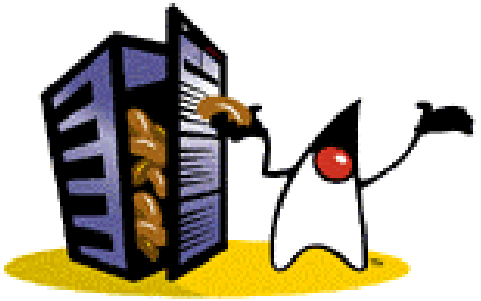
Les Enterprise Java Beans



Didier DONSEZ

Université Joseph Fourier (Grenoble 1)

IMA – LSR/ADELE

Didier.Donsez@imag.fr, Didier.Donsez@ieee.org

Plan

- Rappels
  - Programmation par Composant
  - Architecture multi-tiers
- Les EJB
- Un exemple

Rappel
La programmation par composant

- Limites de la programmation usuelle

« programming in the small »

- tout est la la charge du programmeur
  - construction des différents modules
  - définition des instances
  - interconnexions des modules
- structure de l'application peu visible
  - ensemble des fichiers de codes nécessaire
- évolution / modification difficile
  - changement du mode de communication
  - évolution, ajout, suppression de fonctionnalités
  - modification du placement
- développement, génération des exécutables, déploiement
  - pas ou peu d'outils pour les applications réparties

Rappel
La programmation par composant

- **Programmation constructive** (ou par composition)

  « programming in the large »

  - Motivation : réutilisation de logiciel
    - intégration de modules logiciels existants
    - construction d'applications réparties par assemblage de modules logiciels existants
    - programmation à gros grain ("programming in the large")
  - Approche : description de l'architecture de l'application à l'aide d'un langage déclaratif
    - modèle de construction des composants

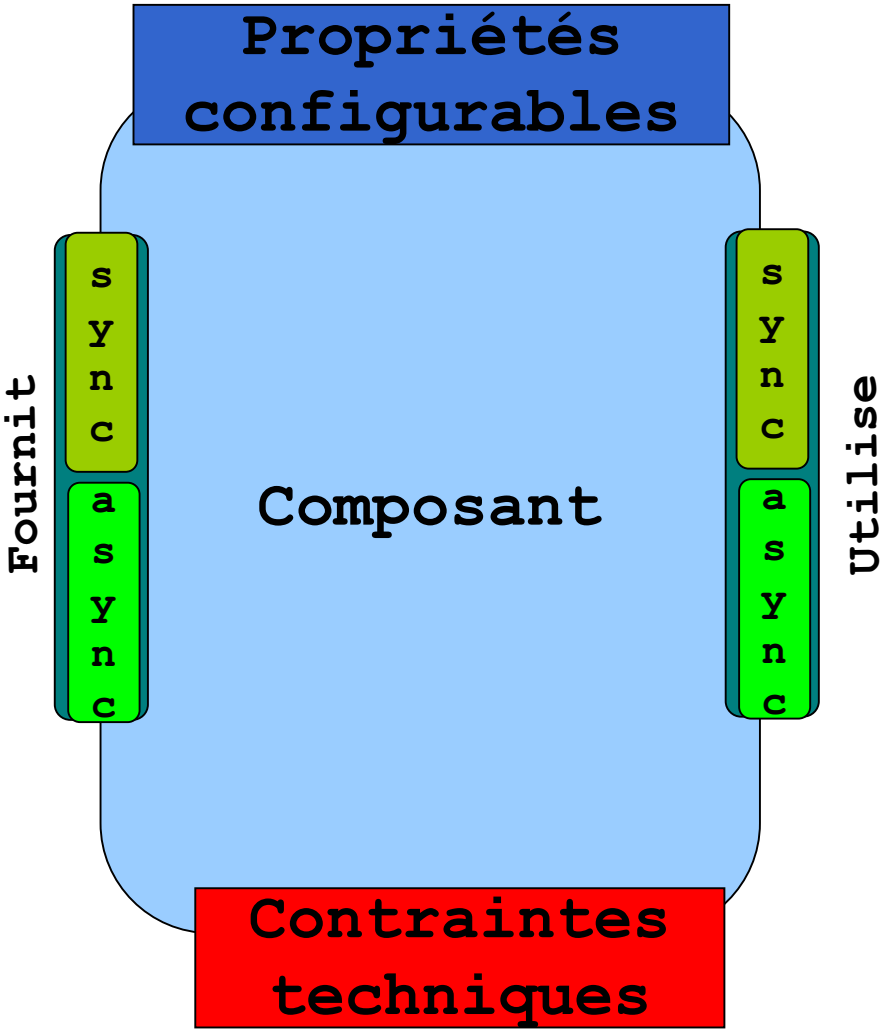
      composants : interfaces, attributs, implémentation
    - description des interactions entre composants (connecteurs)
    - description de variables d'environnement (placement, regroupement, sécurité, etc.)

Les composants...

- Qu'est-ce que c'est ?
  - **Définition usuelle**
    - module logiciel autonome pouvant être installé sur différentes plates-formes
    - qui exporte différents attributs, propriétés ou méthodes
    - qui peut être configuré
    - capable de s'auto-décrire
  - **Intérêt :** être des briques de base configurables pour permettre la construction d'une application par composition
- Quelques composants célèbres (ou qui vont l'être)
  - COM / DCOM, Java Beans,
    Enterprise Java Beans, Composants CORBA

Les modèles de composants : caractérisation d'un composant

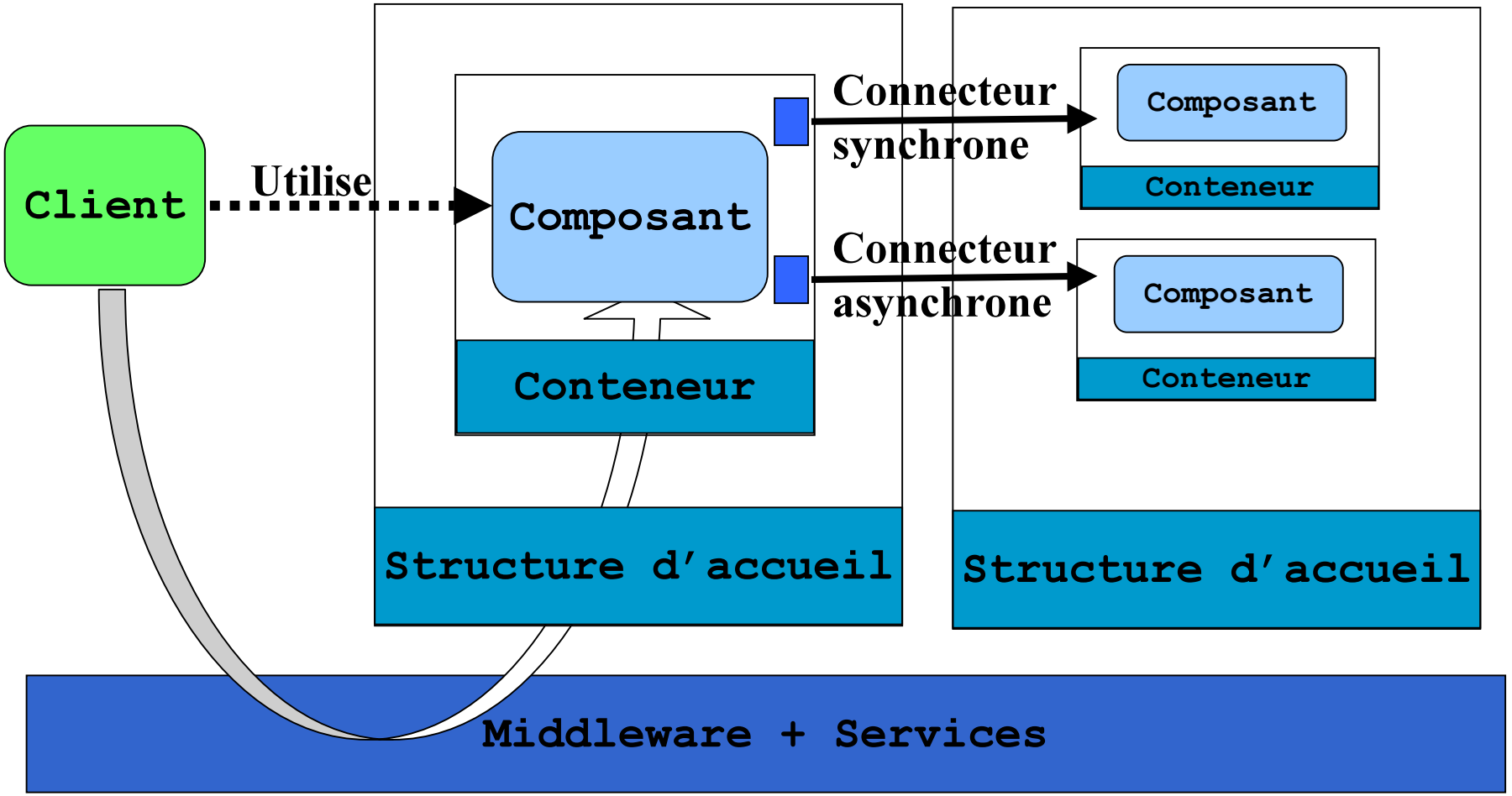
- Comment coopère un composant
  - Ce que fournit le composant (entrées)
    - composantes, interfaces, opérations, propriétés
  - Ce qu'utilise le composant (dépendances)
    - composition et références aux autres composants
  - modes de communication des connecteurs (synchrone, asynchrone, flots)
- Propriétés configurables du composant
- Contraintes techniques (QoS)
  - middleware : placement, sécurité, transaction
  - internes : cycle de vie, persistance
  - implantation : OS, bibliothèques, version



Les modèles de composants : conteneurs et structures d'accueil

- Conteneur
  - encapsulation d'un composant (et ses composantes)
  - prise en charge (masque) les services systèmes
    - nommage, sécurité, transaction, persistance ...
  - prise en charge partielle des connecteurs
    - invocations et événements
  - techniquement par interposition (ou délégation)
- Structures d'accueil
  - espace d'exécution des conteneurs et des composants
  - médiateur entre les conteneurs et les services systèmes
  - des + comme le téléchargement de code (navigateur)

Les modèles de composants : conteneurs et structures d'accueil



Les modèles de composants : de l'installation à l'introspection

- Installer les composants
  - technologie de packaging
  - production des conteneurs
- Créer les composants
  - par des fabriques (maisons / « home »)
  - configuration des valeurs initiales
- Retrouver les composants
  - services de désignation (Nommage ou Vendeur) ou maisons
- Utiliser
  - invocation synchrone et événements
- Introspection
  - découvrir leurs APIs (fonctionnelle)
  - découvrir les connecteurs (structurelle)

Construction par assemblage de composants

- Construction par assemblage plutôt que ingénierie de développement
  - réduire les besoins en compétence technique
  - focaliser l'expertise sur les problèmes du domaine
- Langage de description d'Architecture
  - capturer les composants
    - fonctionnalités et besoins
  - capturer les connecteurs
    - composition et modes de communication
    - impédance entre composants => adaptateurs
  - C'est le point faible des solutions industrielles !

Rappel
Les Architectures Multi-tiers

- Clients légers
  - Browsers Web (HTTP/HTML)
  - Applets (RMI)
  - Contrôle ActiveX (DCOM)
  - Clients CORBA (IIOP)
- Serveurs Applicatifs
  - Présentation, Outils Métiers
  - CGI, Servlet, …
- Source de Données
  - Bases de Données Relationnelles ou Objets
  - Intégrées (ERP), Legacy,
  - Annuaires (LDAP),
  - Flux (fournisseurs comme Reuter, Bloomberg) ...

Motivation pour les Serveurs d ’Application

- Objectifs
  - Simplifier le développement d ’architectures multi-tiers
    - présentation ←→ traitement (logique) ←→ données
- Principe
  - Le développer se concentre sur la logique de son application
  - le reste est réalisé par la plate-forme d ’accueil
    - Sessions, Transactions, Persistance, Securité, Nommage, Charge, ...
- Programmation par composition de la logique application
  - Solutions propriétaires
  - Solution « ouverte » : DCOM, EJB, ...
    - Indépendance Composants /¨Plateforme d ’accueil

Qu 'est que les EJB ?

Enterprise Java Bean

- 1) Architecture permettant la création d'applications réparties
- 2) Composant exécuté sur un serveur et appelé un client distant

- Remarque
  - rien à voir avec les JavaBean
    - qui sont des composants côté client

Enterprise JavaBeans

- Rendre une application
  - **facile à développer**, **déployer** et **administrer**
  - **indépendamment** de la plate-forme permettant son exécution

  - Un EB (EnterpriseBean) n'est pas spécifique de la plate-forme dans laquelle il est utilisé
  - Le déploiement d'un EB se fait sans recompilation ou modification du code source

Enterprise Java Beans

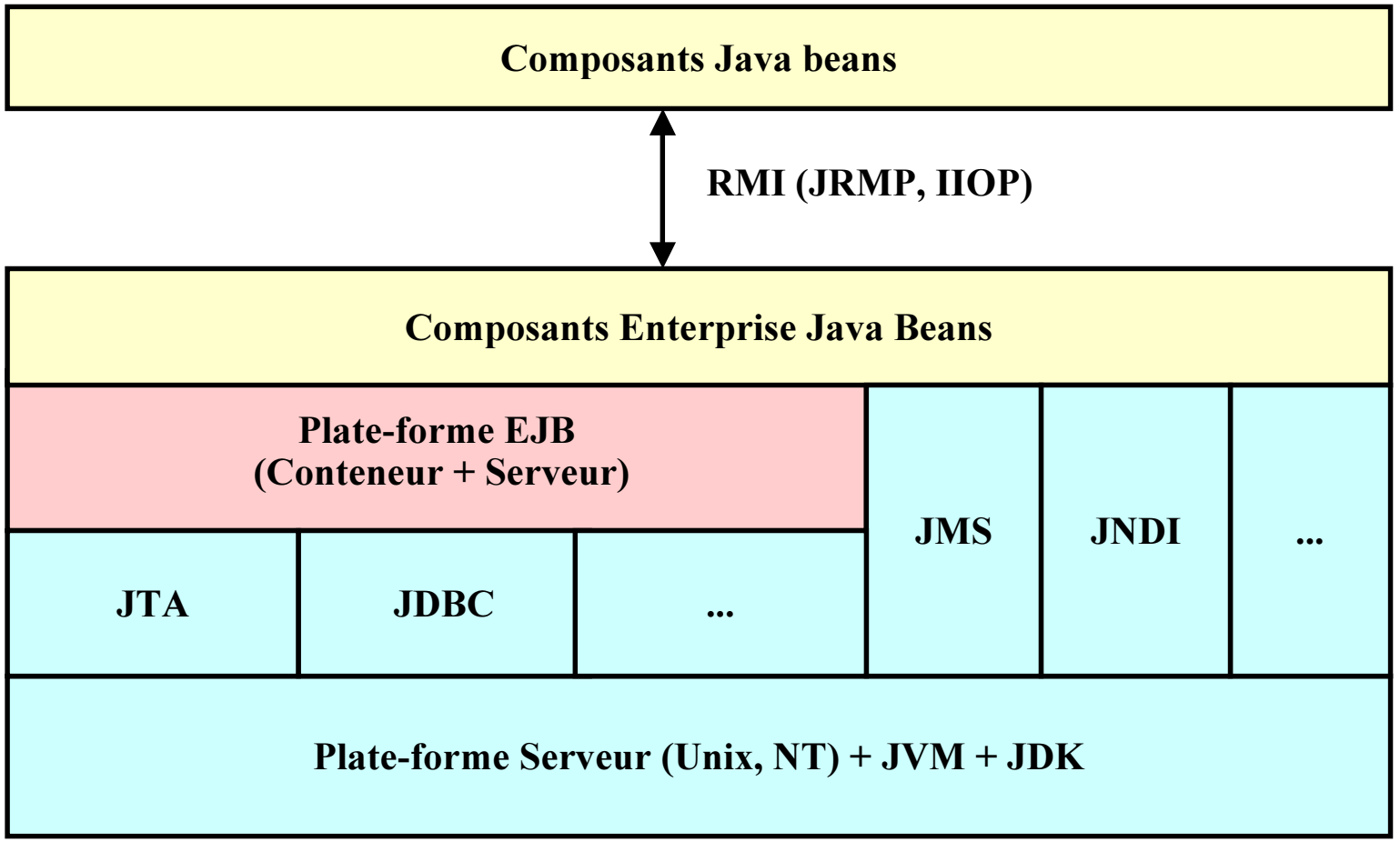
- EJB spécifications

  = **définition d'une architecture** pour construire une application en Java dont la partie serveur est construite à à l'aide de composants appelé Entreprise Beans (EB)

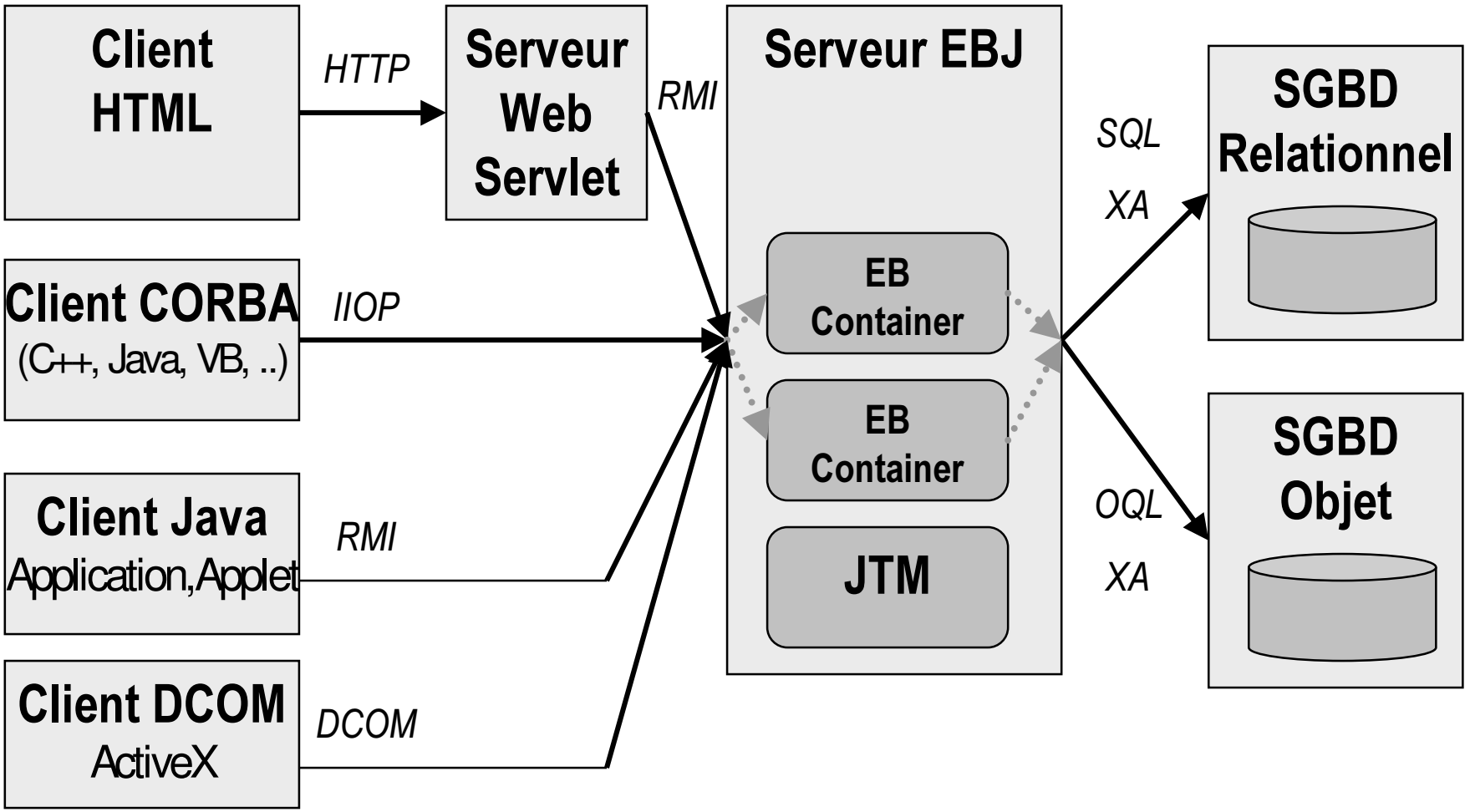
  - Architecture = ensemble d'interfaces

