J2EE

Java2 Enterprise Edition

Pierre-Yves Gibello - pierreyves.gibello@experlog.com

Mise à jour : 30 Septembre 2004

J2EE - Objectifs

- Faciliter le développement de nouvelles applications à base de composants
- Intégration avec les systèmes d'information existants
- Support pour les applications « critiques » de l'entreprise
  - Disponibilité, tolérance aux pannes, montée en charge, securité ...

J2EE - C ’est quoi?

- http://java.sun.com/j2ee
- Spécifications
- Modèle de programmation
- Implémentation de référence
- Suite(s) de tests
- Label J2EE Sun (qualification de plateformes)

Offre commerciale

- BEA WebLogic (haut de gamme)
- IBM Websphere (no 1)
- Sun Java System App. Server
- Borland Enterprise Server
- Oracle 9i Application Server
- Macromedia jRun
- SAP Web application server
- Iona Orbix E2A

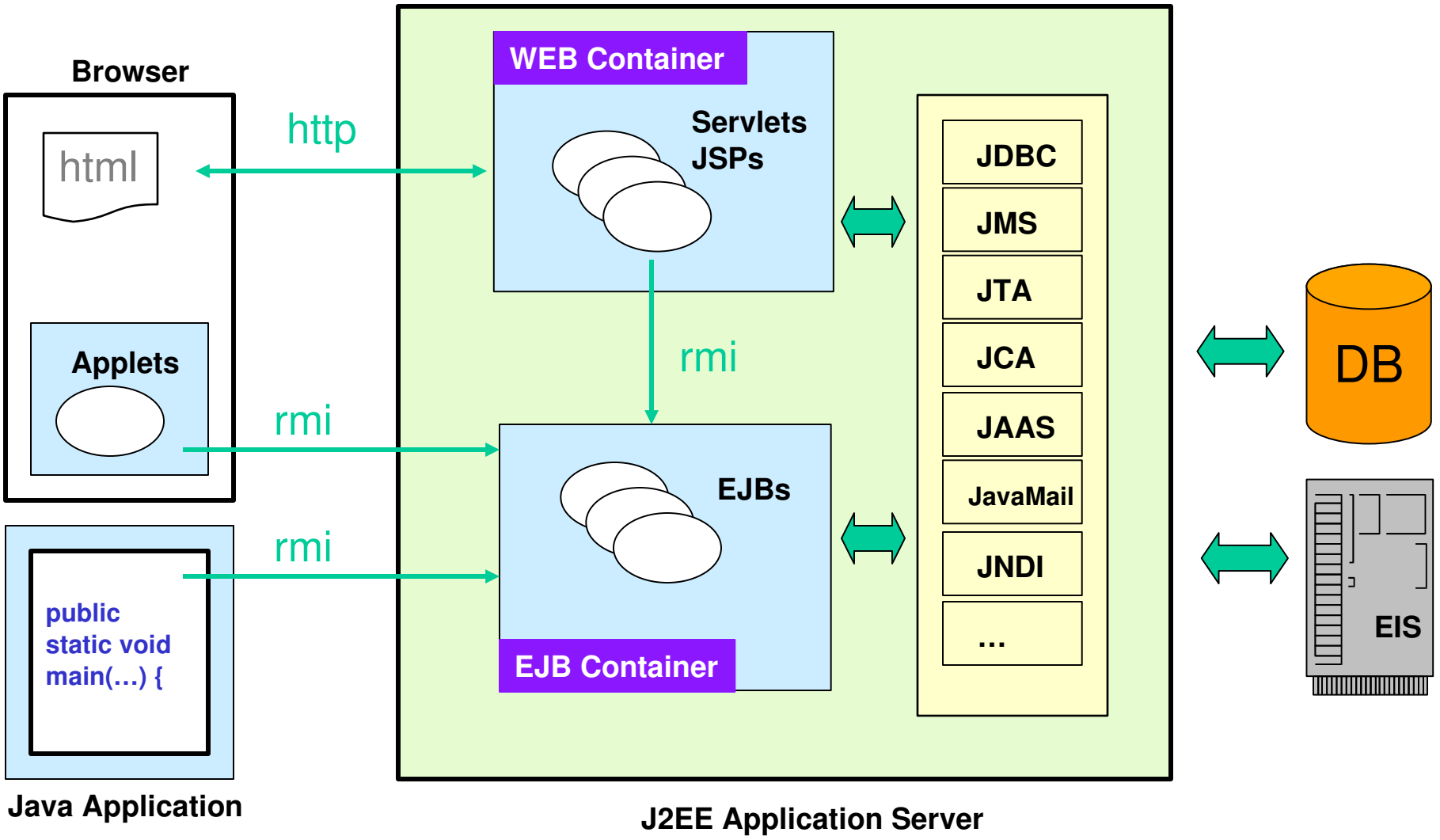
Offre open-source

- JBoss (no 1 en nombre de déploiements)
- ObjectWeb JOnAS(no 2, intégré à plusieurs distro Linux Entreprise)
- Apache Geronimo (démarrage fin 2003)
- openEjb
- ejBean

J2EE sous l ’œil de Darwin...

- Standard en évolution depuis 1997
  - J2EE 1.0 à 1.4 en 2003, etc...
- Au départ, applications Web n-tiers
  - Présentation (Servlets puis JSP), essentiellement HTTP
  - Logique métier : EJB
  - Données : JDBC
- Puis infrastructure de support standard pour EAI
  - Facteurs de rationnalisation majeurs (JTA, JMS, JCA, Web Services)
  - Evolution de progiciels existants vers J2EE

