).

Le point de vue du client

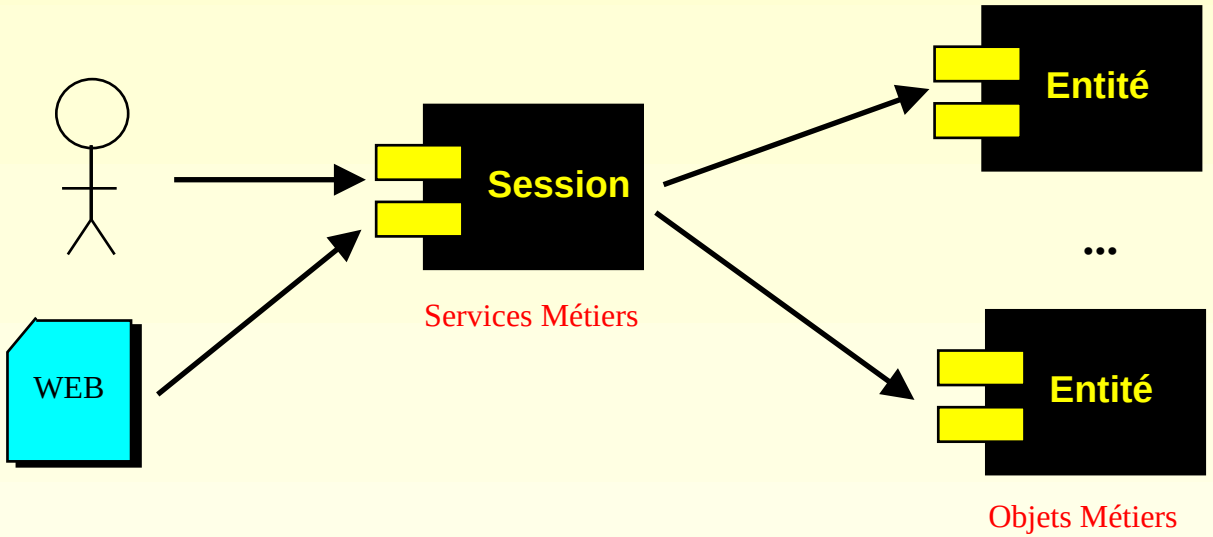


- √ Seules les fabriques et interfaces sont connues par le ou les clients
- √ Selon les cas, un seul des modes d'accès est requis (local ou distant).
- √ La norme spécifie des règles d'interaction entre l'interface distante, la fabrique et l'implémentation.

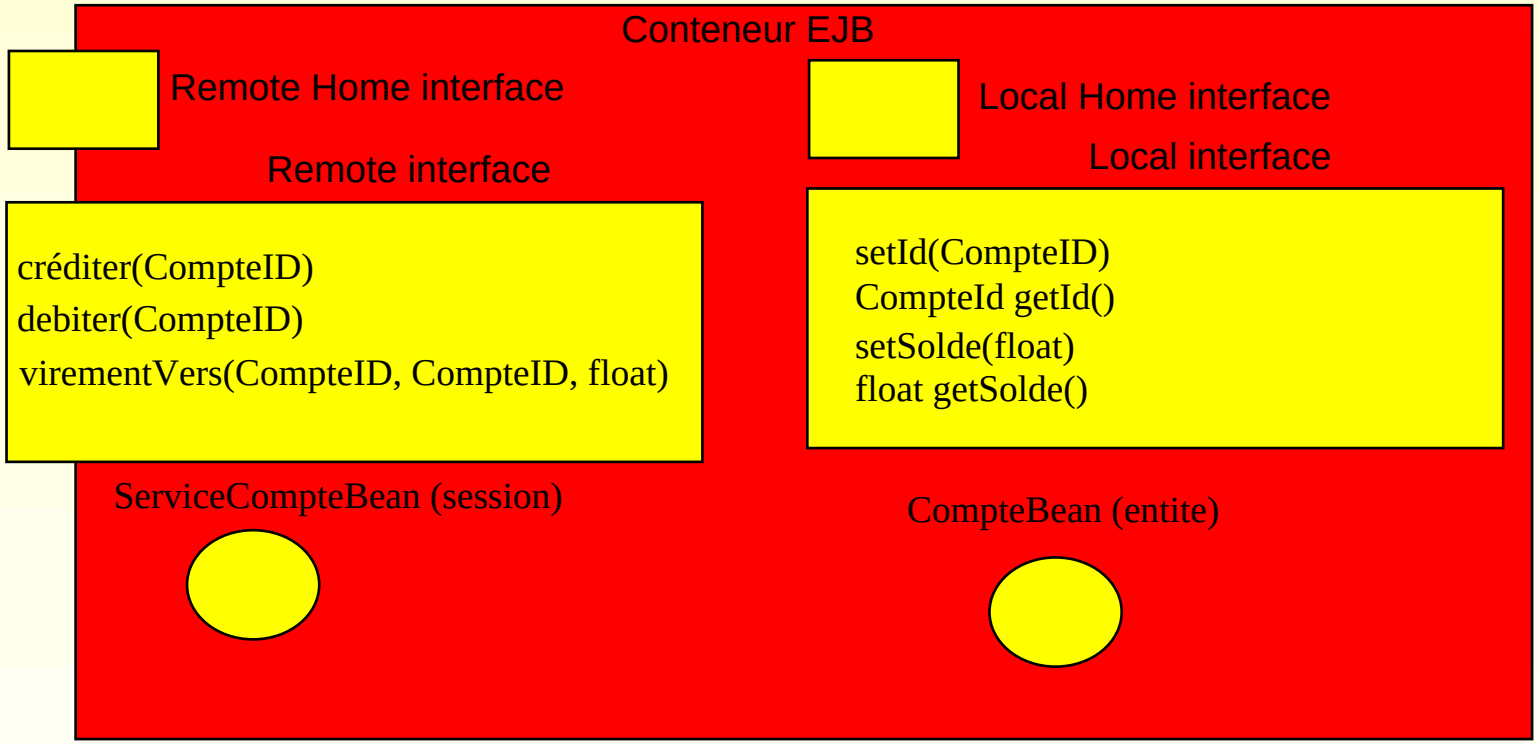
Les types de composants EJB

- √ Les composants Session
  - Assurent les services métiers
  - interface avec les composants web ou clients
  - Avec ou sans état transactionnel (StateLess, StateFul)
- √ Les composants Entités
  - Représentent les objets métiers
  - Interface avec les bases de données
  - Persistance gérée ou pas par le conteneur (CMP, BMP)
- √ Les composants "Message Driven" (norme EJB 2.0)
  - Gestion asynchrone, évènementielle

Conception et Structuration des composants (1/3)



Conception et Structuration des composants (2/3)



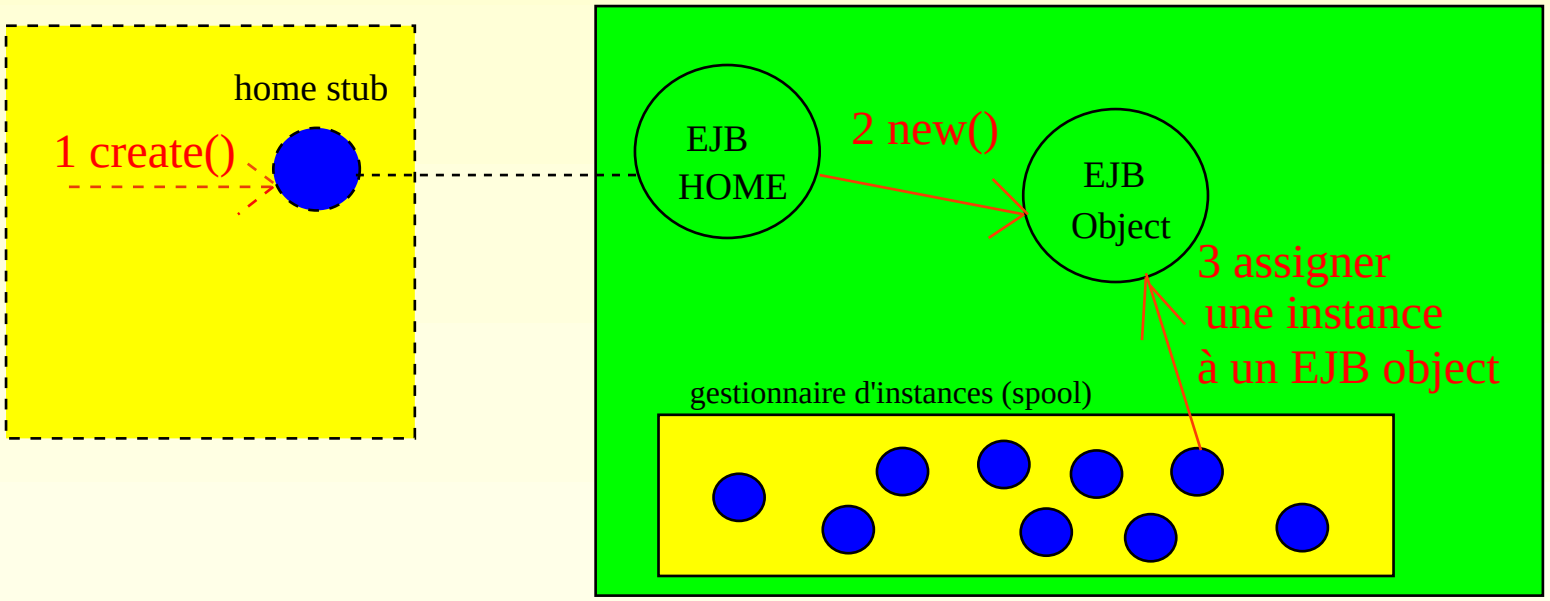
Conception et Structuration des composants (3/3)

