Java Enterprise Edition

Université Pierre & Marie Curie
Licence DANT

Transparents originaux de Lionel Seinturier

Plan

1. Introduction
2. Servlet
3. EJB 3

Introduction

Java EE : Java Enterprise Edition

Ensemble de concepts pour le développement d'applications réparties

•défini par Sun
•Basé sur Java
• un ensemble de spécifications (JSR)
Voir liste :`http://www.oracle.com/technetwork/java/javaee/tech/`

• en évolution "permanente" depuis 1996/97
J2EE 1.0 (servlet + EJB + JDBC), …, 1.5
Java EE 6 depuis 2009

•Domaines applicatifs visés
- E-commerce (B2B & B2C)
- systèmes d'informations
- sites web
- plates-formes de service (Audio-visuel, telco, …)

Introduction

Java EE : Java Enterprise Edition

Plates-formes existantes

-implémentation de référence : Java EE 6 SDK GlassFish
-commerciales
WebSphere (IBM), WebLogic (BEA), NEC, Oracle, SAP, Sun, …
-*open source*
JBoss, JOnAS, Geronimo, OpenEJB, JFox, …
Voir `http://www.oracle.com/technetwork/java/javaee/community/`

Processus de certification mis en place par Sun

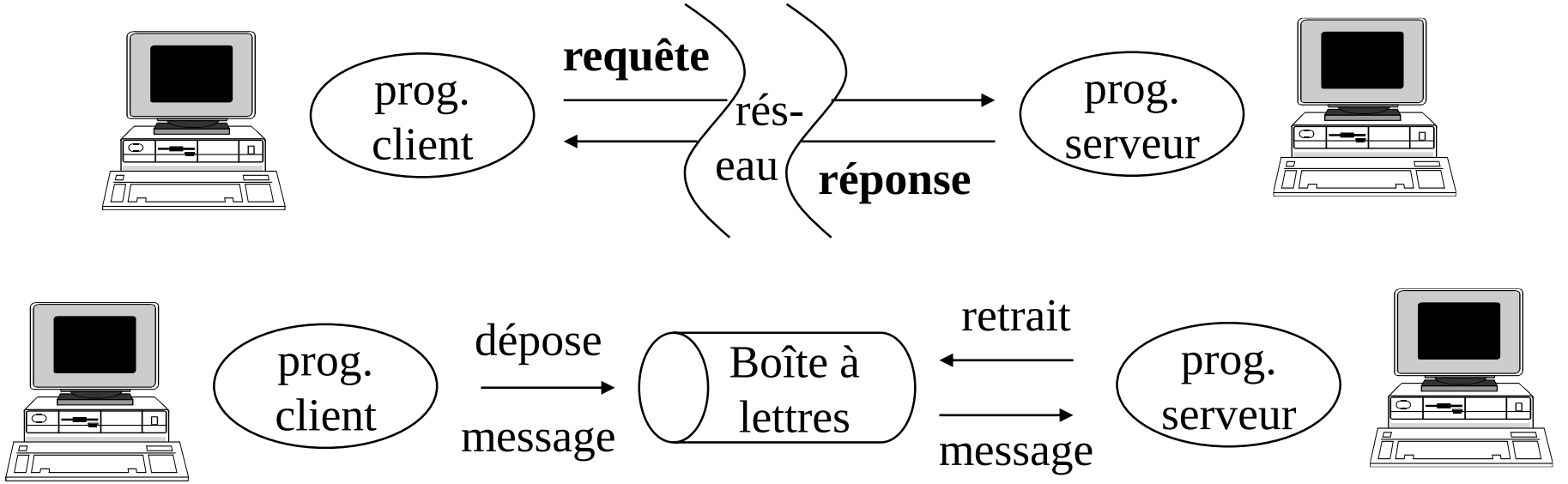
- TCK (Test Compatibility Kit)
- Payant sauf pour plates-formes *open-source*
- Assez lourd (~20 000 tests) à mettre en oeuvre

Introduction

Java EE : Java Enterprise Edition

Un ensemble de technologies *middleware* pour la construction d'applications réparties

- communications distantes
  - RMI-IIOP : requête/réponse (TCP + IIOP + sérialisation Java)
  - JAX-WS : requête/réponse Web Service (HTTP + SOAP + XML)
  - JMS : MOM (*message oriented middleware*) : message + boîte à lettres



Introduction

Java EE : Java Enterprise Edition

Un ensemble de technologies *middleware* pour la construction d'applications réparties

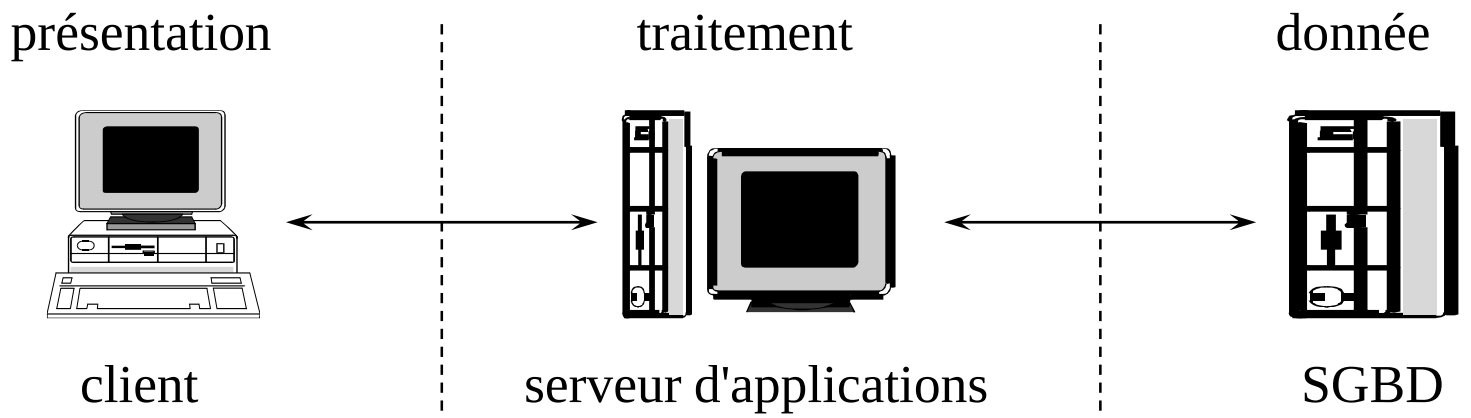
- services systèmes
  - JNDI : annuaire
  - JTA : gestion de transactions
  - JPA : gestion de la persistence des données
  - JASS : sécurité, contrôle d'accès (rôles, droits)
  - JMX : administration de la plate-forme

- un système de connecteur (JCA) pour permettre à la plate-forme d'interagir
  - JDBC : accès client/serveur aux SGBD
  - JavaMail : envoi/réception de mail

Introduction

Java EE : Java Enterprise Edition

Mise en oeuvre du principe des architectures 3 tiers



Introduction

Architecture 3 tiers

Client

- riche : application Java
- léger : navigateur Web

Serveur d'applications

- hébergent des applications à base de
  - composants EJB : classes Java conformes au modèle EJB
  - composants Web : servlet ou JSP

SGBD

- fournit un support de stockage pour les données de l'application
  - 80% : SGBDR (Oracle, SQL Server, PostGreSQL, ...)
  - 20% : autres applications de stockage

Introduction

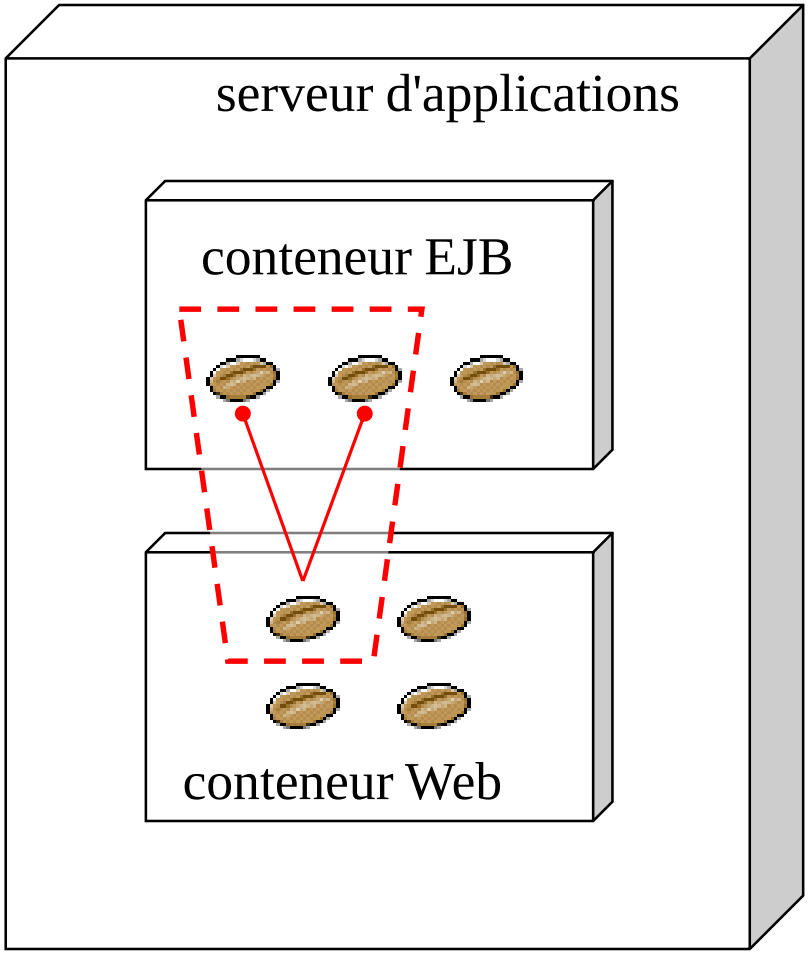
Serveur d'applications

Application JEE =

- 0, 1 ou +sieurs composants EJB
- 0, 1 ou +sieurs composants Web

Plusieurs rôles dans le développement

- développeur de composants Web
- développeur de composants EJB
- assembleur d'applications
- déployeur et gestionnaire d'applications



