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

Olivier Caron

Polytech Lille
Avenue Paul Langevin Cité Scientifique Lille 1
59655 Villeneuve d'Ascq cedex

http://ocaron.plil.fr
Olivier.Caron@polytech-lille.fr

© Dave Barry

Nous ne pouvons pas prédire où nous conduira la révolution informatique. Tout ce que nous savons avec certitude, c'est que, quand on y sera enfin, on n'aura pas assez de RAM.

Sources

- Spécifications Oracle EJB 2.1, EJB 3.1 , `http://www.oracle.com/technetwork/java/javaee/documentation`

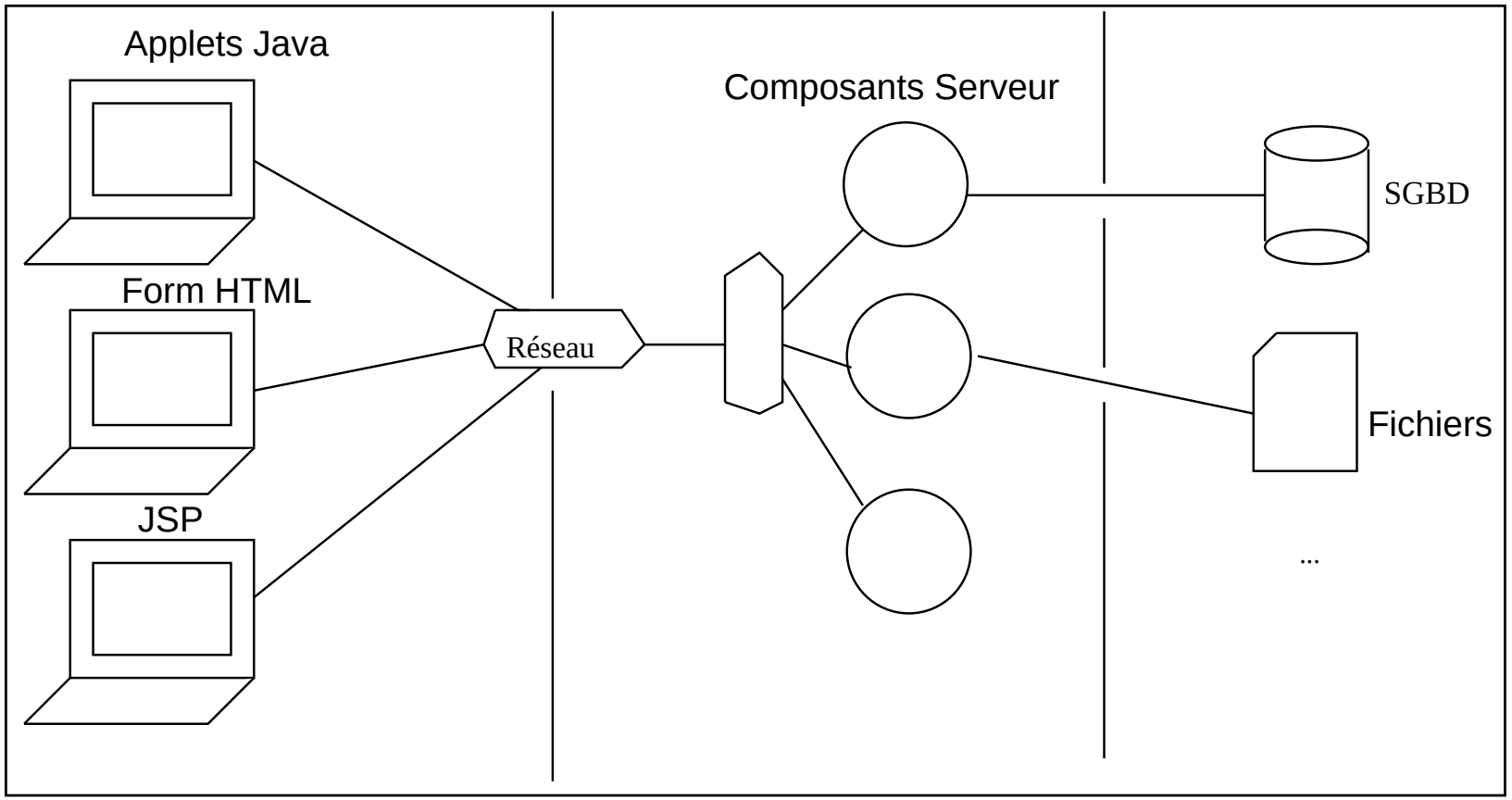
Sources

- Spécifications Oracle EJB 2.1, EJB 3.1 , `http://www.oracle.com/technetwork/java/javaee/documentation`
- Documentation JBoss, `http://www.jboss.org` ou `http://wildfly.org`

Sources

- Spécifications Oracle EJB 2.1, EJB 3.1 , `http://www.oracle.com/technetwork/java/javaee/documentation`
- Documentation JBoss, `http://www.jboss.org` ou `http://wildfly.org`
- Documentation Oracle (ex SUN, Javasoft), `http://docs.oracle.com/javaee/7/tutorial`

Les architectures 3-tiers



Les modèles de composants serveur

- Le modèle de composants JEE-EJB

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)
- Le modèle de composants CORBA

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)
- Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plates-formes

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)
- Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plates-formes
  - Des langages de spécifications de composants (IDL3)

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)
- Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plates-formes
  - Des langages de spécifications de composants (IDL3)
  - Des langages de spécifications de déploiement (CIDL)

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)
- Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plates-formes
  - Des langages de spécifications de composants (IDL3)
  - Des langages de spécifications de déploiement (CIDL)
- Le modèle .Net

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)
- Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plates-formes
  - Des langages de spécifications de composants (IDL3)
  - Des langages de spécifications de déploiement (CIDL)
- Le modèle .Net
  - Multi-langages (C++, VBScript, C#)

Les modèles de composants serveur

- Le modèle de composants JEE-EJB
  - Orienté bases de données, multi-serveur d'applications, mono-langage (Java)
- Le modèle de composants CORBA
  - Multi-serveurs, multi-langages, multi-plates-formes
  - Des langages de spécifications de composants (IDL3)
  - Des langages de spécifications de déploiement (CIDL)
- Le modèle .Net
  - Multi-langages (C++, VBScript, C#)
  - Multi-plates-formes windows mais aussi plateforme linux : `http://www.mono-project.com`

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

- Des objets trop complexes

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 (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)

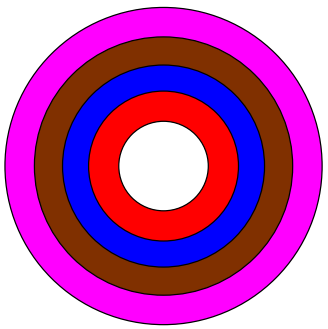
- 1ère étape :une meilleure structuration objet

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

- 1ère étape :une meilleure structuration objet

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

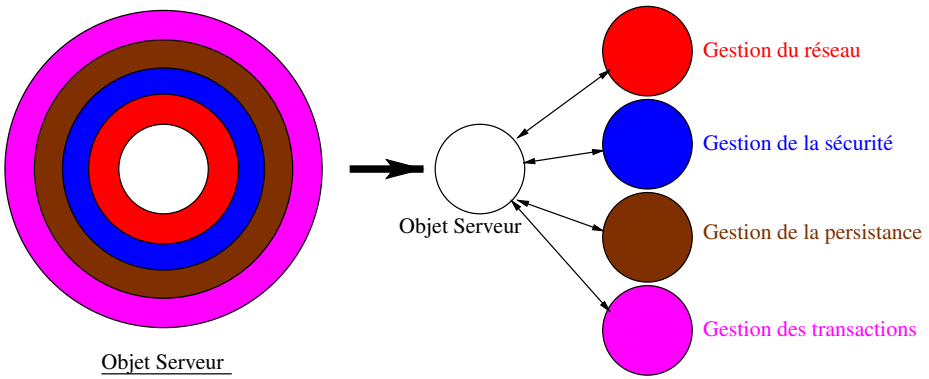
- 1ère étape :une meilleure structuration objet



Objet Serveur

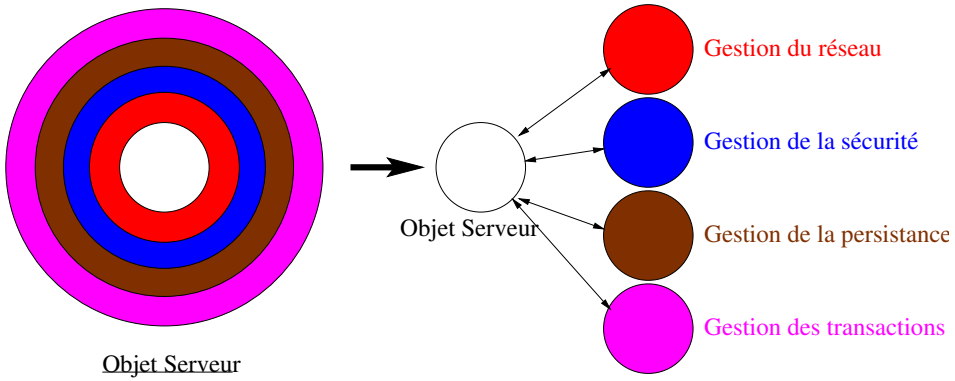
Des objets aux composants d'entreprise (2/3)

- 1ère étape :une meilleure structuration objet



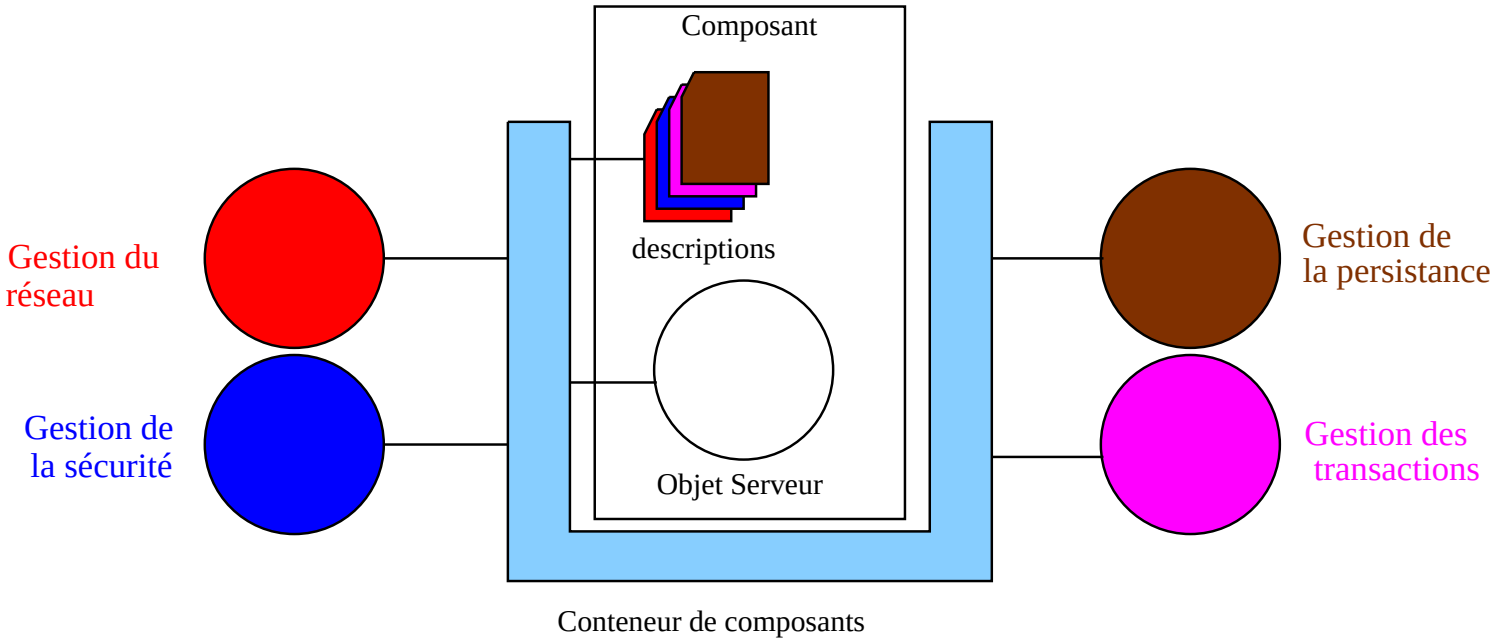
Des objets aux composants d'entreprise (2/3)

- 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 :

Les composants d'entreprise : en résumé

- Applications cibles trop complexes :
  - Gérer le réseau, la sécurité, les transactions, les bases de données, le déploiement,. . .