- √ Use Case et diagrammes de séquences définissent les composants sessions
- √ Structure du système d'information définit les composants entité
- √ Souhait de structuration entre composants, pas une obligation !
- √ Notion d'interfaces locales et distantes (performances)

Gestion des composants par le serveur

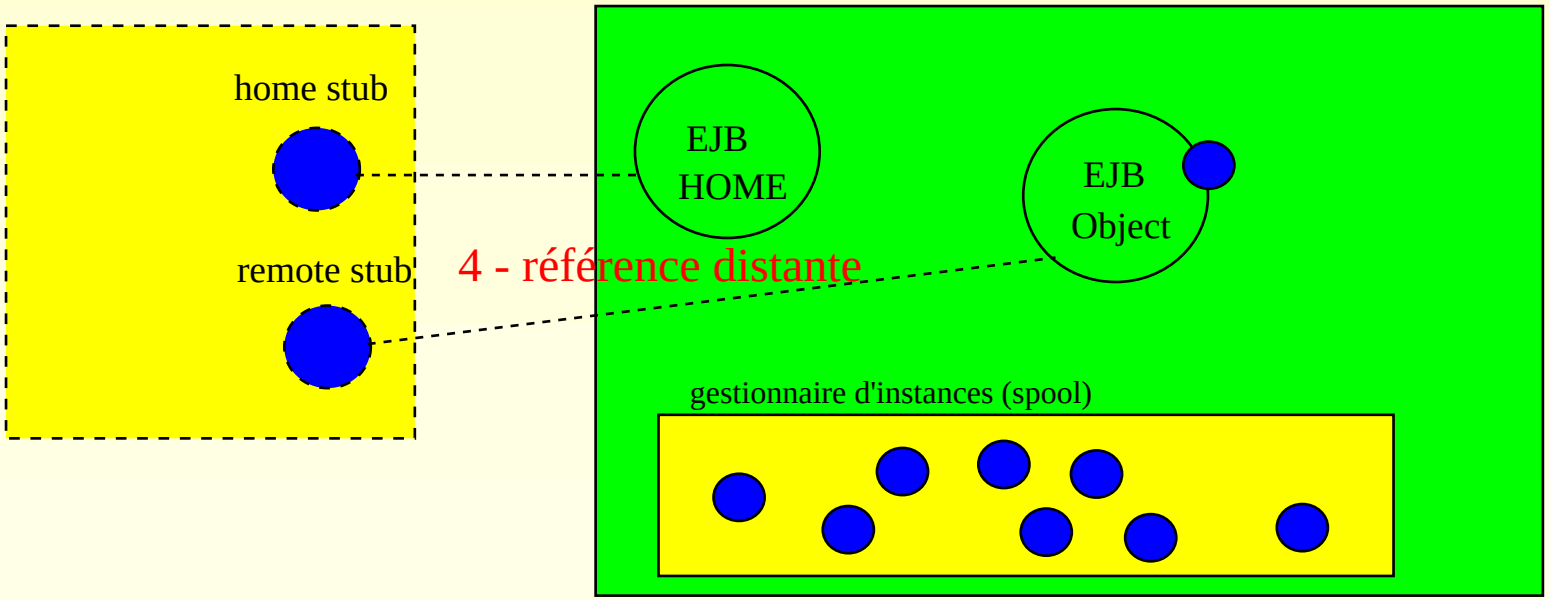
- √ Objectif : le serveur doit pouvoir gérer des milliers, millions d'objets distribués simultanément !
- √ Le type du composant (entité, session avec état, session sans état) implique un cycle de vie différent, un mode de fonctionnement particulier
- √ Le client n'accède jamais directement au composant, cela autorise des mécanismes de swap selon le composant.

Cycle d'exécution d'un composant entité EJB (1/2)



√ Plus rapide de gérer des instances que de multiplier les connexions à la base de données.

Cycle d'exécution d'un composant entité EJB (2/2)

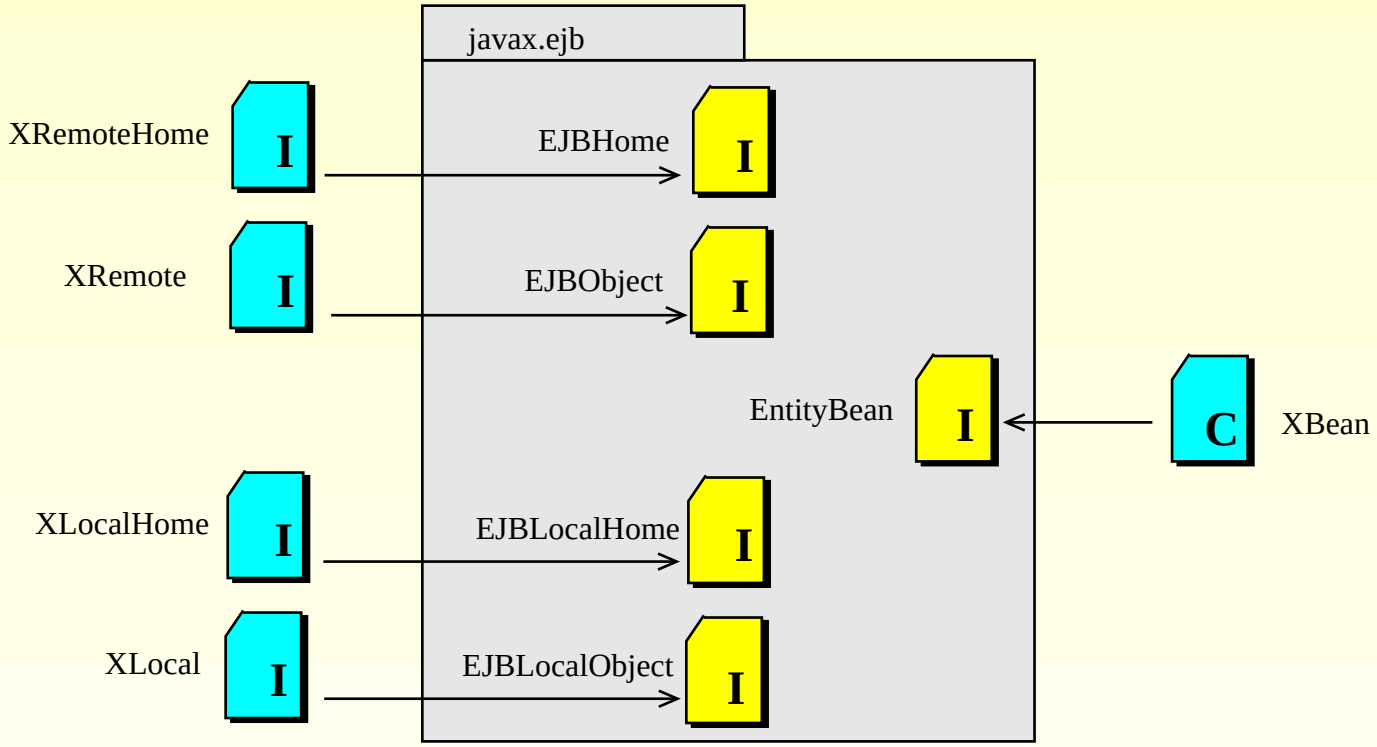


√ Mécanismes de mise à jour dans la base (load, store)