J2EE - Architecture

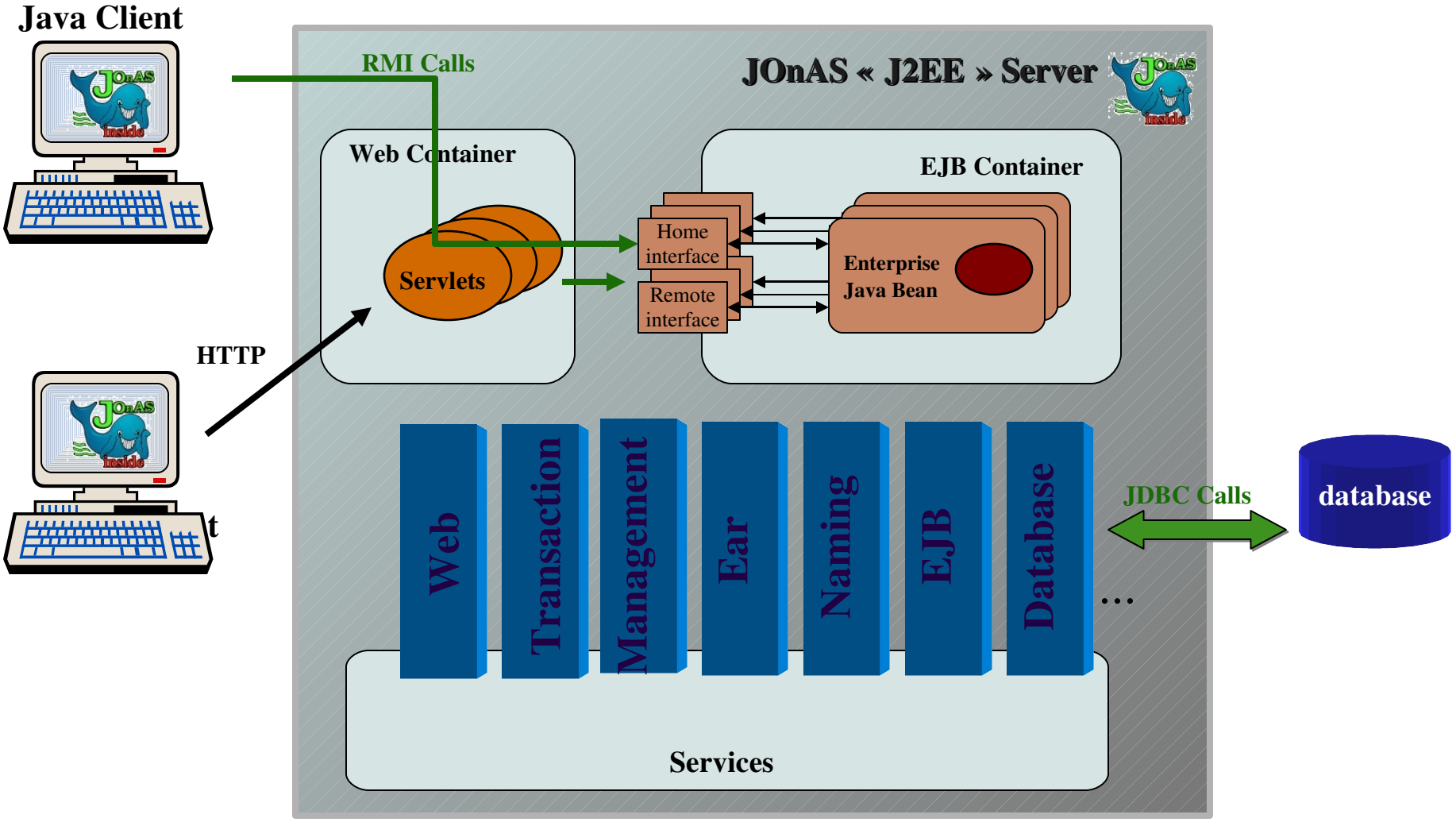


Architecture multi-tiers

- Client
  - Léger (Web, browser)
  - Lourd (Application java, Applet…)
  - Architecture orientée service (Application répartie sans présentation)
- Serveur d ’applications
  - Conteneur EJB + logique métier
  - Services non fonctionnels
- EIS ou Base de données

Un serveur J2EE

Source : Bull/ObjectWeb (JOnAS)



Conteneur Web

- Servlets
  Code java exécuté sur le serveur
  Equivalent du CGI
  Génération de contenu Web dynamique
- JSP: Java Server Pages
  Mélange de HTML/XML et de code java
  Librairies d ’extension (« taglibs »)
  Précompilation en servlet

RMI

- Remote Method Invocation
  - Java seulement, mais passerelles
- « RPC objet » (appels sur objets distants)
- Service de nommage (RMI registry)
- Sécurité paramétrable (SecurityManager)
- Garbage Collection distribuée
- Téléchargement de code
- Fonctions avancées
  - Activation d ’objets persistents, Réplication

JNDI

- Service de nommage / annuaire
  - Java Naming and Directory Interface
- API accès aux annuaires
  - javax.naming
  - « Service Provider » par annuaire cible (LDAP, NIS, RMI registry…)
- Utilisation avec les EJB
  - Accès à l ’interface « home » pour initialiser
  - Accès à diverses ressources (UserTransaction, Queues JMS, DataSources…)

JMS

- Java Messaging Service
- JMS Provider : inclus dans J2EE
  - Transport synchrone ou asynchrone, Garantie de livraison
  - « Messaging domains » point à point ou « publish/subscribe »
- Lien avec EJB : « message-driven bean »
  - Pour échanges asynchrones

API J2EE de transactions : JTA

- Java Transaction API
- Package javax.transaction
  - TransactionManager : begin(), commit(), rollback() …
  - Transaction : commit(), rollback(), enlistResource(XAResource), registerSynchronisation(Synchronization) ...
  - Synchronization : beforeCompletion(), afterCompletion(commit | rollback)

JTA : Participants

- XAResource
  - Conformes à la spécification XA
  - Enregistment avec transaction.enlistResource()
- Synchronization
  - Pour les ressources « non transactionnelles » (EAI…)
  - Participant averti des frontières de transaction
  - enregistrement : transaction.registerSynchronization()
  - beforeCompletion() équivaut à prepare()
  - afterCompletion(état = commit | rollback) équivaut à commit | rollback

API XA de J2EE

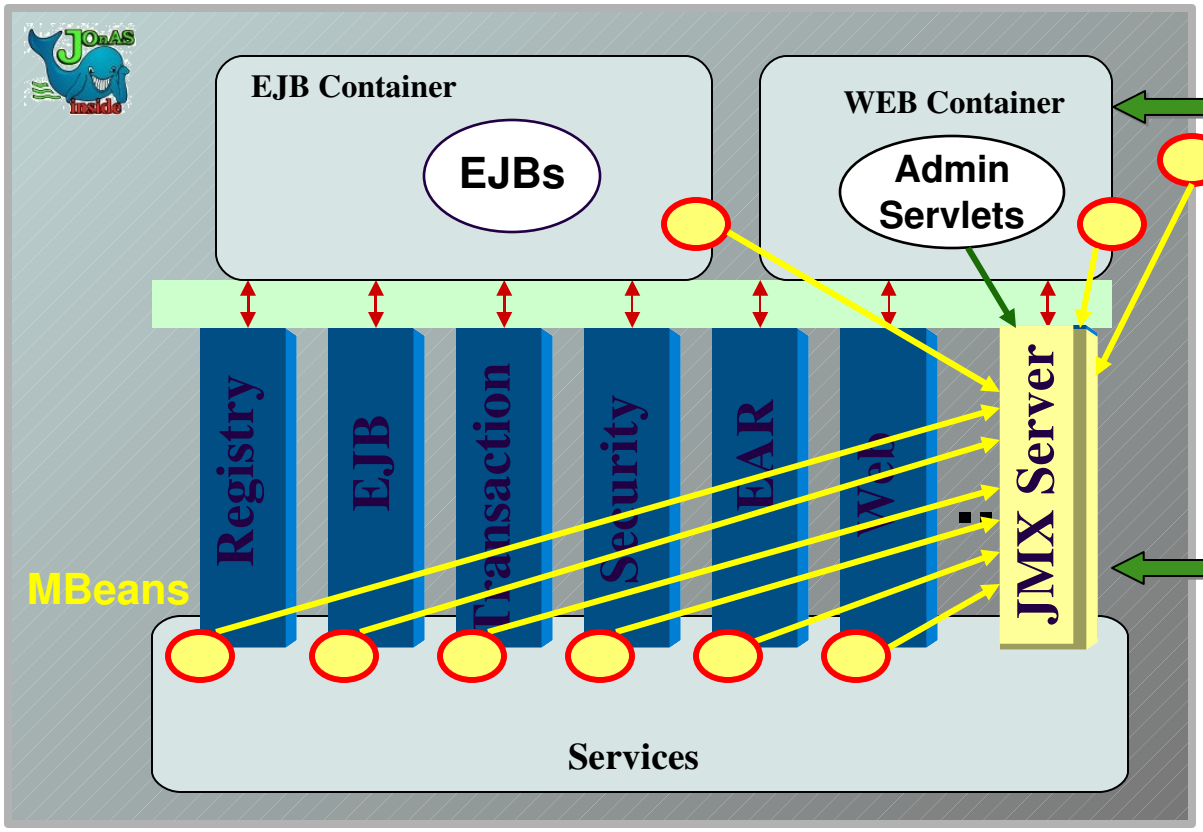
- Package javax.transaction.xa
  - XAResource (prepare(), commit(), rollback()...
  - Xid (identifiant de transaction XA)
- Package javax.sql
  - Extension de JDBC (bases de données)
  - XADataSource (getXAConnection())
  - XAConnection (PooledConnection, avec getConnection() et getXAResource())