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 :

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

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)

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 ”vrai” code !

Objectifs de la norme JEE-EJB (Enterprise Java Beans)

- Conçue par Sun Microsystems (depuis 1998)

Objectifs de la norme JEE-EJB (Enterprise Java Beans)

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

Objectifs de la norme JEE-EJB (Enterprise Java Beans)

- Conçue par Sun Microsystems (depuis 1998)
- Modèle différent des Java Beans !
- Objectifs :

Objectifs de la norme JEE-EJB (Enterprise Java Beans)

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

Objectifs de la norme JEE-EJB (Enterprise Java Beans)

- Conçue par Sun Microsystems (depuis 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 JEE-EJB (Enterprise Java Beans)

- Conçue par Sun Microsystems (depuis 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

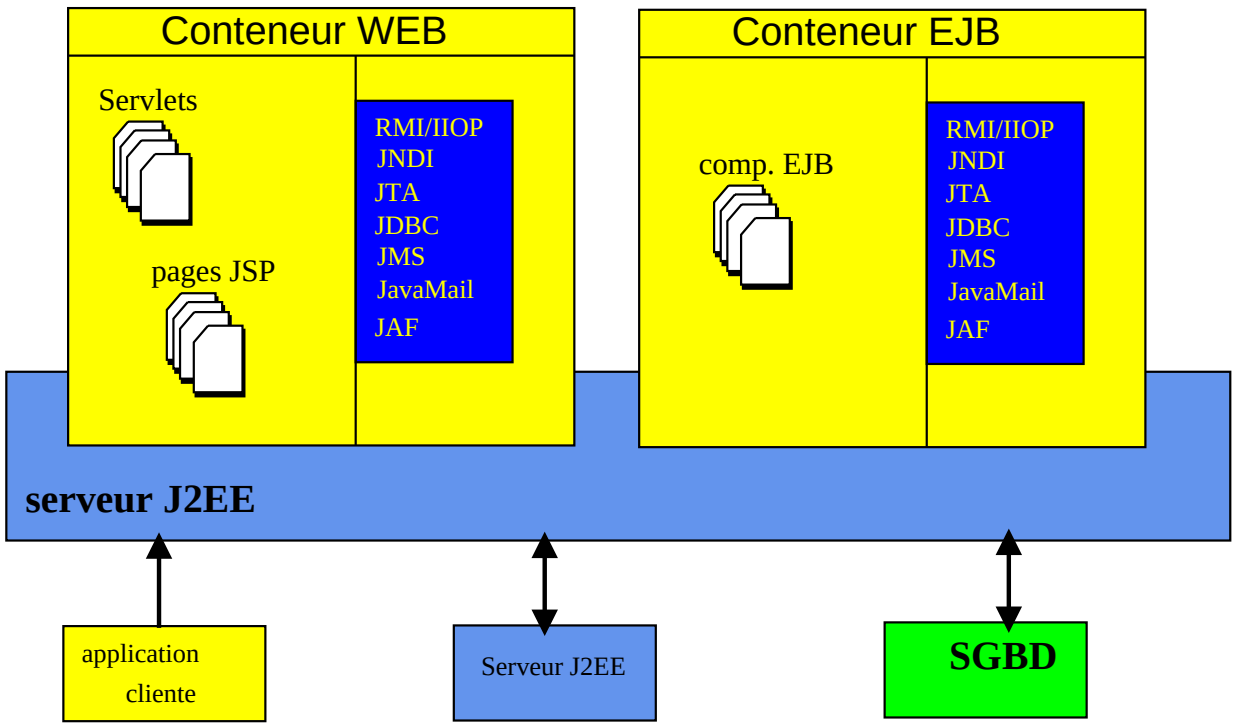
Objectifs de la norme JEE-EJB (Enterprise Java Beans)

- Conçue par Sun Microsystems (depuis 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 JEE-EJB (Enterprise Java Beans)

- Conçue par Sun Microsystems (depuis 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 JEE



Une multitude de serveurs d'applications EJB

- Oracle Glassfish (SGBDR, modèle objet-relationnel)

Une multitude de serveurs d'applications EJB

- Oracle Glassfish (SGBDR, modèle objet-relationnel)
- Inprise, JBoss, JONAS,. . .

Une multitude de serveurs d'applications EJB

- Oracle Glassfish (SGBDR, modèle objet-relationnel)
- Inprise, JBoss, JONAS,. . .
- L'écriture d'une application EJB est indépendante d'un serveur EJB

Evolution des EJB

- des EJB 1.0 aux EJB 3.1

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services
  - Définition et amélioration d'un framework

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services
  - Définition et amélioration d'un framework
- Arrivée des EJB 3.0 (2006), EJB 3.1 (2011)

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services
  - Définition et amélioration d'un framework
- Arrivée des EJB 3.0 (2006), EJB 3.1 (2011)
  - Etat février 06 : phase finale de spécification

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services
  - Définition et amélioration d'un framework
- Arrivée des EJB 3.0 (2006), EJB 3.1 (2011)
  - Etat février 06 : phase finale de spécification
  - Utilisation des mécanismes Java 1.5

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services
  - Définition et amélioration d'un framework
- Arrivée des EJB 3.0 (2006), EJB 3.1 (2011)
  - Etat février 06 : phase finale de spécification
  - Utilisation des mécanismes Java 1.5
  - Libérer le programmeur de tout framework

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services
  - Définition et amélioration d'un framework
- Arrivée des EJB 3.0 (2006), EJB 3.1 (2011)
  - Etat février 06 : phase finale de spécification
  - Utilisation des mécanismes Java 1.5
  - Libérer le programmeur de tout framework
  - Substituer la notation XML par des annotations Java

Evolution des EJB

- des EJB 1.0 aux EJB 3.1
  - Grammaires XML dédiées aux services
  - Définition et amélioration d'un framework
- Arrivée des EJB 3.0 (2006), EJB 3.1 (2011)
  - Etat février 06 : phase finale de spécification
  - Utilisation des mécanismes Java 1.5
  - Libérer le programmeur de tout framework
  - Substituer la notation XML par des annotations Java
  - Garder la compatibilité EJB

Les types de composants EJB

- Les composants Session

Les types de composants EJB

- Les composants Session
  - Assurent les services métiers

Les types de composants EJB

- Les composants Session
  - Assurent les services métiers
  - interface avec les composants web ou clients

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 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

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

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

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)

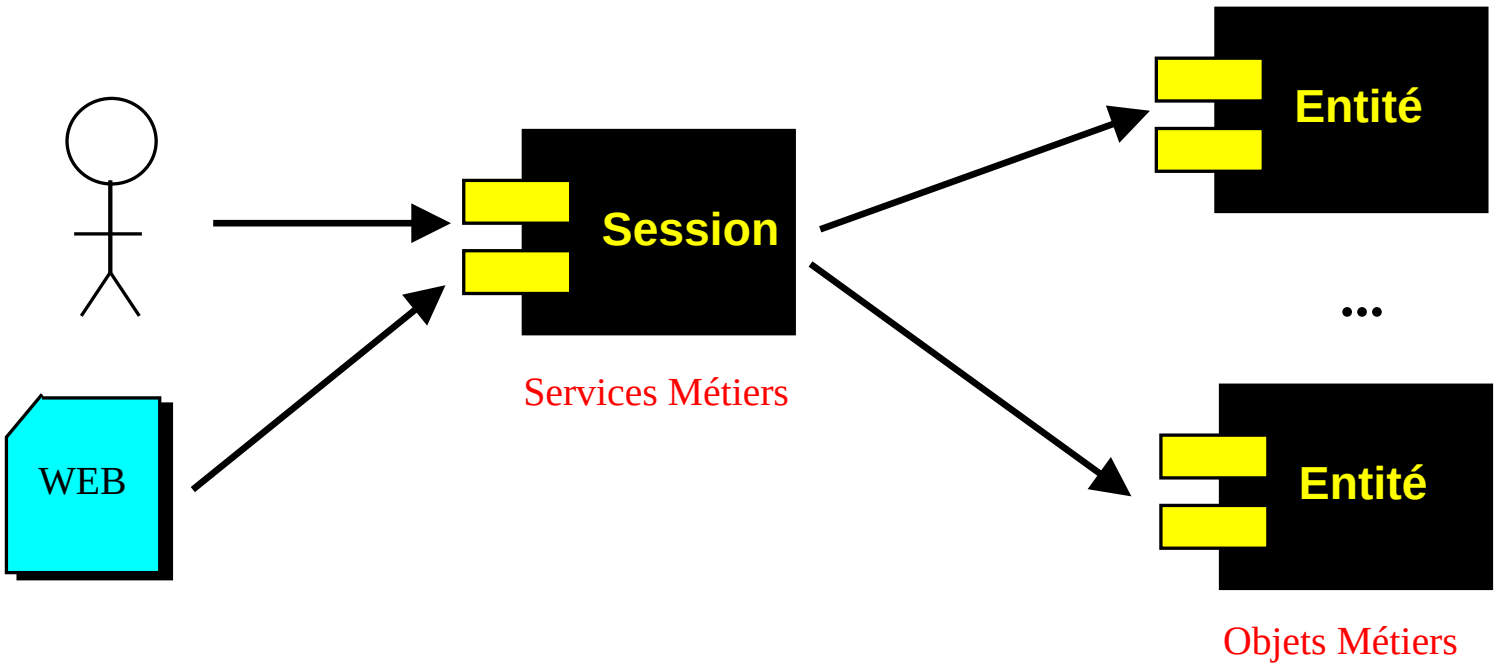
Les types de composants EJB

- 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” (à partir de la norme EJB 2.0)

Les types de composants EJB

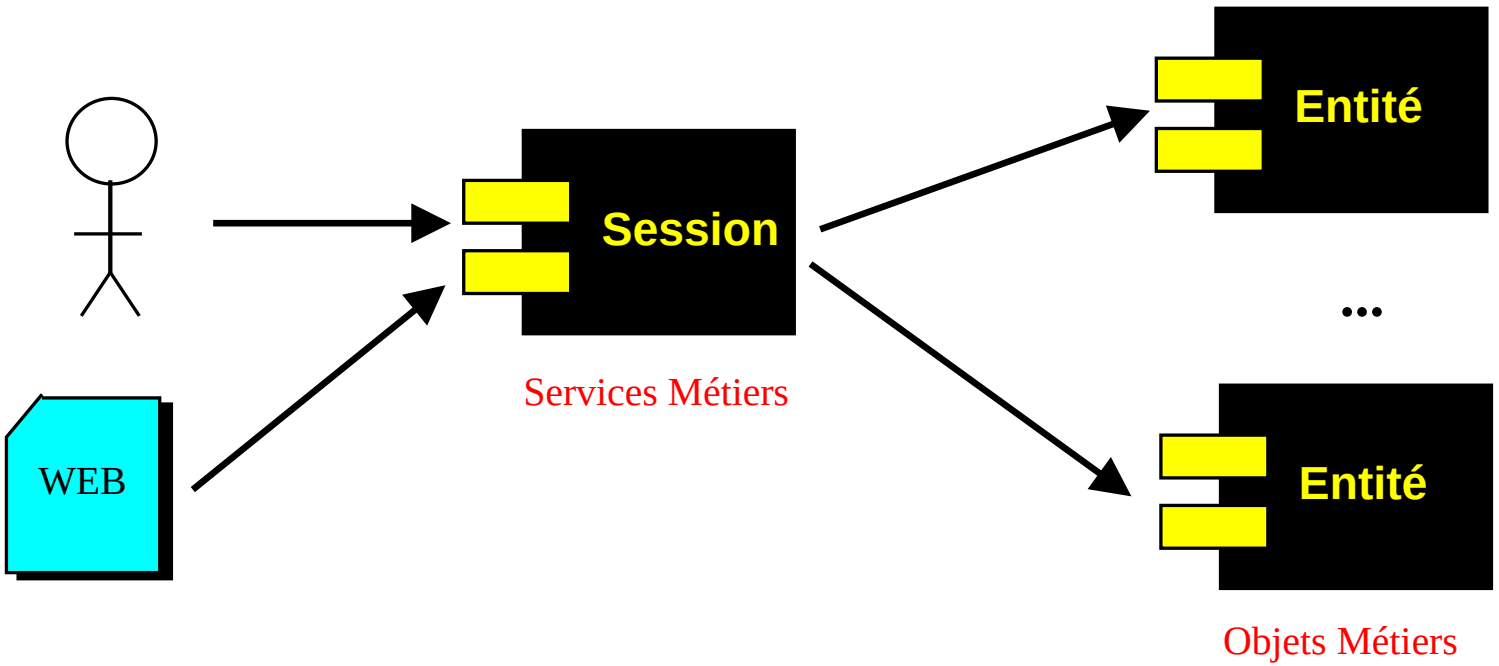
- Les composants Session
  - Assurent les services métiers
  - interface avec les composants web ou clients
  - Avec ou sans état transactionnel (Stateless, Stateful)
- Les composants Entités
  - Représentent les objets métiers
  - Interface avec les bases de données
  - Persistance gérée ou pas par le conteneur (CMP, BMP)
- Les composants ”Message Driven” (à partir de la norme EJB 2.0)
  - Gestion asynchrone, événementielle