Introduction

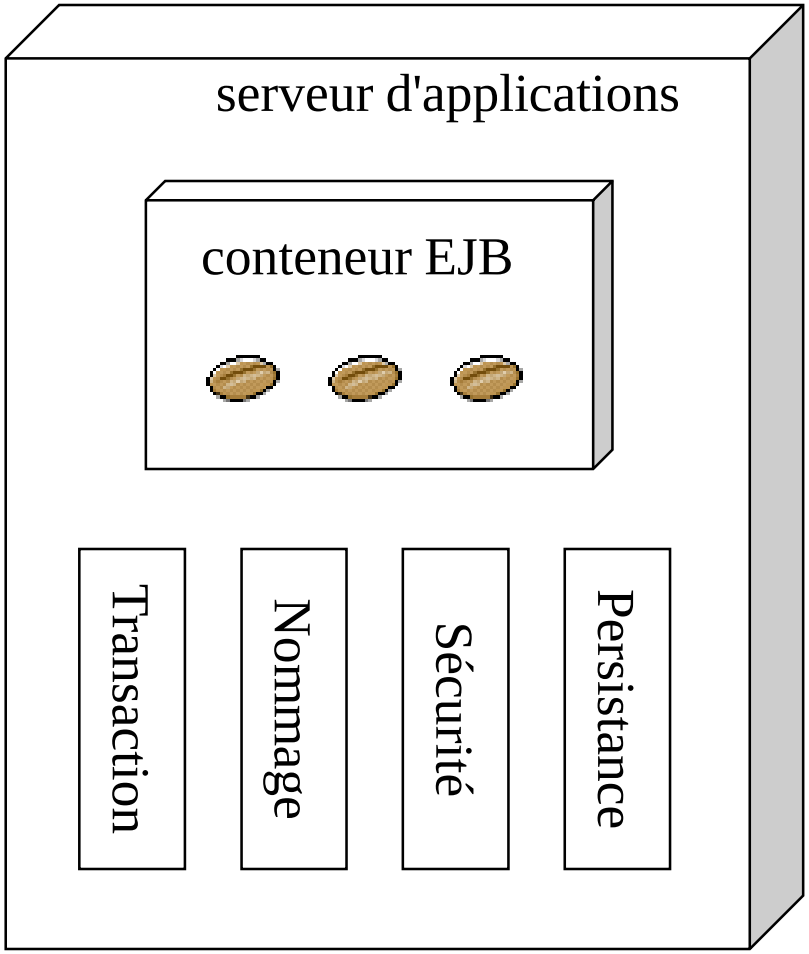
Serveur d'applications

4 services fournis par le serveur
au conteneur EJB

- persistance (JPA)
- transaction (JTA)
- nommage (JNDI)
- sécurité (JASS)

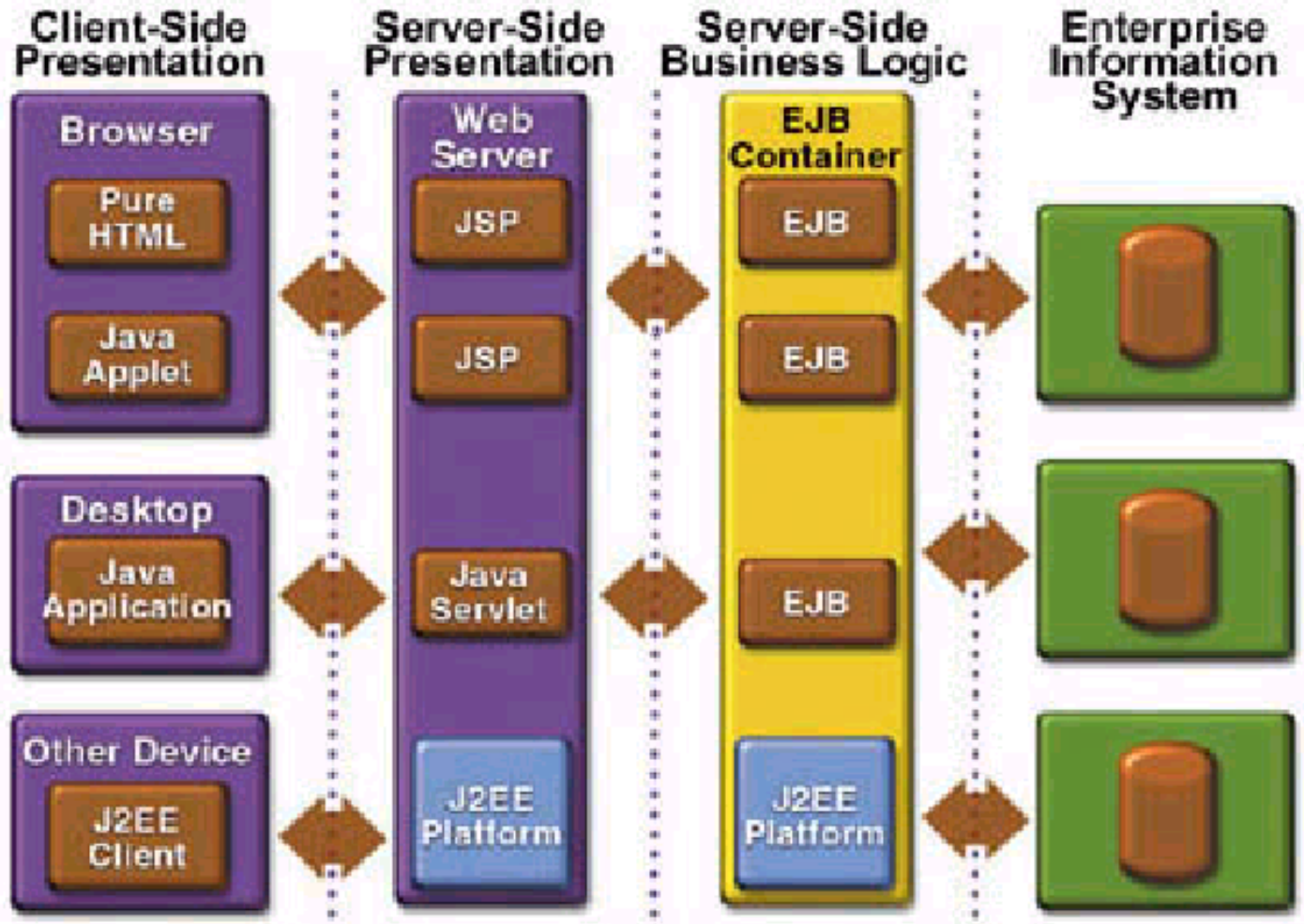
≠ *middleware* style CORBA

ces services sont intégrés dès le départ
à la plate-forme



Introduction

Serveur d'applications



Introduction

Composants

Constat sur les applications réparties orientées objet (CORBA-*like*)

- mélange code fonctionnel – code non fonctionnel
  code métier "noyé" dans un ensemble d'appels à des services techniques
    - sécurité, transaction, persistance données, annuaire, …

➢ difficile à concevoir, comprendre, réutiliser

Besoins par rapport aux intergiciels basés objet ("CORBA-*like*")

- configuration
- déploiement
- empaquetage (*packaging*)
- assemblage
- dynamicité
- gestion des interactions et des dépendances

Introduction

Composants

3 notions importantes des intergiciels basés composant

- **séparation des préoccupations**
  - développer le métier indépendamment des préoccupations non fonctionnelles
  - composants plus facilement réutilisables

- **inversion du contrôle**
  - contenu prend en charge l'exécution du code métier (composant)
  - conteneur assure le lien avec la partie technique
  - ***configurer*** plutôtque programmer
  - approche ***framework*** vs bibliothèque

- **injection de dépendances**
  - vers d'autres composants, vers des services techniques
  - retirer du code métier la gestion des liens vers les autres composants métiers
  - faire gérer l'architecture applicative par le conteneur

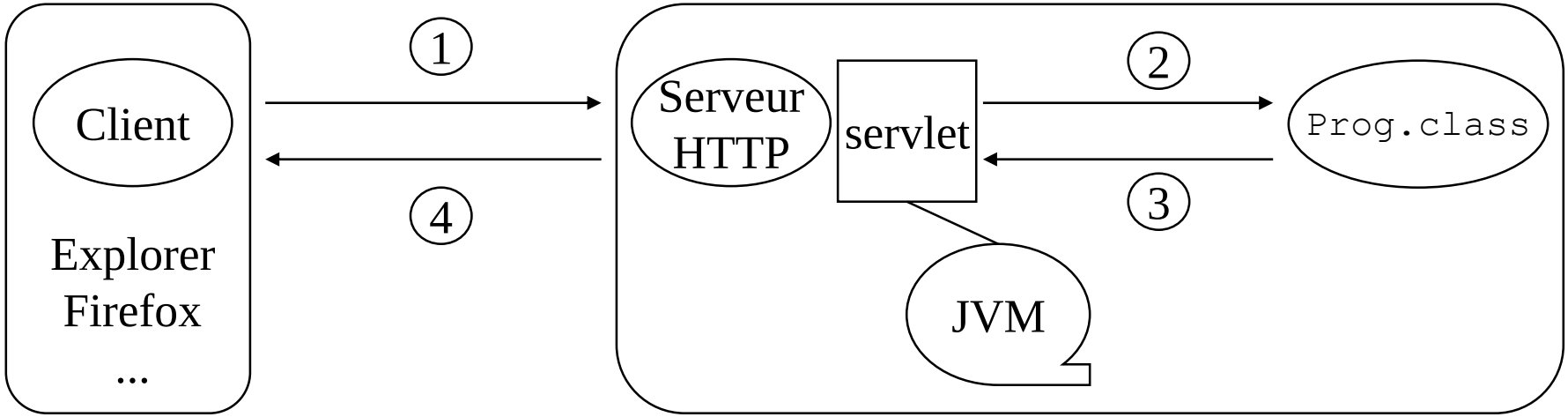
Servlet

Université Pierre & Marie Curie
Licence DANT

Servlet

Principe

- les fichiers de *bytecode*(`.class`) sont stockés sur le serveur
- ils sont désignés par une URL ex. : `http://www.upmc.fr/servlet/Prog`
- le chargement de l'URLprovoque l'exécution du servlet
  - ⇨Servlets étendent le comportement du serveur Web
  - ⇨Sont exécutées par un "moteur" (ex. Tomcat)



Servlet

Exemple de servlet

```
package myPackage;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloServlet extends HttpServlet {

 public void service( HttpServletRequest request,
                      HttpServletResponse response )
    throws ServletException, IOException {

    response.setContentType( "text/html" );
    PrintWriter out = response.getWriter();

    out.println( "<html><body>" );
    out.println( "<h1>Hello depuis une servlet</h1>" );
    out.println( "</body></html>" );
} }
```

Servlet

Fonctionnalités

- possibilité de traitements différenciés selon les requêtes HTTP (`doGet`, `doPost`, …)
- contenu retourné non nécessairement HTML (binaire, .gif, .pdf, …)

- instance de servlet persistante

- notion de session utilisateur (`request.getSession()`)
- données attachables à une session (`session.setAttribute()`)
- manipulation de *cookies* (`request.getCookies()`, `response.setCookies()`)
- espace global de données partagé entre servlets

- format d'archivage : fichier `.war` (`.jar` + descripteur `web.xml`)

- principaux moteurs (conteneurs) de servlet : Tomcat, Jetty, Resin

Enterprise Java Beans 3

Université Pierre & Marie Curie
Licence DANT

Plan

1. Composant EJB
   - 1.1 Session Bean
   - 1.2 Entity Bean
   - 1.3 Message Driven Bean
   - 1.4 Fonctionnalitésavancées