JMX

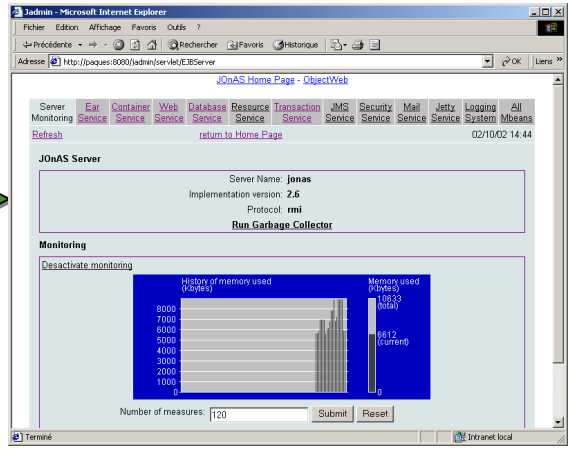
- Java Management eXtensions
  - API unique pour applications de management
- Mbeans avec accesseurs get/set
  - Typage faible, attributs nommés
- Serveur JMX
  - Enregistrement des Mbeans
  - Les applis d ’administration dialoguent avec le serveur JMX
- Instrumenter un composant
  - Fournir un ou des Mbeans
  - Les enregistrer auprès du serveur JMX

JMX : Exemple d ’un serveur J2EE

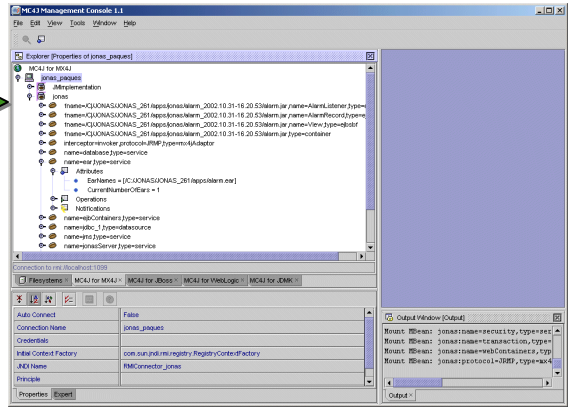
Source : ObjectWeb JOnAS



Admin console



MC4J



EJB: Architecture

- JavaBeans pour l' Enterprise
- Pas des JavaBeans (pas de représentation graphique)
- Logique métier
- S'appuie sur J2SE et les APIs de J2EE
  - JNDI, JTA/JTS, JDBC, JMS , JAAS
- Gestion déclarative (personnalisation sans toucher au code source)
- Portable sur les différents conteneurs EJB

EJB: Gamme de services implicites

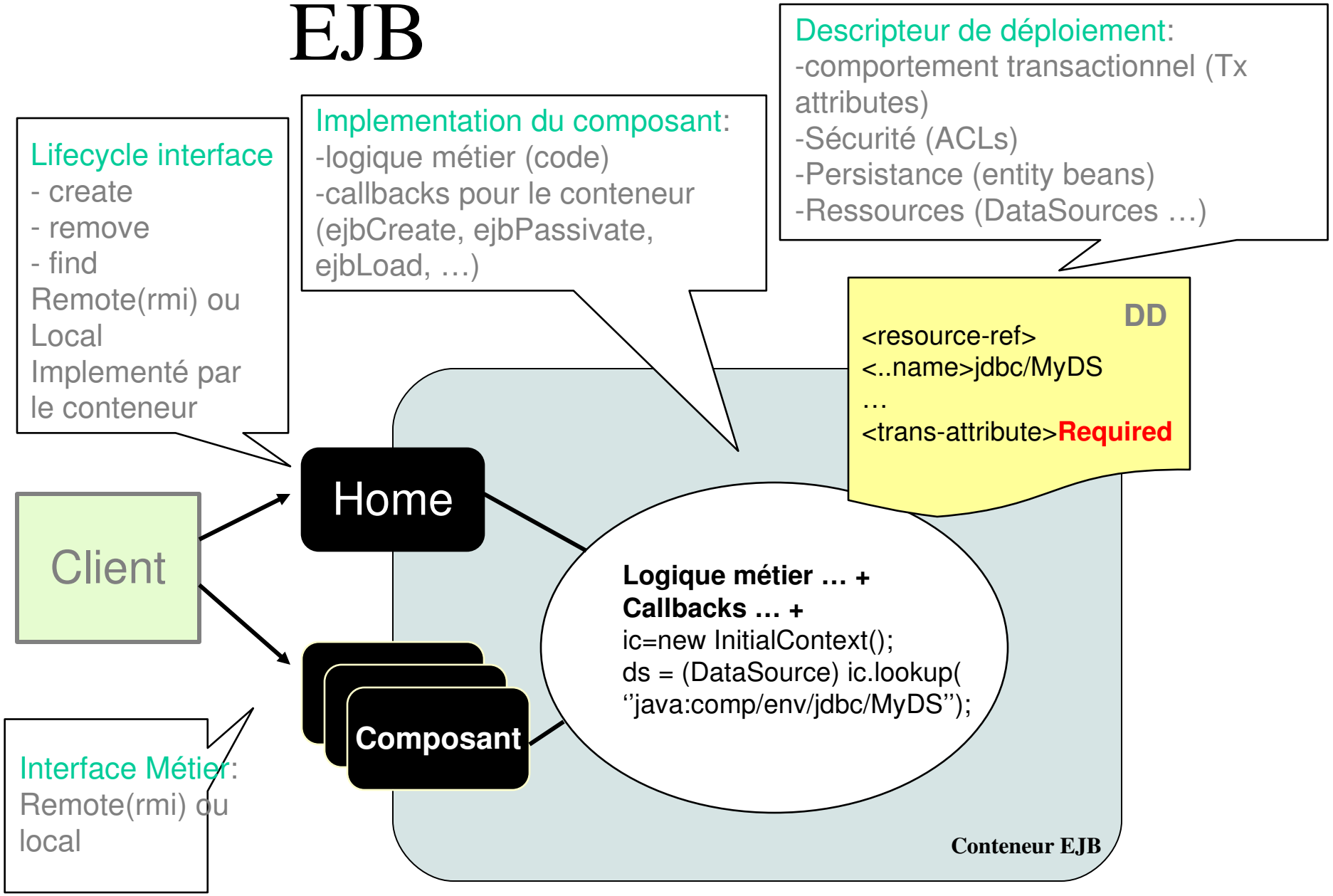
- Gestion du cycle de vie
- Gestion de l'état
- Sécurité
- Transactions
- Persistance
- Localisation des composants transparente

(comparable à objets distribués CORBA)