- Caractéristiques principales des EB
  - « écrit une fois, s'exécute partout »
  - composants « serveurs » spécialisés écrits en Java
    - c.f. objets métiers de CORBA
- EJB est la partie centrale de la plate-forme

Plate-forme Enterprise Java



Architecture de Serveur EBJ



EJB
Caractéristiques principales

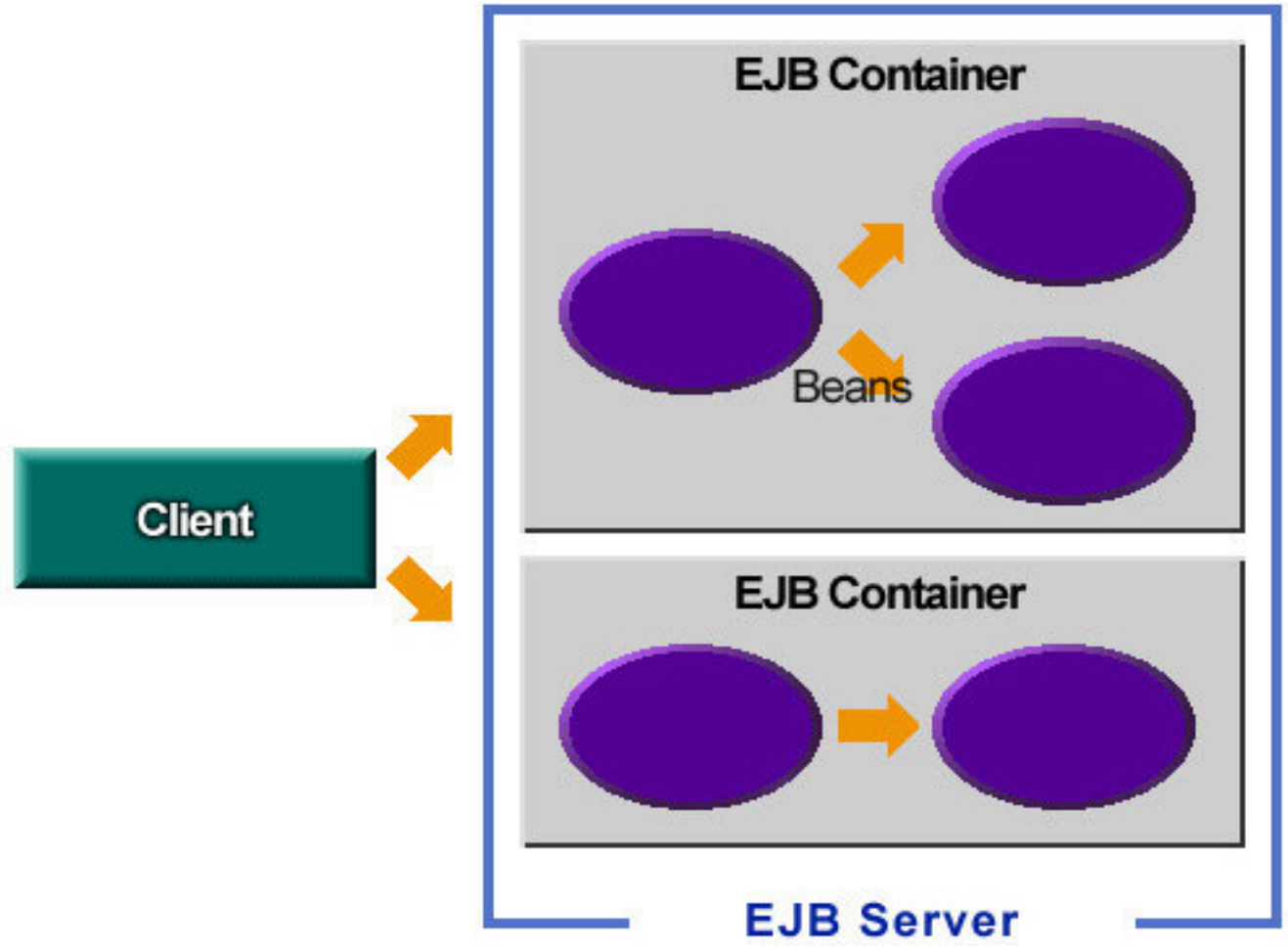
- L'architecture EJB identifie les éléments suivants :
  - composants logiciels ou *beans* (EB),
  - *conteneurs*,
  - *serveurs*,
  - *clients*.

Les conteneurs isolent les beans du client et d'une implémentation spécifique d'un serveur

  - Rappel : les beans sont dans la partie serveur

- Conteneurs et serveurs implémentent les mécanismes de bas niveau utilisés par les applications
  - transactions, persistance, gestion mémoire, sécurité, …

EJB
Modèle d'exécution

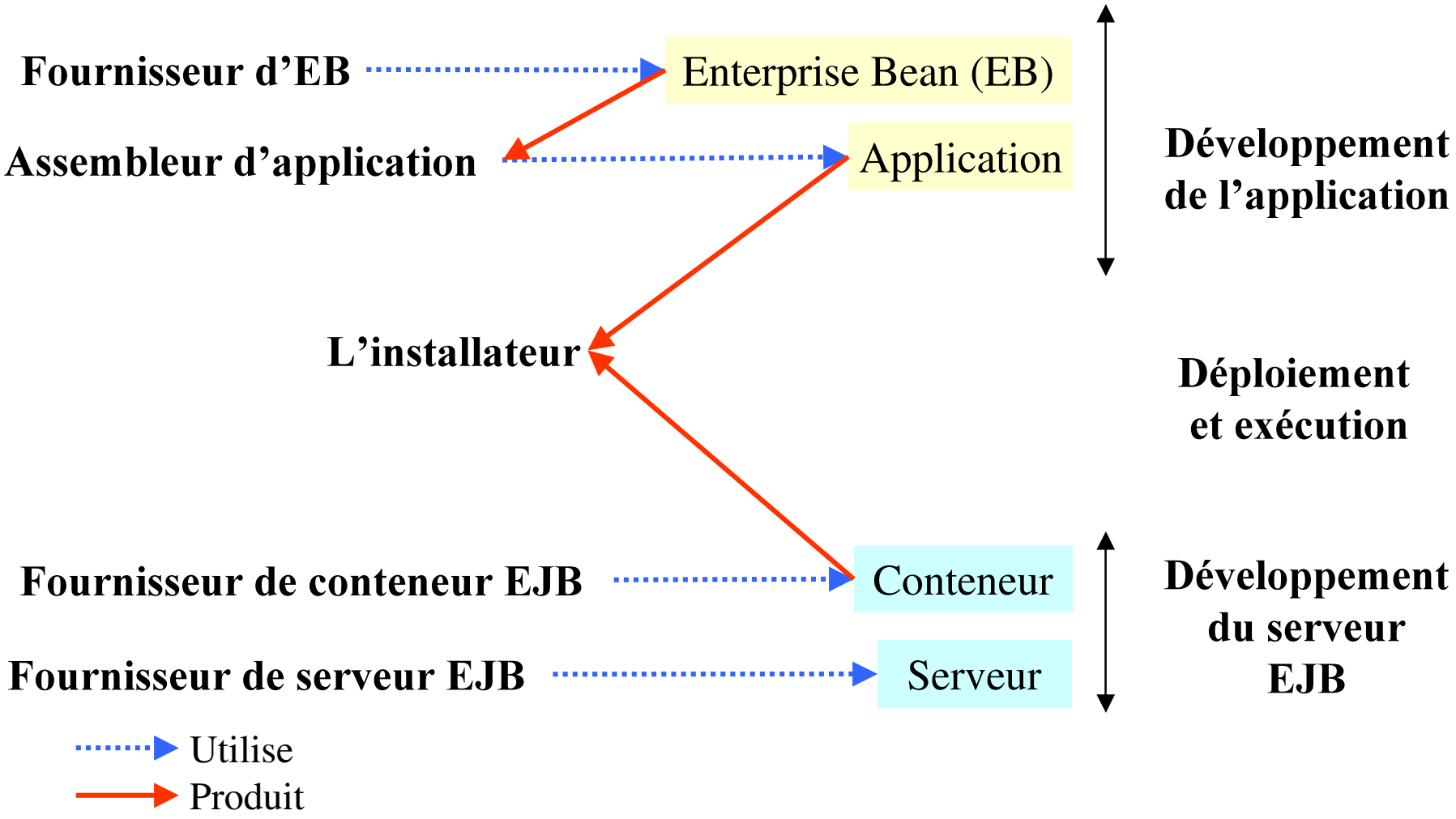


EJB
Caractéristiques principales

- EJB s'intéresse aux activités liées au **développement**, au **déploiement** et à l'**exécution** d'une application
- EJB définit différents **rôles** associés aux différentes parties intervenant dans la production d'une application
- EJB définit des **contrats** associés à un *bean*
  - Ces contrats sont passés entre le *conteneur* et les *clients* qui utilisent le bean.

  => ce sont des règles (obligations) qui doivent être respectées par le fournisseur de l'*EB* et de *conteneur*

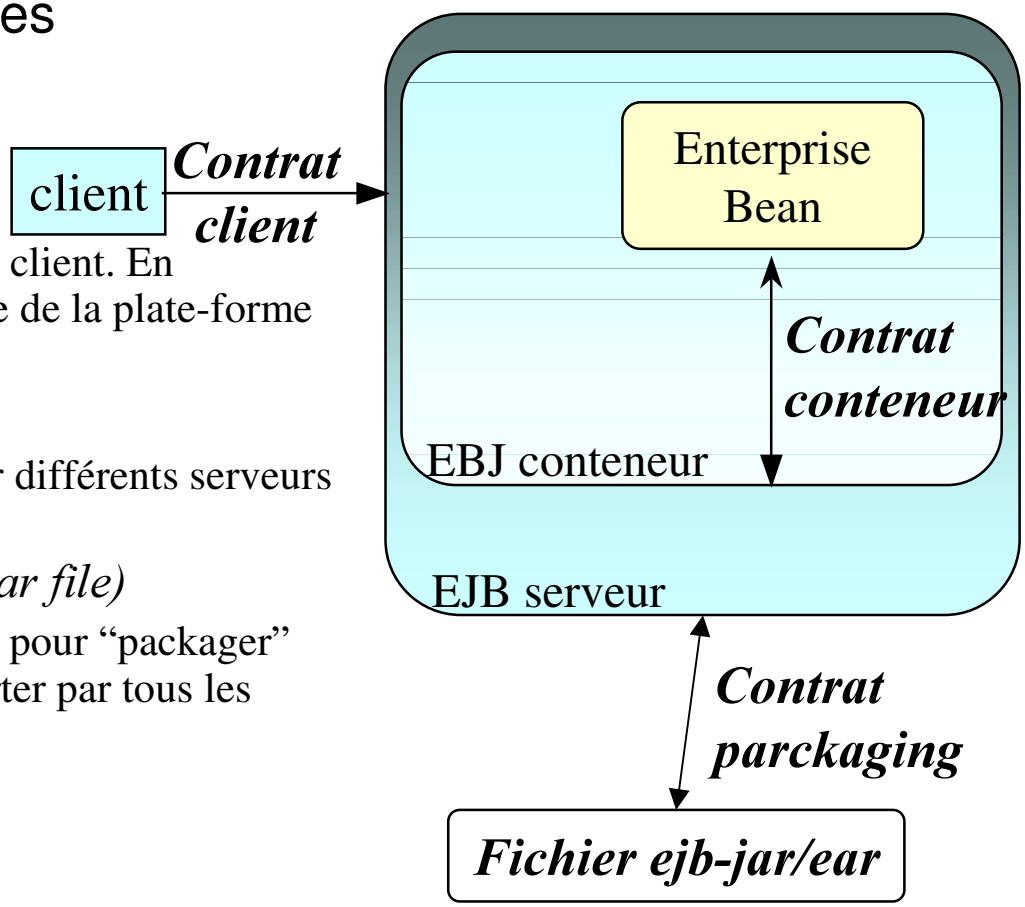
EJB
Les différents rôles



EJB contrats

- Fournir un modèle de développement uniforme pour les applications qui utilisent les composants EB

  - *Contrat coté client*
    - fournir une vue uniforme du bean au client. En particulier cette vue est indépendante de la plate-forme de déploiement
  - *Contrat coté conteneur*
    - permettre la portabilité des beans sur différents serveurs EJB
  - *Contrat coté "packaging" (ejb-jar/ear file)*
    - fournir un format de fichier standard pour "packager" les beans. Ce format doit être supporter par tous les outils liés aux EJB



EJB
Le contrat coté client

- Localiser le bean
  - utilisation de JNDI
- Utiliser le bean
  - utilisation de l'interface standard fournie par l'EB provider
    - *Home* Interface

      méthodes liées à la gestion du bean : *create*, *remove*, *finder*, ...)
    - *Remote* Interface (méthodes de l'application)

Le container implémente le mécanisme de délégation permettant de "faire suivre" l'appel au bean

- le client ne communique pas directement avec le bean mais avec le container

EJB
Le contrat du “conteneur”

- L’EJB conteneur permet
  - gestion du cycle de vie, gestion de l’état, sécurité, transaction distribuée, concurrence, extensibilité
  - ces services appellent des méthodes fournies par le bean (callback methods)
- Les conteneurs gèrent 2 types de beans
  - **Session beans**
  - **Entity beans**

Session beans

- sont non persistant (short-lived)
- associé à **un seul client**
- **un flot d'exécution est créé pour**
  - chaque appel de méthode

    stateless sessions bean (sans état)

    pas de donnée interne, inutile de le rendre passif, peut être partagé par plusieurs clients
  - plusieurs appels de méthodes en provenance du même client

    stateful sessions bean (avec état)
- détruits après un arrêt (ou une panne) du serveur EJB

- Remarque
  - les stateful SB peuvent être inclus dans une transaction

Entity beans

- Représentent les données d'une base de données
  - en général, une ligne d 'une table relationnelle (SGBD-R) ou un objet persistant (SGBD-OO)
- sont persistant (long-lived)
  - la gestion de la persistance peut être faite par le bean (*bean managed persistence*) ou déléguée à son conteneur (*container managed persistence*)
- acceptent les accès multiples effectués par plusieurs clients
  - gestion de la concurrence
    - différents niveaux d 'isolation
- peuvent participer à des transactions
- survivent aux pannes d'un serveur EJB
  - les pannes sont transparentes aux clients

Cycle de vie d'un bean

- Le conteneur gère le cycle de vie d'un bean
- Il fournit
  - administration du bean ('Home implementation')

    permet aux clients

    de créer, détruire et rechercher un objet EJB

    appelle les opérations correspondantes

    fournies par le bean (callbacks)
  - gestion de l'état

    Activation

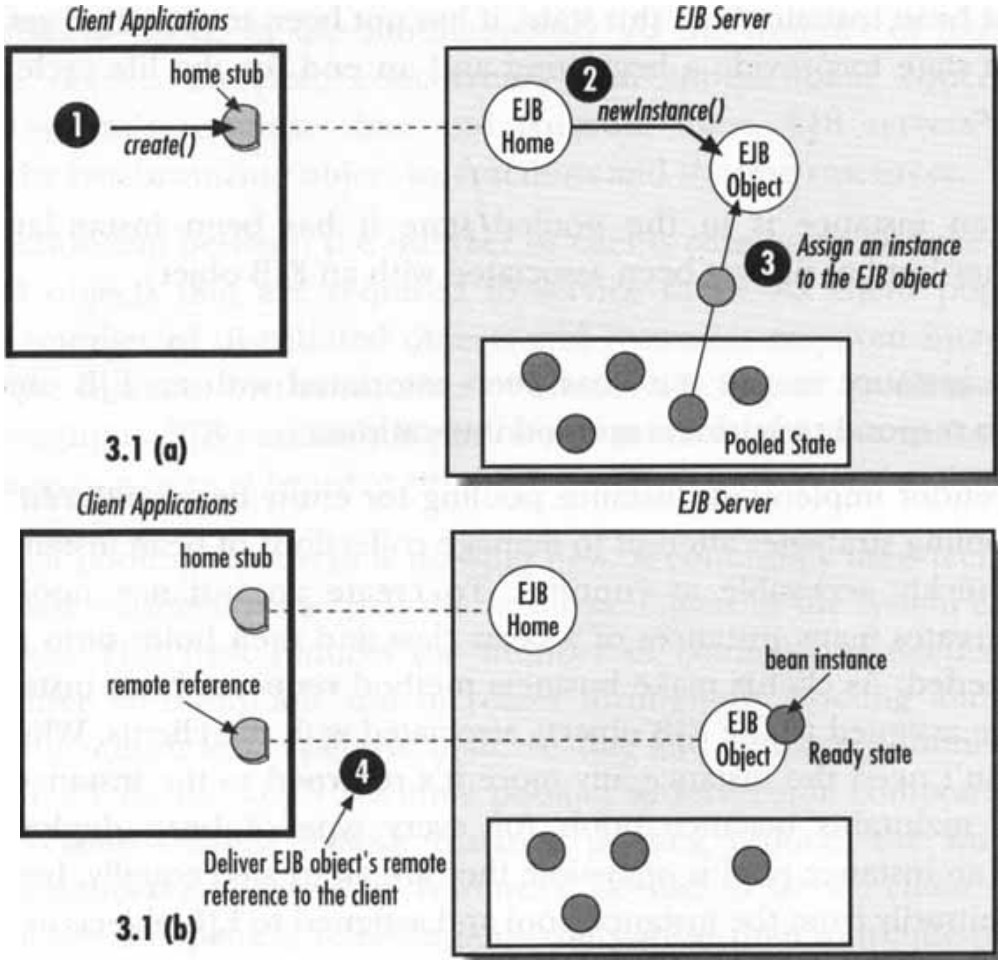
    » le bean est chargé en mémoire

    Passivation (Désactivation)