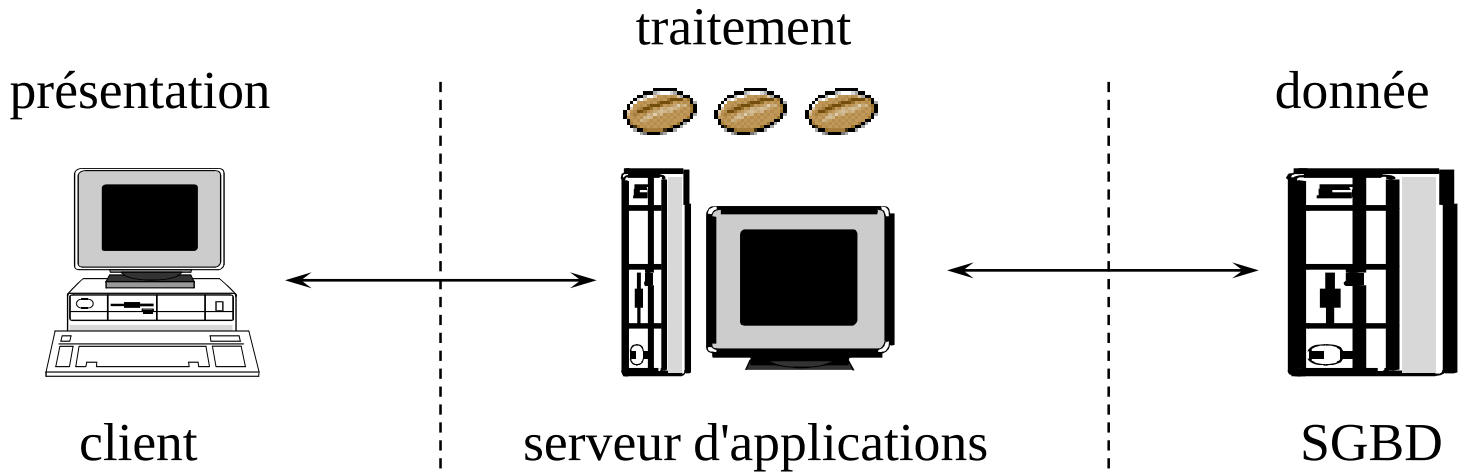
2. Services
   - 2.1 Annuaire
   - 2.2 Transaction
   - 2.3 Sécurité

3. Conclusion

1. Composant EJB

Enterprise Java Bean (EJB)

Composants applicatifs pour le développement d'applications réparties



1. Composant EJB

Enterprise Java Bean (EJB)

A server-side component that encapsulates the business logic of an application

- on se focalise sur la logique applicative
- les services systèmes sont fournis par le conteneur
- la logique de présentation est du ressort du client

Vocabulaire dans ce cours : *bean* = EJB = composant

Types d'EJB

- Session : *performs a task for a client*
- Entity (deprecated) : *represents a business entity object that exists in persistent storage*
- Message-Driven : *listener processing messages asynchronously*

Plusieurs versions : actuellement EJB 3.2 (depuis juin 2013)

1. Composant EJB

EJB 3

- succès Java EE en général
- mais
  - trop compliqué, lourd, contraignant
  - concepts objets (héritage, typage, polymorphisme, …) difficilement exploitables
  - mapping objet/relationel limité
  - trop de fichiers XML fastidieux à écrire, maintenir, comprendre
- passage EJB 2 vers EJB 3
  - utilisation des annotations et de la généricité Java 5
  - pour simplifier l'écriture des *beans*
  - éviter autant que faire se peut l'écriture de fichiers XML
- versions récentes 3.1 et 3.2
  - améliorations des possibilités de spécification des annotations

1. Composant EJB

EJB 3 `http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/`

Table 1: Summary of Findings

Application Name	Item Measured	J2EE 1.4 Platform	Java EE 5 Platform	Improvement
AdventureBuilder	Number of classes	67	43	36% fewer classes
	Lines of code	3,284	2,777	15% fewer lines of code
RosterApp	Number of classes	17	7	59% fewer classes
	Lines of code	987	716	27% fewer lines of code
	Number of XML files	9	2	78% fewer XML files
	Lines of XML code	792	26	97% fewer lines of XML code

1. Composant EJB

Annotations Java 5

- mécanisme standard dans le langage Java depuis version 5 (1.5)
- idée similaire aux commentaires Javadoc
  - informations attachées à des éléments de programme (classe, méthode, attributs, …)
  - pour ajouter de l'information sur cet élément

- `@`*`Identificateur`*
- éventuellement des paramètres : `@`*`Identificateur`*`(`*`name=value`*`,...)`
  - types autorisés
    - primitifs, String, Class, annotation
    - tableaux de primitifs, String, Class, annotation
- éventuellement plusieurs annotations par éléments

Exemple

```
@Resource(name="myDB", type=javax.sql.DataSource.class)
@Stateful
public class ShoppingCartBean implements ShoppingCart { ... }
```

Annotations Java 5

- chaque annotation est un type (au même titre qu'une classe ou qu'une interface)
- défini dans un package (ex. : `javax.ejb.Stateless`)

- de nouvelles annotations peuvent être définies par le programmeur
  - mot clé Java 5 : `@interface`
  - 1 méthode par paramètre avec éventuellement une valeur par défaut

```
public @interface MyAnnot {
  int value() default 12;
}
```

```
@MyAnnot(value=15)
public class MyClass { ... }
```

Annotations Java 5

- annotation = type
  - ➢les déclarations d'annotations peuvent être annotées

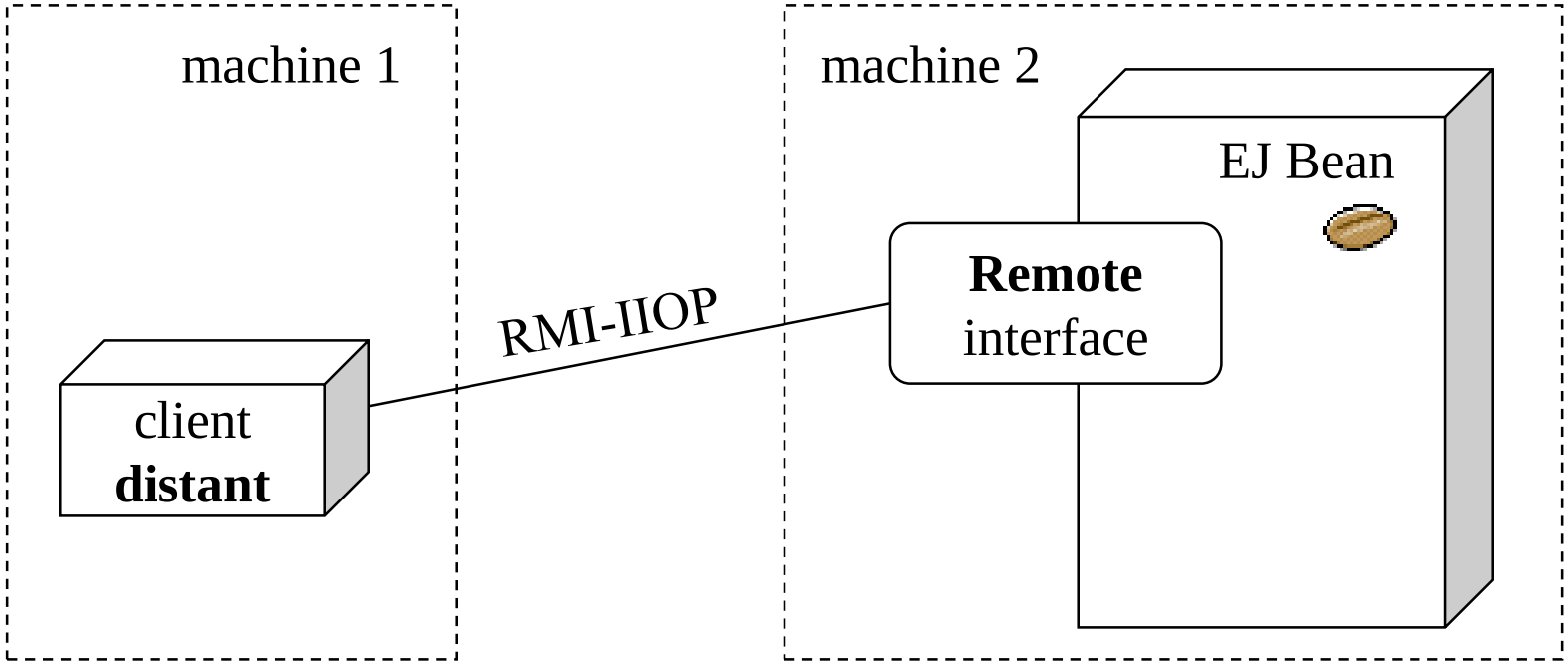
```
@Retention(value=RUNTIME)                    // annotations présentes au runtime
@Target(value=CLASS)                         // annotation de classe
public @interface MyAnnot {
int value() default 12;
}
```

- source, class, runtime
- annotation, constructeur, attribut, variable locale, méthode, package, paramètre, type

Pour + d'informations, voir :
`http://download.oracle.com/javase/1.5.0/docs/guide/language/annotations.html`

1. Composant EJB

Enterprise Java Bean

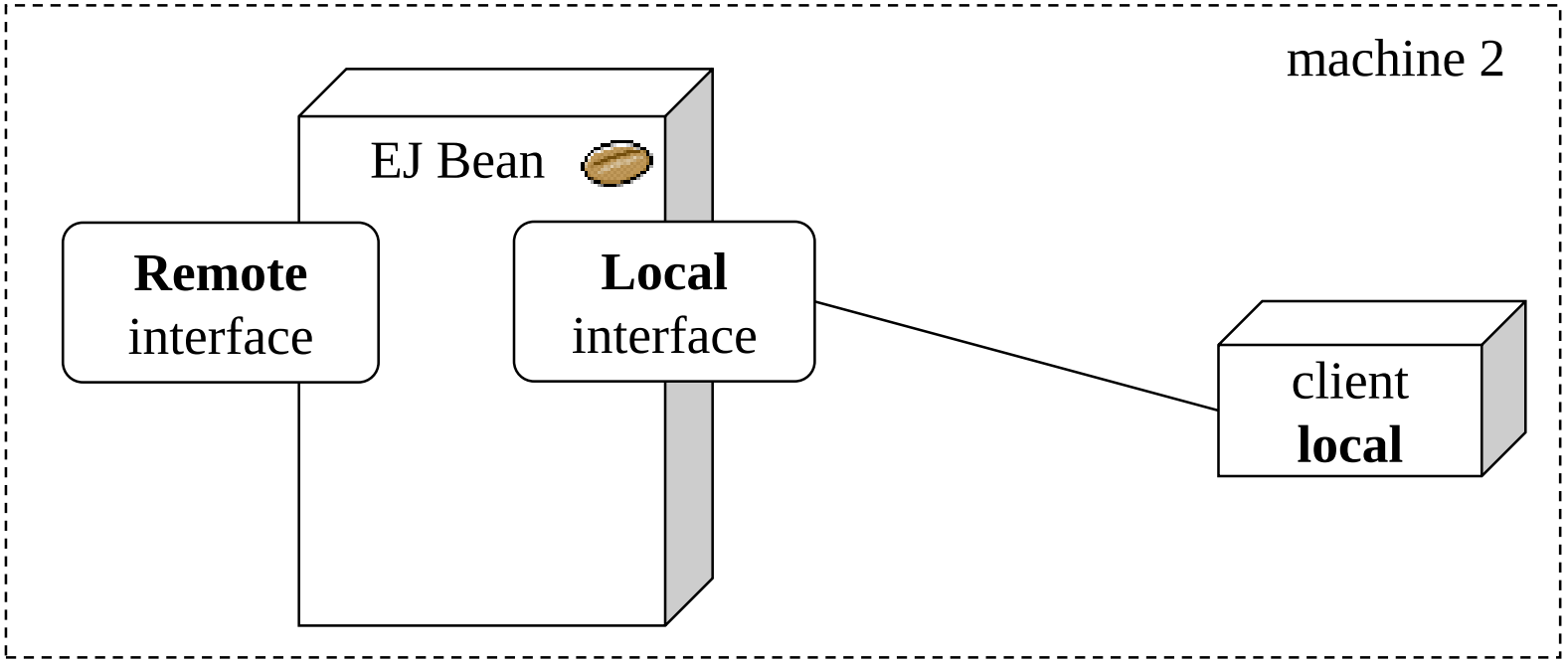


Chaque EJ Bean fournit 1 interface d'accès **distant**

• les services (méthodes) offerts par le bean à ces client

1. Composant EJB

Enterprise Java Bean



+ éventuellement 1 interface d'accès **local** (àpartir EJB 2.0)