Conception et Structuration des composants (1/2)



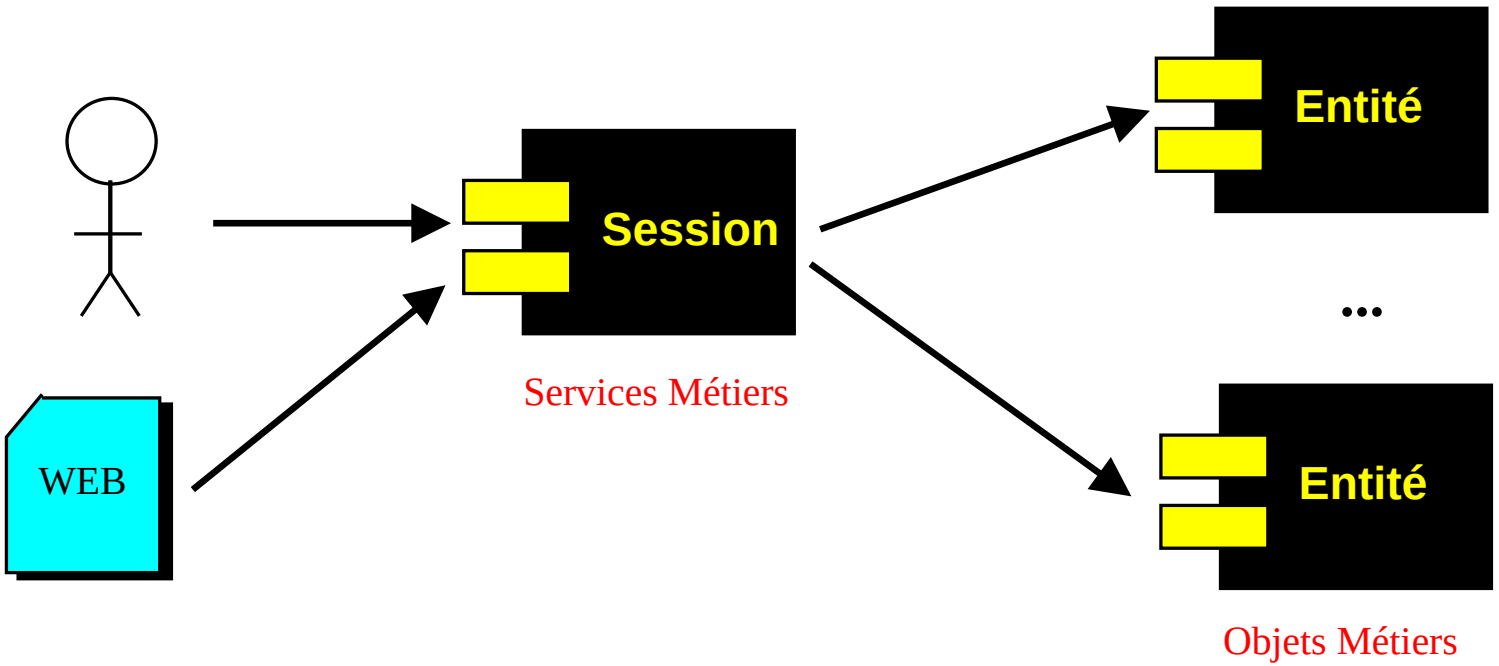
- Exemple banque :

Conception et Structuration des composants (1/2)



- Exemple banque :
  - virement de comptes $\rightarrow$ service (session)

Conception et Structuration des composants (1/2)



- Exemple banque :
  - virement de comptes $\rightarrow$ service (session)
  - info solde $\rightarrow$ données (entité)

Conception et Structuration des composants (2/2)

- Use Case et diagrammes de séquences définissent les composants sessions

Conception et Structuration des composants (2/2)

- Use Case et diagrammes de séquences définissent les composants sessions
- Structure du système d'information définit les composants entité

Conception et Structuration des composants (2/2)

- Use Case et diagrammes de séquences définissent les composants sessions
- Structure du système d'information définit les composants entité
- 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 !

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

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'accéde jamais directement au composant, cela autorise des mécanismes de swap selon le composant.

Norme EJB 3.0

- Concepts de base :

Norme EJB 3.0

- Concepts de base :
  - Plus de framework (ou minimal), liberté de programmation
    Des ”objets POJO” (Plain Old java Objects)

Norme EJB 3.0

- Concepts de base :
  - Plus de framework (ou minimal), liberté de programmation
    Des ”objets POJO” (Plain Old java Objects)
  - Héritage, polymorphisme, etc : tout est possible.

Norme EJB 3.0

- Concepts de base :
  - Plus de framework (ou minimal), liberté de programmation
    Des ”objets POJO” (Plain Old java Objects)
  - Héritage, polymorphisme, etc : tout est possible.
  - Ecriture XML éventuellement remplacée par annotations Java

Norme EJB 3.0

- Concepts de base :
  - Plus de framework (ou minimal), liberté de programmation
    Des ”objets POJO” (Plain Old java Objects)
  - Héritage, polymorphisme, etc : tout est possible.
  - Ecriture XML éventuellement remplacée par annotations Java
- Les pré-requis : Java 5

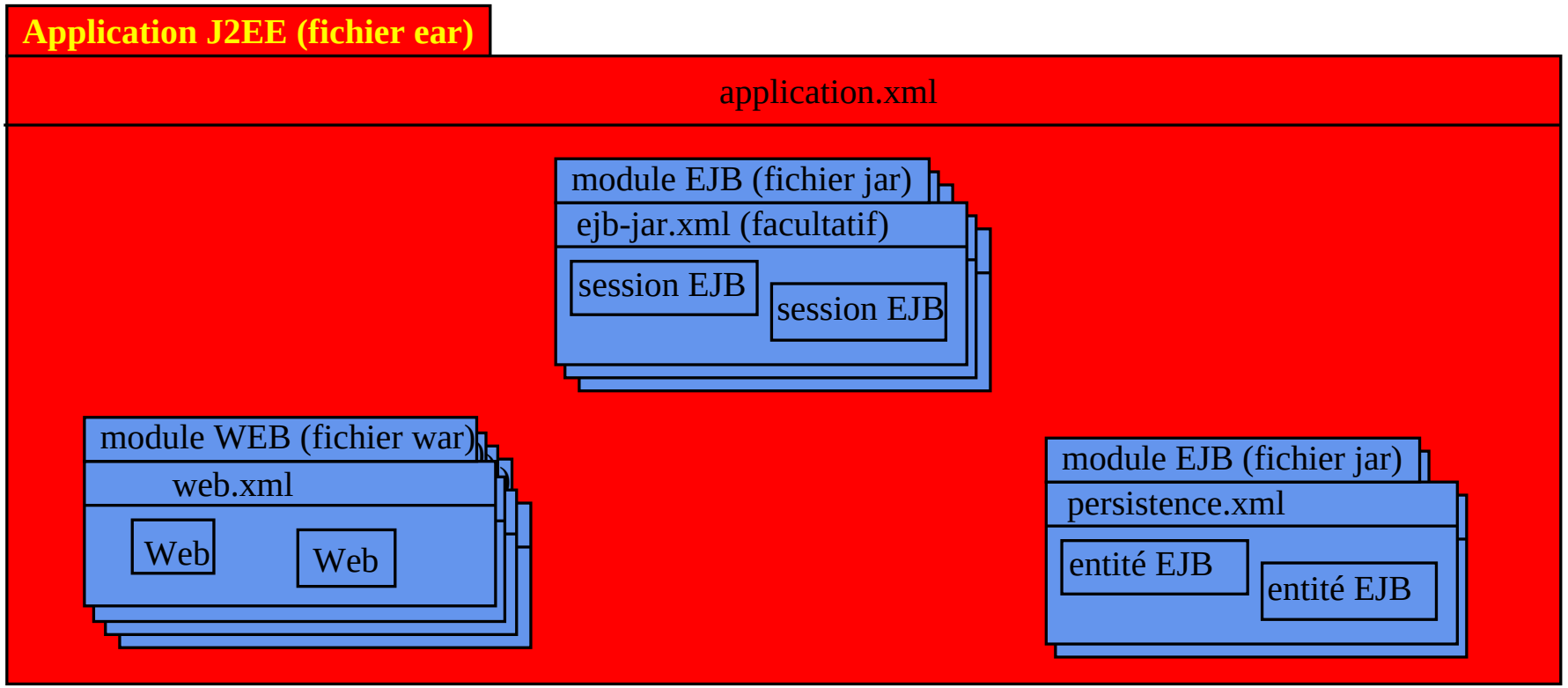
Norme EJB 3.0

- Concepts de base :
  - Plus de framework (ou minimal), liberté de programmation
    Des ”objets POJO” (Plain Old java Objects)
  - Héritage, polymorphisme, etc : tout est possible.
  - Ecriture XML éventuellement remplacée par annotations Java
- Les pré-requis : Java 5
  - Les annotations Java

Norme EJB 3.0

- Concepts de base :
  - Plus de framework (ou minimal), liberté de programmation
    Des ”objets POJO” (Plain Old java Objects)
  - Héritage, polymorphisme, etc : tout est possible.
  - Ecriture XML éventuellement remplacée par annotations Java
- Les pré-requis : Java 5
  - Les annotations Java
  - La généricité Java (gestion des associations)

Une application JEE - EJB 3 (1/3)



Une application JEE - EJB 3 (2/3)

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

jar cvf monAppli.ear
    monAppli/*
```

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="6"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 ____________http://java.sun.com/xml/ns/javaee/application_6.xsd">
  <display-name>monAppli</display-name>
  <module>
    <web>
      <web-uri>mesComposantsWeb.war</web-uri>
      <context-root>repertoireRacine</context-root>
    </web>
  </module>
  <module> <ejb>mesEJBSessions.jar</ejb>
  </module>
  <module> <ejb>mesEJBEntites.jar</ejb>
  </module>
</application>
```

Les composants sessions EJB 3.1

- Assurent les services métiers.

Les composants sessions EJB 3.1

- Assurent les services métiers.
- Doivent disposer d'une interface métier (décrite par une interface Java)

Les composants sessions EJB 3.1

- Assurent les services métiers.
- Doivent disposer d'une interface métier (décrite par une interface Java)
- On distingue :

Les composants sessions EJB 3.1

- Assurent les services métiers.
- Doivent disposer d'une interface métier (décrite par une interface Java)
- On distingue :
  - les composants sessions avec ou sans état transactionnel (Stateful, Stateless)

Les composants sessions EJB 3.1

- Assurent les services métiers.
- Doivent disposer d'une interface métier (décrite par une interface Java)
- On distingue :
  - les composants sessions avec ou sans état transactionnel (Stateful, Stateless)
  - Les composants accessibles à distance et/ou localement.