Le composant Entité

- √ Représente les objets métiers
- √ Lié aux bases de données :
  - ▶ Dispose d'une clé primaire (SGBDR)
  - ▶ Langage EJB-QL
- √ Persistance gérée ou pas par le conteneur (CMP ou CMB)
- √ Notion de schéma de persistance abstrait
  (EJB2.0, relations, intégrité référentielle)

Le framework EJB (entité)



L'entité Bean Compte (1/6)

√ L'interface distante :

```
package exempleEntite ;

import java.rmi.RemoteException;
import javax.ejb.EJBObject ;

public interface CompteRemote extends EJBObject {
    public String getNom() throws RemoteException ;
    public void setSolde(double solde) throws RemoteException ;
    public double getSolde() throws RemoteException ;
}
```

L'entité Bean Compte (2/6)

- L'interface locale :

```
package exempleEntite ;

import javax.ejb.EJBLocalObject ;

public interface CompteLocal extends EJBLocalObject {
    public String getNom() ;
    public void setSolde(double solde) ;
    public double getSolde() ;
}
```

- Un composant peut avoir une interface distante **et/ou** une interface locale.
- Les interfaces peuvent être différentes.

Règles de construction

√ Pour l'interface distante :

- L'interface doit hériter de `javax.ejb.EJBObject`
- Chaque méthode doit prendre en compte l'exception `java.rmi.RemoteException`

Règles de construction

√ Pour l'interface distante :
- L'interface doit hériter de `javax.ejb.EJBObject`
- Chaque méthode doit prendre en compte l'exception `java.rmi.RemoteException`

√ Pour l'interface locale :
- L'interface doit hériter de `javax.ejb.EJBLocalObject`

L'entité Bean Compte (3/6)

√ L'objet Fabrique distant :

```
package exempleEntite ;
import java.rmi.RemoteException ;
import javax.ejb.CreateException ;
import javax.ejb.FinderException ;
import javax.ejb.EJBHome ;

public interface CompteRemoteHome extends EJBHome {
    public CompteRemote create(String nom, double solde)
        throws CreateException, RemoteException ;
    public CompteRemote findByPrimaryKey(String nom)
        throws FinderException, RemoteException ;
}
```

L'entité Bean Compte (4/6)

√ L'objet Fabrique local :

```
package exempleEntite ;

import javax.ejb.CreateException ;
import javax.ejb.FinderException ;
import javax.ejb.EJBLocalHome ;

public interface CompteLocalHome extends EJBLocalHome {
    public CompteLocal create(String nom, double solde)
        throws CreateException ;
    public CompteLocal findByPrimaryKey(String nom)
        throws FinderException ;
}
```

Règles de construction de l'interface Home distante

√ Etendre l'interface `javax.ejb.EJBHome`

Règles de construction de l'interface Home distante

- √ Etendre l'interface `javax.ejb.EJBHome`
- √ Définir **au moins une méthode** `create<nom>`
  - ▶ Peut contenir un ou plusieurs paramètres
  - ▶ L'un des paramètres (un objet) définit la clé
  - ▶ Prendre en compte l'exception `javax.ejb.CreateException`
- √ Prendre en compte l'exception `java.rmi.RemoteException`
- √ Possibilité de définir des méthodes d'accès et des "méthodes home"

Les méthodes d'accès des fabriques d'entités

- √ Conventions de nommage
- √ Recherche d'un élément :

  `ObjectInterface findByPrimaryKey(ObjectKey)`
- √ Recherche d'élément par critère :

  `Collection find<nomExplicite>(...)`

  Exemple : `Enumeration findBySolde(double solde)`

Les méthodes home

√ Des méthodes non liés à des instances

Les méthodes home

- √ Des méthodes non liés à des instances
- √ Equivalent à des méthodes statiques

Les méthodes home

- √ Des méthodes non liés à des instances
- √ Equivalent à des méthodes statiques
- √ Règles : le nom de ces méthodes ne doit pas commencer par `create`, `find` ou `remove`
- √ Exemple : `int getNombrePersonnes()`

Supprimer un élément

√ Méthode `remove` définie au niveau de `EJBHome` et `EJBLocalHome` :

```
public interface EJBHome extends Remote {
  ...
  void remove(Object primaryKey)
    throws RemoteException, RemoveException;
}
```

L'entité bean Compte
Bean Managed Persistence (1/2)

- √ La classe doit implémenter `javax.ejb.EntityBean`
  - ▶ Gérer la persistance (JDBC, SQLJ, . . .)
- √ Doit être déclarée `public` et non `abstract`
- √ Doit contenir une fonction constructeur sans paramètre

L'entité bean Compte
Bean Managed Persistence (2/2)

√ Doit implémenter des méthodes liées à la fabrique
ces méthodes sont publiques et non statiques

`type create<nom>(...)`	`type_key ejbCreate<nom>(...)`
	`void ejbpostCreate(...)`
`type find<nom>(...)`	`type ejbFind<nom>(...)`
`type <nom>(...)`	`type ejbHome<nom>(...)`

L'entité Bean Compte (5/6)
Version Container Management Persistence

```
package exempleEntite ;
import java.rmi.RemoteException ;
import javax.ejb.CreateException ;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;

public abstract class CompteBean implements EntityBean {
    // définitions des attributs
    public abstract String getNom() ;
    public abstract void setNom(String value) ;
    public abstract double getSolde() ;
    public abstract void setSolde(double solde) ;
```

L'entité Bean Compte (6/6)

```
    public String ejbCreate(String nom, double solde)
      throws CreateException {
      this.setNom(nom) ;  this.setSolde(solde) ;
      return nom ;
    }
    public void ejbPostCreate(String nom, double solde)
      throws CreateException {}
    public void setEntityContext(EntityContext ctx) {}
    public void unsetEntityContext() {}
    public void ejbRemove() {}
    public void ejbActivate() {} public void ejbPassivate() {}
    public void ejbLoad() {} public void ejbStore() {}
}
```

Règles de construction de l'entité Bean (CMP)

√ La classe est déclarée `public` et `abstract`

Règles de construction de l'entité Bean (CMP)

- √ La classe est déclarée `public` et `abstract`
- √ La classe doit implémenter l'interface `javax.ejb.EntityBean`

Règles de construction de l'entité Bean (CMP)

- √ La classe est déclarée `public` et `abstract`
- √ La classe doit implémenter l'interface `javax.ejb.EntityBean`
- √ Des fonctions accesseurs abstraites sont définies pour chaque attribut persistant (set et get)

Règles de construction de l'entité Bean (CMP)

- √ La classe est déclarée `public` et `abstract`
- √ La classe doit implémenter l'interface `javax.ejb.EntityBean`
- √ Des fonctions accesseurs abstraites sont définies pour chaque attribut persistant (set et get)
- √ Les méthodes (autres que définitions d'attributs) définies dans les interfaces distantes et locales doivent être implémentées.

Règles de construction de l'entité Bean (CMP)

- √ La classe est déclarée `public` et `abstract`
- √ La classe doit implémenter l'interface `javax.ejb.EntityBean`
- √ Des fonctions accesseurs abstraites sont définies pour chaque attribut persistant (set et get)
- √ Les méthodes (autres que définitions d'attributs) définies dans les interfaces distantes et locales doivent être implémentées.
- √ Les méthodes `ejbCreate<nom>(...)` retourne le type de la clé primaire
- √ Implémenter les méthodes home `ejbHome<nom>`

Les méthodes d'une entité Bean

- √ `setEntityContext` : c'est la passerelle du Bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité,. . .