• les services offerts par le bean à ses clients locaux
• les mêmes (ou d'autres) que ceux offerts à distance
⇒optimisation

1.1 Session Bean

1. Définition
2. Développement
3. Client local
4. Client distant
5. Stateful session bean

1.1 Session Bean

Définition

Session Bean : représente un traitement (services fournis à un client)

1. ***Stateless*** session bean

   - sans état
   - ne conserve pas d'information entre 2 appels successifs
   - 2 instances qcqs d'un tel *bean* sont équivalentes
   - 1 instance par invocation

2. ***Stateful*** session bean

   - avec état(en mémoire)
   - même instance pendant toute la durée d'une session avec un client
   - 1 instance par client

3. ***Singleton*** session bean : idem stateful, mais ***une instance pour tous les clients***

1.1 Session Bean

Développement

1 interface (éventuellement 2 : Local + Remote) + 1 classe

Interface

- annotations `@javax.ejb.Local` ou `@javax.ejb.Remote`

```
import javax.ejb.Remote;

@Remote
public interface CalculatriceItf {
  public double add(double v1,double v2);
  public double sub(double v1,double v2);
  public double mul(double v1,double v2);
  public double div(double v1,double v2);
}
```

1.1 Session Bean

Développement

Classe

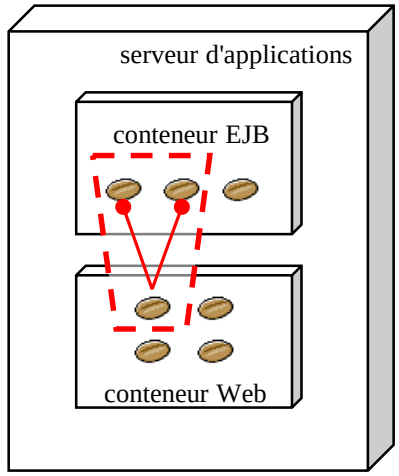
- annotation `@javax.ejb.Stateless` ou `@javax.ejb.Stateful` ou `@javax.ejb.Singleton`

```
import javax.ejb.Stateless;

@Stateless
public class CalculatriceBean implements CalculatriceItf {
  public double add(double v1,double v2) {return v1+v2;}
  public double sub(double v1,double v2) {return v1-v2;}
  public double mul(double v1,double v2) {return v1*v2;}
  public double div(double v1,double v2) {return v1/v2;}
}
```

- possibilité de nommer les *beans* : `@Stateless(name="foobar")`
- par défaut, le nom de la classe

1.1 Session Bean

Client local



- typiquement une servlet ou une JSP
  colocalisée sur le même serveur que le *bean*

- mécanisme dit "injection de dépendance"
  - attribut du type de l'interface
  - annoté `@EJB` éventuellement `@EJB(name="foobar")`

```
public class ClientServlet extends HttpServlet {

  @EJB(name="foobar")
  private CalculatriceItf myBean;

  public void service( HttpServletRequest req, HttpServletResponse resp ) {
     resp.setContentType("text/html");
     PrintWriter out = resp.getWriter();
     double result = myBean.add(12,4.75);
     out.println("<html><body>"+result+"</body></html>");
} }
```

1.1 Session Bean

Client distant

1. Récupération de la référence vers l'annuaire JNDI
2. Recherche du *bean* dans l'annuaire
3. Appel des méthodes du bean

```
public class Client {
  public static void main(String args[]) throws Exception {
        javax.naming.Context ic = new javax.naming.InitialContext();
        CalculatriceItf bean = (CalculatriceItf) ic.lookup("foobar");
        double res = bean.add(3,6);
  }
}
```

1.1 Session Bean

Stateful Session Bean

- instance du *bean* reste en mémoire tant que le client est présent
- expiration au bout d'un délai d'inactivité
- utilisation type
  - gestion d'un panier électronique sur un site de commerce en ligne
  - rapport sur l'activité d'un client

2 annotations principales

- `@Stateful` : déclare un *bean* avec état
- `@Remove`
  - définit la méthode de fin de session
  - la session expire à l'issuede l'exécution de cette méthode

Stateful Session Bean

```
@Stateful
public class CartBean implements CartItf {

  private List items = new ArrayList();
  private List quantities = new ArrayList();

  public void addItem( int ref, int qte ) { ... }
  public void removeItem( int ref ) { ... }

  @Remove
  public void confirmOrder() { ... }
}
```

1.2 Entity Bean

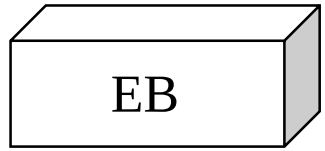
1. Définition
2. Développement
3. Gestionnaire d'entité
4. Relation
5. Autres annotations

1.2 Entity Bean

Définition

Représentation d'une donnée manipulée par l'application

- donnée typiquement stockée dans un SGBD (ou tout autre support accessible en JDBC)



Nom	Solde
John	100.00
Anne	156.00
Marcel	55.25

- correspondance objet – tuple relationnel (*mapping O/R*)
- possibilité de définir des clés, des relations, des recherches
- avantage : manipulation d'objets Java pluôt que de requêtes SQL

- mis en oeuvre à l'aide
  - d'annotations Java 5
  - de la généricité Java 5
  - de l'API JPA (Java Persistence API)

1.2 Entity Bean

Développement

- annotation `@Entity` : déclare une classe correspondant à un *entity bean* (EB)

- Chaque classe d'EB est mise en correspondance avec une table
  - par défaut table avec même nom que la classe
  - sauf si annotation `@Table(name="...")`

- 2 modes (exclusifs) de définition des colonnes des tables
  - *property-based access* : on annote les méthodes *getter*
  - *field-based access* : on annote les attributs

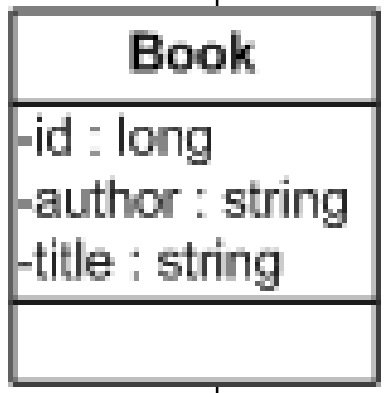
  - par défaut colonne avec même nom que *field/property*
  - Sauf si annotation `@Column(name="...")`

  - annotation `@Id` : définit une clé primaire
  - types supportés
    - primitifs (et leurs types Class correspondants), String, Date

1.2 Entity Bean

Développement

```
@Entity
public class Book {

    private long id;
    private String author;
    private String title;

    public Book() {}
    public Book(String author, String title) {
        this.author = author;
        this.title = title; }

    @Id
    public long getId() { return id; }
    public void setId(long id) { this.id = id; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; } }
```



Développement

Possibilité de définir des champs auto-incrémentés

- annotation `@GeneratedValue`

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
public long getId() { return id; }
```

- `GenerationType.AUTO` numéros de séquence choisis automatiquement
- `GenerationType.SEQUENCE` générateur de numéros de séquence est à fournir

1.2 Entity Bean

Gestionnaire d'entités

Entity Manager

- assure la correspondance entre les objets Java et les tables relationnelles
- point d'entrée principal dans le service de persistance

- permet d'ajouter des enregistrements
- permet d'exécuter des requêtes

- accessible via une injection de dépendance
  - attribut de type `javax.persistence.EntityManager`
  - annoté par `@PersistenceContext`

1.2 Entity Bean

Gestionnaire d'entités

Exemple

- création de trois enregistrements dans la table des livres

```
@Stateless
public class MyBean implements MyBeanItf {

    @PersistenceContext
    private EntityManager em;

    public void init() {
        Book b1 = new Book("Honore de Balzac","Le Pere Goriot");
        Book b2 = new Book("Honore de Balzac","Les Chouans");
        Book b3 = new Book("Victor Hugo","Les Miserables");

        em.persist(b1);
        em.persist(b2);
        em.persist(b3);
}   }
```

- de façon similaire `em.remove(b2)` retire l'enregistrement de la table

Gestionnaire d'entités

Recherche par clé primaire

- méthode find du gestionnaire d'entités

```
Book myBook = em.find(Book.class,12);
```

- retourne `null` si la clé n'existe pas dans la table
- `IllegalArgumentException`
  - si 1er paramètre n'est pas une classe d'EB
  - si 2ème paramètre ne correspond pas au type de la clé primaire

1.2 Entity Bean

Gestionnaire d'entités

Recherche par requête

- requêtes SELECT dans une syntaxe dite EJB-QL étendue
- mot clé OBJECT pour désigner un résultat à retourner sous la forme d'un objet
- paramètres nommés (préfixés par `:`) pour configurer la requête

```
Query q =
em.createQuery("select OBJECT(b) from Book b where b.author = :au");

String nom = "Honore de Balzac";
q.setParameter("au",nom);
List<Book> list = (List<Book>) q.getResultList();
```

- méthode `getSingleResult()` pour récupérer un résultat unique
  - `NonUniqueResultException` en cas de non unicité

1.2 Entity Bean

Gestionnaire d'entités

Recherche par requête pré-compilée

- création d'une requête nommée attachée à l'EB

```
@Entity
@NamedQuery(name="allbooks",query="select OBJECT(b) from Book b")
public class Book { ... }

Query q = em.createNamedQuery("allbooks");
List<Book> list = (List<Book>) q.getResultList();
```

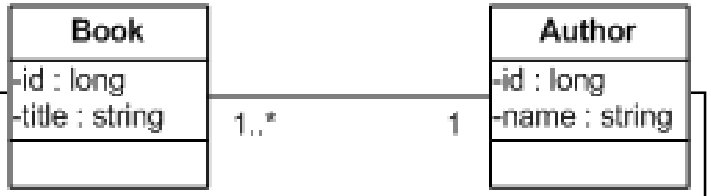
- paramètres peuvent être spécifiés (voir transparent précédent)
- plusieurs requêtes nommées peuvent être définies

```
@Entity
@NamedQueries(
  value={ @NamedQuery("q1","..."), @NamedQuery("q2","...") }
public class Book { ... }
```

1.2 Entity Bean

Relation

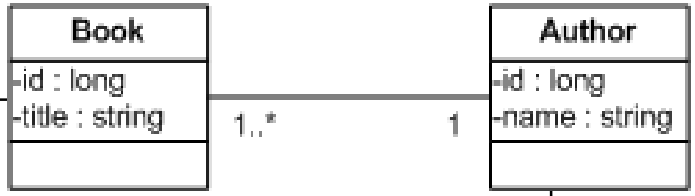
2 catégories principales : 1-n et n-n