Accès aux composants sessions

- Une application JEE (.ear) ne peut être déployée que sur un seul serveur

Accès aux composants sessions

- Une application JEE (.ear) ne peut être déployée que sur un seul serveur
- Une application peut communiquer avec d'autres applications situées sur le même serveur ou pas

Accès aux composants sessions

- Une application JEE (.ear) 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) :

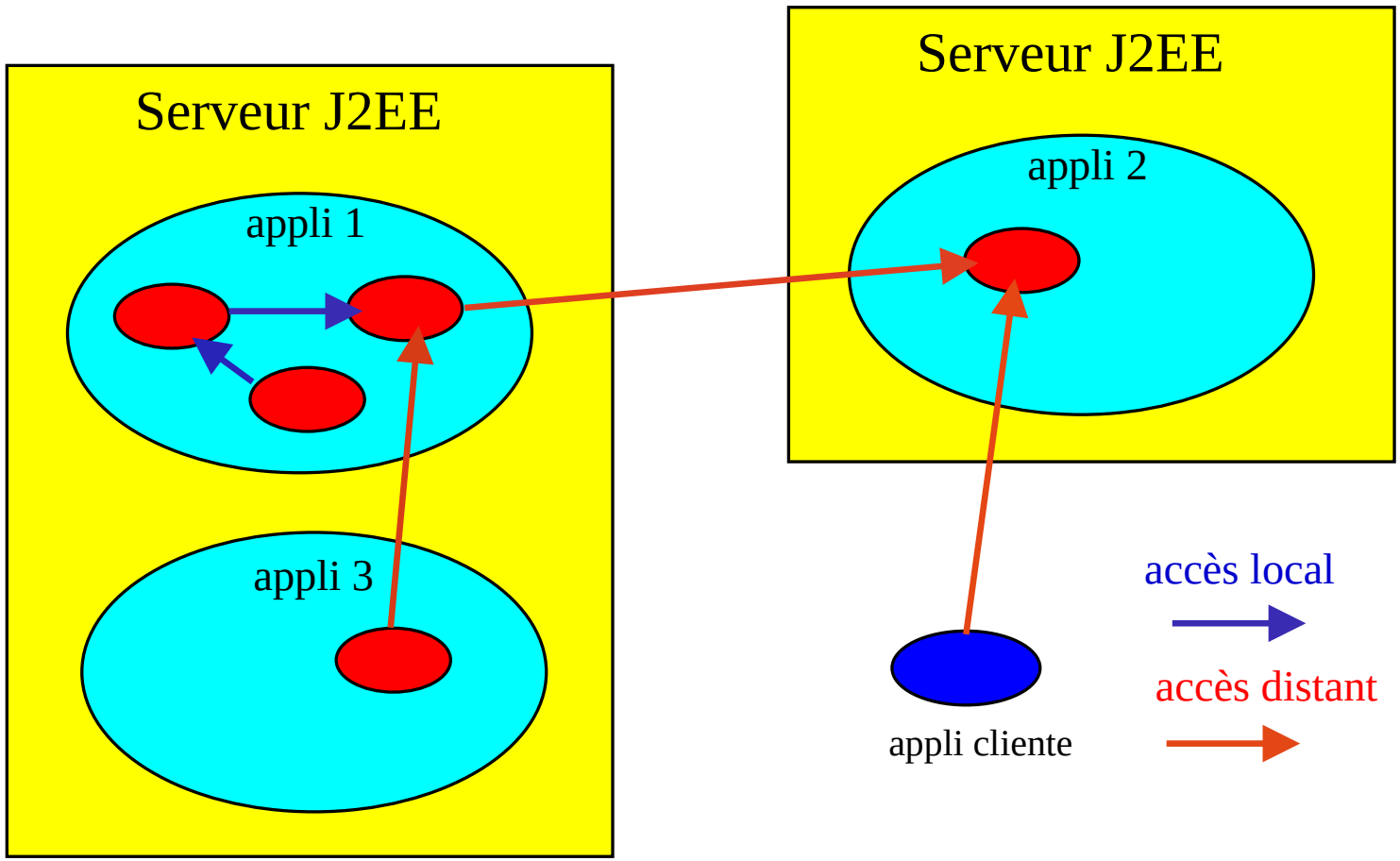
Accès aux composants sessions

- Une application JEE (.ear) 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 session EJB ou Web situés dans une même application

Accès aux composants sessions

- Une application JEE (.ear) 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 session EJB ou Web situés dans une même application
  - Les accès distants : composants sessions entre plusieurs applications (protocole RMI/IIOP)

Accès distants ou locaux



Exemple de composant session sans état (1/3)

```
package ocaron.exempleSession ;

import javax.ejb.Stateless ;

@Stateless
public class CalculSalaireBean
  implements CalculSalaireRemote, CalculSalaireLocal {

  public CalculSalaireBean() {}

  public double getSalaire(int nbreHeures) {
    return nbreHeures*tauxHoraire ;
  }
}
```

Exemple de composant session sans état (2/3)

```
package ocaron.exempleSession ;

public interface CalculSalaire {
   public final static double tauxHoraire = 8.03 ;
   public double getSalaire(int nbreHeures) ;
}
```

Exemple de composant session sans état (3/3)

```
package ocaron.exempleSession ;

import javax.ejb.Remote ;
@Remote public interface CalculSalaireRemote
  extends CalculSalaire {}
```

```
package ocaron.exempleSession ;

import javax.ejb.Local ;
@Local public interface CalculSalaireLocal
  extends CalculSalaire {}
```

Ce composant est accessible à distante et de manière locale avec les mêmes fonctionnalités (même interface java).

Composant session : programmation du bean

- C'est une classe java pure : elle n'hérite pas d'une quelconque classe du standard Java JEE.

Composant session : programmation du bean

- C'est une classe java pure : elle n'hérite pas d'une quelconque classe du standard Java JEE.
- Doit disposer d'une fonction constructeur sans paramètre

Composant session : programmation du bean

- C'est une classe java pure : elle n'hérite pas d'une quelconque classe du standard Java JEE.
- Doit disposer d'une fonction constructeur sans paramètre
- Convention de nommage : le nom de la classe termine par "Bean", "Impl", "Implementation" ou "EJB".

Composant session : programmation du bean

- C'est une classe java pure : elle n'hérite pas d'une quelconque classe du standard Java JEE.
- Doit disposer d'une fonction constructeur sans paramètre
- Convention de nommage : le nom de la classe termine par "Bean", "Impl", "Implementation" ou "EJB".
- Cette classe est annotée soit par : `@Stateless` ou `@Stateful`.

Composant session : programmation du bean

- C'est une classe java pure : elle n'hérite pas d'une quelconque classe du standard Java JEE.
- Doit disposer d'une fonction constructeur sans paramètre
- Convention de nommage : le nom de la classe termine par "Bean", "Impl", "Implementation" ou "EJB".
- Cette classe est annotée soit par : `@Stateless` ou `@Stateful`.
- Le nom du composant est le nom de la classe

Composant session : programmation du bean

- C'est une classe java pure : elle n'hérite pas d'une quelconque classe du standard Java JEE.
- Doit disposer d'une fonction constructeur sans paramètre
- Convention de nommage : le nom de la classe termine par "Bean", "Impl", "Implementation" ou "EJB".
- Cette classe est annotée soit par : `@Stateless` ou `@Stateful`.
- Le nom du composant est le nom de la classe
- Possibilité d'attribuer un nom différent par l'attribut `name` de l'annotation `@Stateless` ou `@Stateful`.

Composant session : les interfaces métiers

- Si le bean n'implémente qu'une seule interface, alors elle est considérée comme l'interface métier **locale**.

Composant session : les interfaces métiers

- Si le bean n'implémente qu'une seule interface, alors elle est considérée comme l'interface métier **locale**.
- Si le bean n'implémente aucune interface, la classe joue le rôle d'interface locale et expose toutes ses méthodes publiques.

Composant session : les interfaces métiers

- Si le bean n'implémente qu'une seule interface, alors elle est considérée comme l'interface métier **locale**.
- Si le bean n'implémente aucune interface, la classe joue le rôle d'interface locale et expose toutes ses méthodes publiques.
- Si le bean implémente plusieurs interfaces, alors chacune des interfaces doit être explicitement marquée par les annotations `Remote` ou `Local`.

Le déploiement des EJB sessions

1. Réception du fichier ear

Le déploiement des EJB sessions

1. Réception du fichier ear
2. Analyse du fichier *application.xml*

Le déploiement des EJB sessions

1. Réception du fichier ear
2. Analyse du fichier *application.xml*
3. Analyse du fichier archive jar des composants

Le déploiement des EJB sessions

1. Réception du fichier ear
2. Analyse du fichier *application.xml*
3. Analyse du fichier archive jar des composants
4. Inscription des interfaces métiers au serveur de noms (une fabrique complètement générée coté serveur)

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC
  - Définit une interface (API) unifiée pour gérer un service de noms

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC
  - Définit une interface (API) unifiée pour gérer un service de noms
  - De multiples implémentations (LDAP, DNS, etc)

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC
  - Définit une interface (API) unifiée pour gérer un service de noms
  - De multiples implémentations (LDAP, DNS, etc)
- Autorise une structuration arborescente des services,
  exemple :`chemin/sous-chemin/sous-sous-chemin/nom`

Spécifications JNDI pour composants EJB 3.1

- La spécification EJB 3.1 comporte un nommage standardisé pour accéder aux composants sessions.

Spécifications JNDI pour composants EJB 3.1

- La spécification EJB 3.1 comporte un nommage standardisé pour accéder aux composants sessions.
- Plusieurs noms sont possibles et dépendent de la ”distance” du composant.

Spécifications JNDI pour composants EJB 3.1

- La spécification EJB 3.1 comporte un nommage standardisé pour accéder aux composants sessions.
- Plusieurs noms sont possibles et dépendent de la ”distance” du composant.
- Trois espaces de noms sont possibles : `Global`, `Application` et `Module`

JNDI : espace de noms standard `Global`

- L'espace de noms `global` permet d'accèder à des composants d'une autre application JEE.
- La syntaxe des noms JNDI :

`java:global[/<app-name>]/<module-name>/<bean-name>[!<interface-FQN>]`

app-name	le nom du fichier application (hors suffixe ".ear")
module-name	le nom du module du composant session (hors suffixe ".jar")
bean-name	le nom du composant session (nom de la classe)
interface-FQN	nom complet de l'interface du composant

- Exemple :

`monAppli/mesSessions/CalculSalaireBean!ocaron.exempleSession.CalculSalaireBeanRemote`

- Si le composant ne dispose que d'une seule interface ou aucune, le serveur doit également enregistrer le nom JNDI suivant :

`java:global[/<app-name>]/<module-name>/<bean-name>`

JNDI : espace de noms standard `App` et `Module`

- L'espace de noms `app` permet d'accèder à des composants de la même application JEE.
- La syntaxe des noms JNDI :

  `java:app/<module-name>/<bean-name>[!<interface-FQN>]`

- L'espace de noms `module` permet d'accèder à des composants issus du même module.
- La syntaxe des noms JNDI :

  `java:module/<bean-name>[!<interface-FQN>]`

WildFly 9 et EJB 3.1

- Programmation: WildFly 9 est compatible avec la norme EJB 3.1.
- JNDI : WildFly a sa propre convention de nommage pour l'accès aux composants distants à partir d'une application Java cliente.
- L'accès distant aux composants est sécurisé : nécessite de fournir un utilisateur (login/password) référencé par le serveur WildFly.
- Conventions de nommage WildFly pour des composants Stateless :

  ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<**interface**-FQN>

- Conventions de nommage WildFly pour des composants Stateful :

  ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<**interface**-FQN>?stateful

- `distinct name` est un nom spécifique qui peut être spécifié lors du déploiement, chaîne vide si n'existe pas.
- L'interface ne peut être que l'interface annotée par `@Remote`

  Ex : ejb:monAppli/mesSessions//*CalculSalaireBean*!*ocaron.exempleSession.CalculSalaireBeanRemote*

Un programme client

```
import javax.naming.InitialContext;
import javax.naming.NamingException ;
import ocaron.exempleSession.CalculSalaireRemote ;

public class MainSession {
  public static void main(String[] args) {
    String adresseJNDI="ejb:monAppli/mesSessions//CalculSalaireBean!"+
                       "ocaron.exempleSession.CalculSalaireBeanRemote" ;
    try {
      InitialContext ctx = new InitialContext();
      Object obj = ctx.lookup(adresseJNDI);
      CalculSalaireRemote salaire = (CalculSalaireRemote) obj ;
      System.out.println("salaire : "+
                         salaire.getSalaire(Integer.parseInt(args[0])))  ;
    } catch(NamingException e1) {
      System.err.println("erreur, acces au serveur de noms") ;
} } }
```

Exécution avec plate-forme WildFly

- Code précédent portable (à l'exception de l'adresse externalisable)
- Le classpath doit contenir les classes réseaux WildFly pour accéder au serveur de noms JNDI WildFly (JBoss)
- librairie JNDI, fichier `jndi.properties` :

```
java.naming.factory.url.pkgs=org.jboss.ejb.client.naming
```

- Localisation serveur, mode d'accès, sécurité définies dans fichier `jboss-ejb-client.properties` :

```
endpoint.name=client-endpoint
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port = 8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.connection.default.username=jeeadm
remote.connection.default.password=secret-Gis4
```

Traitement des erreurs

- L'exécution des méthodes des interfaces métiers peuvent provoquer des erreurs.