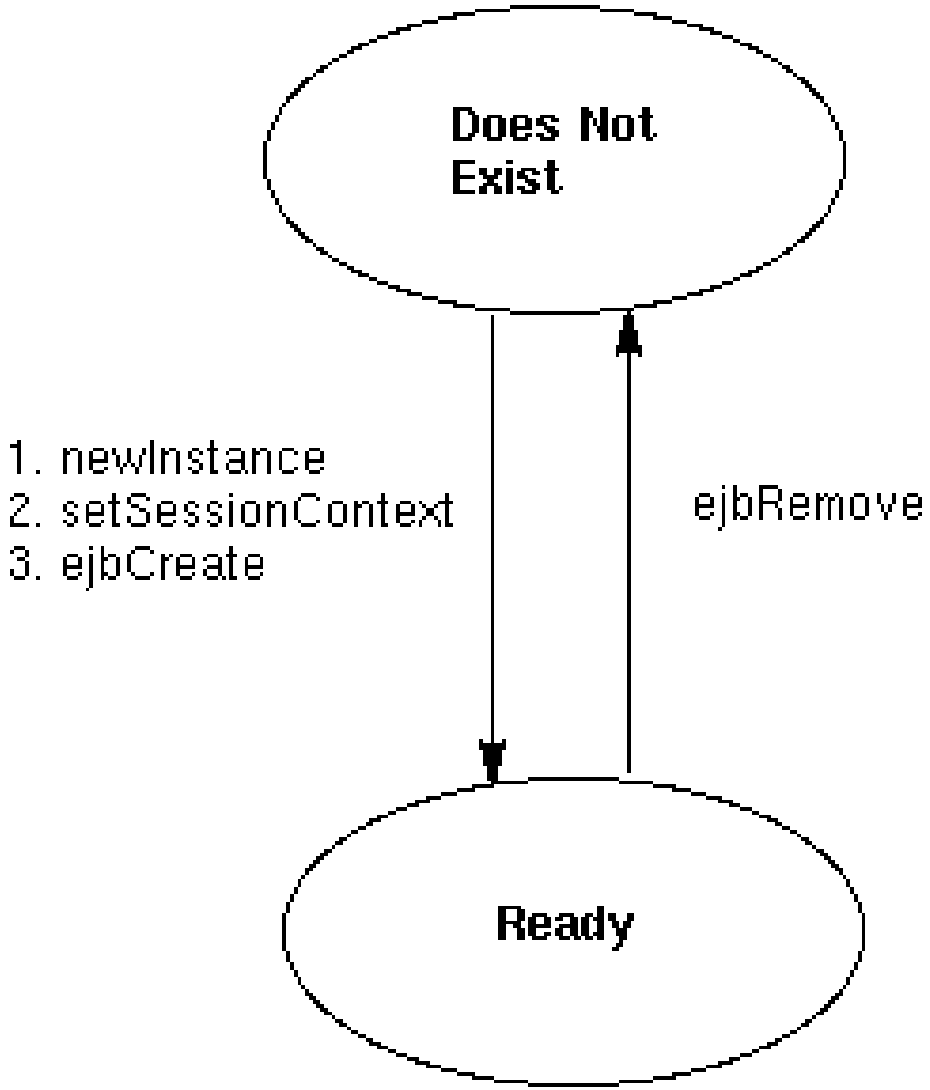
    » le conteneur peut sauvegarder l'état du bean

Gestion des Ressources « Pool » d ’instances

- Le serveur EJB maintient un pool d ’instance de bean

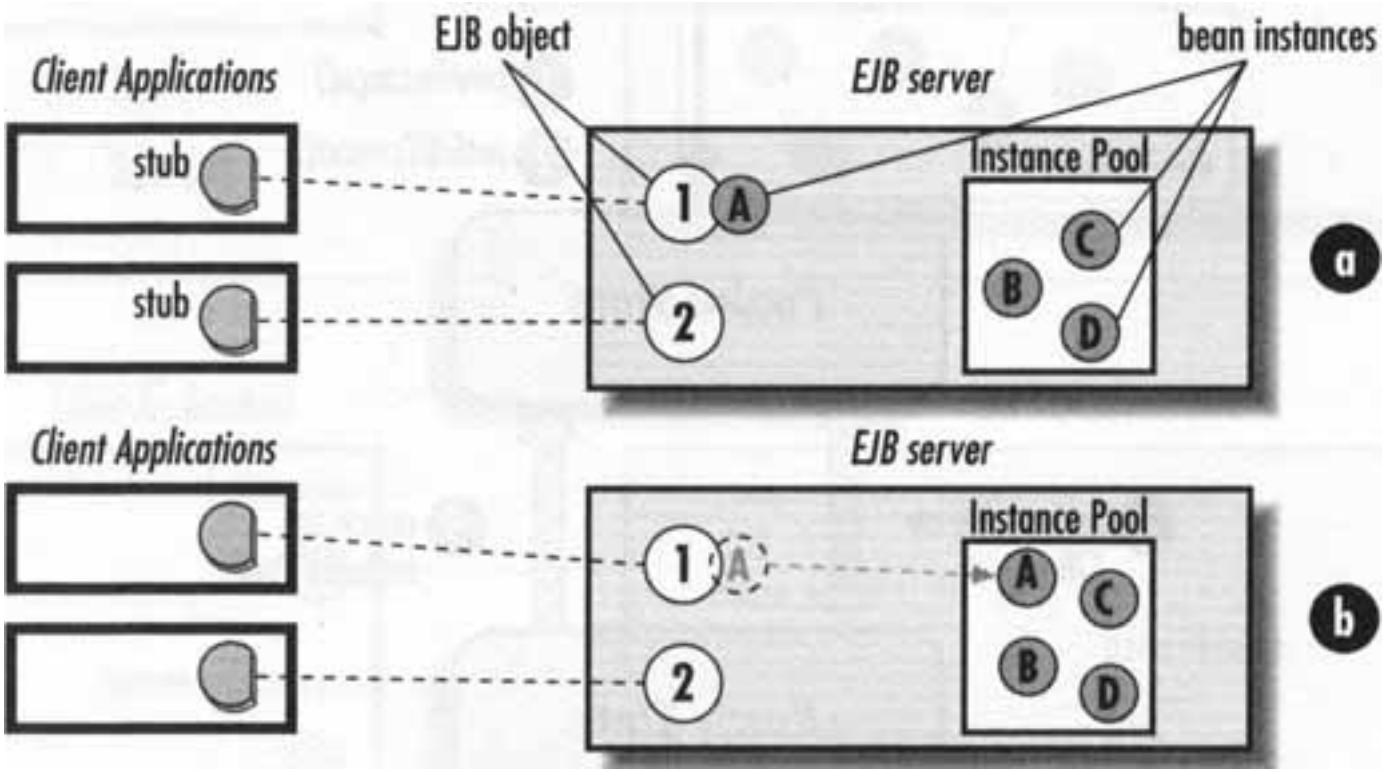


Cycle de Vie d ’un Session Bean Stateless (sans état)

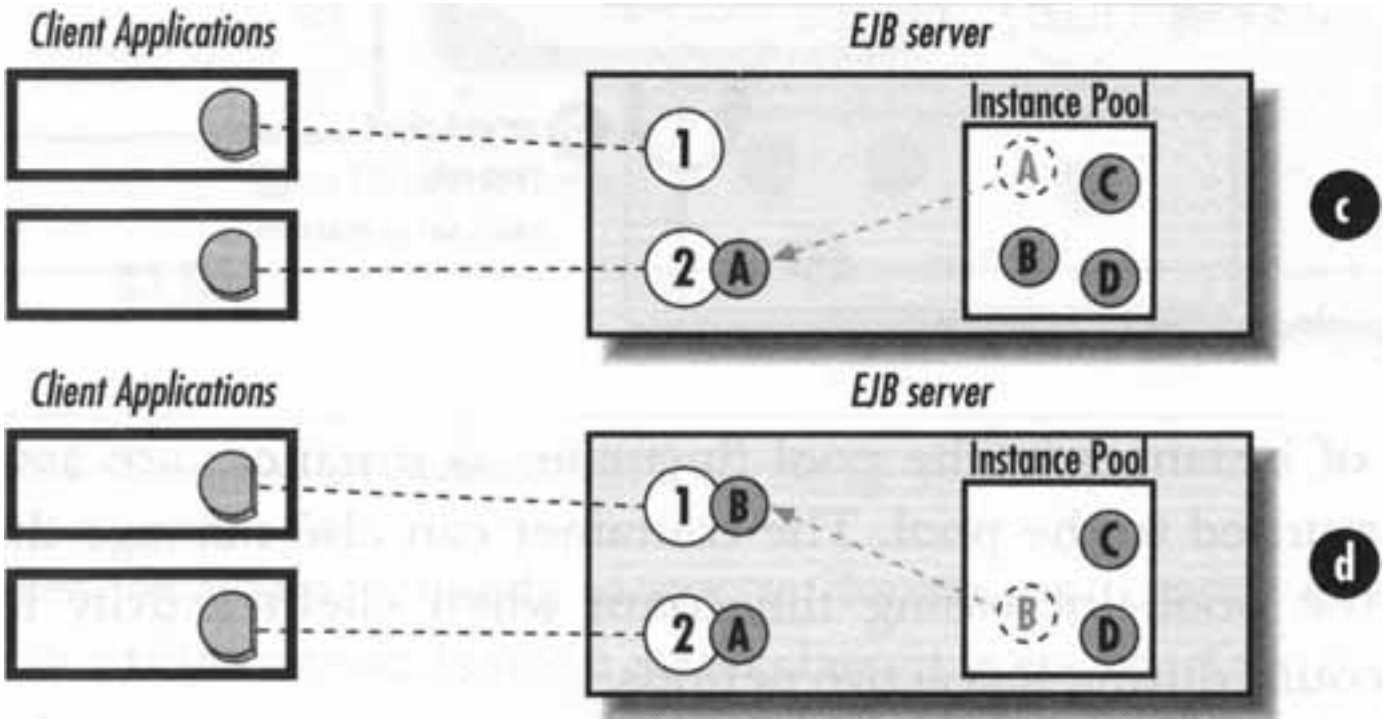


Gestion des Ressources
Instance Swapping (i)

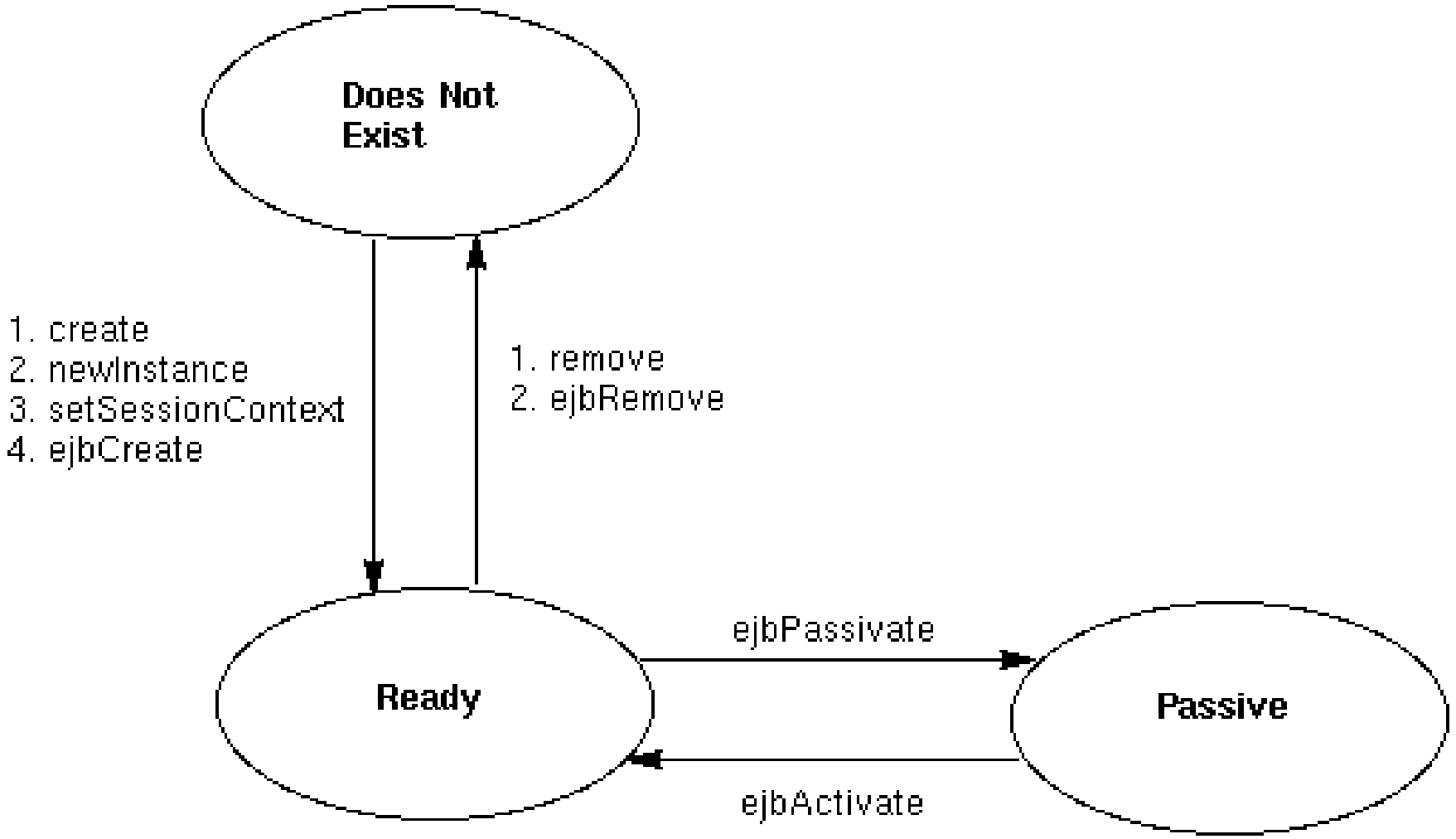
- Les instances de stateless SB sont pris dans un pool à chaque appel puis remis dans le pool après



Gestion des Ressources
Instance Swapping (ii)



Cycle de Vie d ’un Session Bean Stateful (avec état)



Gestion des Ressources

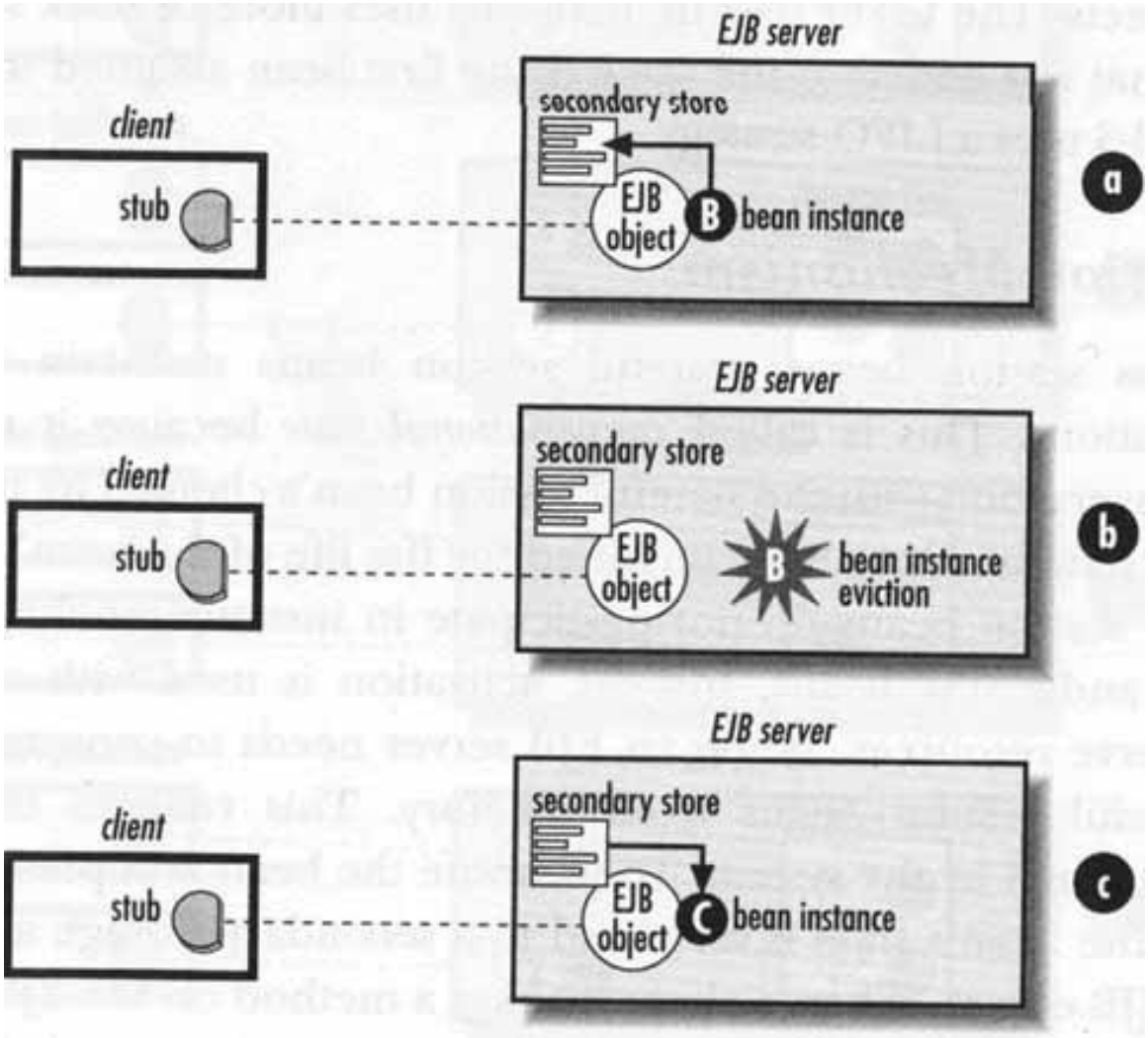
Le mécanisme d 'Activation/Passivation

- Rappel : le Stateful SB maintient un état conversationnel
- Pour accepter un très grand d 'instances simultanément, le serveur d 'EJB sauvegarde l 'état d 'instances non actives (passivation) et restaure leur état lors qu 'un client appelle une méthode sur l 'instance (activation)
- méthodes « callback » appélées par le container
  - ejbPassivate(), ejbActivate()
    - implanté par sérialisation ou autre
      vers un fichier de swap, une table de swap dans un SGBD, ...