```
@Entity
public class Author {

    private long id;
    private String name;
    private Collection<Book> books;

    public Author() { books = new ArrayList<Book>(); }
    public Author(String name) { this.name = name; }

    @OneToMany
    public Collection<Book> getBooks() { return books; }

    public void setBooks( Collection<Book> books ) { this.books=books; }
    ...
}
```

1.2 Entity Bean

Relation 1-n



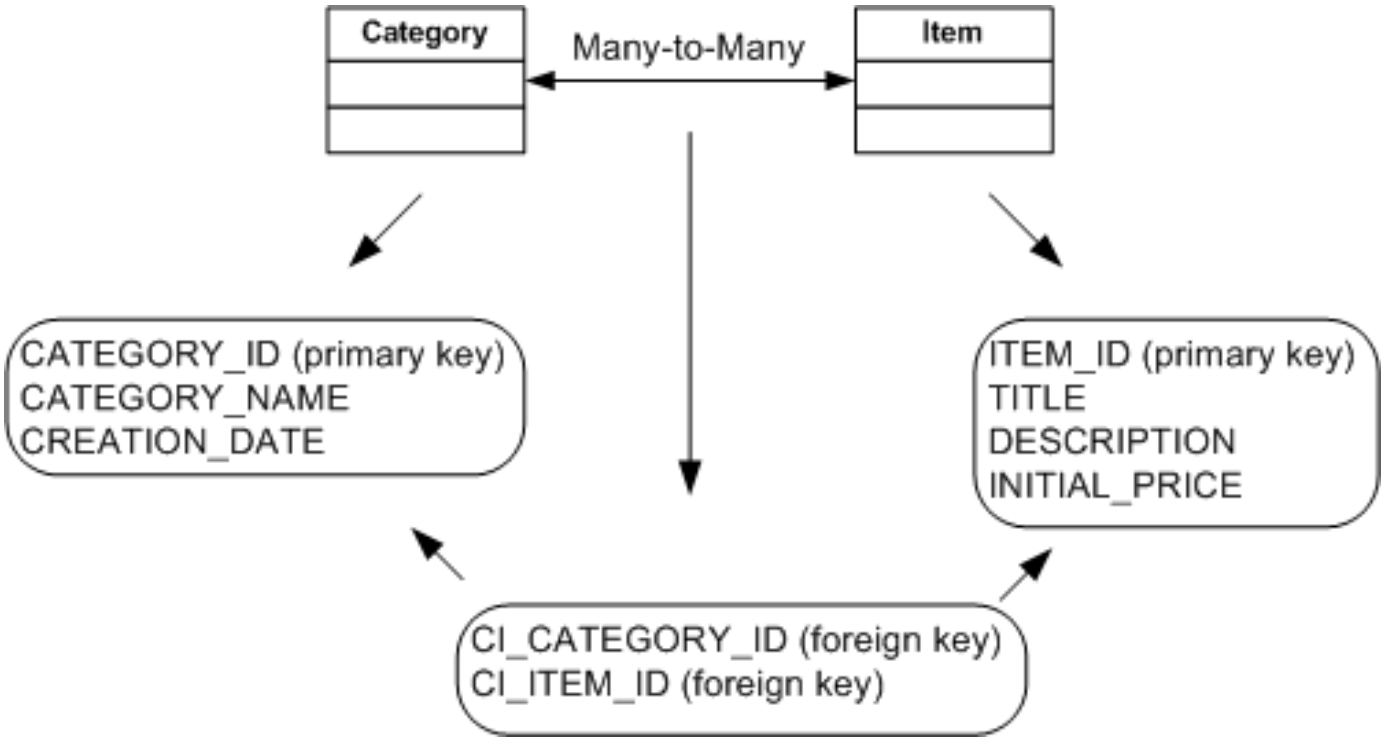

```
@Entity
public class Book {

    private long id;
    private Author author;
    private String title;

    public Book() {}
    public Book(Author author, String title) {
        this.author = author;
        this.title = title; }

    @ManyToOne
    @JoinColumn(name="Author_id")
    public Author getAuthor() { return author; }
    public void setAuthor(Authro author) { this.author = author; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    ...
}
```

Nom de la colonne de jointure

1.2 Entity Bean

Relation n-n

- notion de table de jointure



1.2 Entity Bean

Relation n-n

```
@Entity
public class Category {
 @Id
 @Column(name="CATEGORY_ID")
 protected long categoryId;

 @ManyToMany
 @JoinTable(name="CATEGORIES_ITEMS",
  joinColumns=
   @JoinColumn(
     name="CI_CATEGORY_ID",
     referencedColumnName="CATEGORY_ID"),
  inverseJoinColumns=
   @JoinColumn(
     name="CI_ITEM_ID",
     referencedColumnName="ITEM_ID"))
 protected Set<Item> items;
```

```
@Entity
public class Item {
 @Id
 @Column(name="ITEM_ID")
 protected long itemId;

 @ManyToMany(mappedBy="items")
 protected Set<Category> categories;
```

1.2 Entity Bean

Autres annotations

@Enumerated : définit une colonne avec des valeurs énumérées
EnumType : ORDINAL (valeur stockée sous forme int), STRING

```
public enum UserType {STUDENT, TEACHER, SYSADMIN};

@Enumerated(value=EnumType.ORDINAL)
protected UserType userType;
```

@Lob : données binaires

```
@Lob
protected byte[] picture;
```

@Temporal : dates
TemporalType : DATE (java.sql.Date), TIME (java.sql.Time)
TIMESTAMP (java.sql.Timestamp)

```
@Temporal(TemporalType.DATE)
protected java.util.Date creationDate;
```

1.2 Entity Bean

Autres annotations

@SecondaryTable : mapping d'un EB sur plusieurs tables

```
  @Entity
  @Table(name="USERS")
@SecondaryTable(name="USER_PICTURES"
        pkJoinColumns=@PrimaryKeyJoinColumn(name="USER_ID"))
  public class User { ... }
```

@Embeddable et @Embedded : embarque les données d'une classe dans une table

```
  @Embeddable
  public class Address implements Serializable {
private String rue; private int codePostal; }
```

```
  @Entity
  public class User {
    private String nom;

    @Embedded
    private Address adresse;
  }
```

1.2 Entity Bean

Autres annotations

@IdClass : clé composée

```
@Entity
@IdClass(PersonnePK.class)
public class Personne {
    @Id public String getName() { return name; }
    @Id public String getFirstname() { return firstname; }
}

public class PersonnePK implements Serializable {
  private String name;
  private String firstname;
  public PersonnePK( String n, String f ) { ... }
  public boolean equals(Object other) { ... }
  public int hash() { ... }
}
```

1.2 Entity Bean

Informations complémentaires :

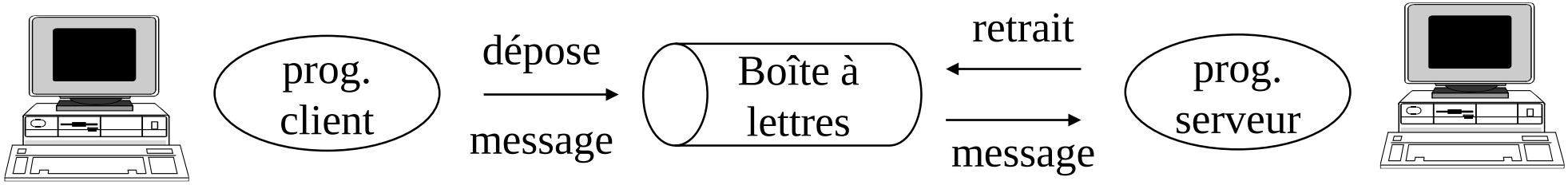
- Les Entity Beans sont maintenant “deprecated” lorsque crées en tant que Beans
- Ils sont désormais des POJOs (plain old java objects) important la JPA.
- Cela permet de concevoir un modèle plus puissant qu’auparavant (pas de nécessité de dépendre d’un framework quelconque)

Plan

1.3 Message-driven Bean

Message-driven bean (MDB)

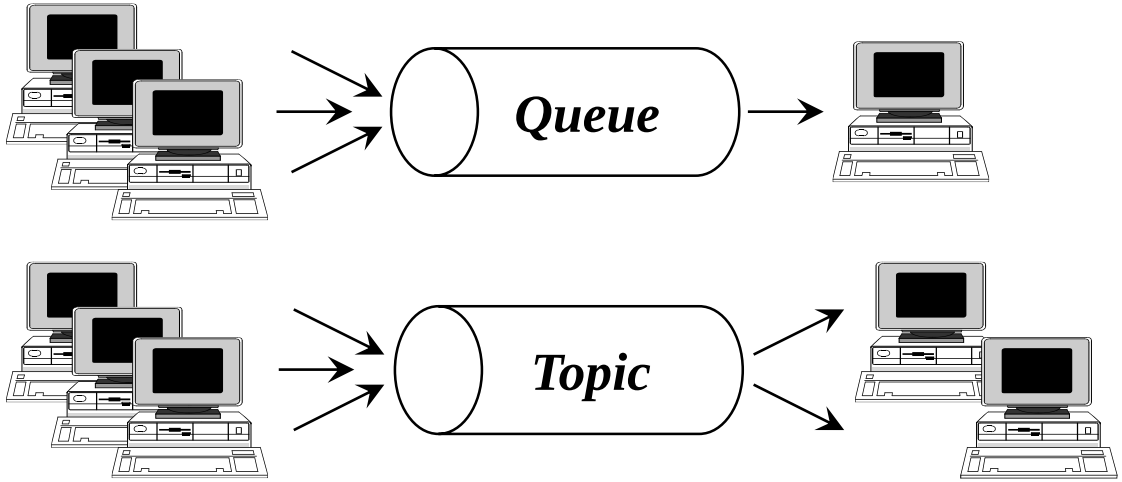
Interaction **par envoi message asynchrone** (MOM : *Message-Oriented Middleware*)



≈ CORBA COSEvent

2 modes

-n vers 1 (*queue*)
-n vers m (*topic*)



1.3 Message-driven Bean

Caractéristiques

- consomme des messages asynchrones
- pas d'état (≡ *stateless session bean*)
- toutes les instances d'une même classe de MDB sont équivalentes
- peut traiter les messages de clients différents

Quand utiliser un MDB

- éviter appels bloquants
- découpler clients et serveurs
- besoin de fiabilité : protection *crash* serveurs

Vocabulaire : producteur/consommateur

1.3 Message-driven Bean

Concepts

MDB basé sur Java Messaging Service (JMS) `java.sun.com/jms`

Connection Factory : fabrique pour créer des connexions vers une *queue/topic*
Connection : une connexion vers une *queue/topic*
Session : période de temps pour l'envoi de messages dans 1 *queue/topic*
Peut être rendue transactionnelle
similitude avec les notions de sessions JDBC, Hibernate, …

Processus

1. Création d'une connexion
2. *Création d'une session* *(éventuellement plusieurs sessions par connexion)*
3. Création d'un message
4. Envoi du message
5. Fermeture session
6. Fermeture connexion

1.3 Message-driven Bean

Producteur