Traitement des erreurs

- L'exécution des méthodes des interfaces métiers peuvent provoquer des erreurs.
- Le conteneur EJB propage ces erreurs par l'exception standard `javax.ejb.EJBException`

Cycle de vie des composants session sans état (1/2)

- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode

Cycle de vie des composants session sans état (1/2)

- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode
- Il peut donc être associé successivement à plusieurs clients

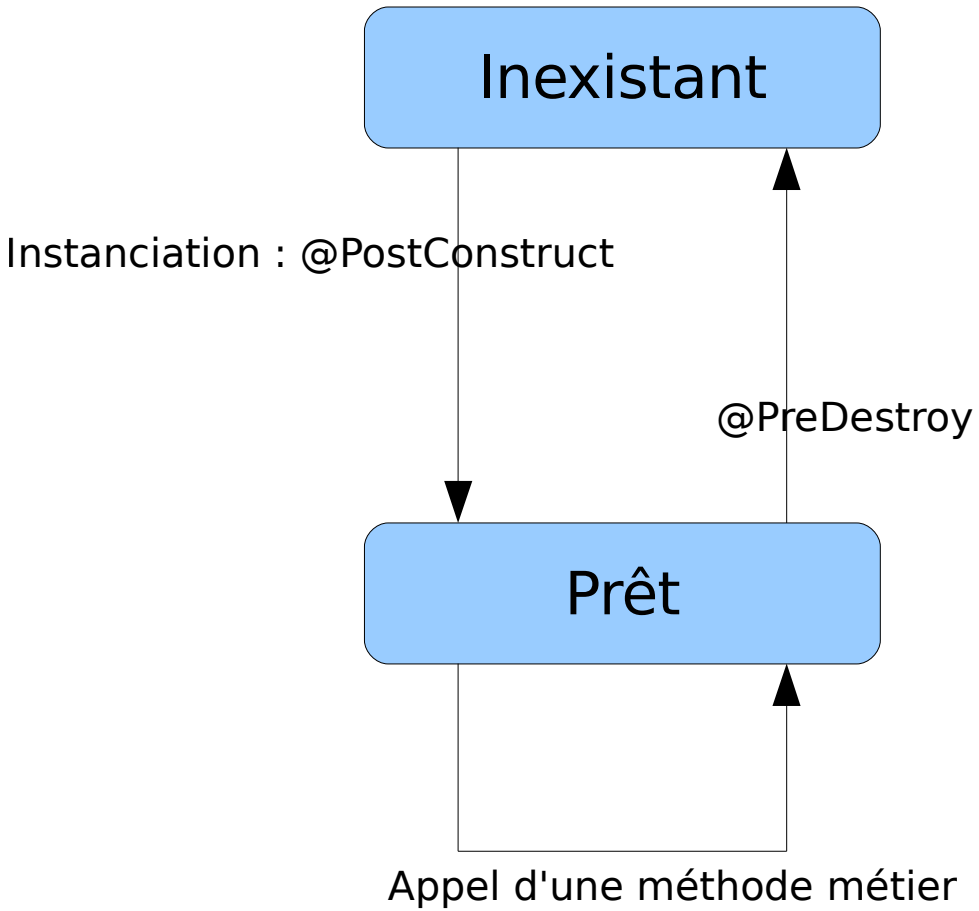
Cycle de vie des composants session sans état (1/2)

- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode
- Il peut donc être associé successivement à plusieurs clients
- Il peut être supprimé par le conteneur en cas de montée en charge par exemple.

Cycle de vie des composants session sans état (1/2)

- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode
- Il peut donc être associé successivement à plusieurs clients
- Il peut être supprimé par le conteneur en cas de montée en charge par exemple.
- Possibilité de gérer le cycle de vie par les méthodes "callbacks"

Cycle de vie des composants session sans état (2/2)



Les méthodes callbacks

- les méthodes doivent respecter la signature suivante :

```
void nomMethode() ;
```

Les méthodes callbacks

- les méthodes doivent respecter la signature suivante :
  `void nomMethode() ;`
- Utilisation d'annotations Java.

Les méthodes callbacks

- les méthodes doivent respecter la signature suivante :
  `void nomMethode() ;`
- Utilisation d'annotations Java.
- méthodes callbacks **applicables aux** `Stateless` :

Les méthodes callbacks

- les méthodes doivent respecter la signature suivante :
  `void nomMethode() ;`
- Utilisation d'annotations Java.
- méthodes callbacks **applicables aux** `Stateless` :
  - `javax.annotation.PostConstruct` : automatiquement appelé après initialisation du composant.

Les méthodes callbacks

- les méthodes doivent respecter la signature suivante :
  `void nomMethode() ;`
- Utilisation d'annotations Java.
- méthodes callbacks **applicables aux** `Stateless` :
  - `javax.annotation.PostConstruct` : automatiquement appelé après initialisation du composant.
  - `javax.annotation.PreDestroy` : automatiquement appelé avant suppression du composant.

Les composants session avec état

- annotés par `javax.ejb.Stateful`

Les composants session avec état

- annotés par `javax.ejb.Stateful`
- après la première invocation de méthode, le composant est associé au client.

Les composants session avec état

- annotés par `javax.ejb.Stateful`
- après la première invocation de méthode, le composant est associé au client.
- Possibilité d'utiliser des variables d'instances entre deux appels de méthodes.

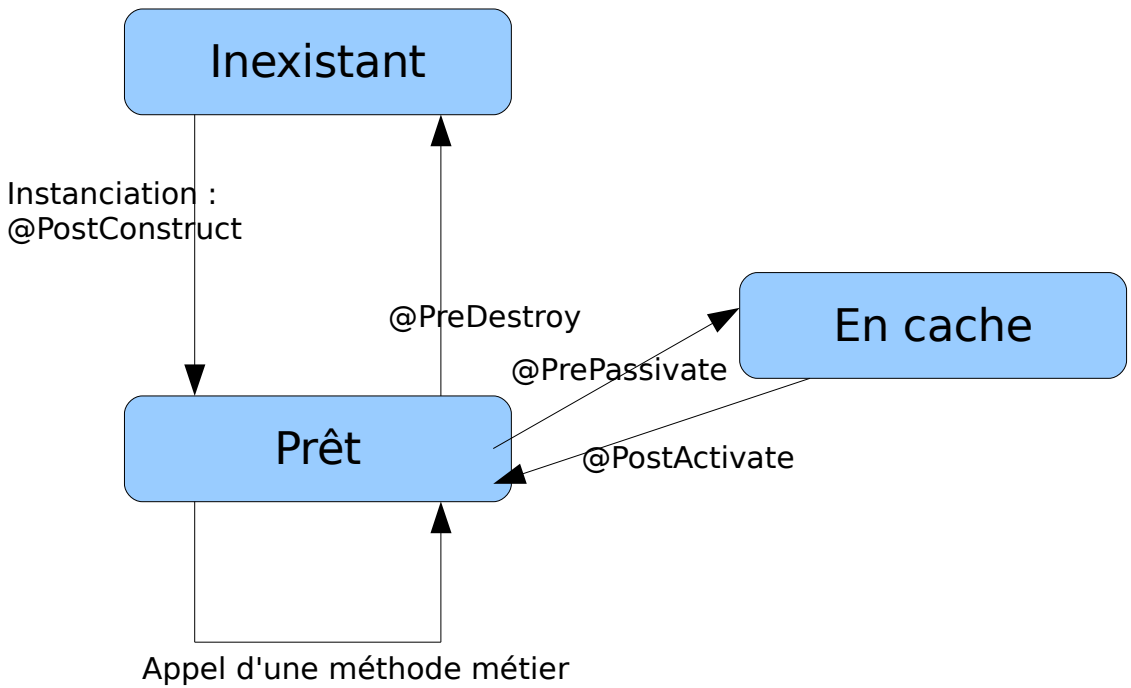
Les composants session avec état

- annotés par `javax.ejb.Stateful`
- après la première invocation de méthode, le composant est associé au client.
- Possibilité d'utiliser des variables d'instances entre deux appels de méthodes.
- Le conteneur peut décider de mettre le composant en mémoire secondaire (montée en charge) : mécanisme de passivation et activation.

Les composants session avec état

- annotés par `javax.ejb.Stateful`
- après la première invocation de méthode, le composant est associé au client.
- Possibilité d'utiliser des variables d'instances entre deux appels de méthodes.
- Le conteneur peut décider de mettre le composant en mémoire secondaire (montée en charge) : mécanisme de passivation et activation.
- Une méthode annotée par `javax.ejb.Remove` notifie le conteneur que le client n'a plus besoin du bean.

Cycle de vie des composants session avec état



Les méthodes callbacks applicables aux composants `Stateful`

- `javax.annotation.PostConstruct` et `javax.annotation.PreDestroy`

Les méthodes callbacks applicables aux composants `Stateful`

- `javax.annotation.PostConstruct` et `javax.annotation.PreDestroy`
- `javax.ejb.PrePassivate` : méthode automatiquement appelée avant la passivation du bean (charge mémoire)

Les méthodes callbacks applicables aux composants `Stateful`

- `javax.annotation.PostConstruct` et `javax.annotation.PreDestroy`
- `javax.ejb.PrePassivate` : méthode automatiquement appelée avant la passivation du bean (charge mémoire)
- `javax.ejb.PostActivate` : méthode automatiquement appelée après la restauration du bean

Exemple d'un composant Stateful (1/2)

```
package ocaron.exempleStateful ;

import javax.ejb.Stateful ;
import javax.annotation.PostConstruct ;
import javax.annotation.PreDestroy ;
import javax.ejb.PrePassivate ;
import javax.ejb.PostActivate ;
import javax.ejb.Remove ;

@Stateful public class SalaireBean
  implements SalaireInterfaceRemote, SalaireInterfaceLocal {
  private double tauxHoraire=8.03 ;

  public SalaireBean() {}

  public void setTauxHoraire(double taux) {
    this.tauxHoraire=taux ;
  }
```

Exemple d'un composant Stateful (2/2)

```
public double getTauxHoraire() { return this.tauxHoraire ; }

public double getSalaire(int nbreHeures) {
  return this.getTauxHoraire()*nbreHeures ;
}
@PostActivate public void postActivate() {
  System.out.println("PostActivate...") ; }
@PrePassivate public void prePassivate() {
  System.out.println("PrePassivate...") ; }
@PostConstruct public void postConstruct() {
  System.out.println("PostConstruct...") ; }
@PreDestroy public void preDestroy() {
  System.out.println("PreDestroy...") ; }
@Remove public void remove() {
  System.out.println("Remove...") ; }
}
```

Déploiement : la sécurité

- Notion de rôle (ou profil d'utilisation)

Déploiement : la sécurité

- Notion de rôle (ou profil d'utilisation)
- Pour chaque rôle, il faut préciser la ou les méthodes utilisables.

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

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 ou annotations java incluent 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.

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 :

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 maniére globale.

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 maniére globale.
  - Cohérent : Avant et après une transaction (réussie ou pas), l'état de la base doit être cohérente.

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 maniére 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.

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 maniére 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) :

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.

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`

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.

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.

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

```
import javax.ejb.Stateful ;
import javax.ejb.TransactionManagement ;
import javax.ejb.TransactionAttribute ;
import javax.ejb.TransactionManagementType ;
import javax.ejb.TransactionAttributeType ;
@Stateful
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class SalaireBean
  implements SalaireInterfaceRemote, SalaireInterfaceLocal {
  ...
  @TransactionAttribute(value=TransactionAttributeType.REQUIRED)
  public void setTauxHoraire(double taux) { this.tauxHoraire=taux ; }
  @TransactionAttribute(value=TransactionAttributeType.REQUIRES_NEW)
  public double getTauxHoraire() { return this.tauxHoraire ; }
}
```

Les composants EJB entité

- EJB entité et JPA ne font qu'un !