  - Remarque
    - même principe que la mémoire virtuelle des SE

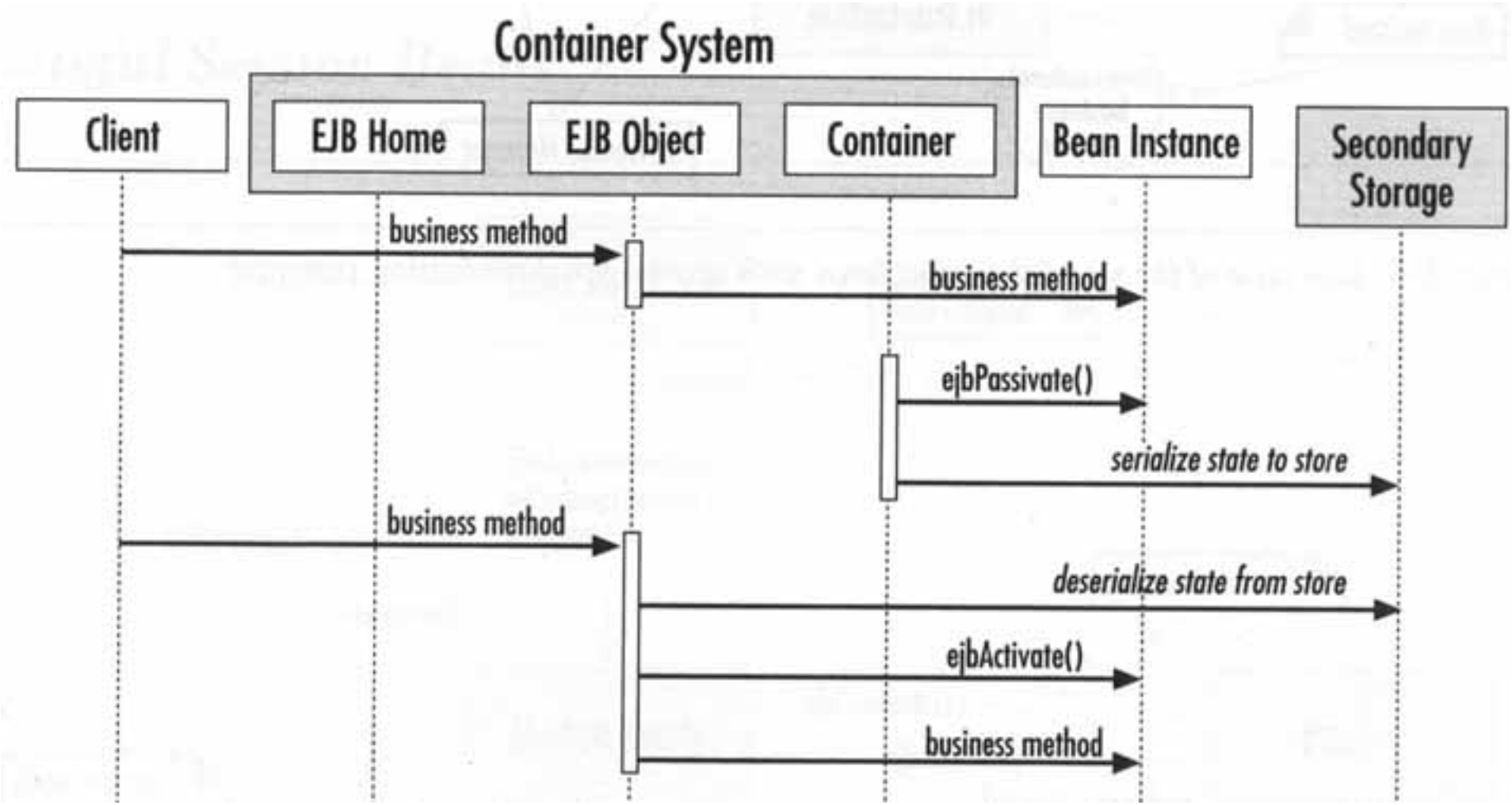
Gestion des Ressources

Le mécanisme d ’Activation/Passivation

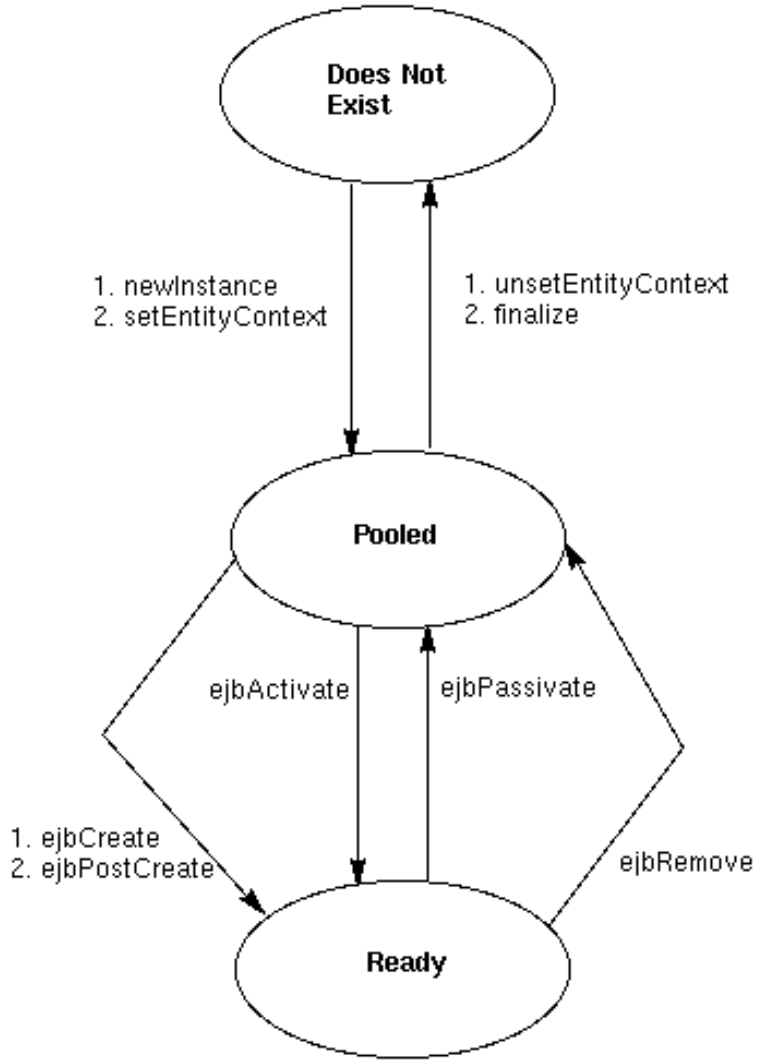


Gestion des Ressources

Le mécanisme d ’Activation/Passivation



Cycle de Vie d ’un Entity Bean



Persistance (Entity Bean)

- L ’Entity Bean correspond à une donnée stockée dans une source de données
- Il faut assurer la correspondance (mapping) avec la source de données
  - Base de Données Relationnelle
    - correspondance (mapping) entre chaque champ d ’une EB vers une colonne d ’une ligne d ’une table, d ’une vue, d ’une requête multi-table
    - indiquer la clé primaire (PK) qui identifie la ligne correspondante à un EB
  - Base de Données Objet
    - plus ou moins direct
  - Base de Données Objet-Relationnelle
  - Legacy
    - Fichiers (IMS, CICS, Btrieve,…), ERP, Annuaire LDAP, …
    - requiert des « wrappers » spécifiques

Persistance (Entity Bean)
Prise en charge de la persistance

- Persistance gérée par le bean (bean-managed)
  - le fournisseur de l'Entity Bean écrit les opérations d'accès aux données permettant de gérer la persistance dans les callback appropriés (*ejbCreate, ejbStore, ejbLoad, ejbFind* ...)
  - en utilisation par exemple JDBC
- Persistance gérée par le conteneur (container-managed)
  - le fournisseur de l'Entity Bean utilise les services du conteneur ("container managed fields")
  - et précise le support utilisé pour la persistance

    le code d'accès à la base de donnée est délégué au conteneur
- Dans les 2 cas
  - le conteneur est responsable de la cohérence entre l'état du bean et de l'état sauvegardé dans la base de données

Transaction
Gestion Declarative

- Support de l'architecture XA (X/Open DTP)
  - extensions standards à JDBC 2.0
  - Java Transaction API (JTA)
- Modèle de transaction plat
  - pas de transactions imbriquées (nested)
- Gestion "déclarative" des transactions
  - l'attribut transactionnel associé aux méthodes des Ebs est affecté
    soit lors de la phase d'implémentation du bean,
    soit lors de son déploiement
  - un bean peut aussi contrôler explicitement les transactions (JTA)
    - un client peut aussi contrôler explicitement les transactions (JTA)
  - concerne les Entity Beans et éventuellement les Stateful Session B.

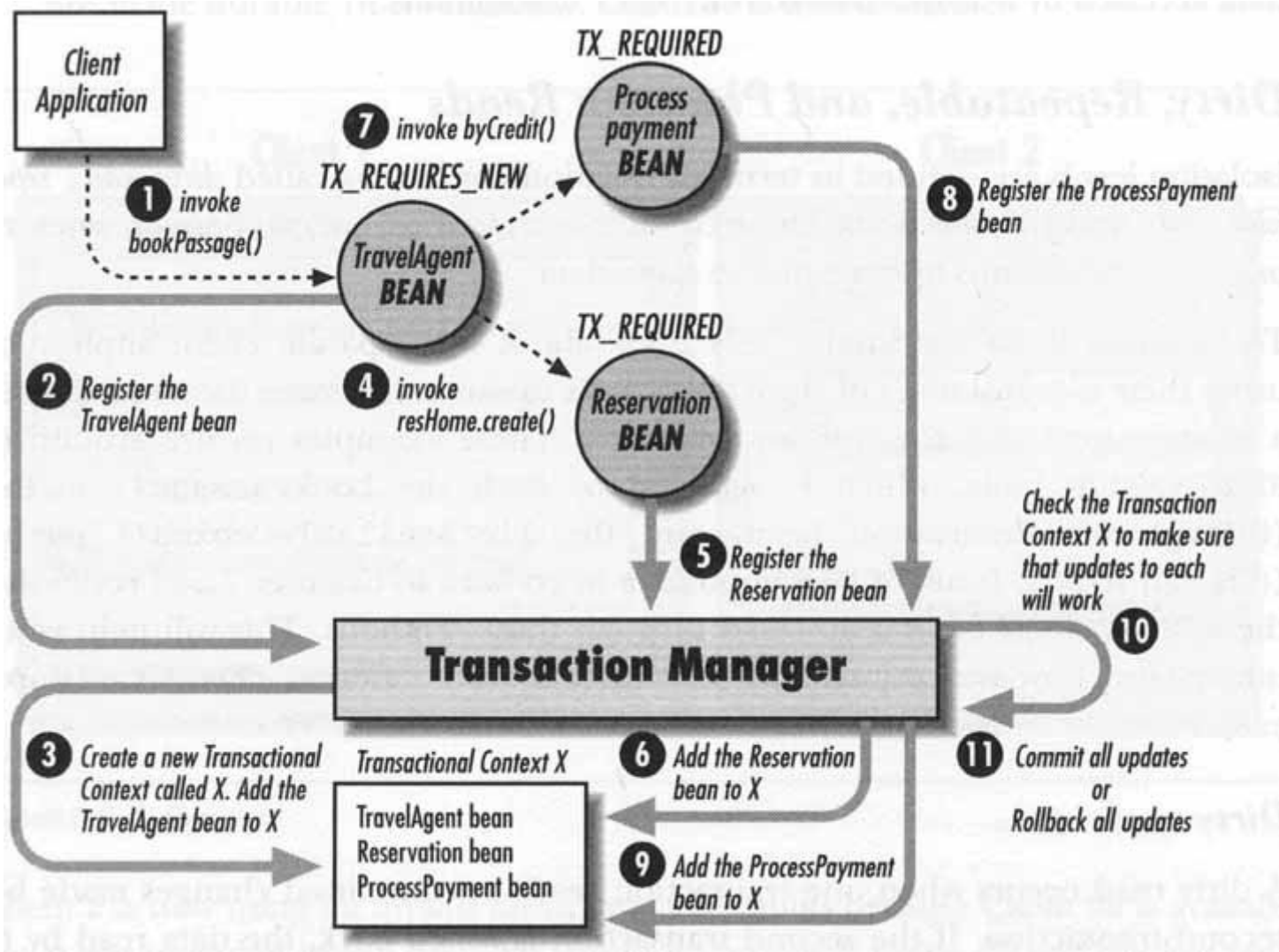
From Gérard Vandome, BullSoft

Transaction
Attributs transactionnels

Beans Transaction Attribute	Client's Transaction	Transaction associated bean's method
TX_NOT_SUPPORTED	None T1	None None
TX_NEVER	None T1	None Error
TX_REQUIRED	None T1	T2 T1
TX_SUPPORTS	None T1	None T1
TX_REQUIRES_NEW	None T1	T2 T2
TX_MANDATORY	None T1	Error T1
TX_BEAN_MANAGED	None T1	

Transaction Exemple

[Monson, p231]



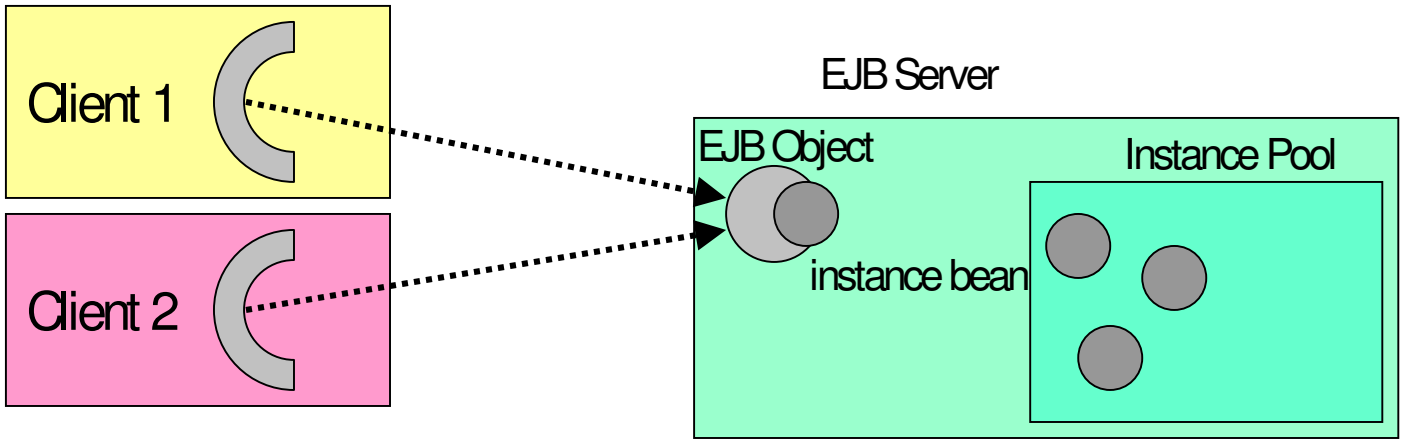
Interface javax.ejb.SessionSynchronization

- Interface optionnelle pour les Stateful Session Bean
- Méthodes de callback

  réagissant au comportement de la transaction
  - afterBegin() juste après le UserTransaction.begin()
  - beforeCompletion() avant le commit()
  - afterCompletion(boolean flag) après le commit() ou rollback()
- Usage
  - Rendre persistant l'état d'un SessionBean
  - Fiabiliser l'envoi de mail
  - …

Gestion de la Concurrence

- Accès de plusieurs clients à un Entity Bean



- Plusieurs problèmes
  - Lecture Sale (Dirty Read)
    - lecture des modifications d ’une autre transaction qui avortera
  - Lecture non répétable (Non repeatable read)
    - la transaction lit successivement deux valeurs différentes d ’une même donnée
  - Lecture Fantôme (Phantom Read)
    - une transaction ne lit pas des données insérées précédenment

Gestion de la Concurrence

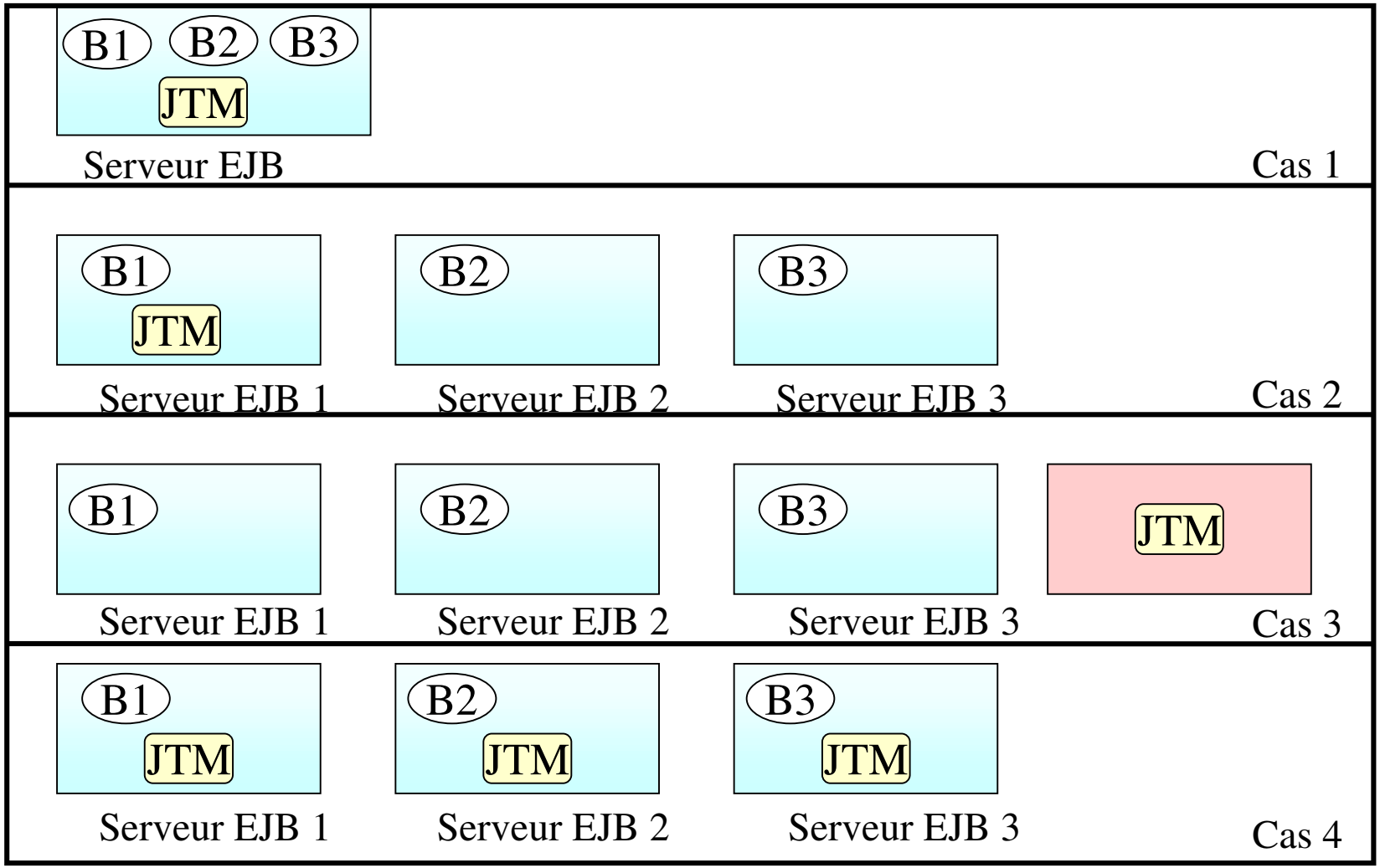
- La source de données « verrouille » la donnée correspondant en fonction du type d ’accès
  - Read Lock
    - la valeur n ’est pas changée par les autres transactions
  - Write Lock
    - la valeur n ’est pas changée par d ’autres transactions
  - Exclusive Write Lock
    - les autres transactions sont bloquées jusqu ’au relachement
  - Snapshot
    - chaque transaction possède une valeur figée
- Une transaction sera bloquée jusqu ’au relâchement du verrou en fonction du niveau d ’isolation souhaité

Gestion de la Concurrence

- Définition des niveaux d ’isolation de la transaction
  - ISOLATION LEVEL (même définition que SQL et JDBC)

Isolation Level	Sale	Non Répétable	Fantômes
TRANSACTION_READ_UNCOMMITED	oui	oui	oui
TRANSACTION_READ_COMMITED	non	oui	oui
TRANSACTION_REPEATABLE_READ	non	non	oui
TRANSACTION_SERIALIZABLE	non	non	non