Les méthodes d'une entité Bean

- √ `setEntityContext` : c'est la passerelle du Bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité,. . .
- √ `ejbPassivate` : méthode automatiquement appelée lorsqu'il y a trop de beans instanciés. Il faut donc stocker l'état du bean (ejbStore)

Les méthodes d'une entité Bean

- √ `setEntityContext` : c'est la passerelle du Bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité,. . .
- √ `ejbPassivate` : méthode automatiquement appelée lorsqu'il y a trop de beans instanciés. Il faut donc stocker l'état du bean (ejbStore)
- √ `ejbActivate` méthode pour restaurer les Beans

Les méthodes d'une entité Bean

- √ `setEntityContext` : c'est la passerelle du Bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité,. . .
- √ `ejbPassivate` : méthode automatiquement appelée lorsqu'il y a trop de beans instanciés. Il faut donc stocker l'état du bean (ejbStore)
- √ `ejbActivate` méthode pour restaurer les Beans
- √ `ejbRemove` : suppression d'un bean

Les méthodes d'une entité Bean

- √ `setEntityContext` : c'est la passerelle du Bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité,. . .
- √ `ejbPassivate` : méthode automatiquement appelée lorsqu'il y a trop de beans instanciés. Il faut donc stocker l'état du bean (ejbStore)
- √ `ejbActivate` méthode pour restaurer les Beans
- √ `ejbRemove` : suppression d'un bean
- √ `ejbLoad` et `ejbStore` relation avec la base de données

Quelques remarques

- √ Le code Java n'est pas suffisant pour décrire le composant :
  - ▶ Exemple : comment établir le lien entre une interface distante et l'objet d'implémentation ?
  - ▶ Exemple : BMP ou CMP ?
  - ▶ Solution : disposer d'un descripteur du composant
- √ La compilation du composant ne fait pas tout :
  - ▶ Exemple : manque méthode create
  - ▶ Exemple : méthode de l'interface distante non implémentée
  - ▶ Solution : disposer d'outils de vérification.

Structuration physique du composant

- √ Un fichier archive (.jar) contient :
  - ▶ Les classes du composants (.class)
  - ▶ Des descripteurs de déploiement (.xml)
- √ Le composant jar est entièrement normalisé (ou presque . . .)
- √ Exploitable pour tout serveur J2EE

Exemple Structure Jar

```
mesComposantsEJB/
  META-INF/
    ejb-jar.xml
    jboss.xml
    jboss-cmpjdbc.xml
  exempleEntite/
    CompteBean.class
    CompteRemoteHome.class
    CompteRemote.class
    CompteLocalHome.class
    CompteLocal.class
```

Descriptions de déploiement jar

- √ Désignation symbolique du jar,
- √ Identification des EJB (noms symboliques, type des EJB)
- √ Structuration physique (désignation des .class)
- √ Informations sur la sécurité
- √ Informations sur les transactions
- √ Information persistance (EJB entité)
- √ Adresses JNDI, référencement des entités

Déploiement : exemple extrait structure EJB

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
<ejb-jar>
  <enterprise-beans>
    <entity >
         <display-name>Entite compte</display-name>
         <ejb-name>CompteBean</ejb-name>
         <home>exempleEntite.CompteRemoteHome</home>
         <remote>exempleEntite.CompteRemote</remote>
         <local-home>exempleEntite.CompteLocalHome</local-home>
         <local>exempleEntite.CompteLocal</local>
         <ejb-class>exempleEntite.CompteBean</ejb-class>
         ...
    </entity>
  </enterprise-beans>
</ejb-jar>
```

Déploiement : la sécurité

- √ Notion de rôle (ou profil d'utilisation)
- √ Pour chaque rôle, il faut préciser la ou les méthodes utilisables.
- √ Des utilisateurs (compte et mot de passe) peuvent être liés à plusieurs rôles
- √ Le descripteur de déploiement inclue les informations liées à la sécurité

Déploiement : les transactions (1/4)

√ Egalement un service déclaratif (descripteur de déploiement), pas de code Java à produire

Déploiement : les transactions (1/4)

- √ Egalement un service déclaratif (descripteur de déploiement), pas de code Java à produire
- √ Assimiler le fonctionnement des transactions pour les décrire.
- √ Les propriétés ACID des transactions :
  - Atomique : les opérations du même ensemble doivent s'exécuter ou échouer de manière globale.
  - Cohérent : Avant et après une transaction (réussie ou pas), l'état de la base doit être cohérente.
  - Isolé : Le programmeur n'a pas à se soucier des autres activités du système.
  - Durable : lors d'une transaction réussie, les résultats sont reportés dans la base de données.

Déploiement : les transactions (2/4)

√ Six options (applicable à chaque méthode) :

- **NotSupported** : non pris en charge.
  La spécification EJB laisse au conteneur le choix de l'accès aux ressources en cas de transaction non définie.
- **Suppports** : prend en charge.
  Si il existe un contexte de transaction, cette option est équivalente à `Required`, sinon `Not Supported`
- **RequiresNew** : nouvelle transaction requise.
  Création d'une nouvelle transaction. Imbrication possible de transactions.

Déploiement : les transactions (3/4)

- √ **Required** : transaction requise.
  Si pas de transaction courante, une transaction est créée.
- √ **Mandatory** : obligatoire.
  Si pas de transaction courante, une exception intervient.
- √ **Never** : jamais.
  Si il exixte une transaction courante, une exception intervient.

Déploiement : exemple extrait transactions

```
<container-transaction >
      <method >
         <ejb-name>Compte</ejb-name>
         <method-intf>Local</method-intf>
         <method-name>getNom</method-name>
         <method-params>
         </method-params>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
```

Déploiement : La persistance (1/3)

- √ Descriptions des attributs persistants et clé primaire
  - description XML standardisé
- √ Description lien avec bases de données distantes
  - Fichier xml spécifique plate-forme supplémentaire
  - Exemple : META-INF/jbosscmp-jdbc.xml
  - Un EJB correspond à une table
  - Un attribut d'un EJB à un élément de cette table
  - Description de clé primaire, contraintes
  - Création ou non des tables lors du déploiement, etc

Déploiement : exemple extrait persistance (2/3)

```
<entity >    ...
        <persistence-type>Container</persistence-type>
        <prim-key-class>java.lang.String</prim-key-class>
        <reentrant>False</reentrant>
        <cmp-version>2.x</cmp-version>
        <abstract-schema-name>Compte</abstract-schema-name>
        <cmp-field >
          <field-name>nom</field-name>
        </cmp-field>
        <cmp-field >
          <field-name>solde</field-name>
        </cmp-field>
        <primkey-field>nom</primkey-field>  ...
```

Déploiement : exemple extrait persistance (3/3)

```
<jbosscmp-jdbc>  ...
   <enterprise-beans>
      <entity>
         <ejb-name>CompteBean</ejb-name>
         <create-table>true</create-table> <remove-table>true</remove-table>
         <pk-constraint>true</pk-constraint>
         <table-name>compte_table</table-name>
         <cmp-field>
            <field-name>nom</field-name> <column-name>nom</column-name>
            <jdbc-type>VARCHAR</jdbc-type>
            <sql-type>char(32)</sql-type>
        </cmp-field> ...
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Déploiement : Accéder aux EJB (1/3)

√ API JNDI : Java Naming Directory Interface

- Serveur de noms :enregistrement, recherche, . . .
- API unique reconnaissant plusieurs protocoles :IMAP, DNS, NIS, LDAP,. . .

Déploiement : Accéder aux EJB (1/3)

- √ API JNDI : Java Naming Directory Interface
  - ▶ Serveur de noms :enregistrement, recherche, . . .
  - ▶ API unique reconnaissant plusieurs protocoles :IMAP, DNS, NIS, LDAP,. . .
- √ plate-forme JBOSS : fichier xml supplémentaire
  fichier : META-INF/jboss.xml
- √ L'enregistrement JNDI se fait au niveau du déploiement
- √ Décrire quel(s) EJB(s) utilisent d'autre(s)EJB(s) et par quel moyen (accès distant ou local

Déploiement : exemple JBOSS JNDI (2/3)

```
<jboss>
   <enterprise-beans>
      <entity>
         <ejb-name>CompteBean</ejb-name>
         <jndi-name>ocaron/Compte</jndi-name>
         <local-jndi-name>ejb/ocaron/Compte</local-jndi-name>
      </entity>
   </enterprise-beans> ...
</jboss>
```

Accès aux composants via JNDI (3/3)

√ Accès aux objets distants : (pas forcément même serveur J2EE) :

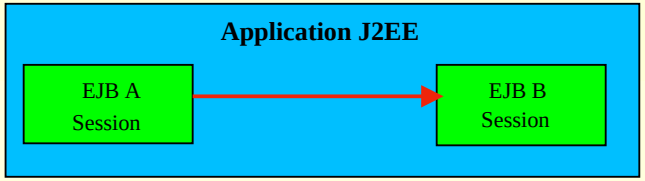
```
InitialContext ctx=new InitialContext() ;
Object obj= ctx.lookup("ocaron/Compte") ;
compteHome = (CompteRemoteHome)
        PortableRemoteObject.narrow(obj, compteRemoteHome.class) ;
```

√ Accès aux objets locaux : (dans le serveur/conteneur J2EE)
Contexte par défaut : `java :comp/env`

```
InitialContext ic = new InitialContext();
Object obj = ic.lookup("java:comp/env/ejb/ocaron/Compte");
CompteLocalHome home = (CompteLocalHome) obj ;
```

Déploiement : référence d'EJB (1/2)

√ Un EJB accède à un EJB de la même application J2EE :



Soit accès distant (?)