```
public class MyProducerBean {

 @Resource(name="jms/QueueConnectionFactory")  // l'id de la factory
 private ConnectionFactory connectionFactory;

 @Resource(name="jms/ShippingRequestQueue")  // l'id de la queue
 private Destination destination;

 public void produce() {

  Connection connection = connectionFactory.createConnection();
  Session session = connection.createSession(true,Session.AUTO_ACKNOWLEDGE)
  MessageProducer producer = session.createProducer(destination);

  TextMessage message = session.createTextMessage();
  message.setText("Hello World!");
  producer.send(message);

  session.close();
  connection.close();
} }
```

1.3 Message-driven Bean

Consommateur

MDB = classe
- annotée `@MessageDriven`
- implantant interface MessageListener
  - méthode `void onMessage(Message)`

```
@MessageDriven(name="jms/ShippingRequestProcessor")
public class MyConsumerBean implements MessageListener{

 public void onMessage( Message m ) {

   TextMessage message = (TextMessage) m;
   ...
} }
```

Plan

1.4 Fonctionnalités avancées

Timer beans

Déclenchement d'actions périodiques

- automatiquement
- programmatiquement

Définition automatique

- annotation `@Schedule` pour définir la périodicité

```
@Stateless
public class NewsLetterGeneratorBean implements NewsLetterGenerator {
    @Schedule(second="0", minute="0", hour="0",
                    dayOfMonth="1", month="*", year="*")
    public void generateMonthlyNewsLetter() {
        // ...
} }
```

➢ Déclenchement tous les 1ers du mois à 00h00

1.4 Fonctionnalités avancées

Timer beans

Définition programmatique des timer beans

- `@Timeout` : méthode exécutée à échéance du *timer*

profil de méthode : `void <methodname>( javax.ejb.Timer timer )`

- `@Resource` : attribut de type `javax.ejb.TimerService`
- utilisation des méthodes de TimerService pour créer des *timers*
  - `createTimer( long initialDuration, long period, Serializable info )`

```
public class EnchereBean {

  @Resource TimerServicets;

  public void ajouterEnchere( EnchereInfoe ) {
    ts.createTimer(1000,25000,e); }

  @Timeout
  public void monitorerEnchere( Timer timer ) {
EnchereInfoe = (EnchereInfo) timer.getInfo(); ... } }
```

1.4 Fonctionnalités avancées

Intercepteurs

Permettent d'implanter des traitements avant/après les méthodes d'un *bean*
- influence de l'AOP (voir AspectJ, JBoss AOP, …)

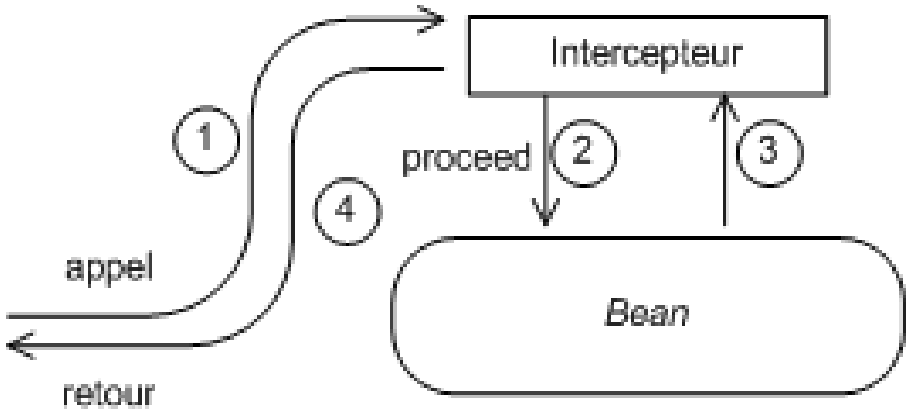
- `@Interceptors` : les méthodes devant être interceptées
- `@AroundInvoke` : les méthodes d'interception

  profil de méthode

```
Object <methodname>( InvocationContext ctx ) throws Exception
```

`javax.interceptor.InvocationContext`

- permet d'obtenir des informations (introspecter) sur les méthodes interceptées
- fournit une méthode `proceed()` pour exécuter la méthode interceptée



1.4 Fonctionnalités avancées

Intercepteurs

Plusieurs méthodes dans des classes ≠ peuvent être associées à `MyInterceptor`

```
public class EnchereBean {

  @Interceptors(MyInterceptor.class)
  public void ajouterEnchere( Bid bid ) { ... } }

public class MyInterceptor {

  @AroundInvoke
  public Object trace( InvocationContext ic ) throws Exception {

    // ... code avant ...

    java.lang.reflect.Method m = ic.getMethod();
    Object bean = ic.getTarget();
    Object[] params = ic.getParameters();
    // éventuellement modification paramètres avec ic.setParameters(...)

    Object ret = ic.proceed();        // Appel du bean (facultatif)

    // ... code après ...

    return ret; } }
```

1.4 Fonctionnalités avancées

Méthode asynchrone

- invocation asynchrone des méthodes d'un *bean*

```
@Stateless
public class OrderBillingServiceBean implements OrderBillingService {
    @Asynchronous
    public void billOrder(Order order) { ... }
}
```

- ne remplace pas *message-driven bean*
-Simplifie le développement pour beaucoup de cas asynchrones simples
-définition d'un objet dit futur en cas de résultat

```
@Asynchronous
public Future<OrderStatus>billOrder(Order order) { ... }
```

- interface `java.util.concurrent.Future` (depuis JDK 6)
- méthodes `isDone()`, `get()`

1.4 Fonctionnalités avancées

Conteneur EJB embarquable

-par ex. directement dans un JDK
-éviter d'avoir à utiliser une plate-forme Java EE complète
-faciliter, promouvoir l' utilisation des EJB
-lien avec la notion de profil

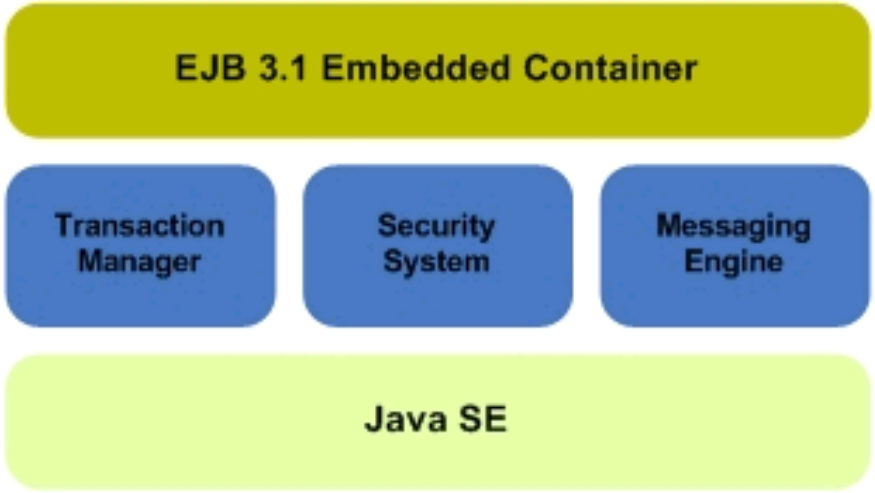


Figure 1: EJB 3.1 Embeddable Containers Concept

```
static void main(String[] args) {
  EJBContainer container =
    EJBContainerFactory.
createEJBContainer();
  Context context =
container.getContext();
  PlaceBid placeBid =
    (PlaceBid) context.lookup
    ("java:global/action-bazaar/
      PlaceBid");
  placeBid.addBid(new Bid("rrahman",
    10059, 200.50));
  container.close(); }
```

1.4 Fonctionnalités avancées

Format standardisé des noms JNDI pour les beans

JBoss global JNDI name	action-bazaar/PlaceBidBean/remote
GlassFish global JNDI name	PlaceBid
WebSphere Community Edition global JNDI name	action-bazaar-ejb/PlaceBidBean/PlaceBid
Oracle Application Server (OC4J) global JNDI name	PlaceBidBean

Table 1: Vendor-specific global JNDI names