Répartition



Sécurité

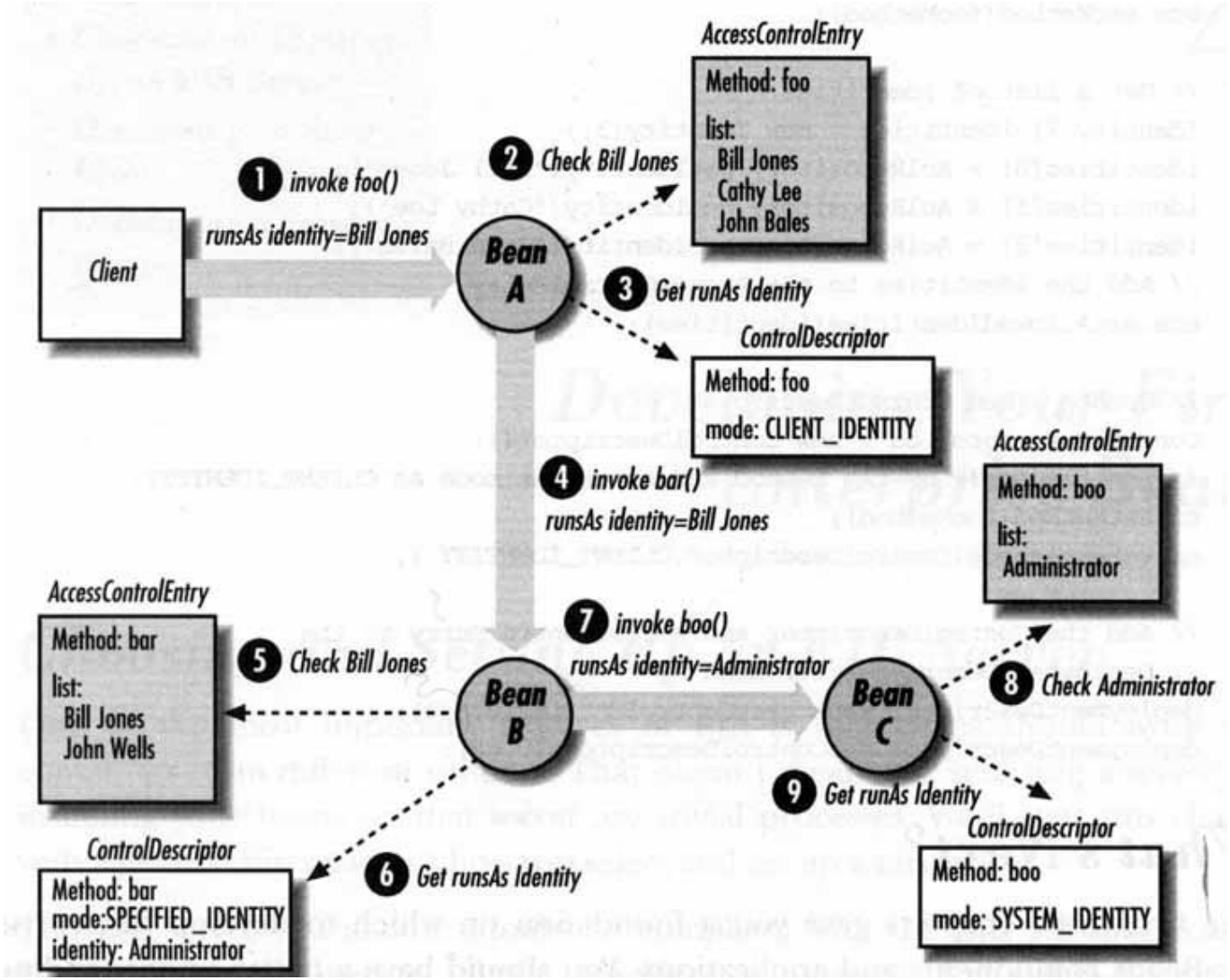
- La gestion de la sécurité est délégué au maximum au conteneur
  - objectif : simplifier la programmation du bean, améliorer la portabilité
- Le support de la sécurité est basé sur
  - l'API sécurité de Java (*javax.security*)
  - les méthodes liées à la sécurité peuvent être implémenté par le conteneur (*javax.ejb.EJBContext* interface)
  - utilisation d'attributs de sécurité défini dans le descripteur du bean utilisé lors de la phase de déploiement

    runAsMode : *CLIENT_IDENTITY, SPECIFIED_IDENTITY, SYSTEM_IDENTITY*

    RunAsIdentity : <identité> si *SPECIFIED_IDENTITY*

Sécurité
Exemple

[Monson, p75]



Principaux bénéfices

- Application complexe “facile” à écrire
  - gestion des transaction de manière déclarative
  - gestion de la persistance
  - gestion intégré de la sécurité
  - gestion de la répartition
- La plate-forme et le bus logiciel (middleware) sont indépendants des applications
- Extensibilité du modèle

Processus de Développement, de Déploiement et de Exécution

- Développement d'un EB
  - Ecrire la classe Primary Key (pour un "entity bean")
  - Ecrire la Home interface
  - Ecrire la Remote interface
  - Ecrire l'implémentation du bean
  - Compiler ces classes et interfaces
- Déploiement du EB
  - Construire le descripteur de déploiement (deploytool)
  - Construire le fichier d'archive .ear
  - Vérifier le fichier d 'archive
- Exécution du EB
  - Activer j2ee
  - Exécuter le client
    - qui peut être une servlet (éventuellement dans un .war déployé)
    - ou une application Java

Exemple:
Beans Bancaires

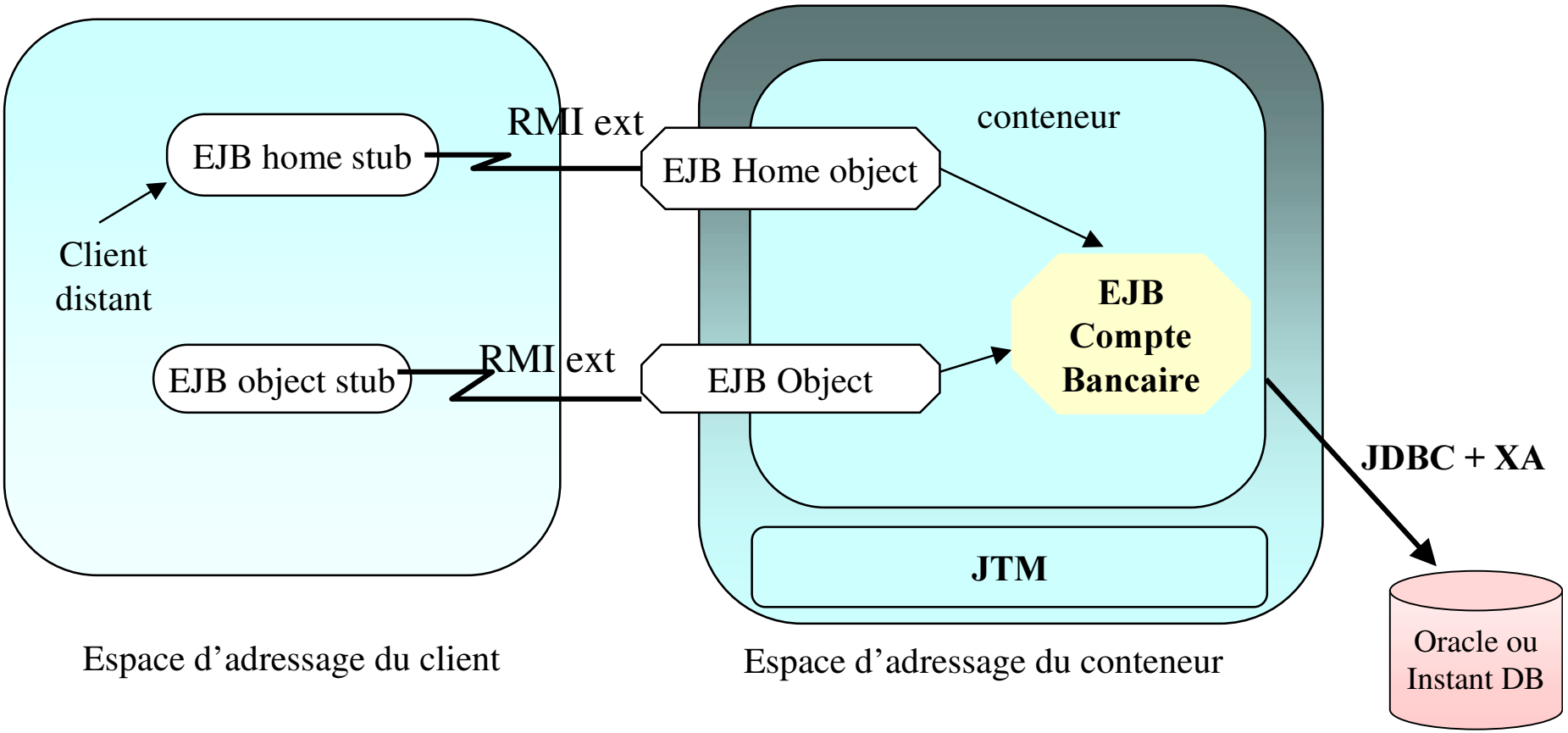
- Gestion de comptes bancaires
  - Compte Bancaire (Composant persistant → Entity Bean)
    - Interface du Bean(vue par le client) → Account.java

```
public interface Account extends EJBObject {
    public double getBalance() throws RemoteException;
    public void   setBalance(double d) throws RemoteException;
    public String getCustomer() throws RemoteException;
    public void   credit(double d) throws RemoteException;
    public void   debit(double d) throws RemoteException;                    }
```

  - Transfert de Fond (Composant session → Session Bean)
    - Interface du Bean (vue par le client) → Transfer.java

```
public interface Transfer extends EJBObject {
    public double transferFund(AccountPK ksrc, AccountPK ktrg,
                               double d) throws RemoteException;
    public double transferFund(Account src, Account trg,
                               double d) throws RemoteException;                    }
```

Architecture de l'application



RMI ext = *extension de RMI afin de propager le contexte de la transaction comme un paramètre supplémentaire (ou utilisation de IIOP)*

Exemple d ’Entity Bean Account

- Spécification du bean
  - entity bean “Account”
  - gestion de la persistance gérée par le conteneur
- Le concepteur doit écrire le code suivant
  - **`AccountHome.java`**

    l’interface de gestion du bean - Home interface
  - **`Account.java`**

    l’interface d’accès à distance - Remote Interface
  - **`AccountPK.java`**

    La classe pour la gestion des clés d’accès aux différents comptes - Primary Key class (nécessaire uniquement pour les “entity beans”, encapsule le champ représentant la clé primaire d’un “entity bean” dans un objet)
  - **`AccountBean.java`**

    le code du bean
  - le descripteur pour le déploiement

AccountPK.java
La clé primaire du compte

- représente la clé primaire
  - n'est pas toujours nécessaire
    cas d'une clé primaire non composée
  - mais la classe doit implementer l'interface Serializable

```
package Bank;
public class AccountPK implements java.io.Serializable {
 public int branchno;
 public int accno;

 public AccountPK(int branchno, int accno) {
  this.branchno = branchno; this.accno = accno; }
 public AccountPK() { }
 public String toString() {
  return String.valueOf(branchno)+"."+ String.valueOf(accno); }
}
```

`Account.java`
l 'interface distance (remote)

- donne les prototypes des méthodes "business"
  - doit étendre l'interface `javax.ejb.EJBObject`

```
package Bank;

public interface Account extends EJBObject {
    public double getBalance() throws RemoteException;
    public void   setBalance(double d)
                        throws RemoteException, AccountException ;
    public String getCustomer() throws RemoteException;
    public void   credit(double d)
                        throws RemoteException, AccountException;
    public void   debit(double d)
                        throws RemoteException, AccountException;
}
```

`Account.java`
l 'interface distance (remote)

- Remarques
  - les méthodes setX et getX correspondent à un attribut X
  - les paramêtres de méthode et les valeurs de retour doivent être de type primitif, serialisable ou étendant/implémentant `java.rmi.Remote`
  - Exceptions
    - les méthodes doivent comporter au minimum `RemoteException`

      exception "système" : SQL, Naming, ...
    - des exceptions applicatives peuvent être ajoutés

```
package Bank;
public class AccountException extends java.lang.Exception {
    public AccountPK pk;
    public AccountException () { super(); }
    public AccountException (AccountPK pk) { super(); this.pk=pk; }
    public AccountException (msg) { super(msg); } }
```

`AccountHome.java`
l 'interface maison (home)

- donne les prototypes des méthodes
  de création/suppresion et de recherche
  équivalent aux Factory de CORBA
  - doit étendre l'interface `javax.ejb.EJBHome`

```
package Bank;
public interface AccountHome extends EJBHome {
public Account create (int branchno, int accno, String customer, double balance)
                                    throws RemoteException, CreateException;
public Account findByPrimaryKey (AccountPK pk)
                                              throws RemoteException, FinderException;
public Enumeration findLargeAccounts (double d)
                                              throws RemoteException, FinderException;
public Enumeration findByCustomer (String str)
                                              throws RemoteException, FinderException;
}
```

`AccountHome.java`
l 'interface maison (home)

Remarques sur les méthodes de création

- comporte `javax.ejb.CreateException`
  - `javax.ejb.DuplicateKeyException` est levée s'il existe déjà une donnée avec la clé primaire
- peut ne pas apparaître dans l'interface Home
  - signifie que les données ne peuvent être créée que depuis la source de données (SGBD)

    par exemple, ordre SQL INSERT

`AccountHome.java`
l 'interface maison (home)

- Remarques sur méthode de recherche
  - comporte `javax.ejb.FinderException`
  - la méthode de recherche sur clé `findByPrimaryKey` retourne un Account
    - `javax.ejb.ObjectNotException` est levée si aucun objet n'est trouvé
  - les autres `findByPrimaryKey` retournent une `Enumeration` de Account ou null si aucun objet n'est trouvé

`AccountBean.java`
l 'implantation de l 'Entity Bean

- doit étendre l'interface `javax.ejb.EntityBean`

- donne l'implantation
  - des méthodes de gestion du cycle de vie ("callback")
    - ejbActivate() appelé lors de l'activation
    - ejbPassivate() appelé lors de la passivation
    - ejbLoad() appelé pour charger l'état de l'EB depuis la BD
    - ejbStore() appelé pour décharger l'état de l'EB vers la BD
    - ejbRemove() appelé quand le client appelle remove()
    - setEntityContext() appelé par le container quand l'instance est crée
    - unsetEntityContext() appelé par le container avant de supprimer l'instance
  - des méthodes "business" de l'interface Account
    - getbalance(), setbalance(), …, credit(), debit() appelées par le client
  - des méthodes de creation de l'interface AccountHome
    - ejbCreate() ejbPostCreate()
  - des méthodes de recherche de l'interface AccountHome
    - ejbFindByPrimaryKey() ejbFindLargeAccounts(), ejbFindByCustomer()

AccountBean.java
l 'implantation de l 'Entity Bean

- Les attributs

```
package Bank;
public class AccountBean implements EntityBean {

 // les attributs

 private transient EntityContext ctx;
 public  int          branchno;    // PK
 public  int          accno;       // PK
 public  String       customer;
 public  double       balance;

...
```

AccountBean.java
l 'implantation de l 'Entity Bean

- Les méthodes de cycle de vie
  - requises par l'interface EntityBean

```
public void ejbActivate() { }
public void ejbDestroy() { }
public void ejbPassivate() { }
public void ejbLoad() { }
public void ejbStore() { }
public void setEntityContext (EntityContext ctx) {   this.ctx = ctx; }
public void unsetEntityContext () {  this.ctx = null; }
...
```

AccountBean.java
l 'implantation de l 'Entity Bean

- Les méthodes "business"
  - correspond à l'interface distante

```
public double getBalance() throws RemoteException { return balance;}
public void setBalance(double d) throws RemoteException, AccountException{
    if (d<0) throw new AccountException(new AccountPK(this.branchno, this.accno));
    balance = d; }
public String  getCustomer() throws RemoteException { return customer;}
public void  credit(double d) throws RemoteException, AccountException{
    if (d<0) throw new AccountException(new AccountPK(this.branchno, this.accno));
    balance += d; }
public void  debit(double d) throws RemoteException, AccountException{
    if (d<0 && (balance-d<0))
      throw new AccountException(new AccountPK(this.branchno, this.accno));
    balance -= d; }
...
```

AccountBean.java
l 'implantation de l 'Entity Bean

- Les méthodes de création

```
public void ejbCreate ( int branchno, int accno,
                        String customer, double balance) throws
    CreateException {
  if(balance<0) throw new CreateException();
  this.branchno=branchno;        this.accno=accno;
  this.customer=customer;        this.balance=balance;
}
public void ejbPostCreate ( int branchno, int accno,
                            String customer, double balance) { }
}
```

- Les méthodes de recherche
  - sont générés au déploiement par le container

ejbFindByPrimaryKey (), ejbFindLargeAccounts (), ejbFindByCustomer ()

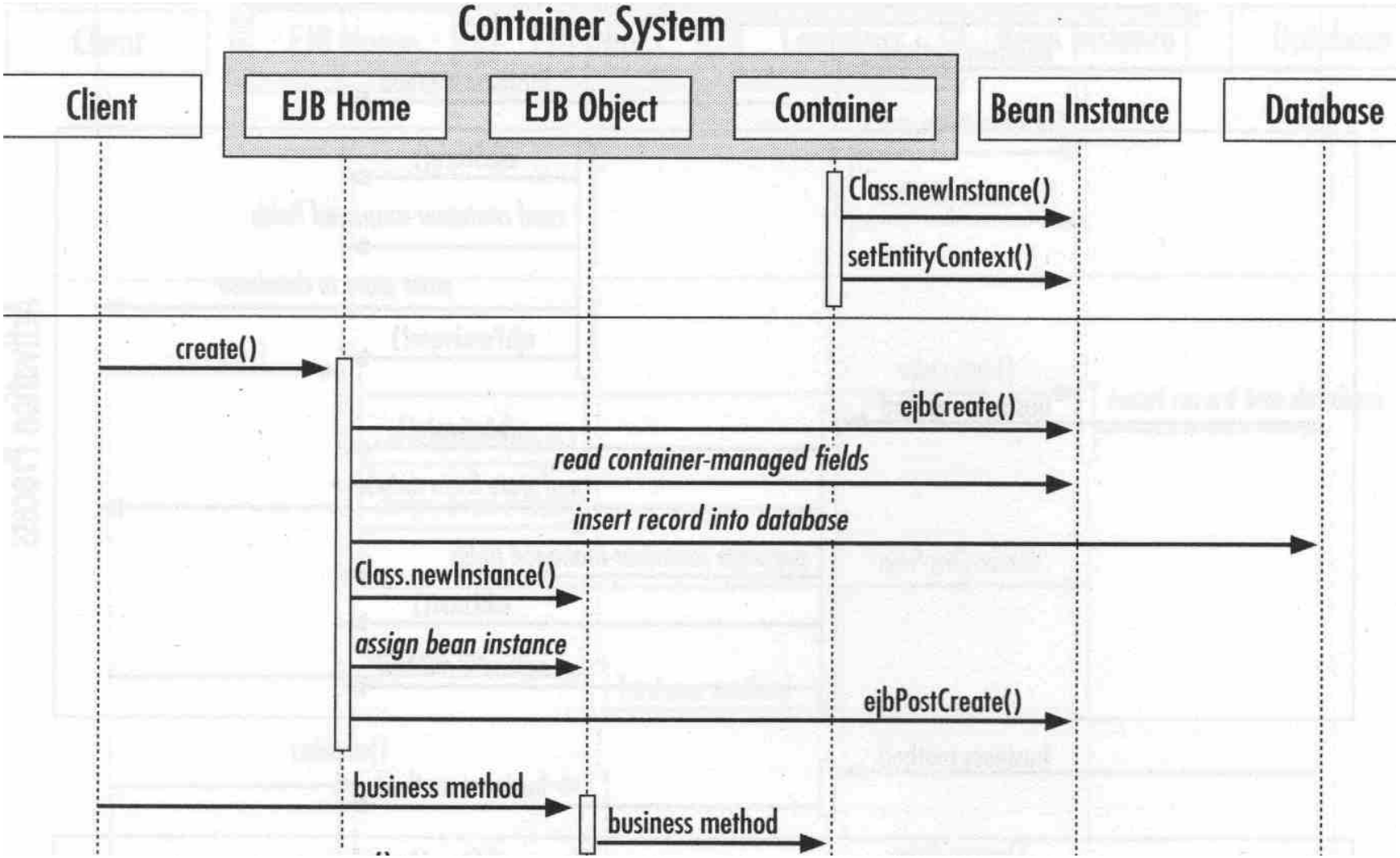
AccountBean.java

Remarque sur les méthodes de création

- pour chaque méthode create(X) de l'interface distante
- il y a un couple ejbCreate(X) et ejbPostCreate(X)
  - ejbPostCreate(X) effectue des compléments à la création

AccountBean.java

Remarque sur les méthodes de création



AccountBean.java

Remarque sur les méthodes de synchronisation ejbLoad() et ejbStore()

- Container-Managed Persistence
  - en principe, ne font rien
  - néanmoins, elles peuvent être complétées pour formater des données commplexes, …

```
public class StringsBean implements EntityBean {
  private transient EntityContext ctx;
  private transient Vector msgvec;
  public  String     msgs; // Persistent field dans une colonne VARCHAR
   public void ejbLoad() {
    StringTokenizer st=new StringTokenizer(msgs,"~");
    while(st.hasMoreTokens()) msgvec.addElement(st.nextToken());
} ... }
```

Descripteur de déploiement (addendum)

- Si la persistance est gérée par le conteneur, le Bean fournisseur/installateur doit spécifier, dans l'environnement les propriétés du Bean
  - le lien avec la base de données

    datasource.name jdbc_oracle1
    db.TableName Account
    db.Field.branchno branchid
    db.Field.accno accountid
    db.Field.customer customer
    db.Field.balance balance
  - la méthode de recherche du bean

    db.Finder.findLargeAccount where balance > ?
    db.Finder.findByName where customer like ?