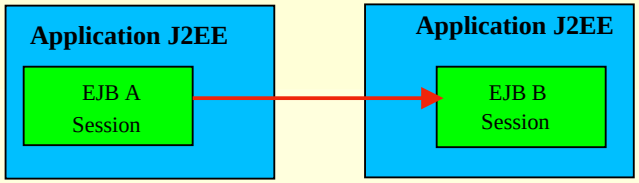
```
<ejb-name>A</ejb-name> ...
<ejb-ref>
 <ejb-ref-name>RefB</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <home>BPack.BRemoteHome</home>
 <remote>BPack.BRemote</remote>
 <ejb-link>B</ejb-link>
</ejb-ref>
```

Soit accès local :

```
<ejb-name>A</ejb-name> ...
<ejb-local-ref >
 <ejb-ref-name>ejb/RefB</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <local-home>BPack.BLocalHome</local-home>
 <local>BPack.BLocal</local>
 <ejb-link>B</ejb-link><!--réf ejb-name-->
</ejb-local-ref>
```

Déploiement : référence d'EJB (2/2)

√ Un EJB accède à un EJB d'une autre application J2EE :



Description de la réf. (ejb-jar.xml)

```
<ejb-name>A</ejb-name> ...
 <ejb-ref>
  <ejb-ref-name>ejb/RefB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>BPack.BRemoteHome</home>
  <remote>BPack.BRemote</remote>
 </ejb-ref>
```

lien avec JNDI distant (jboss.xml)

```
<ejb-ref>
 <ejb-ref-name>ejb/RefB</ejb-ref-name>
 <jndi-name>RefB</jndi-name>
</ejb-ref>
```

Disposer de `BPack.BRemoteHome.class` et `BPack.BRemote.class`

Une application Cliente (1/3)

```
import javax.naming.InitialContext;
import java.rmi.RemoteException ;
import javax.rmi.PortableRemoteObject;
import javax.naming.NamingException;
import javax.ejb.CreateException;
import javax.ejb.FinderException ;
import exempleEntite.* ;

public class CompteMain {
    public static void main(String args[]) {
        CompteRemoteHome compteHome=null ;
        CompteRemote compte ;
        String nomCompte=args[0] ;
        InitialContext ctx ;
```

Une application Cliente (2/3)

```
try {
   ctx=new InitialContext() ;
   Object obj= ctx.lookup("ocaron/Compte") ;
   compteHome=(CompteRemoteHome)
     PortableRemoteObject.narrow(obj,CompteRemoteHome.class) ;
   try {
      compte=compteHome.findByPrimaryKey(nomCompte) ;
   } catch(FinderException e1) {
     try {
      compte=compteHome.create(nomCompte,2000.0) ;
     } catch(CreateException e2) {
       System.err.println("erreur creation") ; return ;
     }
   }
```

Une application Cliente (3/3)

```
    compte.setSolde(compte.getSolde()+200.0) ;
    System.out.println("le solde de "+compte.getNom()
                       +" est "+compte.getSolde()) ;
} catch(NamingException e3) {
    System.err.println("Main:Erreur Naming:comptes") ;
} catch(ArrayIndexOutOfBoundsException e) {
    System.err.println("Erreur usage: java CompteMain nomCompte") ;
} catch(RemoteException e4) {
    System.err.println("Erreur reseau") ;
    e4.printStackTrace() ;
}}}
```

Exécution avec plate-forme JBOSS (1/2)

- √ Le classpath doit contenir les souches réseaux clientes ainsi que les classes J2EE
- √ Adaptation possible à la plate-forme utilisée
- √ localisation serveur JNDI

```
java  -Djava.naming.factory.initial="org.jnp.interfaces.NamingContextFactory"
      -Djava.naming.factory.url.pkgs="org.jboss.naming:org.jnp.interfaces"
      -Djava.naming.provider.url="weppes.priv.eudil.fr" CompteMain dupont
le solde de dupont est 2200.0
java  -Djava.naming.factory.initial="org.jnp.interfaces.NamingContextFactory"
      -Djava.naming.factory.url.pkgs="org.jboss.naming:org.jnp.interfaces"
      -Djava.naming.provider.url="weppes.priv.eudil.fr" CompteMain dupont
le solde de dupont est 2400.0
```

Exécution avec plate-forme JBOSS (2/2)
Version avec Ant

```
<target name="run-client" depends="deploy">
      <java classname="CompteMain">
        <sysproperty key="java.naming.factory.initial"
                     value="org.jnp.interfaces.NamingContextFactory" />
        <sysproperty key="java.naming.factory.url.pkgs"
                     value="org.jboss.naming:org.jnp.interfaces" />
        <sysproperty key="java.naming.provider.url"
                     value="weppes.priv.eudil.fr" />
        <arg value="dupont" />     <classpath refid="classpath" />
      </java>
</target>

ant run-client
le solde de dupont est 2600.0
```

Développement d'EJB

- √ Premier constat :
  - ▶ Empiriquement : 5 lignes de code Java correspond à 200 lignes de code XML.
- √ Solutions :
  - ▶ Environnement de développement générant tout ou partie des descripteurs XML
    JDeveloper (Oracle), JBuilder (Inprise), . . .

Développement d'EJB

- √ Premier constat :
  - ▶ Empiriquement : 5 lignes de code Java correspond à 200 lignes de code XML.
- √ Solutions :
  - ▶ Environnement de développement générant tout ou partie des descripteurs XML
    JDeveloper (Oracle), JBuilder (Inprise), . . .
  - ▶ Outils graphiques de déploiement
    deploytool (SUN)

Développement d'EJB

- √ Premier constat :
  - Empiriquement : 5 lignes de code Java correspond à 200 lignes de code XML.
- √ Solutions :
  - Environnement de développement générant tout ou partie des descripteurs XML
    JDeveloper (Oracle), JBuilder (Inprise), . . .
  - Outils graphiques de déploiement
    deploytool (SUN)
  - tags XDoclet

Présentation plate-forme J2EE Polytech'Lille

- √ Technologie :JBOSS 3.2.1, Ant 1.5, XDoclet 1.2 + éditeur texte de votre choix
- √ Principe de XDoclet : spécifier des commentaires normalisés dans le source afin de générer :
  - ▶ Tous les fichiers de déploiement
  - ▶ et même du code Java !
- √ Tache Ant associé

Exemple Développement J2EE (1/2)

```
/**
 * @ejb:bean  name="CompteBean"
 *            type="CMP"
 *            cmp-version="2.x"
 *            jndi-name="ocaron/Compte"
 *            local-jndi-name="ejb/ocaron/Compte"
 *            view-type="both"
 ...
 * @ejb:home
 *            remote-class="exempleEntite.CompteRemoteHome"
 *            local-class="exempleEntite.CompteLocalHome"
 ...
 */
public abstract class CompteBean implements EntityBean {
```

Exemple Développement J2EE (2/2)

```
<target name="ejbdoclet" depends="prepare">
  ...
  <taskdef name="ejbdoclet" classname="xdoclet.modules.ejb.EjbDocletTask"
           classpathref="xdocpath"
  />
  <ejbdoclet  destdir="${generated.java.dir}" ejbspec="2.0">
    <fileset dir="${src.dir}/ejb">
      <include name="**/*Bean.java" />
    </fileset>
    <remoteinterface />  <homeinterface/>
    <localinterface/> <localhomeinterface/>
    <entitypk /> <entitycmp />
    <deploymentdescriptor destdir="${build.dir}/ejb/META-INF"/>
    <jboss version="3.0" xmlencoding="UTF-8"  validatexml="true"
           destdir="${build.dir}/ejb/META-INF"/>
  </ejbdoclet>
</target>
```

Cycle de vie d'un composant session

- √ Composant session sans état (Stateless Session)
  - La durée de vie minimun d'un composant session est l'exécution d'une méthode !
  - Le serveur peut donc retirer de la mémoire le bean entre deux invocations de méthodes.
- √ Composant session avec état (Stateful Session)
  - Utilisation de la sérialisation pour sauver l'état du bean entre deux invocations de méthodes (si nécessaire)
  - Mécanismes d'activation et de passivation

Le composant Session

√ Les Session Bean

Un composant de type session effectue les opérations, transactionnelles ou non.