- Répartition de charge, pooling

=> Le développeur se focalise sur les aspects métier

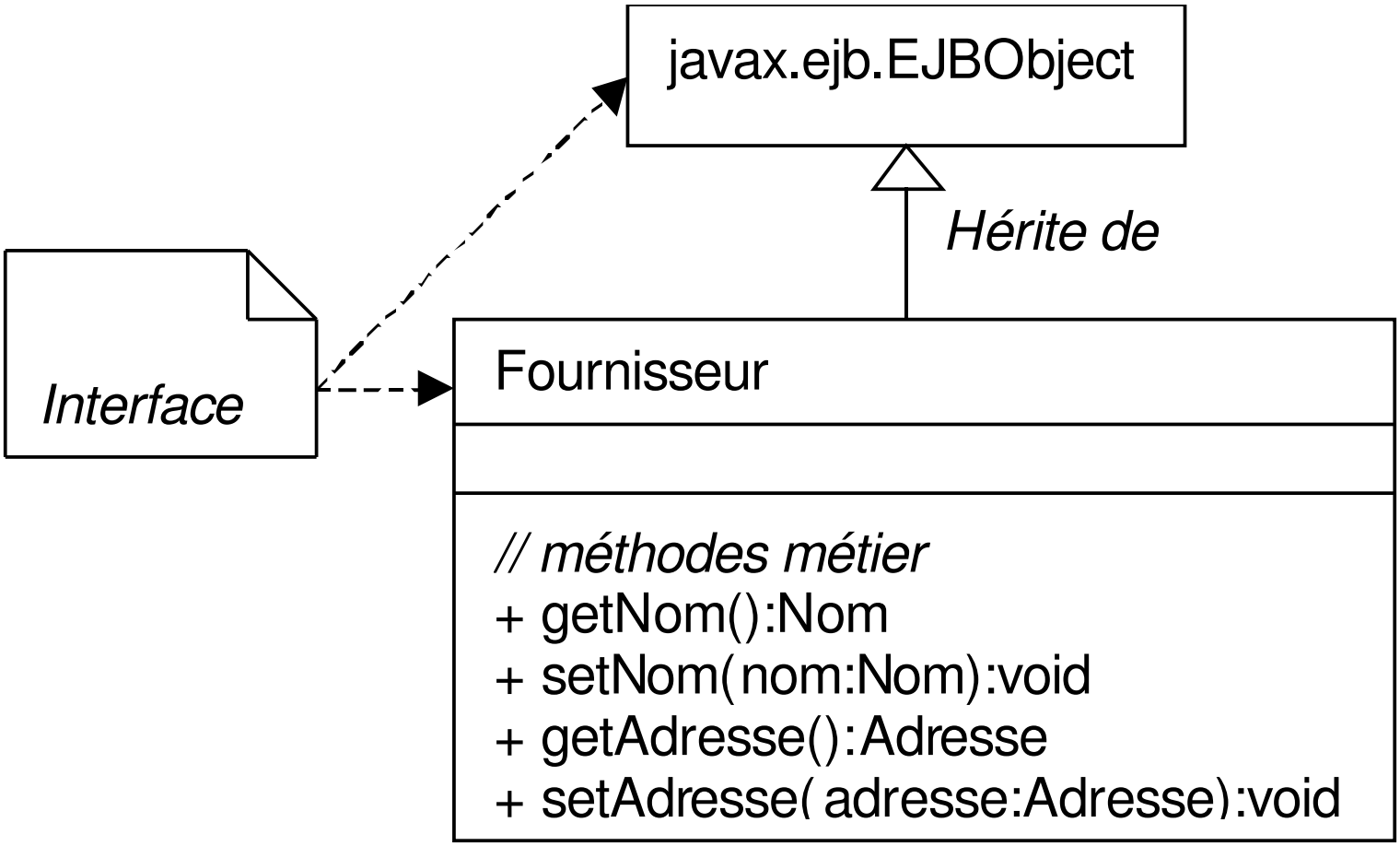
EJB


Descripteur de déploiement:
-comportement transactionnel (Tx attributes)
-Sécurité (ACLs)
-Persistance (entity beans)
-Ressources (DataSources …)
Implementation du composant:
-logique métier (code)
-callbacks pour le conteneur (ejbCreate, ejbPassivate, ejbLoad, …)
Lifecycle interface
- create
- remove
- find
Remote(rmi) ou Local
Implementé par le conteneur
DD
<resource-ref>
<..name>jdbc/MyDS
…
<trans-attribute>Required
Home
Client
Composant
Logique métier … +
Callbacks … +
ic=new InitialContext();
ds = (DataSource) ic.lookup(
"java:comp/env/jdbc/MyDS");
Interface Métier:
Remote(rmi) ou local
Conteneur EJB

EJB: Interface composant

- Remote (RMI) ou Local
- Vue client de l’ EJB
- Declare les méthodes métier
- Implementée par les outils intégrés à la plateforme EJB - au moment du déploiement

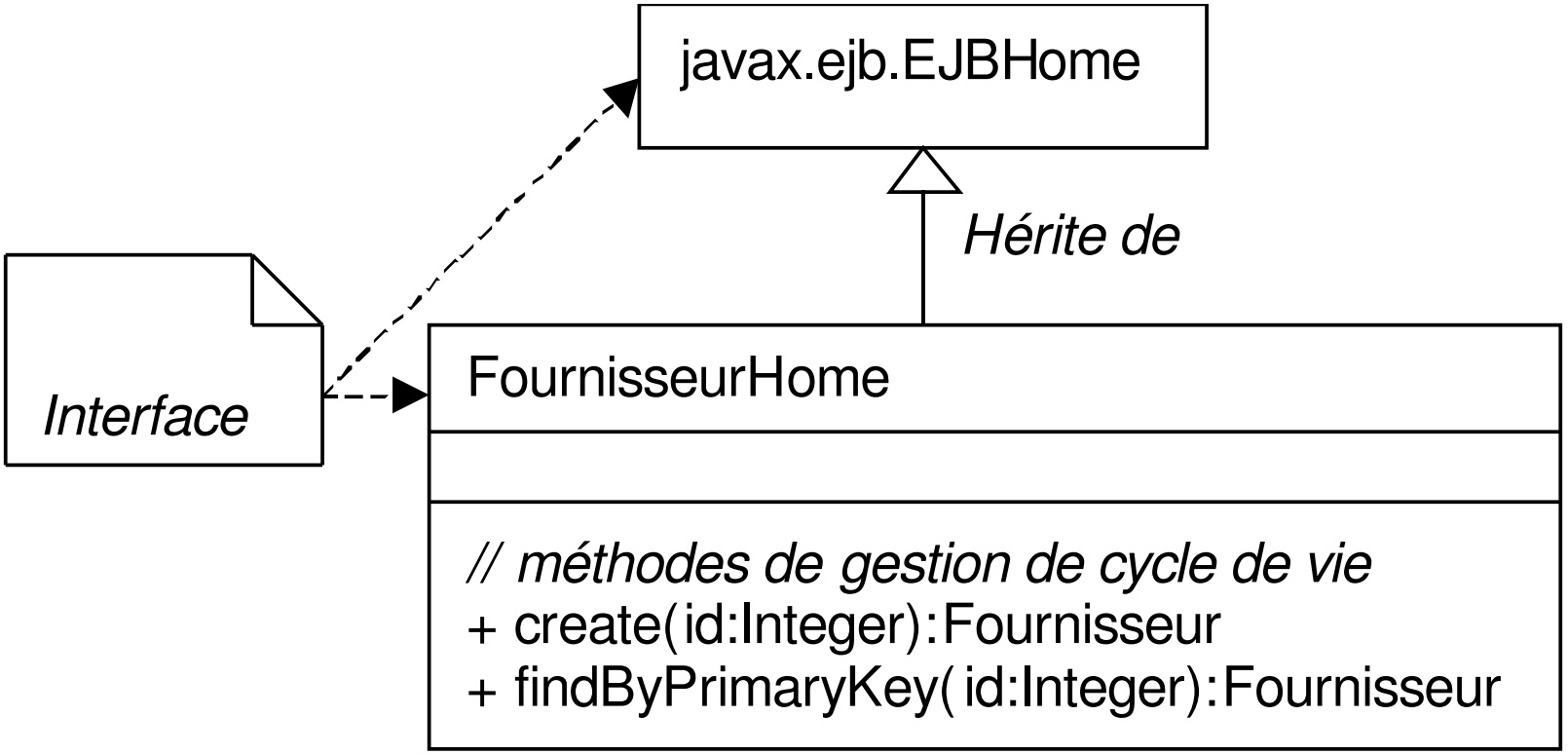
Exemple: Remote Interface



EJB: Home Interface

- Interface composant accessible à distance (RMI)
- Représente le cycle de vie du composant (création, suppression, recherche)
- Implementée par les outils intégrés à la plateforme EJB
- Séparée de la Remote Interface car elle représente des comportements non spécifiques à une instance de bean