AccountBean.java

Remarque sur la persistance gérée par le bean (Bean-Managed Persistence)

- Les méthodes de création, de suppression, de synchronisation et de recherche doivent être codé par le développeur

- Exemple avec ejbCreate()

```
public void ejbCreate ( int branchno, int accno, String customer, double
    balance)
   throws CreateException {
  if(balance<0) throw new CreateException();
  this.branchno=branchno;          this.accno=accno;
  this.customer=customer;          this.balance=balance;
  Connection con = null; PreparedStatement ps = null;
  try { con = getConnection(); ps = con.prepareStatement(
  "insert into Account (branchid,accountid,customer,balance) values (?,?,?,?)");
  ps.setInt(1, branchno);                    ps.setInt(2, accno);
  ps.setString(3,customer);                  ps.setDouble(4,balance);
  if (ps.executeUpdate() != 1) {
          throw new CreateException ("Failed to add Account to database");
  } catch (SQLException se) {  throw new CreateException (se.getMessage());
}
```

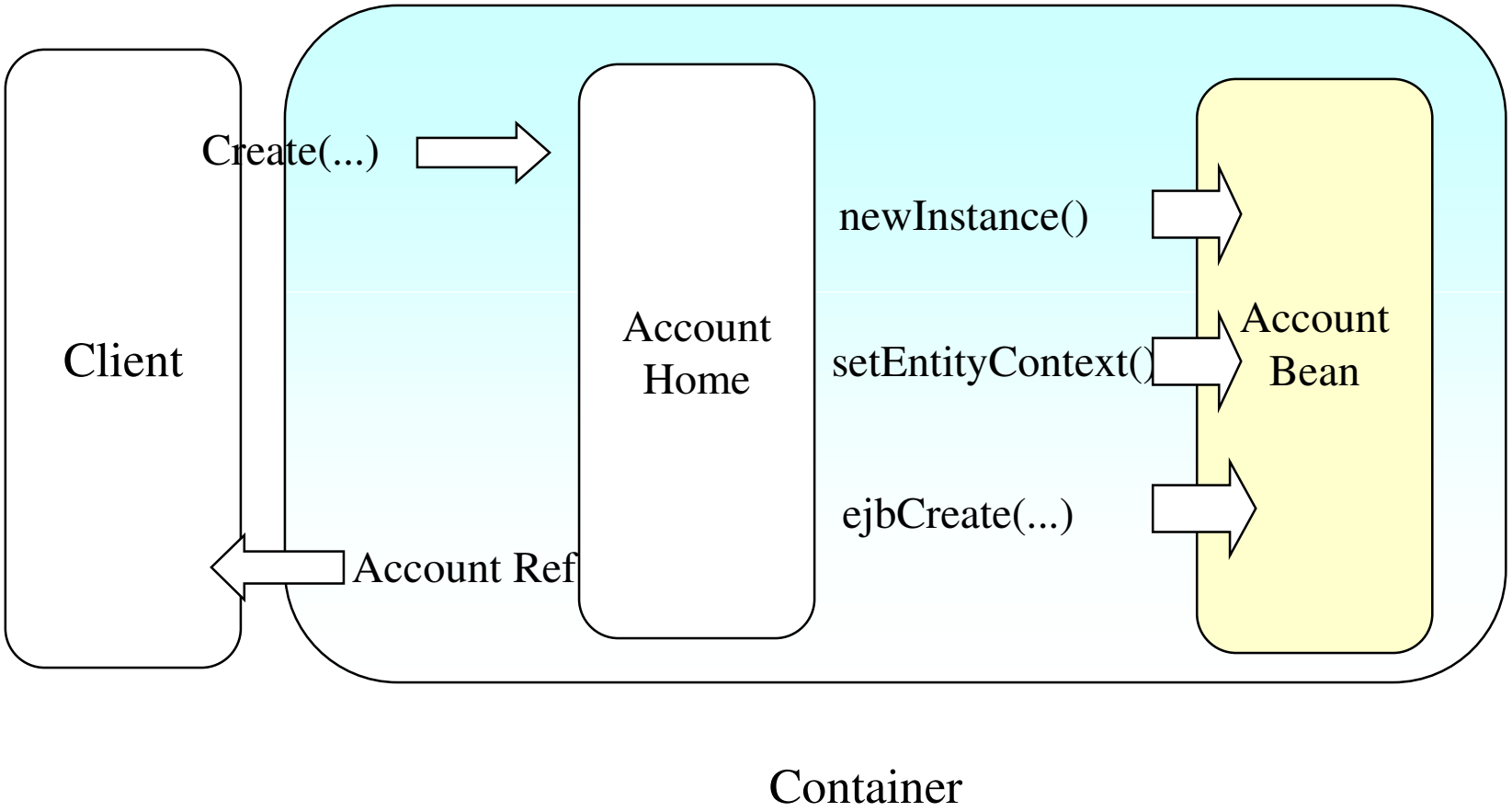
AccountBean.java

Remarque sur la persistance gérée par le bean (Bean-Managed Persistence)

- Exemple avec ejbLoad()

```
public void ejbLoad ()   throws RemoteException{
   AccountPK  pk = (AccountPK) context.getPrimaryKey();
   Connection con = null; PreparedStatement ps = null; ResultSet result = null;
   try { con = getConnection(); ps = con.prepareStatement(
    "select customer,balance from Account where branchid=? and
    accountid=?");
    ps.setInt(1,pk.branchno); ps.setInt(2,pk.accno);
    result = ps.executeQuery();
   if(result.next()){
         branchno=pk.branchno; accno=pk.accno;
         customer = result.getString("customer");
         balance = result.getDouble("balance");
   }else{ throw new ObjectNotFoundException("Cannot find Account "+pk); }
...
```

Creation des instances
Home interface



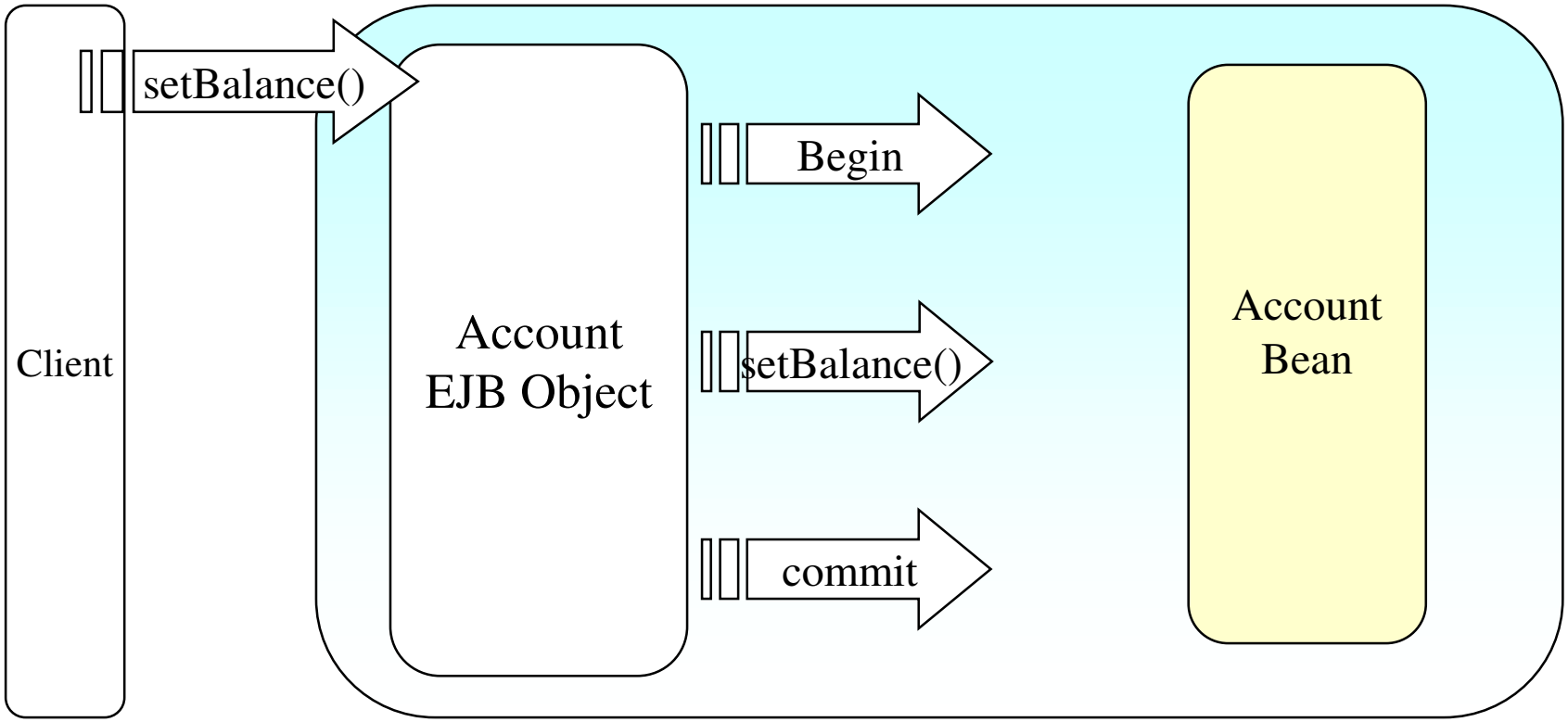
Le client (Application Java)

```
import bank.*; ...
public class ClientCreate{
  public static void main(String[] argv){
  AccountHome ah; Account a;
  try{ // rechercher l'interface EJB home interface
      InitialContext ctx = new InitialContext();
      Object objref = ctx.lookup("BankAccount");
      ah = (AccountHome)PortableRemoteObject.narrow(objref, AccountHome.class);
    }  catch (NamingException) { NamingException.printStackTrace(); }
// créer une instance d'entité
    a=ah.create(argv[0],argv[1],argv[2],0);
// invoquer une méthode "business"
    System.err.println(a.getCustomer());
}
```

Le client (Application Java) EJB1.1

```
import bank.*; ...
public class ClientCreate{
  public static void main(String[] argv){
  AccountHome ah; Account a;
  try{ // rechercher l'interface EJB home interface
      InitialContext ctx = new InitialContext();
      Object objref = ctx.lookup(" java:comp/env/ejb/BankAccount");
      ah = (AccountHome)PortableRemoteObject.narrow(objref, AccountHome.class);
}  catch (NamingException) { NamingException.printStackTrace(); }
// créer une instance d'entité
    a=ah.create(argv[0],argv[1],argv[2],0);
// invoquer une méthode "business"
    System.err.println(a.getCustomer());
}
```

Méthodes de l'application
Remote interface



Le client (Application Java)

```
import bank.*; ...
public class ClientSetBalance{
  public static void main(String[] argv){
  AccountHome ah; Account a;
  try{ // rechercher l'interface EJB home interface
      InitialContext ctx = new InitialContext();
      Object objref = ctx.lookup("BankAccount");
      ah = (AccountHome)PortableRemoteObject.narrow(objref, AccountHome.class);
    }  catch (NamingException) { NamingException.printStackTrace(); }
// créer une instance d'entité
   try{
     a=ah.findByPrimaryKey(new AccountPK(argv[0],argv[1]));
   } catch (FinderException) { FinderException.printStackTrace(); }
// invoquer une méthode "business"
    a.setBalance(argv[2]);
}
```

Le client (Servlet 1/2)

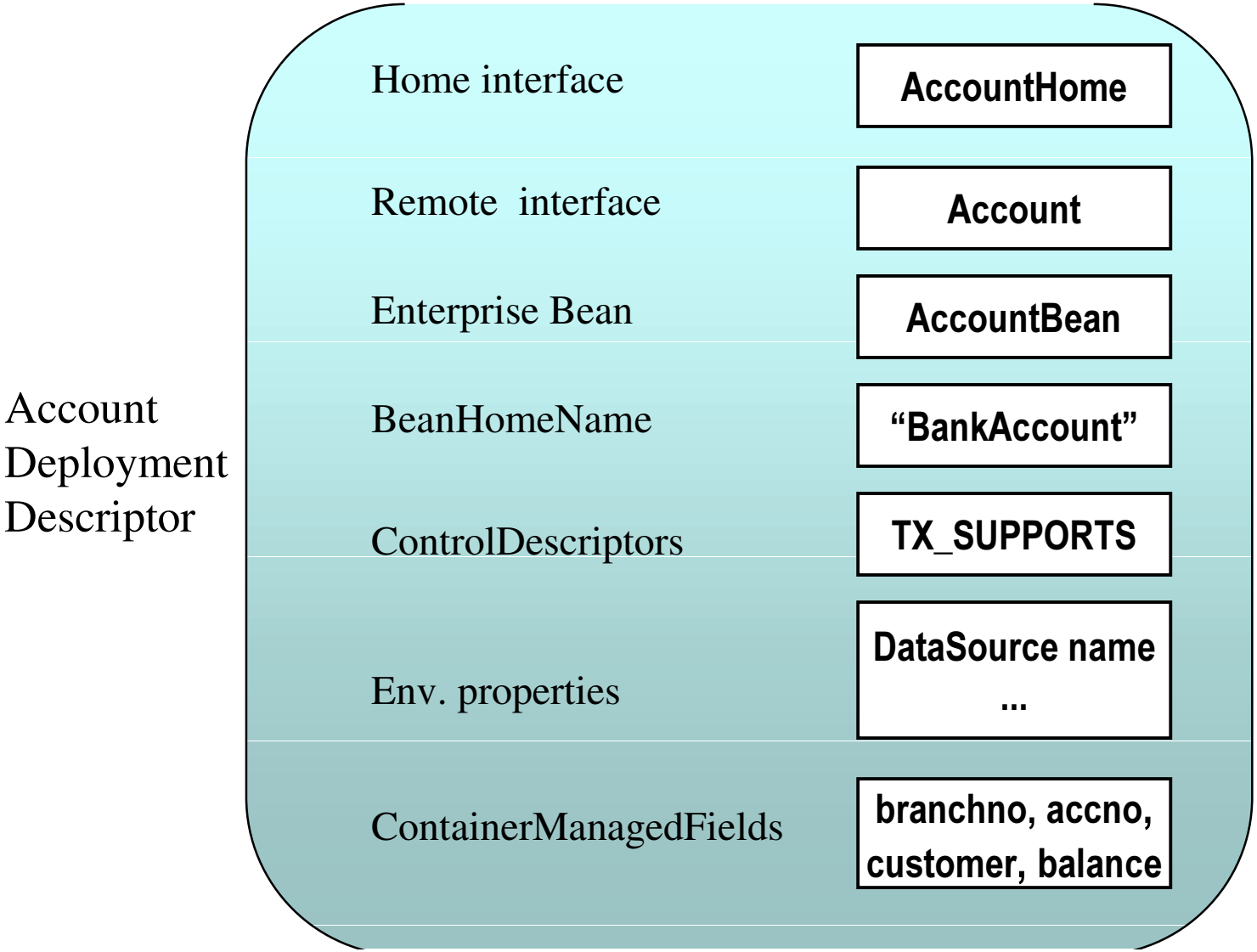
```
import ...
public class SetBalanceServlet extends HttpServlet {
  AccountHome ah;
public void init(ServletConfig config) throws ServletException{
    try{
      InitialContext ctx = new InitialContext();
      Object objref = ctx.lookup("BankAccount");
      ah = (AccountHome)PortableRemoteObject.narrow(objref, AccountHome.class);
    }  catch (NamingException) { NamingException.printStackTrace(); }
}

public void destroy() {
    System.out.println("Destroy");
  }
...
```

Le client (Servlet 2/2)

```
public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException  {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML><HEAD><TITLE>SetBalance</TITLE></HEAD><BODY>");
int accno = (new Integer(request.getParameter("ACCNO")).intValue();
int branchno = (new Integer(request.getParameter("BRANCHNO")).intValue();
AccountPK pk=new AccountPK(branchno,accno);
double newbalance=(new Double(request.getParameter("BALANCE")).doubleValue();
try{
   Account a=ah.findByPrimaryKey(pk);              a.setBalance(newbalance);
   out.println("<H1>New Balance</H1>");
   out.println("<P>Account: " + pk + "<P><P>New Balance: " + a.getBalance() + "<P>");
} catch (FinderException) { out.println("<H1>Problem</H1><P>No Account: " + pk + "<P>"); }
  out.println("</BODY></HTML>"); out.close();
}}
```

Descripteur de déploiement



Gestion déclarative des transactions

- Support du conteneur pour les transactions
  - le conteneur utilise les attributs de transaction pour réaliser la politique souhaitée
- Attributs des transactions

TX_NOT_SUPPORTED
TX_NEVER
TX_REQUIRED
TX_SUPPORTS
TX_REQUIRES_NEW
TX_MANDATORY
TX_BEAN_MANAGED

Gestion des transactions

Le conteneur gère les transaction à l'aide des classes d'interposition qui sont générées automatiquement

BullBean.java

```
preInvoke() {
 case TX_REQUIRES_NEW:
        clientTransaction=current.suspend();
        Current.begin();

 postInvoke(){
   case TX_REQUIRES_NEW:
          Current.commit();
          Current.resume(clientTransaction);
```

BullBeanAccount.java

```
public void setBalance(double d){

        this.preInvoke();
        AccountBean.setBalance(d);
        this.postInvoke();
```

Exemple de Session Bean Transfer

- Spécification du bean
  - session bean “Transfer”
- Le concepteur doit écrire le code suivant
  - **`TransferHome.java`**

    l’interface de gestion du bean - Home interface
  - **`Transfer.java`**

    l’interface d’accès à distance - Remote Interface
  - **`TransferBean.java`**

    le code du bean
  - le descripteur pour le déploiement
  - pas de TransferPK !!