Les composants EJB entité

- EJB entité et JPA ne font qu'un !
- Programmer des composants entités :

Les composants EJB entité

- EJB entité et JPA ne font qu'un !
- Programmer des composants entités :
  - voir cours GIS 4 - S7 - SIO

Les composants EJB entité

- EJB entité et JPA ne font qu'un !
- Programmer des composants entités :
  - voir cours GIS 4 - S7 - SIO
  - `docs.oracle.com/javaee/7/tutorial/doc`

Les composants EJB entité

- EJB entité et JPA ne font qu'un !
- Programmer des composants entités :
  - voir cours GIS 4 - S7 - SIO
  - `docs.oracle.com/javaee/7/tutorial/doc`
- Différence entre un programme JPA autonome et JEE : le conteneur EJB :

Les composants EJB entité

- EJB entité et JPA ne font qu'un !
- Programmer des composants entités :
  - voir cours GIS 4 - S7 - SIO
  - `docs.oracle.com/javaee/7/tutorial/doc`
- Différence entre un programme JPA autonome et JEE : le conteneur EJB :
  - Se charge des transactions

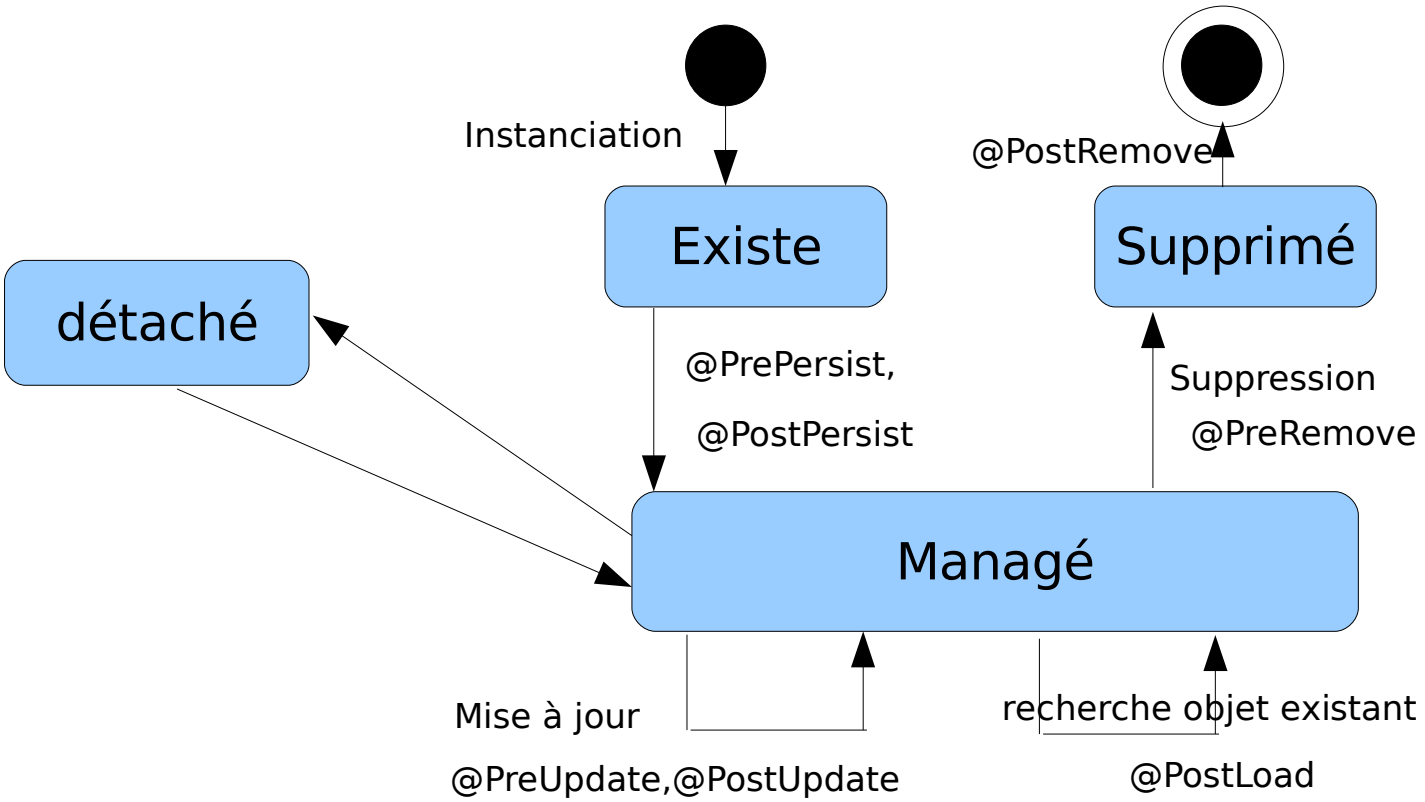
Les composants EJB entité

- EJB entité et JPA ne font qu'un !
- Programmer des composants entités :
  - voir cours GIS 4 - S7 - SIO
  - `docs.oracle.com/javaee/7/tutorial/doc`
- Différence entre un programme JPA autonome et JEE : le conteneur EJB :
  - Se charge des transactions
  - Se charge du gestionnaire d'entités (instanciation)

Les composants EJB entité

- EJB entité et JPA ne font qu'un !
- Programmer des composants entités :
  - voir cours GIS 4 - S7 - SIO
  - `docs.oracle.com/javaee/7/tutorial/doc`
- Différence entre un programme JPA autonome et JEE : le conteneur EJB :
  - Se charge des transactions
  - Se charge du gestionnaire d'entités (instanciation)
  - Attention au cycle de vie des composants selon que l'objet "quitte" ou pas la sphère du conteneur.

Cycle de vie des composants entités (1/3)



Cycle de vie des composants entités (2/3)

- Possibilité de spécifier des méthodes ”callbacks” liées au cycle de vie des composants entités :

Cycle de vie des composants entités (2/3)

- Possibilité de spécifier des méthodes ”callbacks” liées au cycle de vie des composants entités :
  - `javax.persistence.PrePersist` : la méthode annotée est invoquée juste avant la création de l'entité dans la base de donnée.

Cycle de vie des composants entités (2/3)

- Possibilité de spécifier des méthodes ”callbacks” liées au cycle de vie des composants entités :
  - `javax.persistence.PrePersist` : la méthode annotée est invoquée juste avant la création de l'entité dans la base de donnée.
  - `javax.persistence.PostPersist` : la méthode annotée est invoquée juste après la création de l'entité dans la base de donnée.

Cycle de vie des composants entités (2/3)

- Possibilité de spécifier des méthodes ”callbacks” liées au cycle de vie des composants entités :
  - `javax.persistence.PrePersist` : la méthode annotée est invoquée juste avant la création de l'entité dans la base de donnée.
  - `javax.persistence.PostPersist` : la méthode annotée est invoquée juste après la création de l'entité dans la base de donnée.
  - `javax.persistence.PreRemove` : la méthode annotée est invoquée juste avant la destruction de l'entité dans la base de donnée.

Cycle de vie des composants entités (2/3)

- Possibilité de spécifier des méthodes ”callbacks” liées au cycle de vie des composants entités :
  - `javax.persistence.PrePersist` : la méthode annotée est invoquée juste avant la création de l'entité dans la base de donnée.
  - `javax.persistence.PostPersist` : la méthode annotée est invoquée juste après la création de l'entité dans la base de donnée.
  - `javax.persistence.PreRemove` : la méthode annotée est invoquée juste avant la destruction de l'entité dans la base de donnée.
  - `javax.persistence.PostRemove` : la méthode annotée est invoquée juste après la destruction de l'entité dans la base de donnée.

Cycle de vie des composants entités (3/3)

- `javax.persistence.PreUpdate` : la méthode annotée est invoquée juste avant la modification de l'entité dans la base de donnée.

Cycle de vie des composants entités (3/3)

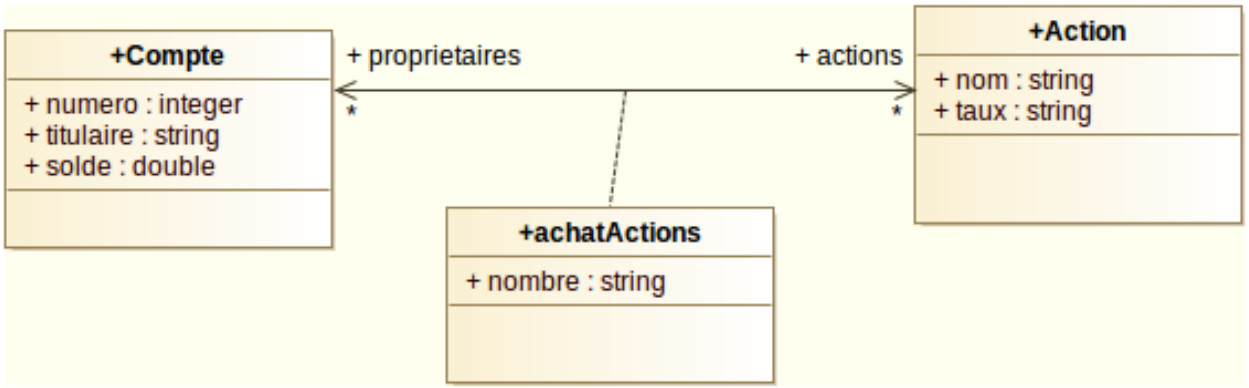
- `javax.persistence.PreUpdate` : la méthode annotée est invoquée juste avant la modification de l'entité dans la base de donnée.
- `javax.persistence.PostUpdate` : la méthode annotée est invoquée juste après la modification de l'entité dans la base de donnée.

Cycle de vie des composants entités (3/3)

- `javax.persistence.PreUpdate` : la méthode annotée est invoquée juste avant la modification de l'entité dans la base de donnée.
- `javax.persistence.PostUpdate` : la méthode annotée est invoquée juste après la modification de l'entité dans la base de donnée.
- `PostLoad` : la méthode annotée est invoquée juste après le chargement des données de la base dans l'entité associée.

Illustration : application banque (1/2)

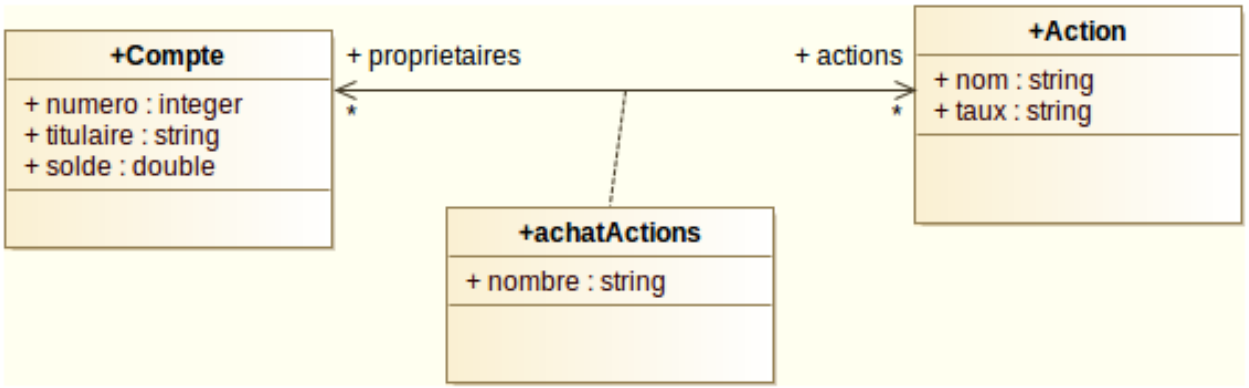
Figure: La modélisation UML des données



- Sources Java dans `ocaron.plil.fr`

Illustration : application banque (1/2)

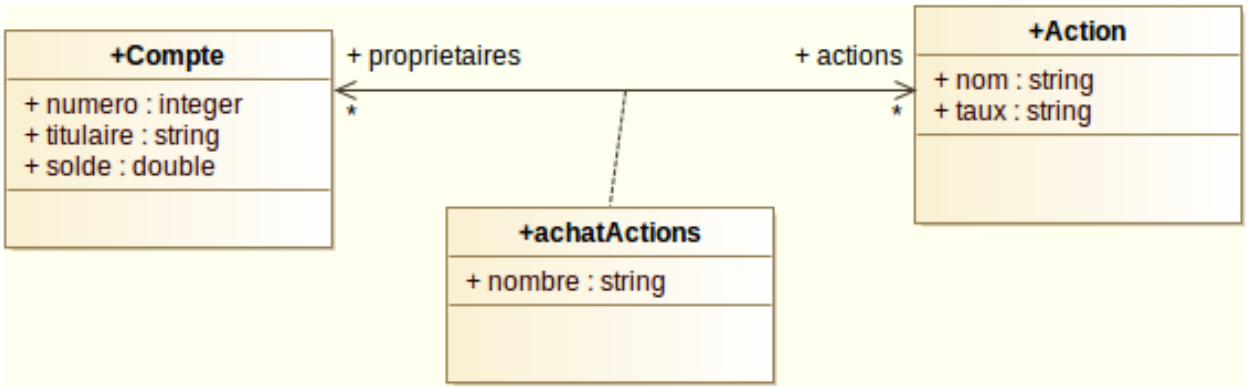
Figure: La modélisation UML des données



- Sources Java dans `ocaron.plil.fr`
- modélisation UML non directement exploitable pour EJB