Exemple: Home Interface

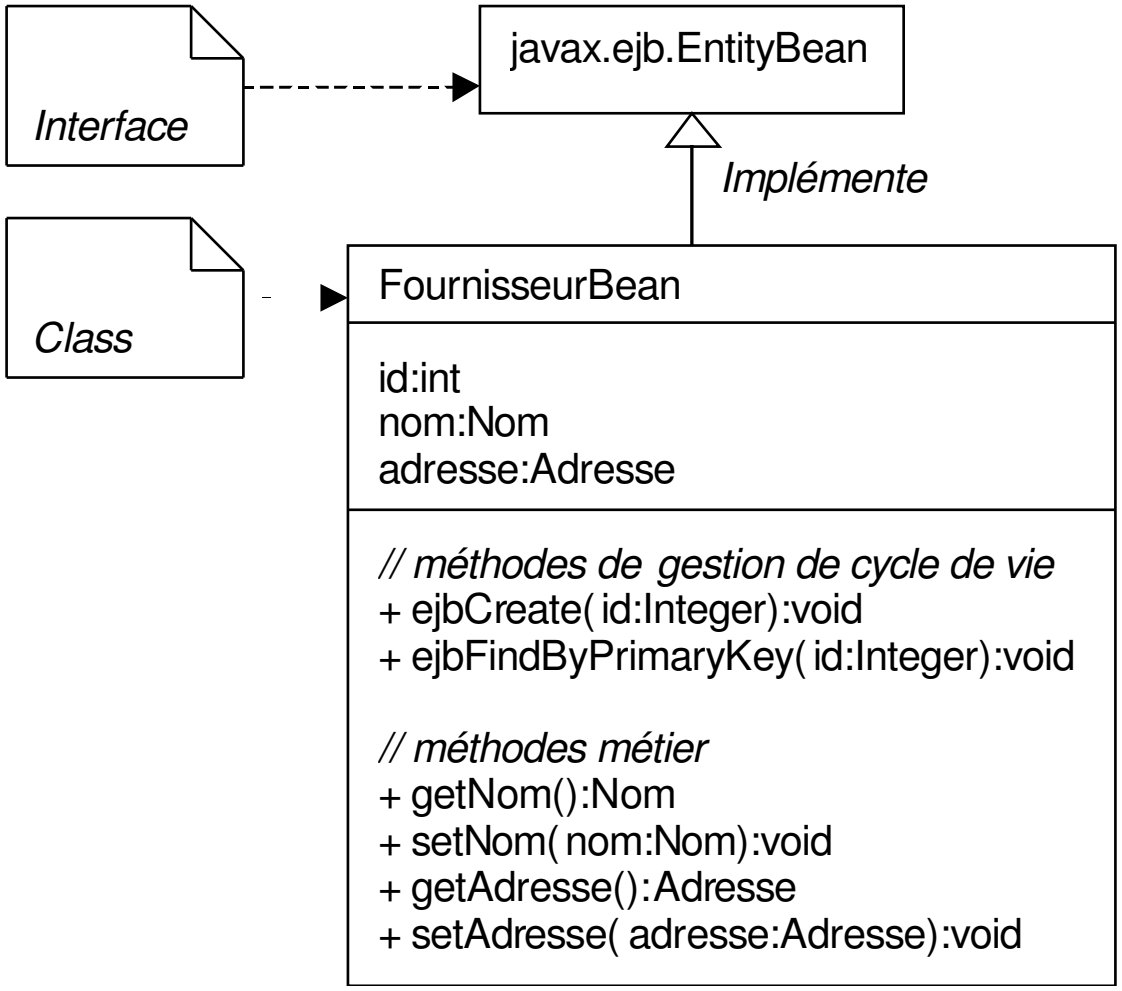


EJB: Implémentation du Bean

- Hérite de EntityBean ou SessionBean mais n'implémente pas les interfaces Home ou Remote

- Pourtant:
- Implémente les méthodes métier de l'interface Remote
- Définit et déclare les methodes correspondant à l'interface Home.

Exemple : Implém. de Bean



EJB: Code client

```
Object ref ;
FournisseurHome home; // Home interface
Fournisseur fourn; // Remote interface, pas le bean
// appel JNDI pour obtenir une référence à l'interface Home
ref = jndiContext.lookup(
            "java:comp/env/ejb/Fournisseur");
home = PortableRemoteObject.narrow(ref,
            FournisseurHome.class);
// créer un Bean
fourn = home.create(idFournisseur);
// appel méthode métier
fourn.setNom(unNom);
```

EJB: Bean Entité

- Représente des données dans la base de données
- Container-Managed Persistence (CMP) or Bean-Managed Persistence (BMP)
  - En mode CMP, le conteneur EJB gère la persistance du bean (pas d'accès BD dans le code).
  - En mode BMP, c'est le développeur du Bean qui gère la persistance (par exemple, en JDBC).

EJB: Bean Session

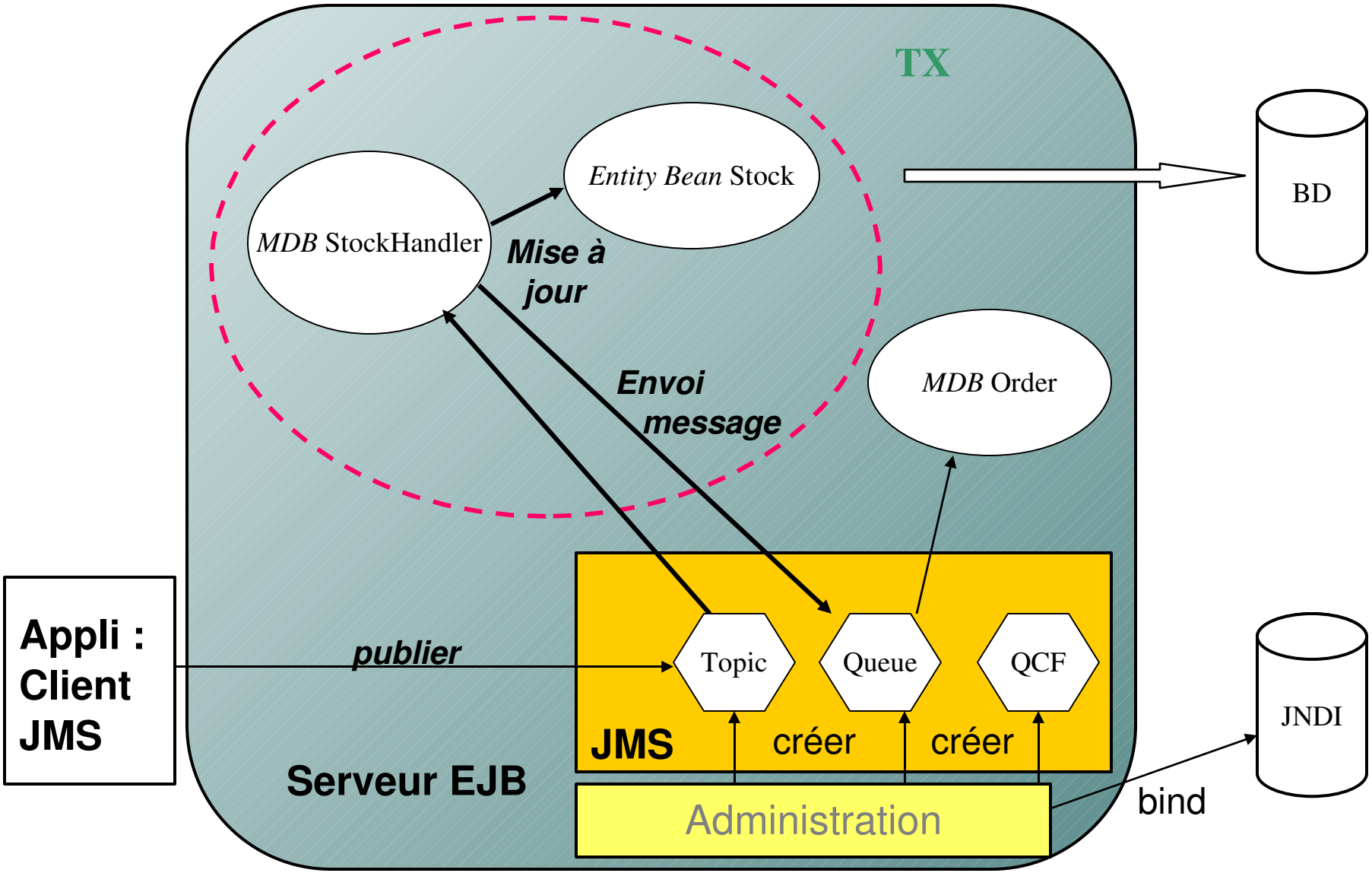
- Gestion des interactions entre beans entité ou session, accès aux ressources, réalisation d'actions sur demande du client
- Objets métier non persistants
- Stateful ou Stateless - maintient ou pas un état interne en mémoire