Il y a une session par programme client

les objets sessions disparaissent lorsque le serveur s'éteint

- Avec état transactionnel (STATEFUL)
- Sans état transactionnel (STATELESS)

Le Session Bean Salaire (1/6)

√ L'interface distante (Remote) :

```
package exempleSession ;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface SalaireRemote extends EJBObject {
    public double getAugmentation(double salaire)
      throws RemoteException ;
}
```

Le Session Bean Salaire (2/6)

- L'interface locale :

```
package exempleSession ;
import javax.ejb.EJBLocalObject;
public interface SalaireLocal extends EJBLocalObject {
    public double getAugmentation(double salaire) ;
}
```

- Même règles de construction qu'un composant entité

Le Session Bean Salaire (3/6)

√ L'objet fabrique distant (Home) :

```
package exempleSession ;

import java.rmi.RemoteException ;
import javax.ejb.CreateException ;
import javax.ejb.EJBHome ;

public interface SalaireRemoteHome extends EJBHome {
    public SalaireRemote create()
        throws CreateException, RemoteException ;
}
```

Le Session Bean Salaire (4/6)

√ L'objet fabrique local (Home) :

```
package exempleSession ;

import javax.ejb.CreateException ;
import javax.ejb.EJBLocalHome ;

public interface SalaireLocalHome extends EJBLocalHome {
    public SalaireLocal create()
        throws CreateException ;
}
```

Règles de construction de l'interface Home

√ Pour l'interface home distante :

- Etendre l'interface `javax.ejb.EJBHome`
- Définir **au moins** une méthode `create`
  peut contenir plusieurs paramètres (StateFul Session Bean)
  prendre en compte les exceptions `javax.ejb.CreateException` et `java.rmi.RemoteException`

Règles de construction de l'interface Home

√ Pour l'interface home distante :
- Etendre l'interface `javax.ejb.EJBHome`
- Définir **au moins** une méthode `create`
  peut contenir plusieurs paramètres (StateFul Session Bean)
  prendre en compte les exceptions `javax.ejb.CreateException` et `java.rmi.RemoteException`

√ Pour l'interface home locale :
- Etendre l'interface `javax.ejb.EJBLocalHome`
- Définir **au moins** une méthode `create`
  peut contenir plusieurs paramètres (StateFul Session Bean)
  prendre en compte l'exception `javax.ejb.CreateException`

Le Session Bean Salaire (3/4)

√ L'EJB :

```
package exempleSession ;

import java.rmi.RemoteException ;
import javax.ejb.SessionBean ;
import javax.ejb.SessionContext;

public class SalaireBean implements SessionBean {
    private SessionContext ctx ;

    public double getAugmentation(double salaire) {
        return salaire*.1 ; // 10 pour cent du salaire
    }
```

Le Session Bean Salaire (4/4)

```
    public void ejbCreate() { }
    public void setSessionContext(SessionContext ctx) {
      this.ctx=ctx ;}
    public void ejbRemove() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
}
```

Règles de construction du SessionBean

√ La classe doit implémenter l'interface `javax.ejb.SessionBean`

Règles de construction du SessionBean

- √ La classe doit implémenter l'interface `javax.ejb.SessionBean`
- √ La classe doit écrire les méthodes définies dans la ou les interfaces

Règles de construction du SessionBean

- √ La classe doit implémenter l'interface `javax.ejb.SessionBean`
- √ La classe doit écrire les méthodes définies dans la ou les interfaces
- √ Ecrire autant de méthodes `void ejbCreate(...)` qu'il y a de méthodes `create(...)` dans la ou les fabriques (home)

Règles de construction du SessionBean

- √ La classe doit implémenter l'interface `javax.ejb.SessionBean`
- √ La classe doit écrire les méthodes définies dans la ou les interfaces
- √ Ecrire autant de méthodes `void ejbCreate(...)` qu'il y a de méthodes `create(...)` dans la ou les fabriques (home)
- √ implémenter les méthodes de l'interface `SessionBean`

Les méthodes d'un SessionBean

- √ `setSessionContext` : c'est la passerelle du bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité, analyser le client. . .

Les méthodes d'un SessionBean

- √ `setSessionContext` : c'est la passerelle du bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité, analyser le client. . .
- √ `ejbCreate(...)` : initialise le session bean.

Les méthodes d'un SessionBean

- √ `setSessionContext` : c'est la passerelle du bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité, analyser le client. . .
- √ `ejbCreate(...)` : initialise le session bean.
- √ `ejbPassivate` : méthode automatiquement appelée lorsqu'il y a trop de beans instanciés. Il faut donc stocker l'état du bean (valable pour les STATEFUL Session Beans)

Les méthodes d'un SessionBean

- √ `setSessionContext` : c'est la passerelle du bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité, analyser le client. . .
- √ `ejbCreate(...)` : initialise le session bean.
- √ `ejbPassivate` : méthode automatiquement appelée lorsqu'il y a trop de beans instanciés. Il faut donc stocker l'état du bean (valable pour les STATEFUL Session Beans)
- √ `ejbActivate` méthode pour restaurer les Beans

Les méthodes d'un SessionBean

- √ `setSessionContext` : c'est la passerelle du bean avec son conteneur, permet de connaitre l'état transactionnel, l'état de sécurité, analyser le client. . .
- √ `ejbCreate(...)` : initialise le session bean.
- √ `ejbPassivate` : méthode automatiquement appelée lorsqu'il y a trop de beans instanciés. Il faut donc stocker l'état du bean (valable pour les STATEFUL Session Beans)
- √ `ejbActivate` méthode pour restaurer les Beans
- √ `ejbRemove` : suppression d'un bean

Une application Cliente (1/3)

```
package exempleSession ;

import javax.ejb.SessionContext;
import javax.naming.InitialContext;
import java.rmi.RemoteException ;
import javax.rmi.PortableRemoteObject;
import javax.naming.NamingException;
import javax.ejb.CreateException;

public class MainSalaire {
    public static void main(String args[]) {
        SalaireRemoteHome salaireHome ;
        SalaireRemote salaire ;
```

Une application Cliente (2/3)

```
try {
  InitialContext ctx=new InitialContext() ;
  Object obj= ctx.lookup("ocaron/Salaire") ;

  salaireHome = (SalaireRemoteHome)
    PortableRemoteObject.narrow(obj, SalaireRemoteHome.class) ;

  salaire=salaireHome.create() ;
  double leSalaire= Double.parseDouble(args[0]) ;
  double augmentation=salaire.getAugmentation(leSalaire) ;
  System.out.println("l'augmentation vaut : "+augmentation) ;
```

Une application Cliente (3/3)

```
    } catch(RemoteException e1) {
     System.err.println("Erreur reseau") ;
    } catch(ArrayIndexOutOfBoundsException e2) {
     System.err.println("Erreur usage: java MainSalaire salaire") ;
    } catch(NamingException e3){System.err.println("Erreur Naming");
    } catch(CreateException e4){System.err.println("Erreur Create");
    }
  }
}
```

Une véritable application 3-tiers

√ Ajout d'un composant client à l'application.