`Transfer.java`
l 'interface distance (remote)

- donne les prototypes des méthodes "business"
  - doit étendre l'interface `javax.ejb.EJBObject`

```
package Bank;

public interface Transfer extends EJBObject {
      public void fundTransfer(AccountPK src, AccountPK trg, double amount)
                  throws RemoteException, AccountException;
}
```

`TransferHome.java`
l 'interface maison (home)

- donne les prototypes des méthodes de création
  - doit étendre l'interface `javax.ejb.EJBHome`

```
package Bank;
public interface TranferHome extends EJBHome {
public Tranfer create () throws RemoteException, CreateException;
}
```

`TransferBean.java`
l 'implantation du Session Bean

- doit étendre l'interface `javax.ejb.SessionBean`

- donne l'implantation
  - des méthodes de gestion du cycle de vie ("callback")
    - ejbActivate() appelé lors de l'activation
    - ejbPassivate() appelé lors de la passivation
    - ejbLoad() {} inutilisé mais présent
    - ejbStore() {} inutilisé mais présent
    - ejbRemove() appelé quand le client appelle remove()
    - setEntityContext() appelé par le container quand l'instance est crée
    - unsetEntityContext() appelé par le container avant de supprimer l'instance
  - des méthodes "business" de l'interface Account
    - fundTransfer() appelée par le client
  - des méthodes de creation de l'interface AccountHome
    - ejbCreate() ejbPostCreate()

TransferBean.java
l 'implantation du Session Bean

```
package Bank;
public class TransferBean implements SessionBean {
  private transient EntityContext ctx;
  private AccountHome ah;

public void fundTransfer(AccountPK src, AccountPK trg, double amount)
                    throws RemoteException, AccountException {
   try { Account asrc=ah.findByPrimaryKey(src);
   } catch(FinderException e){ throw new AccountException(src); }
   try { Account atrg=ah.findByPrimaryKey(trg);
   } catch(FinderException e){ throw new AccountException(trg); }
   asrc.debit(amount);
   atrg.credit(amount);
}
...
```

TransferBean.java
l ’implantation du Session Bean

```
...
//méthodes requises par l’interface SessionBean
  public void ejbActivate() { }
  public void ejbDestroy() { }
  public void ejbPassivate() { }
  public void ejbLoad() { }
  public void ejbStore() { }
  public void setEntityContext (EntityContext ctx) {   this.ctx = ctx; }
  public void unsetEntityContext () {  this.ctx = null; }
...
```

TransferBean.java
l 'implantation du Session Bean

```
public void ejbCreate () throws CreateException {
  try{
    InitialContext ctx = new InitialContext();
    Object objref = ctx.lookup("Transfer");
    this.ah = (AccountHome)PortableRemoteObject.narrow
                                                objref, TransferHome.class);
    }  catch (NamingException) { throw new CreateException() }
}

public void ejbPostCreate ()
        { }
}
```

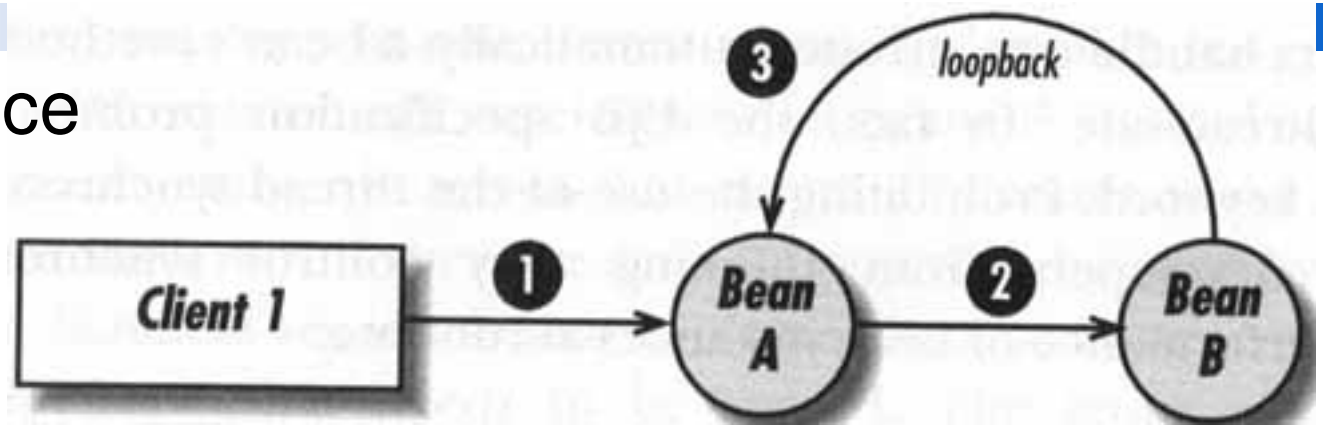
Gestion de la sécurité

Stratégies de Conception

- Amélioration des performances avec un session bean
  - Entity-Bean : grain fin / Session Bean : gros grain
  - réduction du trafic réseau et de la latence

Limites : La réentrance

- La réentrance



- Exemple

```
public class A_Bean implements EntityBean { // A est l 'interface distante
   private transient EntityContext ctx;
   public  void method_1() {
      B b= ... // récupère une référence distante
      A myself=(A)ctx.getEJBObject();
      b.method_2(myself); }};
public class B_Bean implements EntityBean { // B est l 'interface distante
   private transient EntityContext ctx;
   public  void method_2(A a) { a.method_3();/* loopback */ }};
```

Limites : La réentrance

- interdit pour les session bean
- autorisé mais déconseillé pour les entity bean

Problème

- pas de différence
  entre multithreading multiclient et réentrance

Relations entre composants CORBA 3.0 et les EJB

- Un EJB peut être placé dans un conteneur de composants CORBA
- Un composant CORBA écrit en Java et utilisant seulement les APIs EJB peut être mis dans un conteneur EJB
- Correspondance entre les APIs EJB et les APIs CORBA Components

EJB vs MTS

- Voir Gopalan
  - http://members.tripod.com/gsraj/misc/ejbmts/

Outils de Développement

- Bases de Données
- Moniteurs Transactionnels
- Serveurs Applicatifs
- Outils de programmation

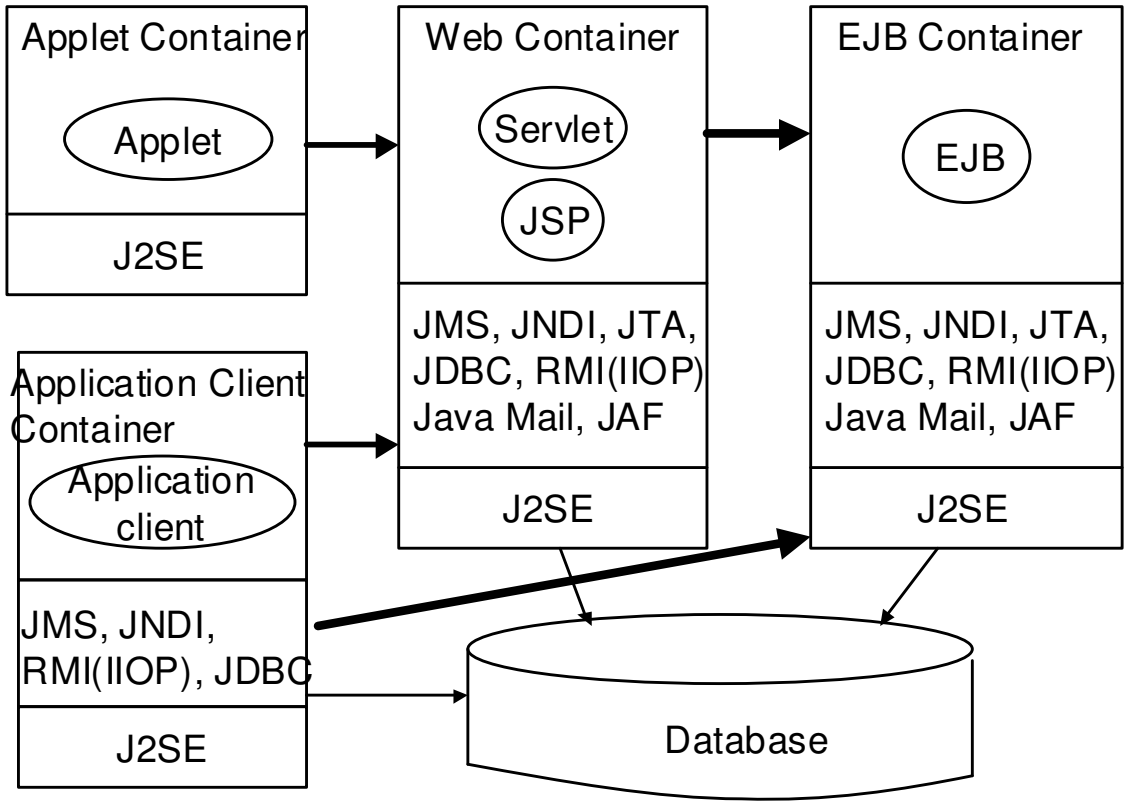
Outils de Développement

- WebLogic
- IBM
- Oracle
- GemStone
- BEA
- Borland
- Netscape
- Lotus
- SilverStream
- Forte
- Progress
- Novell
- Novera
- Borland
- Informix
- IONA
- Allaire
- EJBHome, JOnAS, ...

J2EE
Java 2 Enterprise Edition

- Spécifie la plateforme complet Java pour des applications multi-tiers (vs J2SE,J2ME)

Architecture



J2EE
Java 2 Enterprise Edition

- Deux types de composants J2EE
  - Web Component (fichier JAR .war)
    - Servlet, JSP, HTML (formulaire associé à une servlet)
  - EB Component (fichier JAR .ear)
    - Entity Bean, Session Bean
- Plateforme de référence
  - SUN j2sdkee

Java ONE (Open Net Environment)

- Annoncé le 5/2/2001
- Ajout des « standards » Web Services dans J2EE
  - SOAP, WSDL, UDDI, ebXML, …
- Nouvelles API
  - Context API

    Interface vers les données contextuelles d'un service Web
  - JAX/RPC (Java API for XML based RPC)

    Interface aux protocoles de transport de message XML de type RPC (SOAP, W3C XP, …)
  - JAXB (Java API for XML Data Binding)

    Correspondance automatique entre des objets Java et des données XML
  - JAXM (Java API for XML Messaging)

    Interface aux protocoles de transport de messages XML : SOAP, ebXML Message Service, …
  - JAXP (Java API for XML Processing)

    Gestion des arborescence DOM
  - JAXR (Java API for XML Registries)

    Interface aux annuaires ebXML Registry/Repository et UDDI Business Registry

Les spécifications EJB

- EJB 1.0
- EJB 1.1
  - JDBC 2.0, Descripteurs XML
  - contexte standard JNDI
- EJB 2.0
  - Connector
  - Message Driven Bean (JMS), ...
- La suite
  - Héritage ?
  - Réentrance ?
  - Modèles de transaction avancées ?

EJB 2.0 What' new

- Abstract Persistente State
- What is persistent state ?
- Notion of « dependent classes »
- Can be pluggable
- Relationship
  - EJB1.1 coded by the developper
  - in 2.0 ??
- New methods : Select
    - can b
  - ejBSelect<type>
  - ejbSelect<type>inEntity
- New ejbHome methods

Persistence

- Abstract Persistente State
- What is persistent state ?
- Notion of « dependent classes »
- Can be pluggable
- Relationship
  - EJB1.1 coded by the developper
  - in 2.0 ??
- New methods : Select
    - can b
  - ejBSelect<type>
  - ejbSelect<type>inEntity
- New ejbHome methods

EJB QL (Query Language)

- SQL like
  - used to specified finder and select
- Form of Query
  - [Select_clause] From_clause [Where_Clause]
  - Navigation
- Query
  - static in DD
  - has parameters that corresponds to finder/select methods parameters
  - can use finder methods of other entity beans

EJB QL (Query Language) Example

- No SELECT and navigation

```
FROM OrderBean o, I IN o.lineItems
WHERE I.product.product_type = ?
```

- SELECT and @@ operator

```
SELECT @@o1
FROM OrderBean o1, OrderBean o2
WHERE o1.quantity > o2.quantity AND
o2.customer.lastname = 'Smith' AND
o2.customer.firstname= 'John'
```

- SELECT and @@ operator

```
SELECT I=>product FROM OrderBean AS o, I IN o.lineItems
SELECT I.product FROM OrderBean AS o, I IN o.lineItems -- Illegal
```

JMS Integration

- publisher
  - bean can send message
- subscriber - Message Driven Bean
  - new EJB bean type
  - similar to Stateless Session Bean
  - transaction attribute of onMessage()

    `Only : NotSupported, Required`

    - The use of the other transaction attributes is not meanigful for message-driven beans because there can be no pre-existing transaction context (`RequiresNew`, `Supports`) and no client to handle exceptions (`Mandatory`, `Never`).
- Transaction
  - Receive/Send Messages can be in a transaction scope

MessageDrivenBean

- Subscriber of an JMS Queue or Topic

- ```
  package javax.ejb;
  import javax.jms.Message;
  import javax.jms.MessageListener;

  public interface MessageDrivenBean extends MessageListener{
    public void onMessage(Message message);
    public void ejbCreate();
    public void ejbRemove();
    public void setMessageDrivenContext(MessageDrivenContext mdc);
  }
  ```
  ```

MessageDrivenBean Exemple

```
public class MarketingBean implements javax.ejb.MessageDrivenBean {

    public void onMessage(Message message) {

    ObjectMessage orderMessage = (ObjectMessage)orderMessage:
    OrderDetail orderDetail = (OrderDetail)orderMessage.getObject();

    Integer customerID = orderDetail.getCustomerID();

    InitialContext jndiEnc = new InitialContext();
    CatalogHome catalogHome =
    (CatalogHome)jndiEnc.lookup("java:comp/env/ejb/catalog");