=> Un Bean Stateful encapsule la logique métier et l'état specifiques à un client

EJB : Message Driven Bean

- Composant asynchrone
- Execution sur réception d ’un message JMS
  - Méthode onMessage()
  - Appels à d ’autres beans, etc…
- Descripteur de déploiement
  - Associations Bean / ressource JMS
  - Ressources JMS (ex. Queue ou Topic)

Message Driven Bean : exemple



Message Driven Bean : exemple (2)

```
public class StockHandlerBean  implements MessageDrivenBean, MessageListener {
…
  public void onMessage(Message message) {
     ...
     sh = (StockHome)initialContext.lookup("java:comp/env/ejb/Stock");
     queue = (Queue)initialContext.lookup("java:comp/env/jms/Orders");
     …
     MapMessage msg = (MapMessage)message;
     pid = msg.getString("ProductId");
     qty = msg.getString( "Quantity");
     cid = msg.getString("CustomerId");
     Stock stock = sh.findByPrimaryKey(pid);
     stock.decreaseQuantity(qty);
    …
    qs = session.createSender(queue);
    TextMessage tm = session.createTextMessage();
    String m = "For CustomerId = "+cid+" ProductId= "+pid+" Quantity= "+qty;
    tm.setText(m);
    qs.send(tm);
    ...
    }
}
```

EJB: Configuration & Déploiement

- Interfaces Home et Remote (ou Local), classe qui implémente le Bean
- Descripteur de déploiement (fichier XML)

  <ejb-jar>
  - Description du Bean (Entity ou Session, ...)
  - Ressources (Base de données,...)
  - Securité: permissions et roles
  - Persistance (BMP, CMP)
  - Attributs transactionnels
  - …

  </ejb-jar>

=> Utilisé par l'assembleur d'application et par le conteneur EJB au moment du déploiement

Descripteur de déploiement : Bean

Bean Entité

```
<enterprise-beans>
  <entity>
    <description>EJB Fournisseur ( BMP )</description>
    <ejb-name>Fournisseur</ejb-name>
    <home>logistique.FournisseurHome</home>
    <remote>logistique.Fournisseur</remote>
    <ejb-class>logistique.FournisseurBean</ejb-class>
    <persistence-type>Bean</persistence-type>
    <prim-key-class>java.lang.String</prim-key-class>
    <reentrant>False</reentrant>
    <resource-ref>
      <res-ref-name>jdbc/ClientDB</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
    </resource-ref>
  </entity>
</enterprise-beans>
```

« Indirection » : Lien entre interfaces et implémentation

Persistance : Bean ou Container

Ressource : ici, BD

Ressources et JNDI

- Ressources déclarées dans le descripteur de déploiement (accès via JNDI)
- Convention de nommage
  - Noms préfixés par le type de ressource référencée (ejb, jms, jdbc, mail, url…)
- Exemple

```
fh = (FournisseurHome)initialContext.lookup(
   "java:comp/env/ejb/Fournisseur");
bd = (DataSource)initialContext.lookup(
   "java:comp/env/jdbc/Compta");
```

Indirection

- Contrat du conteneur entre les clients et les EJB
  - Garantit que l ’on passe par le conteneur
  - Exemple : activation / passivation, pool d ’instances...
- La classe d ’implémentation n ’implémente pas les interfaces Home ou Remote; pourtant, elle implémente les méthodes de Home et Remote
  - Le lien est fait par le descripteur de déploiement
- Renommage des méthodes de Home
  - create() -> ejbCreate(), etc...
  - Exception si appel direct ! (méthode inexistante)

Indirection : implémentations

- Génération de code (JOnAS, WebLogic)
  - Code d ’interposition entre le bean et le conteneur
  - Généré à partir des interfaces du bean
  - Paramétré par le descripteur de déploiement
- Dynamic Proxy (JBoss)
  - Pas de génération / compilation
  - Plus souple, mais moins efficace

Optimisations liées à l ’indirection

- Stateless Session Bean : Sans état
  - Pool d ’instances
  - Le serveur peut y accéder via un pool de threads
- Stateful session et Entity beans
  - Activation / Passivation (appel par le container de ejbActivate() après activation / ejbPassivate() avant passivation)

Conséquences de l ’indirection...

- Attention à « this »...
  - Pas de « this.methode() » sur les méthodes d ’interfaces « local » ou « remote »
  - Eviter de passer « this » en paramètre
- Utiliser les références !
  - Remote : context.getEJBObject()
  - Local : context.getEJBLocalObject()
  - Contexte initialisé par le conteneur appel setSessionContext ou setEntityContext après la création (méthode à implémenter).

Descripteur de déploiement : Persistance

Bean-managed (BMP)

<persistence-type>Bean</persistence-type>

Container-managed (CMP)

<persistence-type>Container</persistence-type>
<cmp-field>
  <field-name>codeFournisseur</field-name>
</cmp-field>

Persistance BMP

- Bean-managed persistence
  - Le bean gère sa persistance
  - Exemple : usage direct de JDBC
- Appels de méthodes par le conteneur
  - ejbLoad() - restaurer l ’état
  - ejbStore() - sauvegarder l ’état
  - Méthodes jamais appelées par le Bean ! (ne pas appeler depuis activate / passivate)

Persistance CMP

- Persistance gérée par le conteneur
  - Gestion déclarative (descripteur de déploiement)
  - Champs persistants (avec méthodes get/set)
  - Relations entre instances d ’entités CMP (CMP2) (cardinalité 1-1 ou 1-N, uni ou bi-directionnel)
- EJB-QL (CMP2)
  - Basé sur SQL92 (« select… from… where »)
  - Méthodes « find » associées à des requêtes avec paramètres

Exemple CMP: Home interface

```
public interface BookHome extends EJBHome
{
   public Book create(String id, String author)
      throws RemoteException, CreateException;

   public Book findByPrimaryKey (String id)
      throws RemoteException, FinderException;

   public Collection findByAuthor(String author)
      throws RemoteException, FinderException;
}
```