Illustration : application banque (1/2)

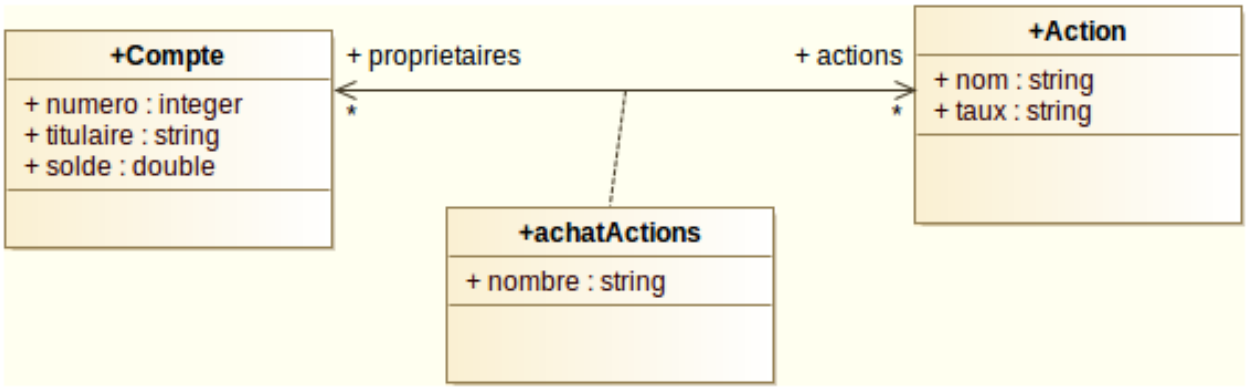
Figure: La modélisation UML des données



- Sources Java dans `ocaron.plil.fr`
- modélisation UML non directement exploitable pour EJB
  - Uniquement associations binaires

Illustration : application banque (1/2)

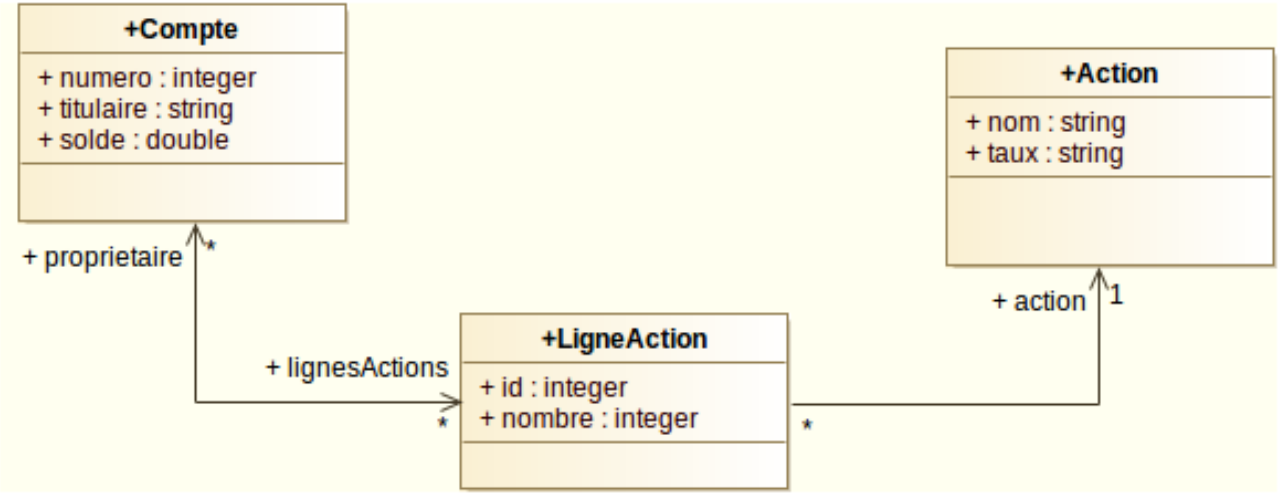
Figure: La modélisation UML des données



- Sources Java dans `ocaron.plil.fr`
- modélisation UML non directement exploitable pour EJB
  - Uniquement associations binaires
  - Pas de classes-associations (propriétés des associations)

Illustration : application banque (2/2)

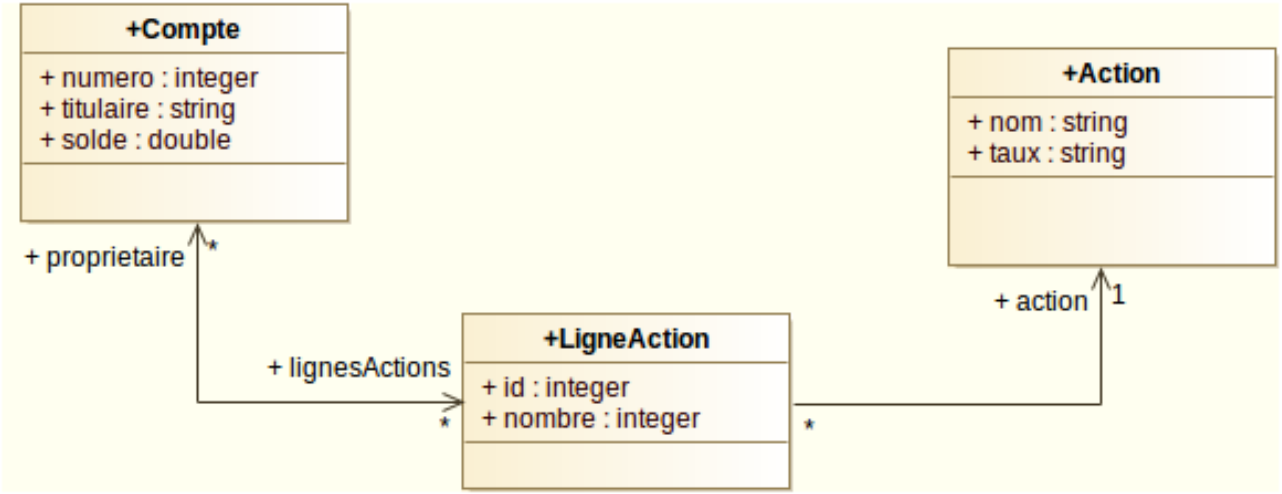
Figure: La modélisation **UML-EJB** des données



- Ajout d'une entité

Illustration : application banque (2/2)

Figure: La modélisation **UML-EJB** des données



- Ajout d'une entité
- Choix d'un attribut `id` mais clé-composé possible

Archivage des composants entités

```
mesEntites/
  META-INF/
    persistence.xml
  ejb/
    entites/
      Compte.class
      Action.class
      LigneAction.class
jar cvf mesEntites.jar
    mesEntites/*
```

Fichier persistence.xml :

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
___http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
  <persistence-unit name="appliBanque">
    <jta-data-source>
      java:jboss/datasources/PostgresDS
    </jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto"
                value="create-drop" />
    </properties>
  </persistence-unit>
</persistence>
```

- Possibilité de définir plusieurs sources BD (**spécifiées au niveau du serveur**)
- Définition d'une unité de persistance

Classe Compte

```
package ejb.entites;

import ...

@Entity public class Compte implements Serializable {
   ...
   @Id public int getNumero() { return numero; }
   public double getSolde() { return solde; }
   public String getTitulaire() { return titulaire; }

   @OneToMany(mappedBy="proprietaire")
   public Set<LigneAction> getLignesActions() { return actions;}

   // méthodes set
   ...
}
```

Classe Action

```
package ejb.entites;

import ...

@Entity public class Action implements Serializable {
   ...
   @Id public String getNom() { return nom; }
   public double getTaux() { return taux; }

   // méthodes set
   ...
}
```

Classe LigneAction

```
package ejb.entites;

import ...

@Entity public class LigneAction implements Serializable {
   ...
   @Id
   @GeneratedValue // permet de générer une valeur de clé
   public int getId() { return id; }
   public int getNombre() { return nombre; }

   @ManyToOne public Compte getProprietaire() {  ... }
   @ManyToOne public Action getAction() {  ... }
   // méthodes set
   ...
}
```

Définition du ou des services

- Services assuré par composants sessions

Définition du ou des services

- Services assuré par composants sessions
- Choix du type de session (Stateless, Stateful)

Définition du ou des services

- Services assuré par composants sessions
- Choix du type de session (Stateless, Stateful)
- Stateless : une méthode devient un programme (POO ?)

Définition du ou des services

- Services assuré par composants sessions
- Choix du type de session (Stateless, Stateful)
- Stateless : une méthode devient un programme (POO ?)
- Quelles sont les fonctions accessibles à distance ou localement ?

Définition du ou des services

- Services assuré par composants sessions
- Choix du type de session (Stateless, Stateful)
- Stateless : une méthode devient un programme (POO ?)
- Quelles sont les fonctions accessibles à distance ou localement ?
- Pour l'exemple : un seul service accessible à distance

L'interface distante (1/2)

```
package ejb.sessions;
import ejb.entites.LigneAction ;

@javax.ejb.Remote
public interface ServiceBanque {
  public void addCompte(int numeroCompte, String nomTitulaire,
                        double soldeDepart)
    throws CompteDejaExistantException ;
  public void addAction(String nomAction, double taux)
        throws ActionDejaExistanteException ;
  public void crediterCompte(int numeroCompte, double montant)
    throws CompteInconnuException ;
  public void debiterCompte(int numeroCompte, double montant)
    throws CompteInconnuException ;
```

L'interface distante (2/2)

```
    public void virementVers(int numCompteDebit, int numCompteCredit,
                             double montant)
      throws CompteInconnuException,ApprovisionnementException ;
    public void acheteActions(int numeroCompte, String nomAction, int nb)
      throws CompteInconnuException, ActionInconnueException,
             ApprovisionnementException ;
    public void vendActions(int numeroCompte, String nomAction, int nb)
      throws CompteInconnuException, ActionInconnueException,
      ApprovisionnementException ;
    public java.util.Set<LigneAction> getActionsAchetees(int numeroCompte)
      throws CompteInconnuException ;
}
```

Exploitation des composants d'entités

- Les entités ne sont accessibles que dans un contexte transactionnel (c'est le cas des composants sessions)

Exploitation des composants d'entités

- Les entités ne sont accessibles que dans un contexte transactionnel (c'est le cas des composants sessions)
- Un gestionnaire d'entités permet de relier les objets Java aux objets bases de données et de gérer leur cycle de vie.

Exploitation des composants d'entités

- Les entités ne sont accessibles que dans un contexte transactionnel (c'est le cas des composants sessions)
- Un gestionnaire d'entités permet de relier les objets Java aux objets bases de données et de gérer leur cycle de vie.
- Une instance de gestionnaire d'entités est associée avec un contexte de persistance (`persistence.xml`) et donc une seule base.

Exploitation des composants d'entités

- Les entités ne sont accessibles que dans un contexte transactionnel (c'est le cas des composants sessions)
- Un gestionnaire d'entités permet de relier les objets Java aux objets bases de données et de gérer leur cycle de vie.
- Une instance de gestionnaire d'entités est associée avec un contexte de persistance (`persistence.xml`) et donc une seule base.
- Une instance de gestionnaire d'entités peut gérer plusieurs EJB entités.

```
@PersistenceContext(unitName="appliBanque")
protected EntityManager em;
```

Le composant session (1/3)

```
package ejb.sessions;

import ...