```
java:global[/<application-name>]/<module-name>/<bean-name>#<interface-name>
```

- améliore la portabilité des applications

Plan

1. Composant EJB
   - 1.1 Session Bean
   - 1.2 Entity Bean
   - 1.3 Message Driven Bean
   - 1.4 Fonctionnalités avancées

2. Services
   - 2.1 Annuaire
   - 2.2 Transaction
   - 2.3 Sécurité

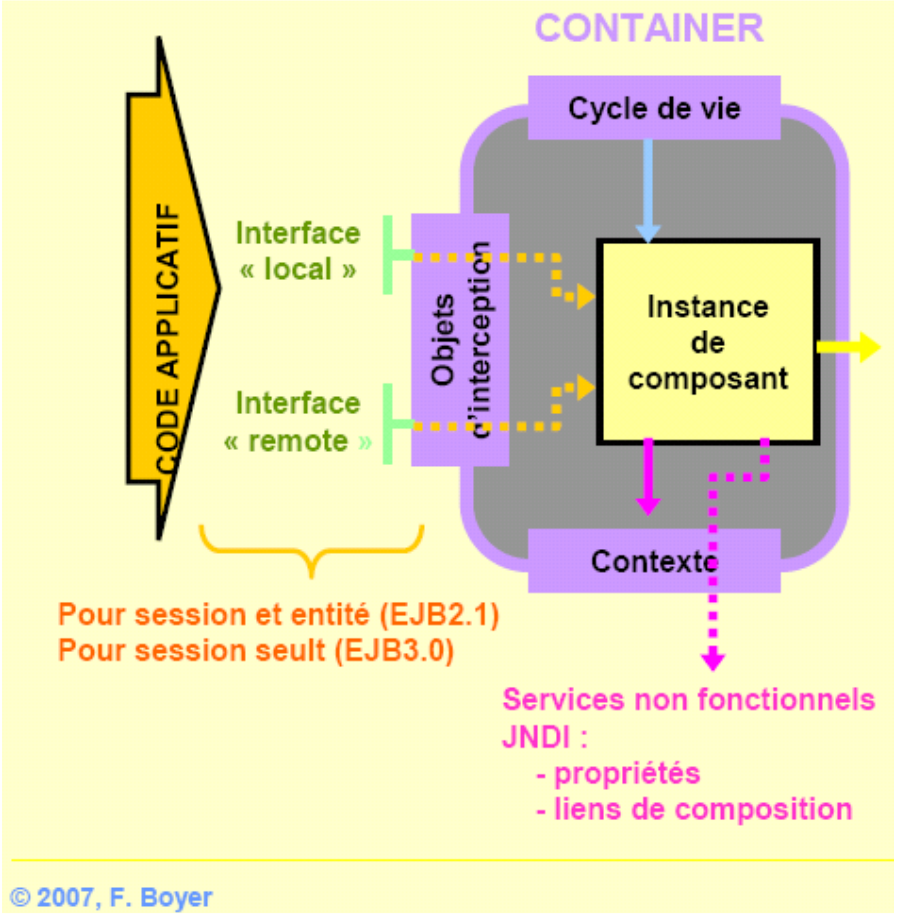
3. Conclusion

2. Services

Conteneur

- environnement de création/gestion des instances de composants
  - notion de cycle de vie

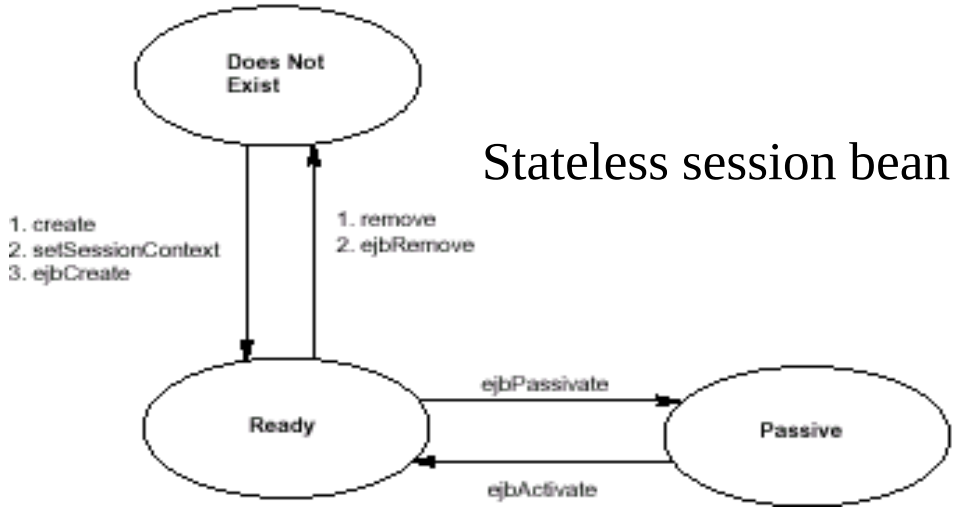
- fournit un contexte qui permet d'accéder
  - aux propriétés de configuration
  - aux références vers les autres comp.
  - aux références vers les services techniques



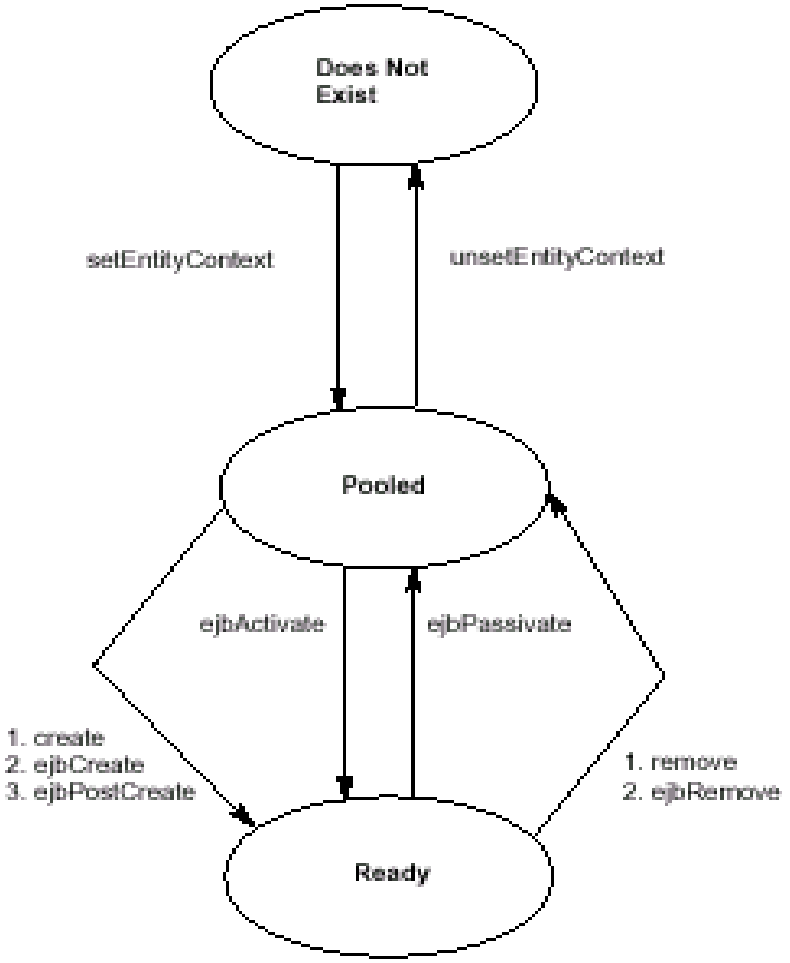
2. Services

Cycle de vie

Stateless session bean



Stateful session bean



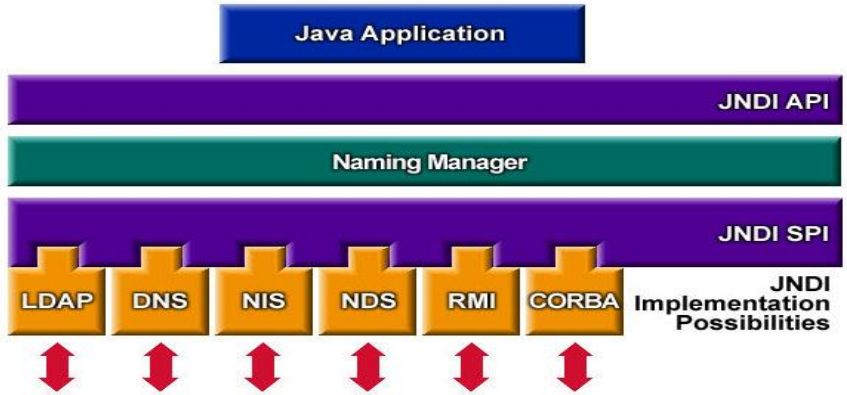
Notification sur changement état

`@PostConstruct` : après la création
`@PreDestroy` : avant la destruction
`@PrePassivate` : avant la passivation
`@PostActivate` : après l'activation

2.1 Annuaire

JNDI

Un service d'annuaire



- références vers composants
- références vers services techniques

- accès distant
- accès local `javax.naming.Context ic = new InitialContext();`
- recherche `Object o = ic.lookup("url");`

- URL JNDI type : chemin/nom
  - RMI-IIOP `iiop://myhost.com/myBean`
  - LDAP `ldap://localhost:389`

- Contexte local Java EE
  - `java:comp/env/`
  - ex. `java:comp/env/myOtherBean`, `java:comp/env/javax.user.Transaction`

2.2 Transactions

Service de transactions

Assure des propriétés **ACID** pour des transactions plates

Exemple classique : un transfert bancaire (débit, crédit)

- atomicité soit les 2 opérations s'effectuent complètement, soit aucune
- cohérence le solde d'un compte ne doit jamais être négatif
- isolation des transferts // doivent fournir le même résultat qu'en séq.
- durabilité les soldes doivent être sauvegardés sur support stable

Support complètement intégré au serveur EJB
Véritable + / aux *middlewares* style CORBA

Granularité des transactions

Comment démarquer (délimiter) les transactions ?

Attribut transactionnel avec 6 valeurs

- REQUIRED
- REQUIRES_NEW
- SUPPORTS
- NOT_SUPPORTED
- MANDATORY
- NEVER

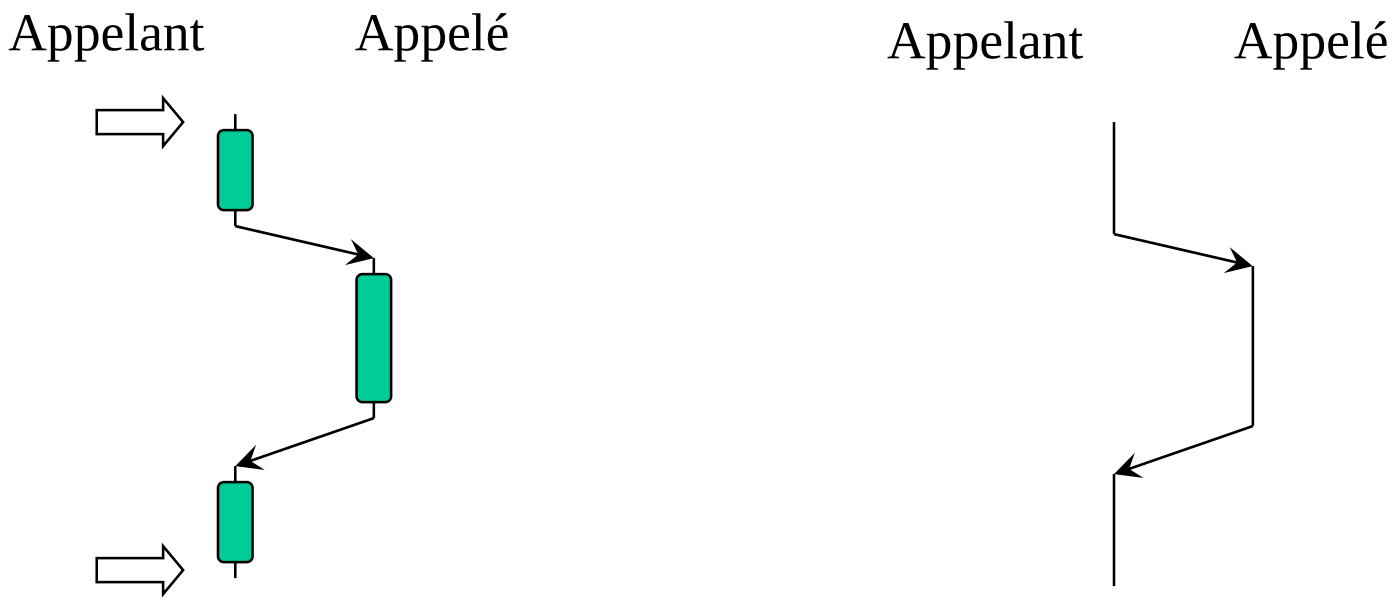
2 cas pour le *bean* appelant

• soit il s'exécute dans une transaction
• soit il s'exécute en dehors de tout contexte transactionnel

Granularité des transactions

SUPPORTS

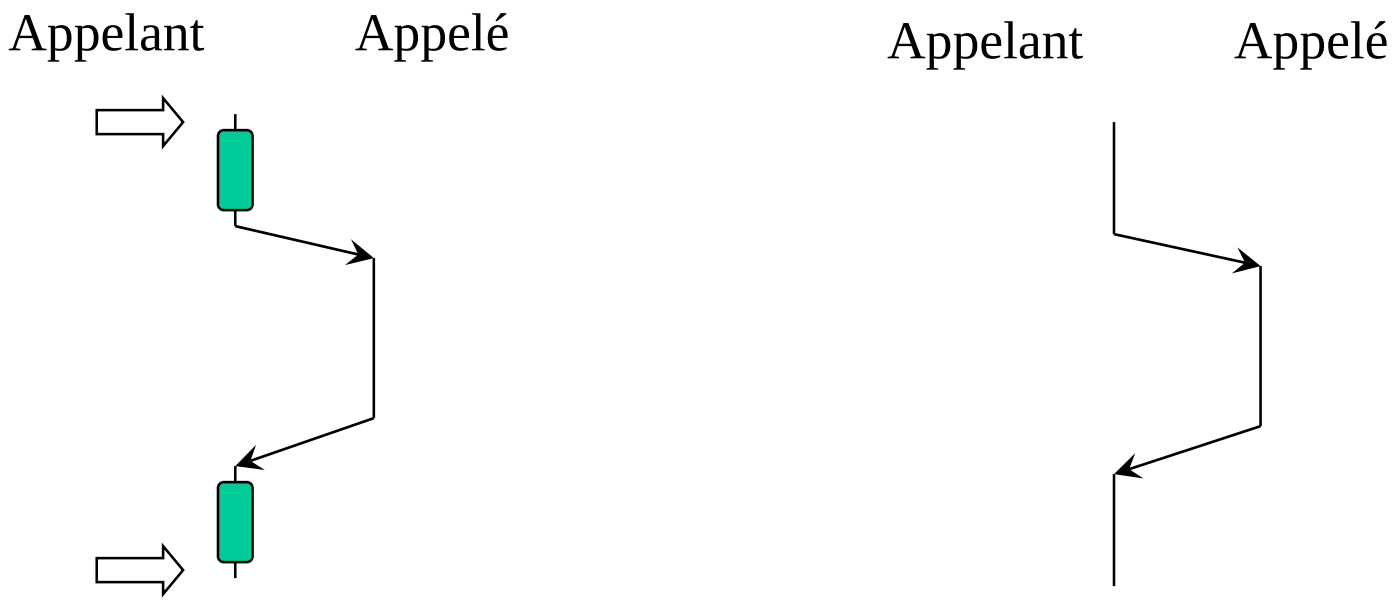