CMP: Descripteur de déploiement

```
<entity>
   <display-name>Book</display-name>  <ejb-name>Book</ejb-name>
   <home>BookHome</home> <remote>Book</remote>  <ejb-class>BookEJB</ejb-class>
   <persistence-type>Container</persistence-type>
   <prim-key-class>java.lang.String</prim-key-class>
   <cmp-field><field-name>title</field-name></cmp-field>
   <cmp-field><field-name>author</field-name></cmp-field>
   <cmp-field><field-name>price</field-name></cmp-field>
   <primkey-field>title</primkey-field>
   <query>
     <description></description>
     <query-method>
      <method-name>findByAuthor</method-name>
      <method-params>
       <method-param>java.lang.String</method-param>
      </method-params>
     </query-method>
     <ejb-ql>select distinct object(b) from Book b where b.author=?1</ejb-ql>
   </query>
</entity>
```

Inutile d 'implémenter la méthode findByAuthor()

CMP: Méthodes find et ejbSelect

- « Finder methods » (findBy…)
  - Retournent 1 bean (interface métier) ou une Collection
- Méthodes ejbSelect
  - Equivalent aux « findBy… »
  - Local à un bean (pas d ’accès externe)

Transactions

- Applicable aux 3 profils de composants
  - Session, Entité, Message driven
    - Limitation pour le MessageDriven (attributs « Required » ou « NotSupported » seulement).
- Gestion explicite
  - Utilisation de javax.transaction.UserTransaction (JTA)
  - Contrôle de la transaction (timeout, « rollbackOnly »)
  - Exemple

```
UserTransaction utx =
  (UserTransaction)ctx.lookup(« java:comp/UserTransaction »);
utx.begin();
…
utx.commit();
```

Descripteur de déploiement : Gestion déclarative des Transactions

```
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>Fournisseur</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>

 ...
</assembly-descriptor>
```

NotSupported
Required
RequiresNew
Mandatory
Supports
Never

Gestion déclarative des transactions

- Au niveau de la méthode du bean ! (démarcation)
- NotSupported
  - Si transaction courante, elle est suspendue
- Required
  - Si pas de transaction, nouvelle transaction
- RequiresNew
  - Nouvelle transaction (si tx courante, suspendue)
- Mandatory
  - Exception si pas de transaction courante
- Supports
  - Si transaction courante, l 'utiliser
- Never
  - Exception si transaction courante

Gestion des évènements transactionnels (Session Bean)

- Interception par un EJB des évènements transactionnels (produits par le conteneur)
  - Implantation de javax.ejb.SessionSynchronization
- Evènements (appelés par le conteneur)
  - afterBegin : appelé après UserTransaction.begin
  - beforeCompletion : appelé avant UserTransaction.commit
  - afterCompletion(true | false) : appelé après commit ou rollback

Descripteur de déploiement : Sécurité

```
<assembly-descriptor>
...
 <security-role>
  <description>Personnel administratif de gestion</description>
  <role-name>administratif</role-name>
 </security-role>
 <method-permission>
  <role-name>administratif</role-name>
  <method>
    <ejb-name>Fournisseur</ejb-name>
    <method-name>*</method-name>
  </method>
 </method-permission>
</ assembly-descriptor>
```

Définition de rôle

Permissions accordées à un rôle

JCA

- Java Connector Architecture
- Intégration avec les SI d ’entreprise (EIS)
  - Applications (ERP, Supply Chain…)
  - Middleware (gestionnaire de transactions…)
- Connecteur composé de :
  - Contrats système
  - API cliente
  - Resource Adapter

Contrats Système

- Interaction entre le SI et le serveur d ’applications
  - Gestion des connexions et pooling
  - Gestion des transactions (par le serveur d ’application et/ou le SI)
  - Sécurité (accès sécurisé au SI)

API clientes

- Standard : CCI (Common Client Interface)
  - Avantage : API standard
  - Inconvénient : trop général et peu compréhensible (pas de « sens » métier), implémentation facultative.
- Spécifique : API spécifique au SI, fournie par le « resource adapter »
  - Avantage : API « métier »
  - Inconvénient : spécifique au SI concerné, donc différente selon le connecteur...

Resource Adapter

- Implémentation de l ’interfaçage avec le SI
  - Classes d ’interface
  - Librairies natives du SI
- Descripteur de déploiement
  - Configuration du connecteur au sein du serveur d ’application
    - Classes d ’implémentation des interfaces standard
    - Fonctionnalités supportées ou non (transactions, sécurité…)

Déploiement

- Packaging en fichier .rar (RA archive)
  - Déployé par le serveur d ’application
  - Format jar avec organisation standard
  - Contient tous les éléments du connecteur
  - Descripteur de déploiement (ra.xml) dans META-INF : configuration du connecteur