Une véritable application 3-tiers

- √ Ajout d'un composant client à l'application.
- √ Composant Web :
  - Déploiement automatique ! (http)
  - Fournir un alias (formulaire html et JSP)
  - Référencer l'EJB.
  - Utiliser les interfaces locales (charge réseau), même serveur

Le composant Web
le formulaire HTML

```
...
<body>
  <h1>Gestion des Salaires</h1>
  <form action="salaireAlias">
    Entrez votre salaire : <input type="text" name=salaire></input>
    <p>
    <input type="submit" value="valider">
  </form>
...
```

Le composant Web - Extrait code JSP

```
<%
    valSalaire=Double.parseDouble(request.getParameter("salaire")) ;
    try {
        InitialContext ic = new InitialContext();
        Object o = ic.lookup("java:comp/env/ejb/ocaron/Salaire");
        SalaireLocalHome home = (SalaireLocalHome) o ;
        SalaireLocal salaire = home.create() ;
        out.println("L'augmentation vaut : " +
        salaire.getAugmentation(valSalaire)) ;
    } catch(Exception e) {
        e.printStackTrace();
        out.println("Erreur augmentation : " + e.toString());
    }
%>
```

Référencement d'EJB dans composant web

- √ Même principe que pour référencement d'EJB entre eux.
- √ Stockage des interfaces distantes dans `WEB-INF/classes`
- √ Un composant Web peut accéder aux composants EJB de la même application par l'interface locale.
- √ Description des références dans web.xml :

```
<web-app>  ...
  <servlet-mapping> ...</servlet-mapping>
  <ejb-local-ref>
    <ejb-ref-name>ejb/ocaron/Salaire</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>ocaron.SalaireLocalHome</local-home>
    <local>ocaron.SalaireLocal</local>
    <ejb-link>Salaire</ejb-link>
</ejb-local-ref>
</web-app>
```

Conception d'applications EJB

- √ En général, pas un mais plusieurs EJB.
- √ Des problèmes ou particularités :
  - Comment accéder aux EJB (distants ou locaux)
  - Référence d'EJB (passage de paramètres, attributs,. . .)
  - Comparer des EJB (via EJBObject)

Comparaison d'EJB

- √ Pour tout composant EJB :

```
interface EJBObject ...
  public boolean isIdentical(EJBObject obj))
```

- √ Pour les Entity Beans

```
interface EJBObject ...
  Object getPrimaryKey()
```

Une application équivaut à un schéma

√ Les composants entité décrivent les données persistantes

Une application équivaut à un schéma

- √ Les composants entité décrivent les données persistantes
- √ Les données persistantes sont stockés dans un SGBD

Une application équivaut à un schéma

- √ Les composants entité décrivent les données persistantes
- √ Les données persistantes sont stockés dans un SGBD
- √ Ces données sont structurées, résultat d'une démarche de conception (enfin, on l'espère !)

Une application équivaut à un schéma

- √ Les composants entité décrivent les données persistantes
- √ Les données persistantes sont stockés dans un SGBD
- √ Ces données sont structurées, résultat d'une démarche de conception (enfin, on l'espère !)
- √ **Solution EJB 2.0 :**

  Redéfinir le schéma entité-association

  - ▶ Gestion automatique des clés étrangères
  - ▶ Gestion automatique de l'intégrité référentielle

schéma abstrait de persistance

- √ Notion de `Container Management Relationship`
- √ Uniquement des associations binaires.
- √ Une association est décrite par :
  - ▶ Association uni-directionnelle : définir un nom de rôle
  - ▶ Association bi-directionnelle : définir les deux noms de rôle
  - ▶ Cardinalités : `one-to-one, one-to-many, many-to many` pas de possibilités d'affinement (CMR)
- √ Ces informations se retrouvent dans le descripteur de déploiement XML

Au niveau du Composant Bean

- √ Introduire des méthodes accesseurs publiques et abstraites.
- √ Convention de nommage en fonction des cardinalités
  - Cardinalité one :

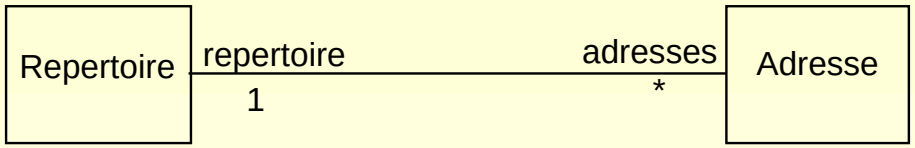
```
public abstract <ForeignClass> get<nomRole>() ;
public abstract void set<nomRole>(<ForeignClass> value) ;
```

  - Cardinalité many :

```
public abstract Collection get<nomRole>() ;
public abstract void set<nomRole>(collection value) ;
```

Un exemple

√ Gestion d'un répertoire d'adresses :



√ Codage EJB :

```
public abstract class RepertoireBean implements EntityBean
  public abstract Collection getAdresses() ;
  public abstract void setAdresses(Collection value) ;
```

```
public abstract class AdresseBean implements EntityBean
  public abstract RepertoireLocal getRepertoire() ;
  public abstract void setRepertoire(RepertoireLocal value) ;
```

Les méthodes métiers

√ Les méthodes métiers exploitent les méthodes abstraites :

```
public void addAdresse(AdresseDetails details) {
  try {
    Collection adresses = this.getAdresses() ;
    AdresseLocal adresse = adresseHome.create(details.getNom(),
                                              details.getEmail()) ;
    adresses.add(adresse) ;
  } catch (Exception ex) {
    throw new EJBException(ex.getMessage());
  }
}
```

Une classe de représentation des données (1/2)

- √ Préserver l'accès des composants entité (interface locale)
- √ But : fournir les informations

```
public class AdresseDetails implements java.io.Serializable {
  private String nom; private String email;

  public AdresseDetails (String nom, String email){
      this.nom = nom; this.email = email;
  }
  public String getNom() { return nom; }
  public String getEmail() { return email; }
  public String toString() { return  nom + " " + email ; }
}
```

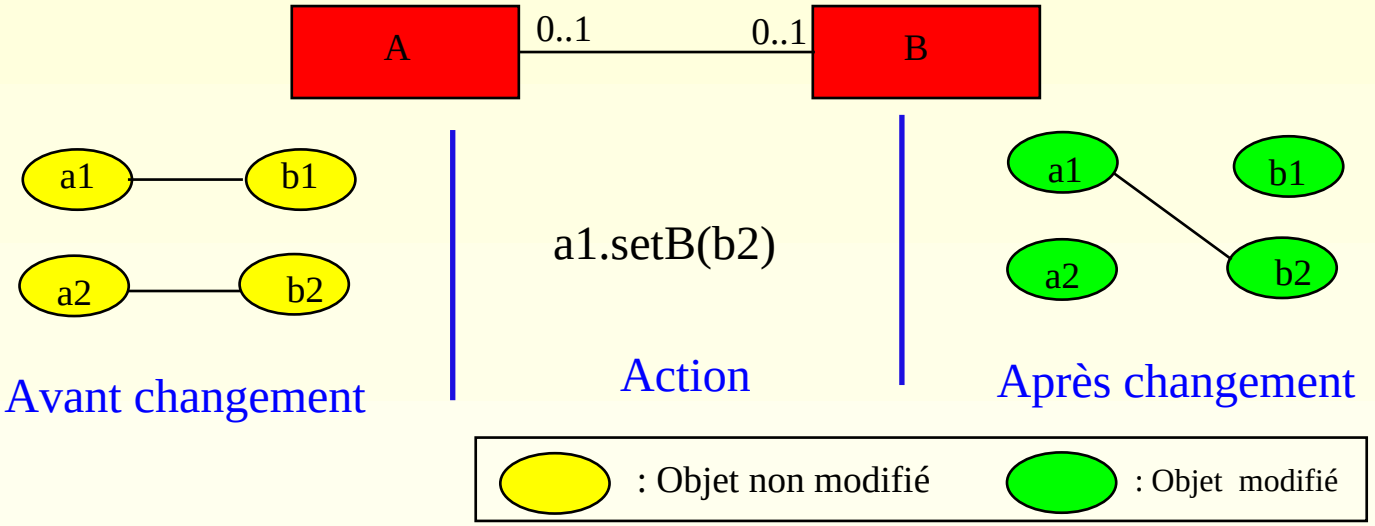
Une classe de représentation des données (1/2)

√ Exemple d'exploitation de la classe de représentation :

```
public ArrayList getAllAdresses() {
    Collection adresses = getAdresses();
    ArrayList detailsList = new ArrayList();
    for (Iterator i = adresses.iterator(); i.hasNext() ; ) {
       AdresseLocal adresse = (AdresseLocal) i.next();
       AdresseDetails details ;
       details = new AdresseDetails(adresse.getNom(),
                                    adresse.getEmail());
       detailsList.add(details);
    }
    return detailsList;
}
```

La puissance de l'intégrité référentielle

- √ Se situe au niveau des liens d'associations (add, set, remove)
- √ Le conteneur est garant de la cohérence pour toute modification
- √ Un exemple :



Vers de meilleurs accès BD

- √ Les méthodes select (EJB 2.0, CMP)
- √ Convention de nommage : `ejbSelect<nom>`
- √ Définie au niveau du bean uniquement, exemple :
  `public abstract Collection ejbSelectAdrStartsWith(String v)`
- √ Introduction au niveau XML de la requête EJB-QL correspondante.
  Exemple :
  `select Object(a) from Adresse a where a.nom like '?1%'`
- √ EJB-QL reste très limité (sous-sous SQL, pas de count ! (sauf EJB 2.1))