@javax.ejb.Stateless
public class ServiceBanqueBean implements ServiceBanque {

  @PersistenceContext(unitName="appliBanque")
  protected EntityManager em ;

  public ServiceBanqueBean() { }
```

Le composant session (2/3)

```
  public Set<LigneAction> getActionsAchetees(int numeroCompte)
    throws CompteInconnuException {
    Compte c = this.getCompte(numeroCompte) ;
    return c.getLignesActions() ;
  }

  private Compte getCompte(int numeroCompte)
    throws CompteInconnuException {
    Compte c= (Compte) em.find(Compte.class, numeroCompte) ;
    if (c==null) throw new CompteInconnuException() ;
    return c;
  }
  ...
}
```

Le composant session (3/3)

```
public void addCompte(int numeroCompte, String nomTitulaire,
                      double soldeDepart)
  throws CompteDejaExistantException {
  try {
    this.getCompte(numeroCompte) ;
    throw new CompteDejaExistantException() ;
  } catch(CompteInconnuException e) {
    Compte c=new Compte() ;
    c.setNumero(numeroCompte); c.setSolde(soldeDepart);
    c.setTitulaire(nomTitulaire);
    em.persist(c);
  }
}  // Par défaut: une méthode équivaut à une transaction
```

Utilisation du service

```
1  public static void main(String[] args) {
2    try {
3      InitialContext ctx = new InitialContext();
4      System.out.println("Accès au service distant") ;
5      Object obj = ctx.lookup("ejb:appliBanque/appliBanqueSessions//"+
6                   "ServiceBanqueBean!ejb.sessions.ServiceBanque");
7      ServiceBanque service = (ServiceBanque) obj ;
8      System.out.println("les␣actions␣du␣compte␣no:␣1") ;
9      for (LigneAction la : service.getActionsAchetees(1))
10       System.out.println(la.getNombre()+"␣action(s)␣"+
11                          la.getAction().getNom()+
12                          "␣au␣taux␣de␣"+la.getAction().getTaux()) ;
13     ...
```

Mode de chargement des instances associées

- Le précédent exemple ne fonctionne pas (ligne 9), pourquoi ?

Mode de chargement des instances associées

- Le précédent exemple ne fonctionne pas (ligne 9), pourquoi ?
  - Les instances de `LigneAction` sont des objets détachés

Mode de chargement des instances associées

- Le précédent exemple ne fonctionne pas (ligne 9), pourquoi ?
  - Les instances de `LigneAction` sont des objets détachés
  - le mode par défaut d'obtention des instances d'entité en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !

Mode de chargement des instances associées

- Le précédent exemple ne fonctionne pas (ligne 9), pourquoi ?
  - Les instances de `LigneAction` sont des objets détachés
  - le mode par défaut d'obtention des instances d'entité en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !
- Deux solutions :

Mode de chargement des instances associées

- Le précédent exemple ne fonctionne pas (ligne 9), pourquoi ?
  - Les instances de `LigneAction` sont des objets détachés
  - le mode par défaut d'obtention des instances d'entité en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !
- Deux solutions :
  1. Forcer le chargement des objets avant l'envoi des données (méthode `getActionsAchetees`)

Mode de chargement des instances associées

- Le précédent exemple ne fonctionne pas (ligne 9), pourquoi ?
  - Les instances de `LigneAction` sont des objets détachés
  - le mode par défaut d'obtention des instances d'entité en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !
- Deux solutions :
  1. Forcer le chargement des objets avant l'envoi des données (méthode `getActionsAchetees`)
  2. Fixer le mode d'acquisition des données en mode non paresseux (`EAGER`)

Version opérationnelle

```
// fichier Compte.java
...
@OneToMany(mappedBy="proprietaire", fetch=FetchType.EAGER)
        public Set<LigneAction> getLignesActions() { ... }
```

```
// fichier LigneAction.java
...
@ManyToOne(fetch=FetchType.EAGER)
public Action getAction() { ... }
```

Session Stateful et JSP (1/2)

- Problème :

Session Stateful et JSP (1/2)

- Problème :
  - Les composants `Stateful` permettent d'établir une session avec leur client.

Session Stateful et JSP (1/2)

- Problème :
  - Les composants `Stateful` permettent d'établir une session avec leur client.
  - Les JSP reposent sur une technologie (http) non orientée session.

Session Stateful et JSP (1/2)

- Problème :
  - Les composants `Stateful` permettent d'établir une session avec leur client.
  - Les JSP reposent sur une technologie (http) non orientée session.
- Solution :

  Programmer la session dans le JSP

Session Stateful et JSP (2/2)

```
<%
  MonStateful composant ;
  composant = (MonStateful) session.getAttribute("unNom") ;
  if (composant==null) {
    try {
      InitialContext ctx=new InitialContext() ;
      String nomJNDI = "java:app/mesSessions/Comp!ILocal" ;
      composant=(MonStateful) ctx.lookup(nomJNDI);
      session.setAttribute("unNom",composant) ;
   } catch ...
```

Message Driven Bean (1/3)

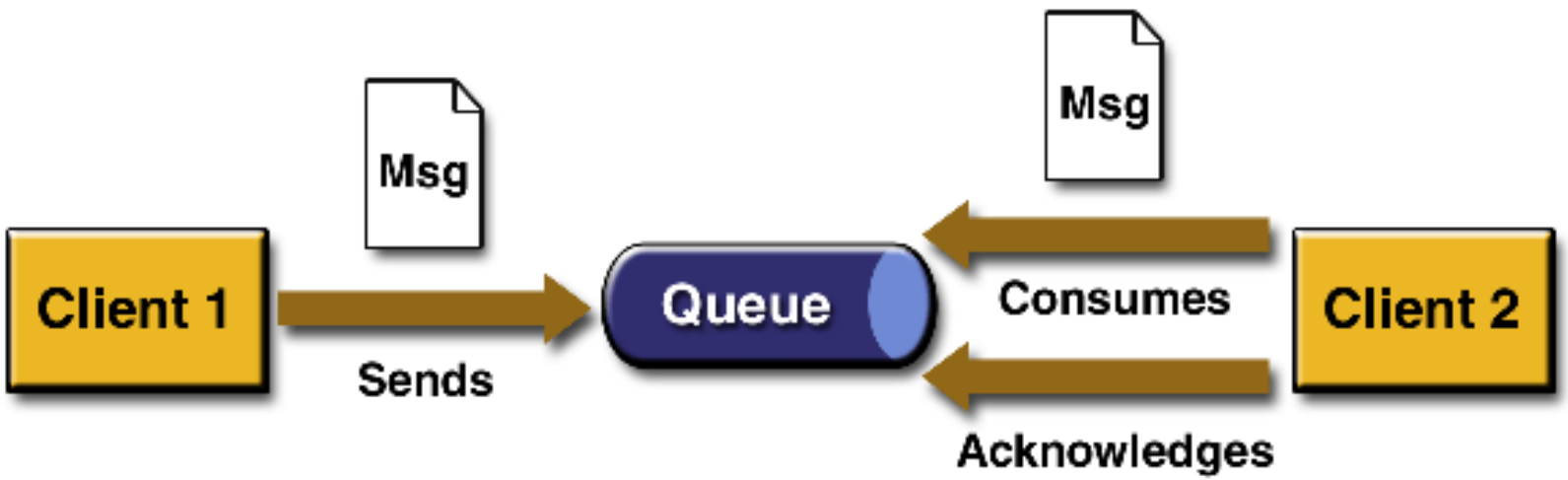
- Envoi de messages asynchrones

Message Driven Bean (1/3)

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

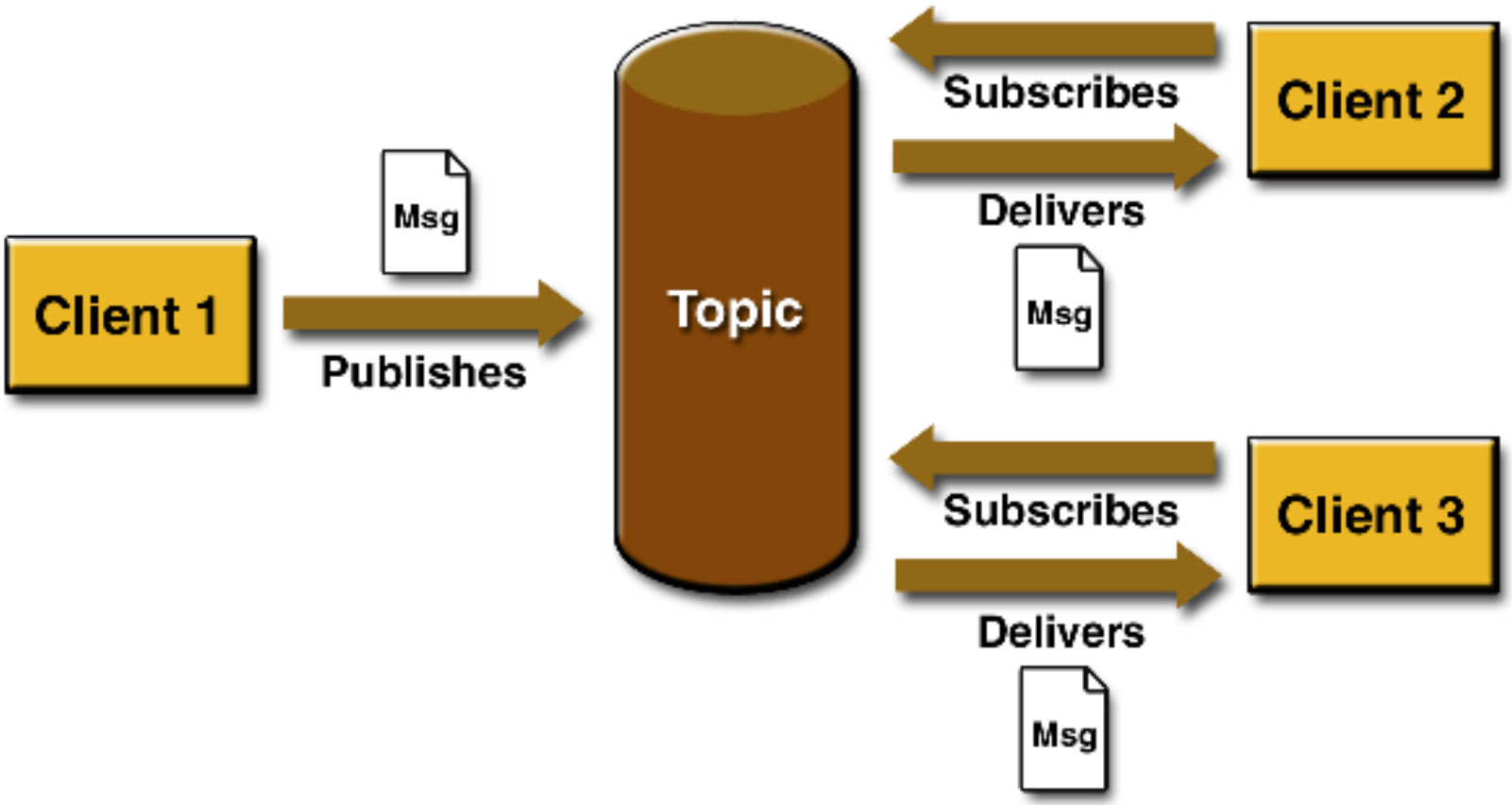
Message Driven Bean (2/3)

- Liaison Point à point :



Message Driven Bean (3/3)

- Diffusion multiple :



Conclusion

- Quelques aspects non abordés :

Conclusion

- Quelques aspects non abordés :
  - Possibilité de programmer des intercepteurs, Introduction à la programmmation par aspects.

Conclusion

- Quelques aspects non abordés :
  - Possibilité de programmer des intercepteurs, Introduction à la programmmation par aspects.
  - Référencement d'EJB

Conclusion

- Quelques aspects non abordés :
  - Possibilité de programmer des intercepteurs, Introduction à la programmmation par aspects.
  - Référencement d'EJB
  - Interaction avec le conteneur

Conclusion

- Quelques aspects non abordés :
  - Possibilité de programmer des intercepteurs, Introduction à la programmmation par aspects.
  - Référencement d'EJB
  - Interaction avec le conteneur
- Avantages :

Conclusion

- Quelques aspects non abordés :
  - Possibilité de programmer des intercepteurs, Introduction à la programmmation par aspects.
  - Référencement d'EJB
  - Interaction avec le conteneur
- Avantages :
  - Simplicité de programmation

Conclusion

- Quelques aspects non abordés :
  - Possibilité de programmer des intercepteurs, Introduction à la programmmation par aspects.
  - Référencement d'EJB
  - Interaction avec le conteneur
- Avantages :
  - Simplicité de programmation
  - Exploite les caractéristiques Java 1.5

Conclusion

- Quelques aspects non abordés :
  - Possibilité de programmer des intercepteurs, Introduction à la programmmation par aspects.
  - Référencement d'EJB
  - Interaction avec le conteneur
- Avantages :
  - Simplicité de programmation
  - Exploite les caractéristiques Java 1.5
  - Support idéal pour des applications Java réseaux, 3-tiers, sécurisées, persistantes et transactionnelles.

Le mot de la fin

© Auteur inconnu

Il y a 10 sortes de personnes, ceux qui comprennent le binaire et ceux qui ne le comprennent pas.