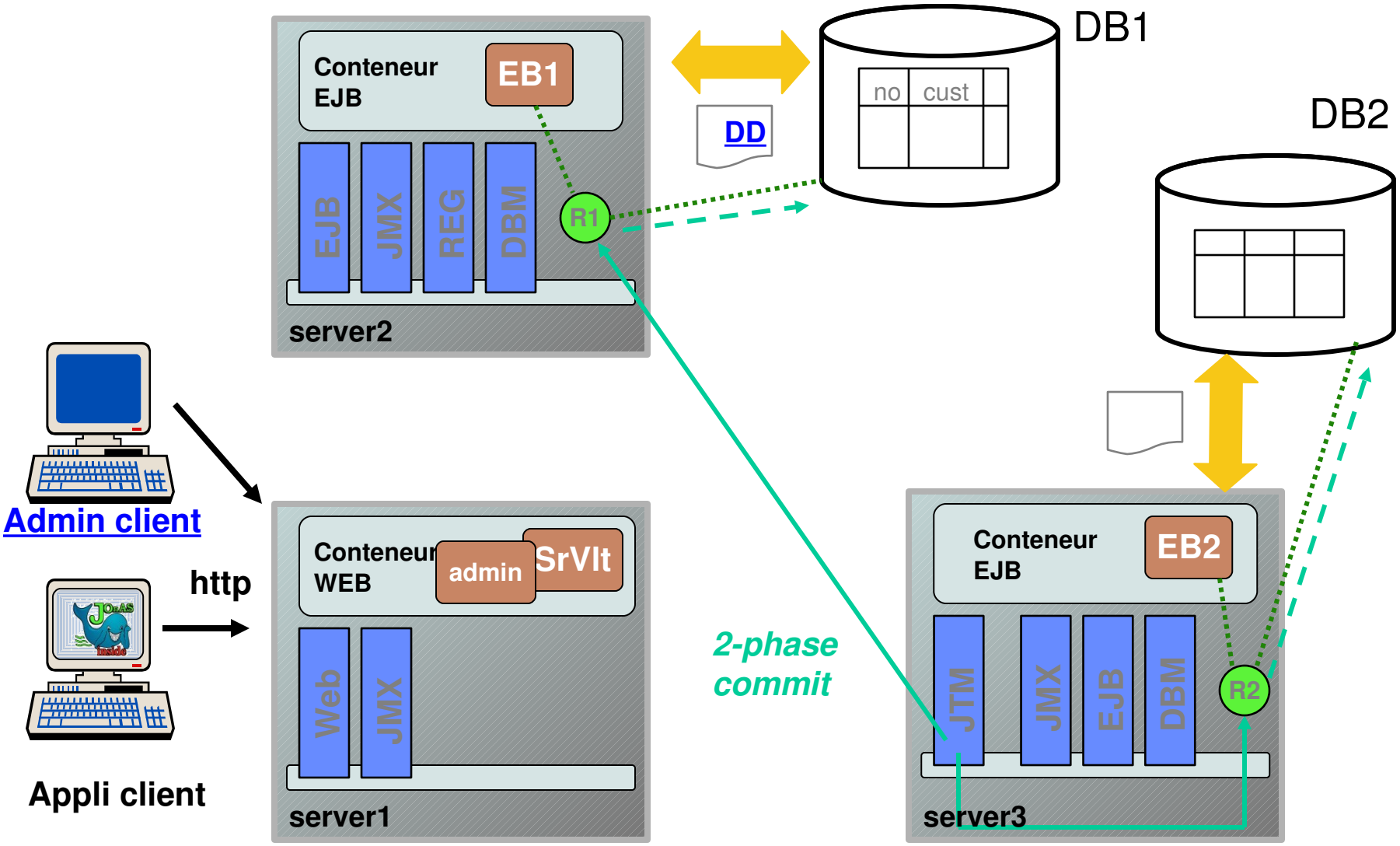
Rôles définis par la spec. EJB

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

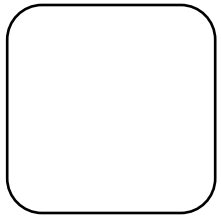
Packaging

- Application J2EE (aggrégarion de différents tiers)
  - Fichier « .ear » + descripteur « application.xml »
- Tiers client
  - Web : fichier « .war » + descripteur « web.xml »
  - Application : fichier « .jar » + descripteur « application-client.xml » (lancement du main() de la classe spécifiée dans le « manifest », attribut « Main-Class »)
- Tiers EJB
  - Fichier « .jar » + descripteur « ejb-jar.xml »
- Tiers « EIS » (connecteurs JCA)
  - Fichier « .rar » + descripteur « rar.xml »

Persistance + Transactions : exemple

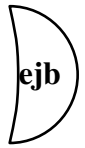


Répartition de charge : notations



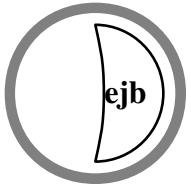
Un noeud (machine) qui héberge un ou plusieurs serveurs

Un conteneur Web



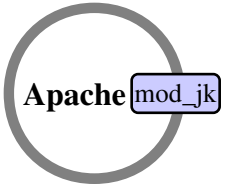
Un conteneur EJB

Un serveur qui héberge un conteneur Web



Un serveur qui héberge un conteneur EJB

Un serveur qui héberge un conteneur Web et un conteneur EJB

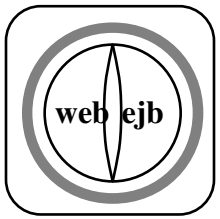


Un serveur Apache avec le module mod_jk

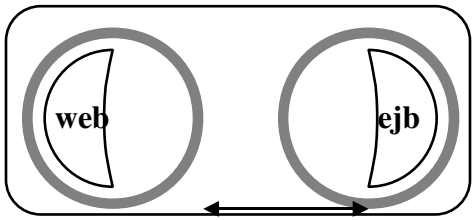
Répartition de charge : scenarii (1)

Répartition du serveur J2EE

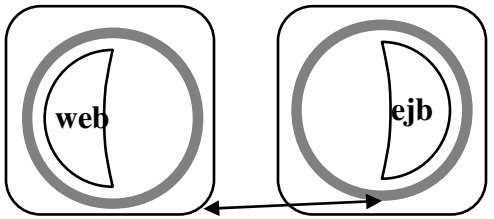
Compact



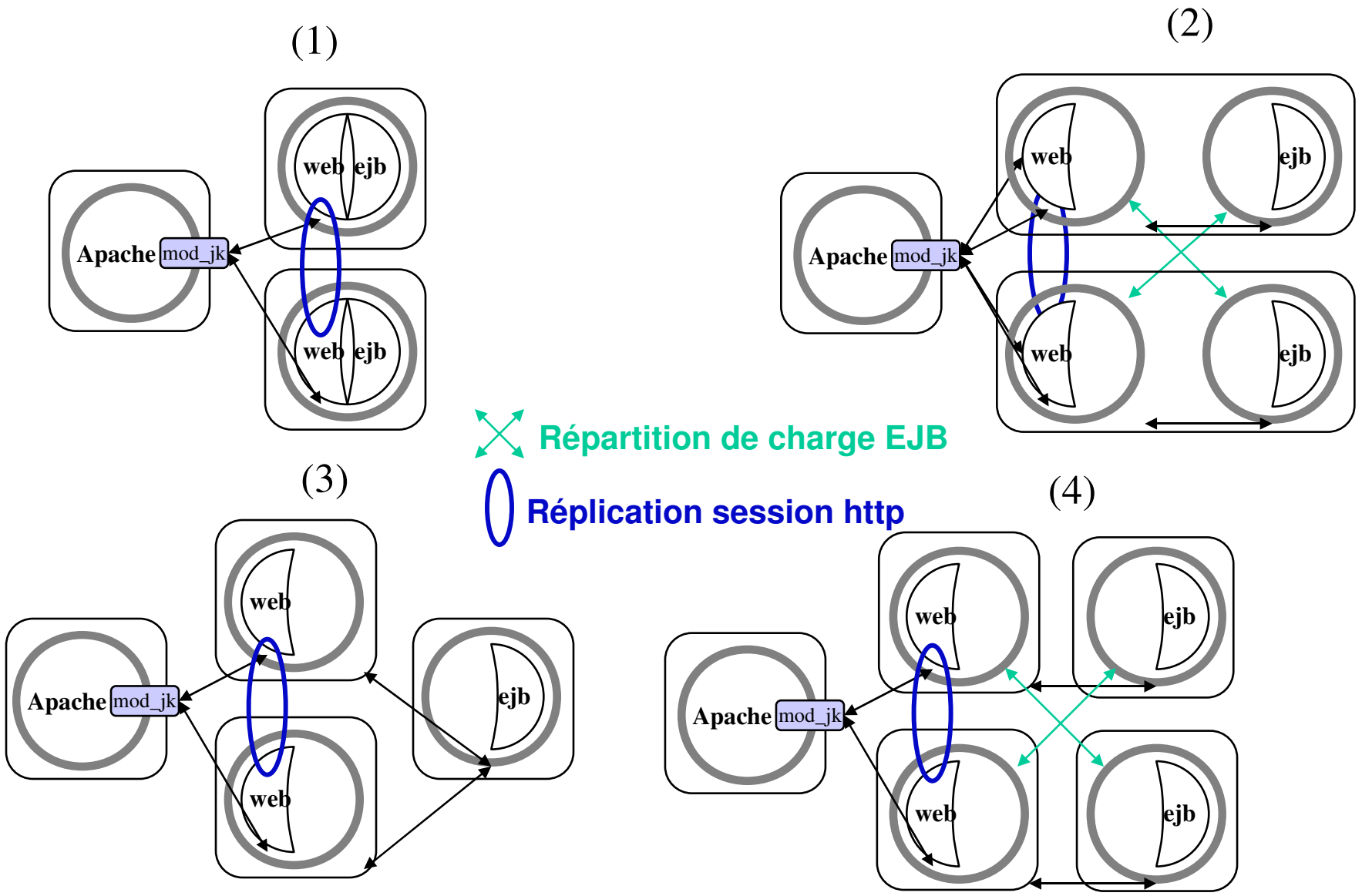
Réparti
(au sein d ’un même nœud)



Réparti



Répartition de charge : scenarii (2)



Répartition de charge et clustering

Source : ObjectWeb JOnAS