Message Driven Bean (1/3)

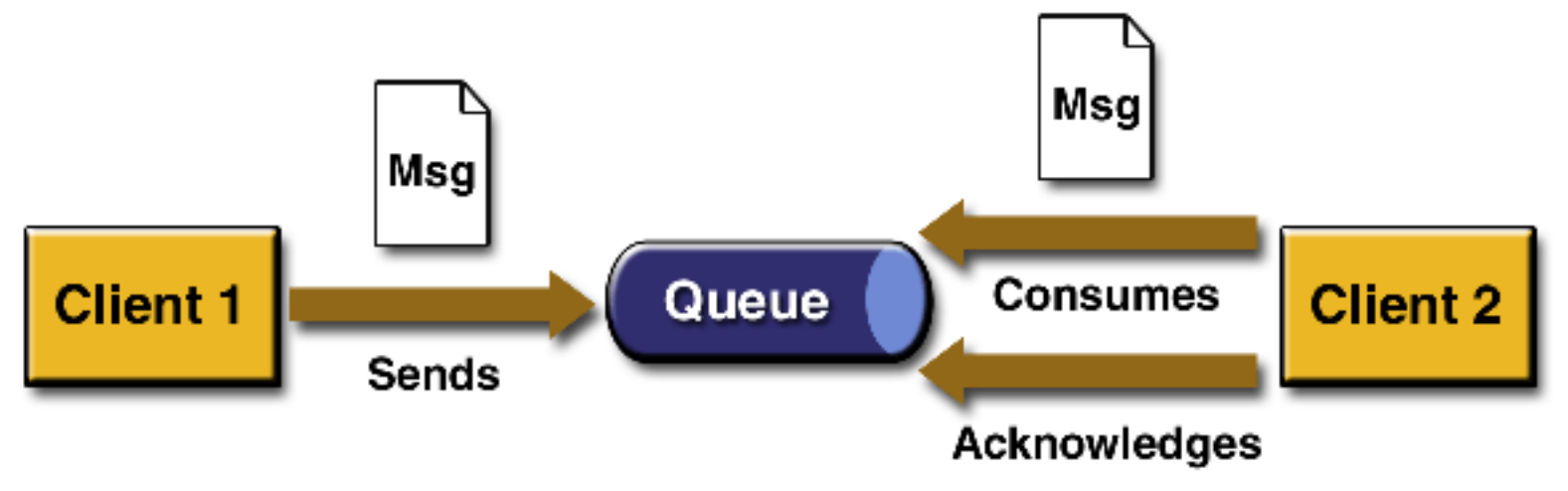
√ Envoi de messages asynchrones

Message Driven Bean (1/3)

- √ Envoi de messages asynchrones
- √ Les composants ont un couplage faible, pas reliés par leur interface

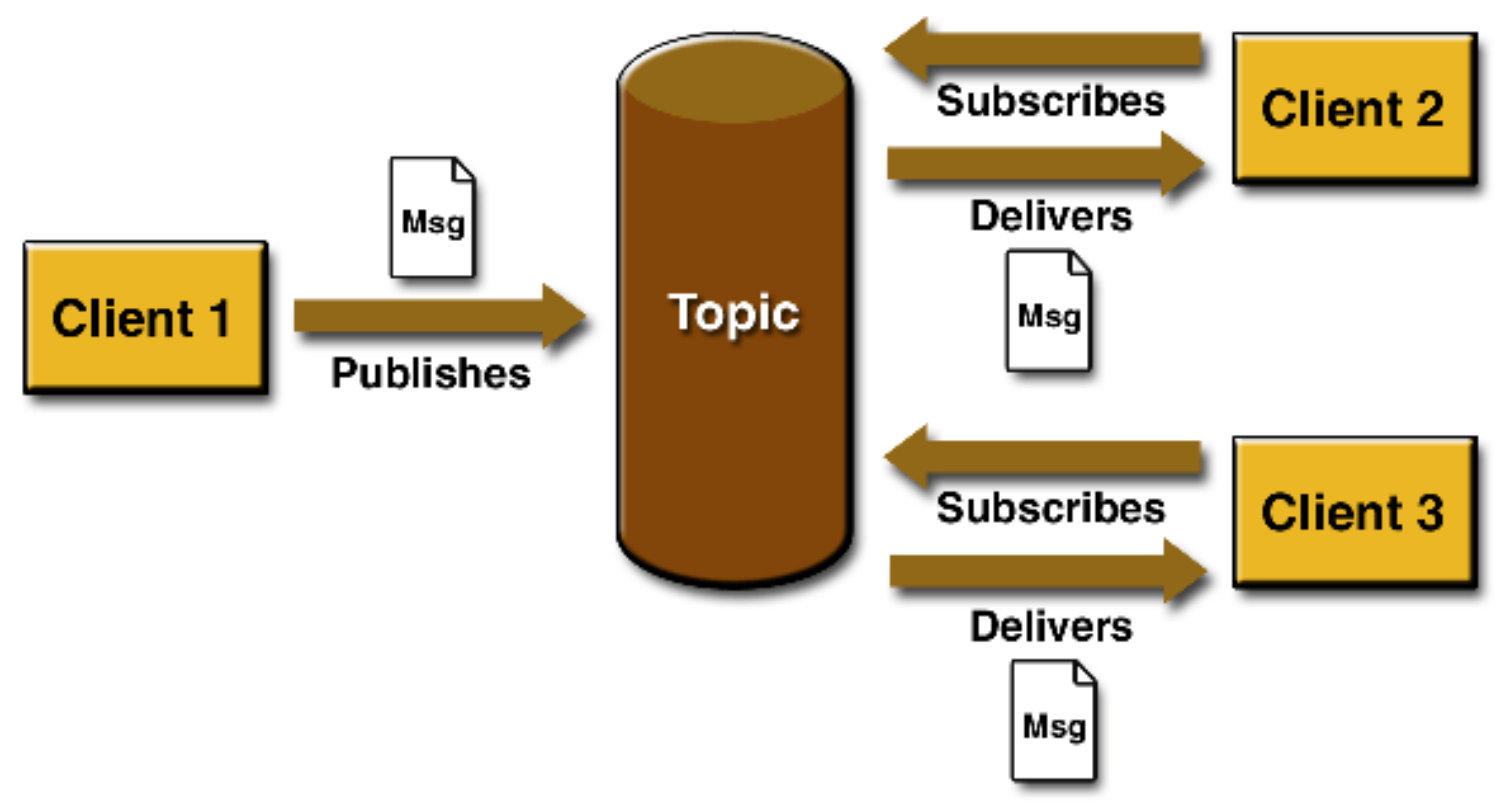
Message Driven Bean (2/3)

√ Liaison Point à point :



Message Driven Bean (3/3)

√ Diffusion multiple :



Conclusion

- √ Les plus :
  - framework Assez simple

Conclusion

√ Les plus :

- framework Assez simple
- bon découpage aspects fonctionnel et techniques

Conclusion

√ Les plus :

- framework Assez simple
- bon découpage aspects fonctionnel et techniques
- Portabilité maximum (si serveur conforme J2EE. . .)

Conclusion

- √ Les plus :
  - framework Assez simple
  - bon découpage aspects fonctionnel et techniques
  - Portabilité maximum (si serveur conforme J2EE. . .)
- √ Les moins :
  - Liens efficaces SGBD, montée en charge

Conclusion

- √ Les plus :
  - framework Assez simple
  - bon découpage aspects fonctionnel et techniques
  - Portabilité maximum (si serveur conforme J2EE. . .)
- √ Les moins :
  - Liens efficaces SGBD, montée en charge
  - Outillage, pas une méthode :
    Quelques "design pattern EJB" (ex : pattern Facade)
  - de UML à EJB ?

Conclusion

- √ Les plus :
  - framework Assez simple
  - bon découpage aspects fonctionnel et techniques
  - Portabilité maximum (si serveur conforme J2EE. . .)
- √ Les moins :
  - Liens efficaces SGBD, montée en charge
  - Outillage, pas une méthode :
    Quelques "design pattern EJB" (ex : pattern Facade)
  - de UML à EJB ?
  - Conteneur non évolutif : prise en compte de nouveaux aspects techniques