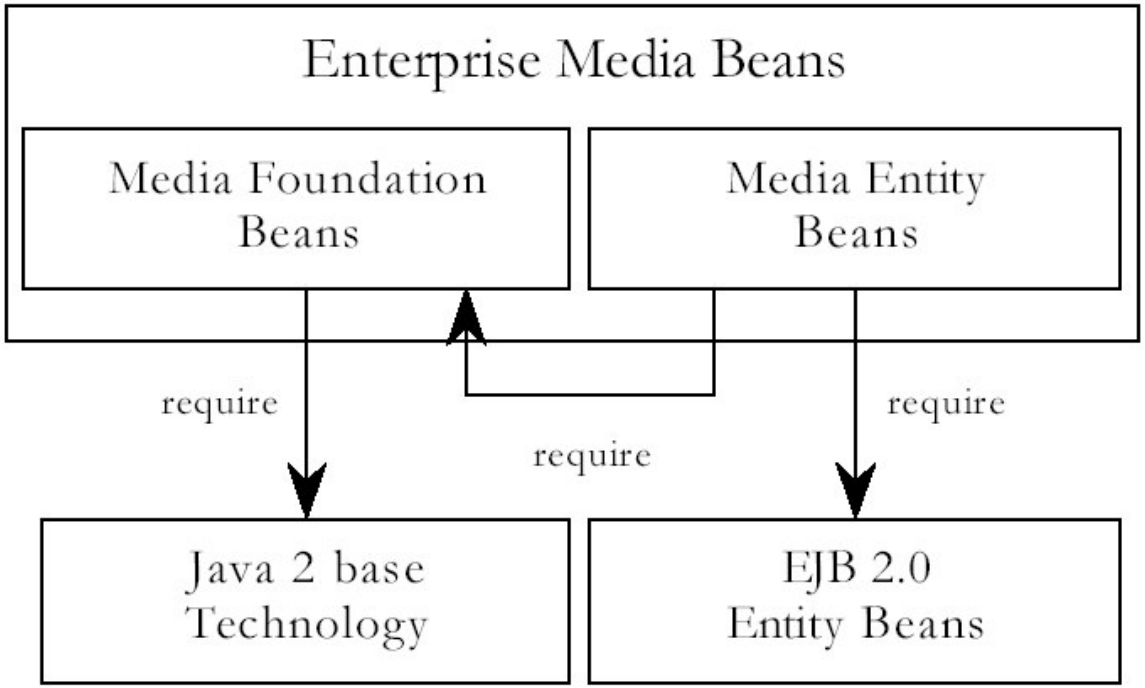
    Iterator productIDs = orderDetail.getProductsIDs();
    while(productIDs.hasNext()){
    Integer productID = (Integer)productIDs.next();
    Catalog cat = CatalogHome.findByProductID(productID);
    cat.addCustomerToMailingList(customerID);
    }
    }
    }
```

Timer Service

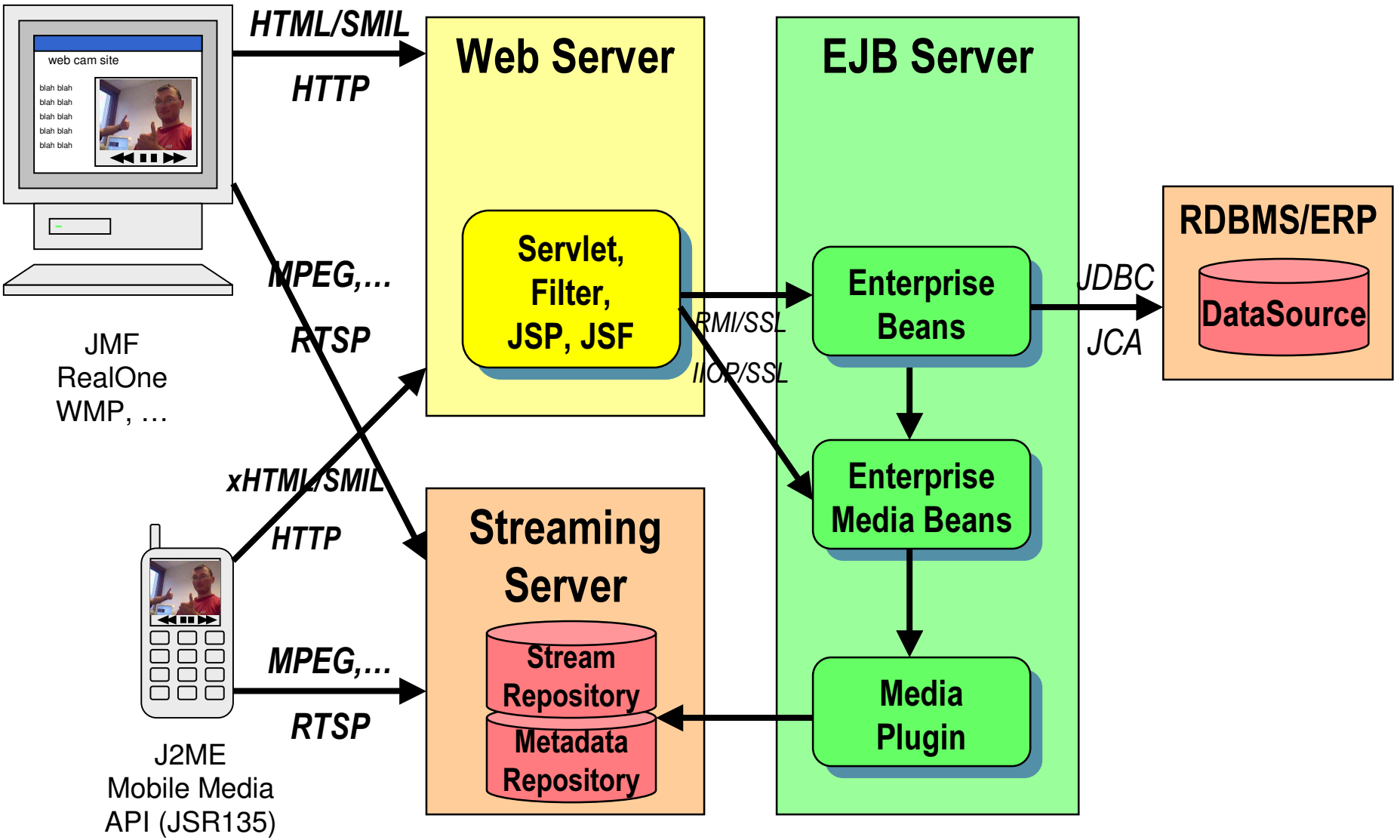
- Business Work-Flows or Process-Flows
  - rely on timed notifications.
- EJB Timer service
  - Notify beans (except SF SB) but must implement javax.ejb.TimedObject
  - Schedule
    - at a specific time (« at 10:30 AM on May 23 »),
    - after a duration of time (« in 30 days »)
    - or at timed intervals (« every 12 hours »).
  - Call the ejbTimeout method of the bean's implementation class defined by the javax.ejb.TimedObject interface
- 4 interfaces in the javax.ejb
  - TimedObject, Timer, TimerHandle, TimerService

EMB (Enterprise Media Beans) *JSR0086*

- Motivations
  - Intégration des données multimedia (orienté Streaming) dans les applications J2EE (EJB-JSP/JSF/Servlet)
- 2 types de composants EMB



Architecture d'applications utilisant des EMB

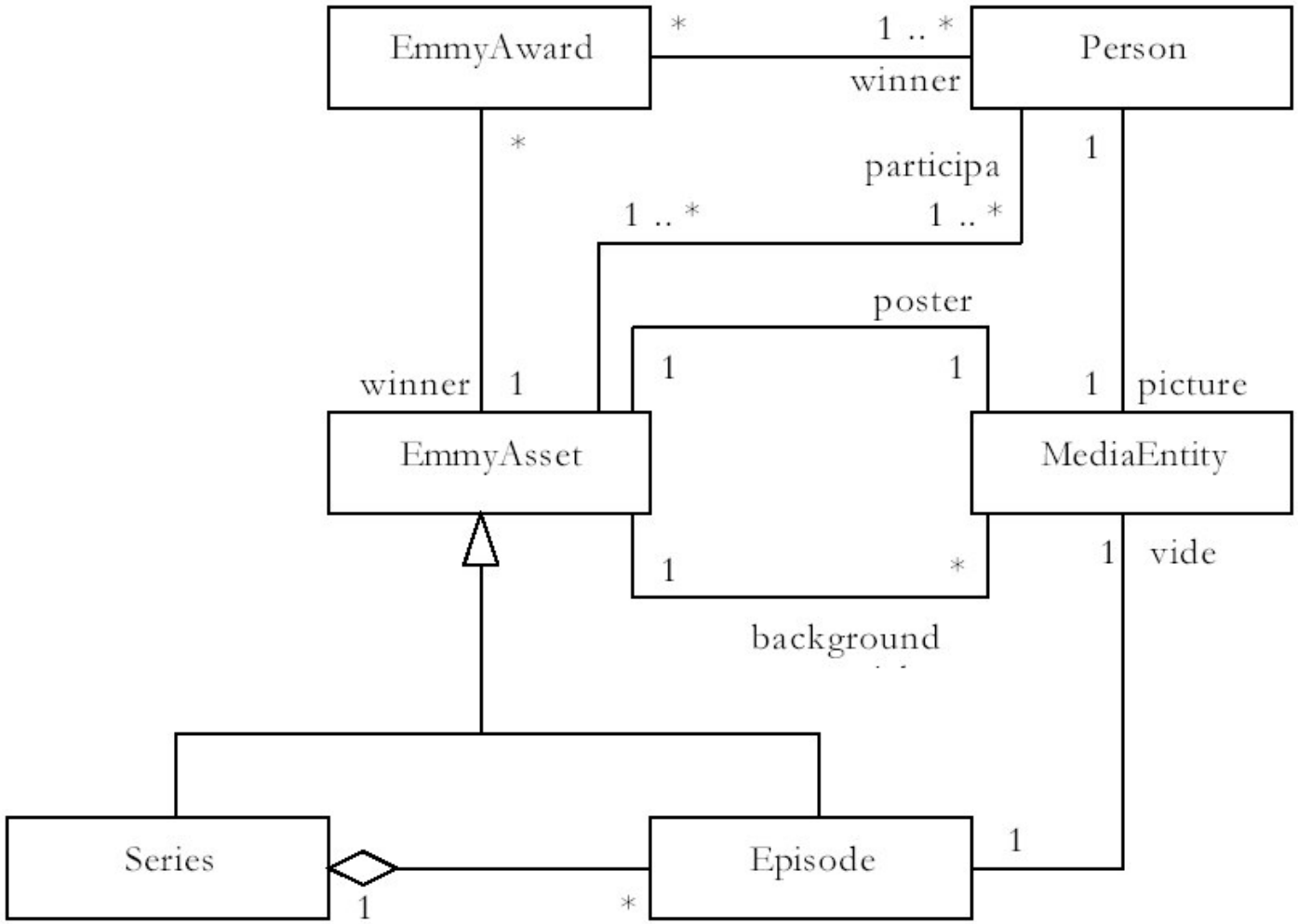


Composants EMB

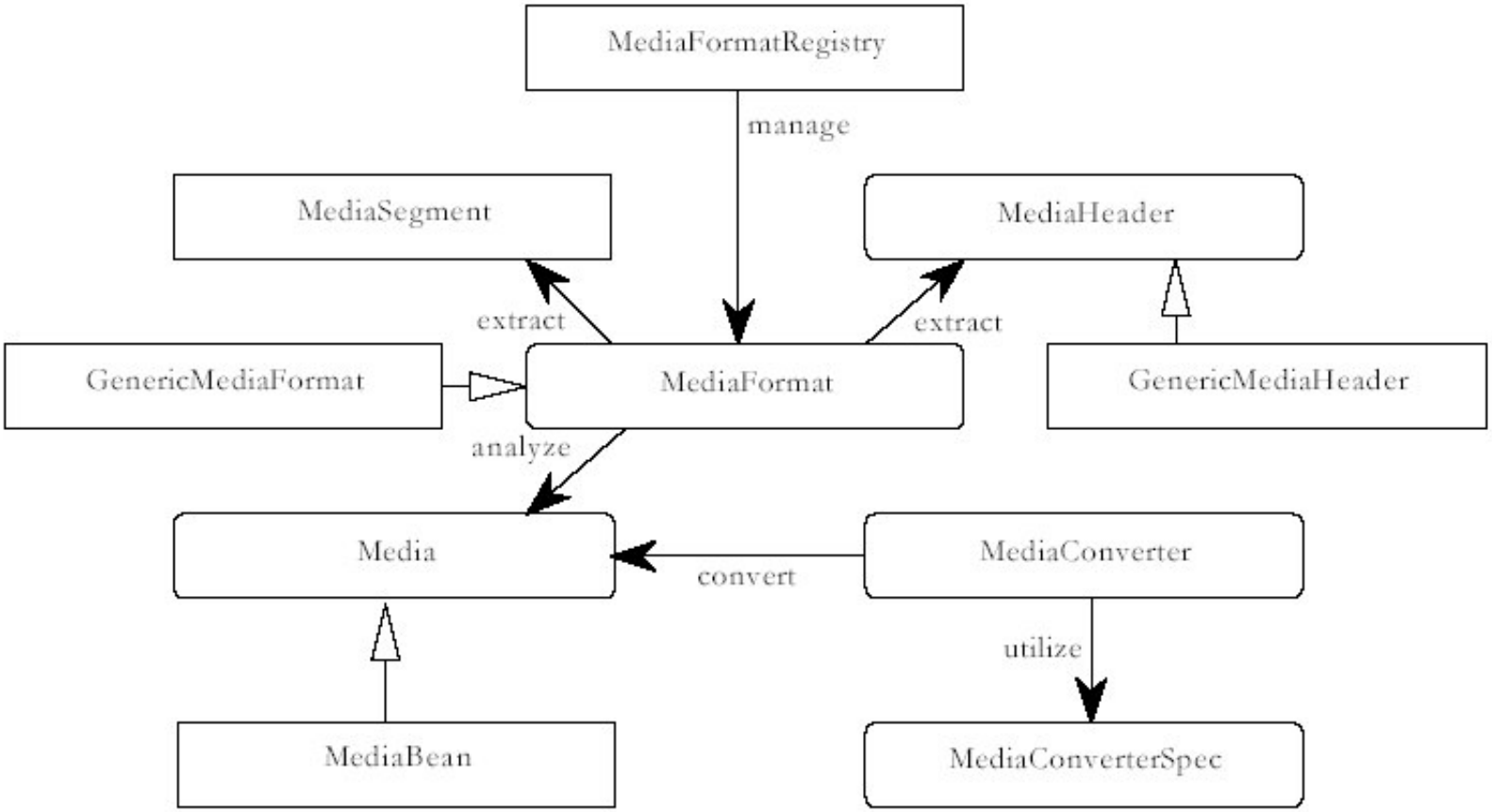
- Media Foundation Beans
  - *Transient, local, immutable*
  - Services
    - Extraction des metadatas (durée d'une chanson, …)
    - Conversion de formats, ajout d'information (watermark, …)
    - Analyse des capacités du terminal (CC/PP)
  - Architectures à plugin
    - Media Format Plug, Media Converter
- Media Entity Beans
  - Représente des objets multimédia
    - *Persistent* (Primary Key) et *modifiable et observable*
  - Les objets multimédia peuvent être liés les uns aux autres
    - Inclusion, prédecesseur/sucesseur, content/representant (thumbnail), …
  - Intégration *seamless* dans une application J2EE
  - Les streams sont opaques et stockés sur des sources externes
    - BLOB, UDT, url http://, url rtsp:// …

Exemple d'application utilisant des EB et des EMB

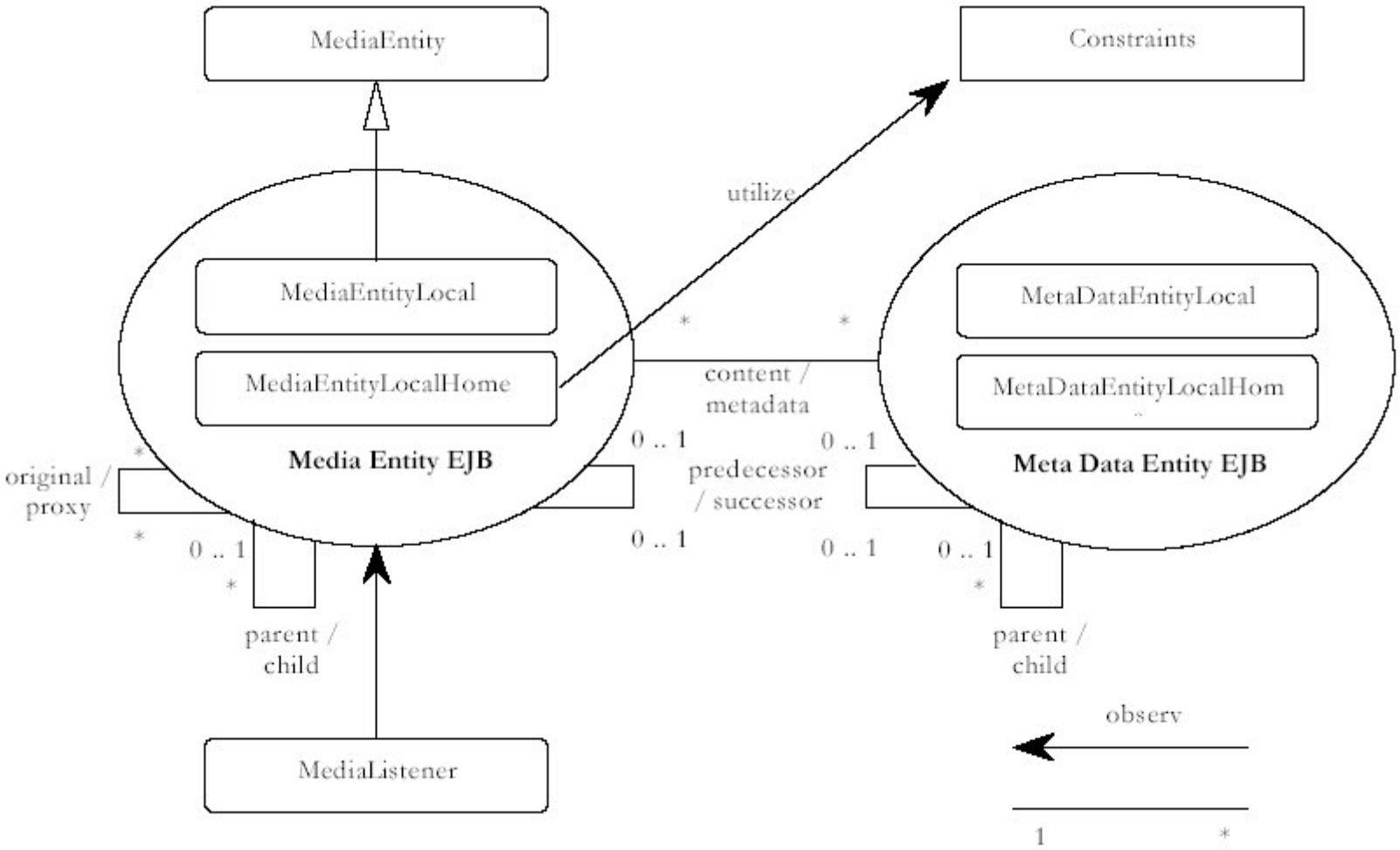
- Emmy Award



EMB - Media Foundation Beans



EMB - Media Entity Beans
Components and interactions



Interoperablity

- Distribution
- Transaction (optional)
  - TX propagation rely on OTS TX propagation
- Naming
  - CoCosNaming
  - Client
- Security
  - CORBA IIOP CSIv2lebvel 1
  - SSL, TSL

Transaction

- Message Driven bean issues
  - not propogation via JMS
  - bean can only NOT_SUPPORTED or REQUIRED
  - onMessage()
- Local Optimisation
  - on commit ?

Connection Factory

Future

- Deferred features
  - Pluggable Persistence Manager
  - Support for method interceptor
  - Support for Component based inheritance
  - Read-Only Beans with CMP (optimisation
  - Aggregate operations for EJB QL finder method
- Related
  - Java Connector Specification

    API
- Question about JOS (

Web Services

- * Support for the use of Message-Driven Beans with JAXM messaging.
  - Enterprise JavaBeans 2.0 introduced the message-driven bean as a new component
  - type in the EJB architecture. Use of message-driven beans in EJB 2.0, however, is
  - limited to the Java Message Service API. With the growing need for Enterprise
  - JavaBeans to support use with web services, it becomes important to extend the use
  - of message-driven beans to support JAXM (the Java APIs for XML Messaging) in
  - addition to JMS. This will support the asynchronous delivery of XML business
  - documents by means of the message-driven bean type.
- * Support for Web Services Usages
  - Enterprise JavaBeans 2.1 will include support for the necessary elements to enable
  - enterprise beans usage with web services. This will include, among other items,
  - expanding the environment support and deployment descriptor infrastructure for
  - enterprise beans that use JAX-RPC and JAXM to access service endpoints or to
  - implement web service endpoints.

Pont avec DCOM et CORBA

- Pont avec DCOM
  - COM Client invoque EJB Server
  - EJB Client invoque COM Server
  - Spécification SUN J2EE CAS

- Pont avec CORBA
  - « **Enterprise JavaBeans™ Components and CORBA Clients** » http://java.sun.com/j2se/1.4.1/docs/guide/rmi-iiop/interop.html

XDocLet

- Motivation
  - Inclure les informations de déploiement dans les commentaires des sources du bean
  - Pour en générer les fichiers de déploiement
- Outillage
  - Basé sur les DocLet (JavaDoc)
  - Tache ANT

```
<taskdef
     name="ejbdoclet"
     classname="xdoclet.modules.ejb.EjbDocletTask"
/>
```

Exemple XDocLet

```
/**
 * @ejb.bean
 *      name="CustomerBean"
 *      type="CMP"
 *      view-type="both"
 *      primkey-field="id"
 * @ejb.home
 *      remote-class="xdoclet.CustomerHomeRemote"
 *      local-class="xdoclet.CustomerHomeLocal"
 * @ejb.interface
 *      remote-class="xdoclet.CustomerRemote"
 *      local-class="xdoclet.CustomerLocal"
 * @ejb:pk
 *      class="java.lang.Integer"
* @ejb:finder
 *    signature= "java.util.Collection findAllCustomers()"
 *    query ="SELECT OBJECT(c) FROM CustomerBean AS c"
* @ejb:transaction
      type="Required"
 */
public abstract class CustomerBean implements EntityBean { …
}
```

Bibliographie

- Homepage EJB et J2EE
  - http://java.sun.com/products/ejb
  - http://java.sun.com/products/j2ee
- Pour pratiquer
  - L'implantation de référence J2EE
    - http://java.sun.com/products/j2ee
  - Le Tutorial de Sun
    - http://java.sun.com/j2ee/tutorial/
  - JOnAS un "J2EE" Open-Source (made in France)
    - http://www.objectweb.org
  - OpenEJB un autre "J2EE" Open-Source (made in France)
    - http://www.openejb.org
  - Tutorial de Gopalan
    - http://www.execpc.com/~gopalan/java/ejb/
  - Le Training de jGuru
    - http://developer.java.sun.com/developer/onlineTraining/EJBIntro/EJBIntro.html

Performances et Benchmarks

- ECPerf Version 1.0 (Draft 02/2001)
  - Pas de charge de travail induite par l'interface utilisateur (génération des pages de présentation HTML)
  - Ni aux performances des serveurs bases de données (Entrée-Sorties disque, Contrôle de concurrence, gestion des buffers, …)
  - S'interesse à la capacité du container d'EJB de gérer l'usage de de la mémoire, le cache , les pools de connexions aux SGBDs, l'activation et la passivation des Beans, …
- Modèle d'applications
  - bases de données et les charges de travail de 4 domaines d'activité d'une entreprise: Fabrication (Manufacturing), Fournisseur (Supplier), Clientèle (Customer), et Société (Corporate).
- Mesures
  - BBops/min (Benchmark Business Operations Per Minute)
    - qui est la somme des transactions de prises de commande du domaine Clientèle et des ordres de fabrication du domaine Fabrication par minute.
  - $/Bbops
    - la mesure économique qui est le ratio performance sur prix du matériel, du logiciel et d'une année de maintenance.
- http://java.sun.com/j2ee/ecperf/, http://ecperf.theserverside.com/,

Bibliographie
Livres

- Ed Roman, Scatt Ambler, Tyler Jewell, Mastering Entreprise Javabeans, End edition, 2002, Ed Wiley, ISBN 0-471-41711-4
  - Très complet
  - Existe en version électronique téléchargeable
- Richard Monson-Haefel, " Enterprise JavaBeans ", 1st Edition June 1999 (est.), ISBN 1-56592-605-6, Ed O'Reilly
  - seconde édition, Mars ??? 2000, ISBN ????
    - Livre consacré aux EJB(spec 1.1). l 'exemple développé dans le livre est de la Réservation de Croisières
- Andrew Patzer , "Programmation Java côté serveur : Servlets, JSP et EJB", Ed Eyrolles-Wrox, 2000, ISBN 1-861002-77-7 (sources des exemples sur www.wroxfrance.com)
  - chapitres 23
- www.theserverside.com
- www.middleware-company.com

Bibliographie
Livres

- Robert Orfali, Dan Harkey, “ Client/Server Programming with Java and Corba ”, 2ème édition, 1998, Ed Wiley, ISBN 0-471-24578-X.
  - Livre plutôt orienté CORBA, le chapitre 34 est consacré aux EJB et l ’exemple développé dans le livre(Réservation de Hôtel/Sejour) est transposé aux EJB
- Andreas Vogel, Madhav Rangarao, “ Programming Enterprise Javabeans, JTS and OTS : Building Distributed Transactions With Java and C++ ”, 1999, Ed John Wiley & Sons; ISBN 0-471-31972-4
  - Livre plutôt orienté OTS/JTS, le dernier chapitre est consacré au EJB et l ’exemple développé dans le livre(Réservation de Billet d ’Avion) pour OTS est transposé aux EJB.

Sources du Cours

- Ce cours a été construit à partir des présentations et des livres de :
  - Michel Riveill
  - Philippe Merle
  - Gérard Vandome
  - Tom Valesky
  - Richard Monson-Haefel
  - Monica Pawlan
  - Gopalan Suresh Raj
  - Marek Prochazka (University of Charles, Pragues)

Limites

- Limited set of Component Mode
  - Entity, session, message-driven
- Change usage of service
  - Eg how to add TX attributes
- Single level component
- EJB app is a set of isolated beans
  - No implicit access restriction
- EjB do not supported shared instances
  - Only a bean's home can be shared
- Proposal EOA
- POA for EJB
  - Intercepts methods invocations