Si l'appelant s'exécute dans une transaction, l'appelé s'y insère
Sinon, l'appelé s'exécute en dehors de toute transaction



Granularité des transactions

NOT_SUPPORTED

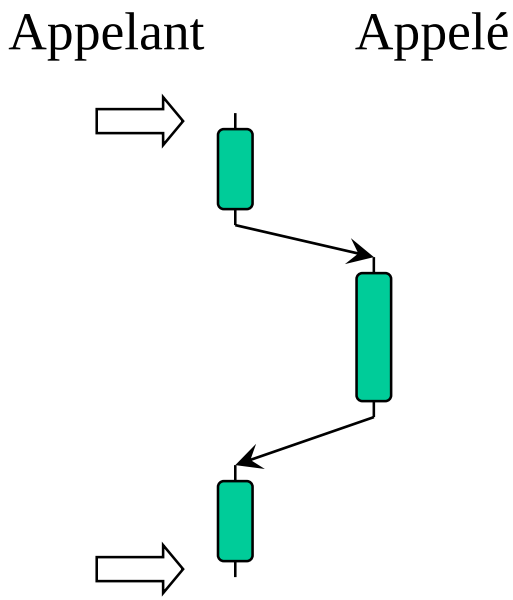
L'appelé s'exécute toujours en dehors d'une transaction

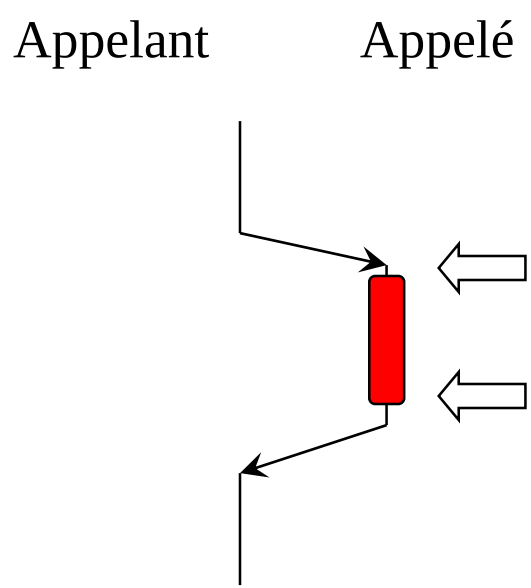


Granularité des transactions

REQUIRED

Si l'appelant s'exécute dans une transaction, l'appelé s'y insère
Sinon, l'appelé débute une nouvelle transaction



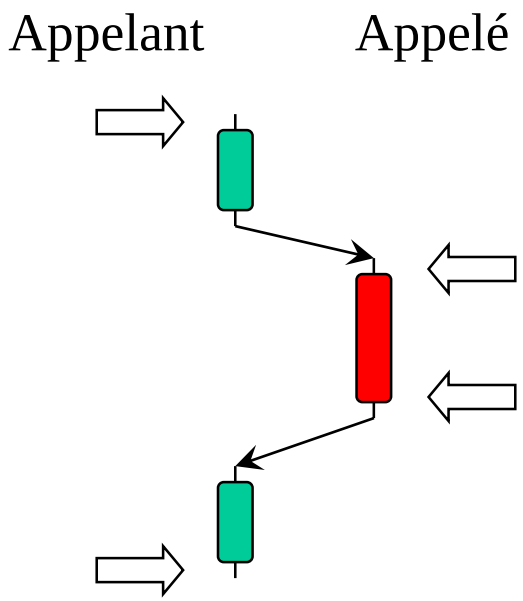


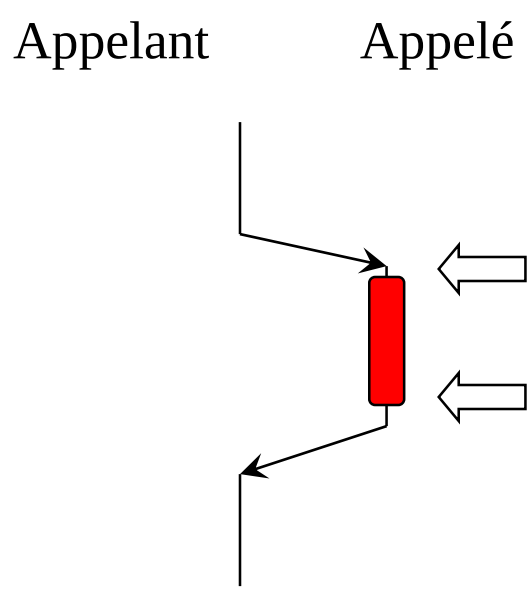
Granularité des transactions

REQUIRES_NEW

Une nouvelle transaction est systématiquement créée

Si l'appelant s'exécute dans une transaction, l'appelé y imbrique la sienne



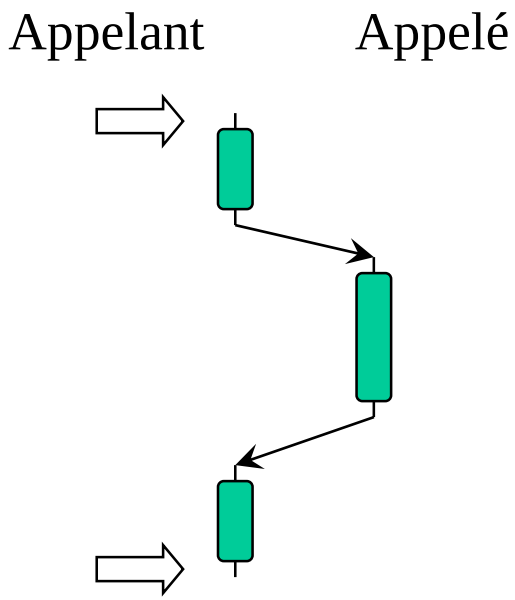


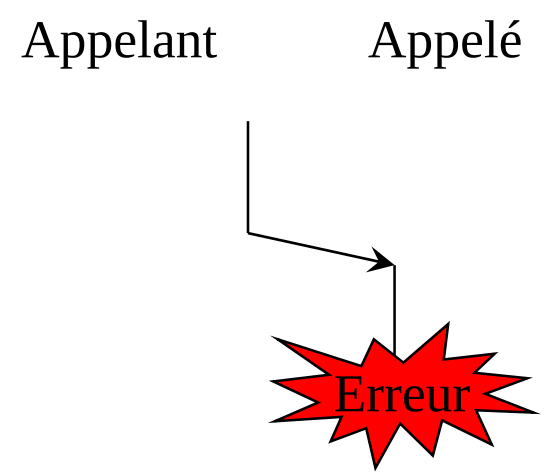
2.2 Transactions

Granularité des transactions

MANDATORY

Si l'appelant s'exécute dans une transaction, l'appelé l'utilise
Sinon, l'appelé lève une exception



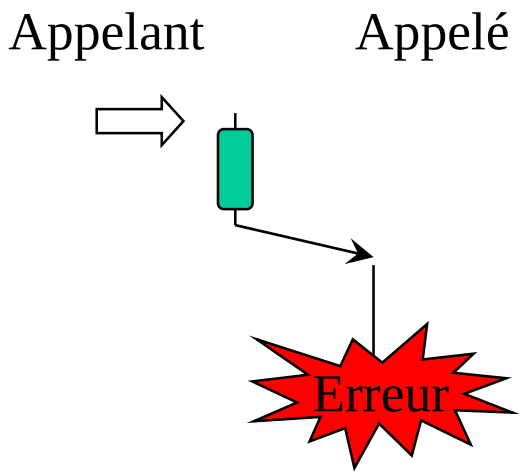


throw
new TransactionRequiredLocalException()
or new TransactionRequiredException()

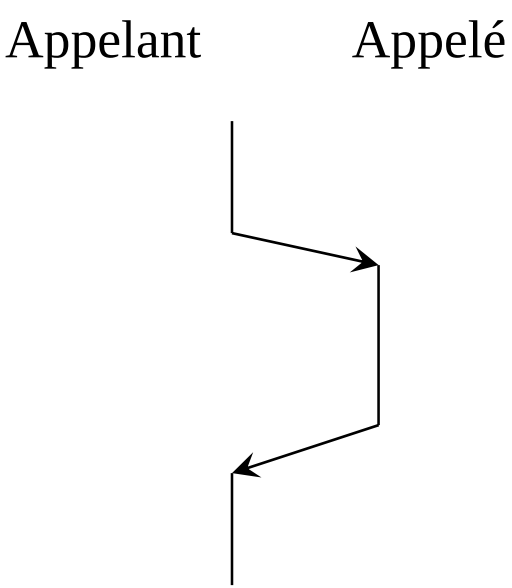
Granularité des transactions

NEVER

L'appelant ne doit pas s'exécuter dans une transaction



throw
new EJBException()
or new RemoteException()



2.2 Transactions

Définition des transactions

2 modes

- CMT (*Container Managed Transaction*) : annotations
- BMT (*Bean Managed Transaction*) : API JTA

CMT

- `@TransactionManagement` sur la classe
- `@TransactionAttribute(TransactionAttributeType.XXX)`
  sur les méthodes transactionnelles
  où `XXX` est 1 des 6 attributs précédents
- toute la méthode est considérée comme un bloc transactionnel
- *commit* par défaut en fin de méthode
- appel `setRollbackOnly()` pour annuler

2.2 Transactions

Définitions des transactions – CMT

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class MyBean implements MyBeanItf {

  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  public void transfert() {
    try {
      Account a1 = em.find(Account.class,"Bob");
      Account a2 = em.find(Account.class,"Anne");
      a1.credit(10.5);
      a2.widthdraw(10.5);
    }
    catch( Exception e ) {
      sc.setRollbackOnly();
  } }

  @PersistenceContext
  private EntityManager em;

  @Resource
  private SessionContext sc; }
```

2.2 Transactions

Définition des transactions

BMT

- démarcation explicite avec begin/commit/rollback
- avantage : possibilité granularité plus fine qu'en CMT

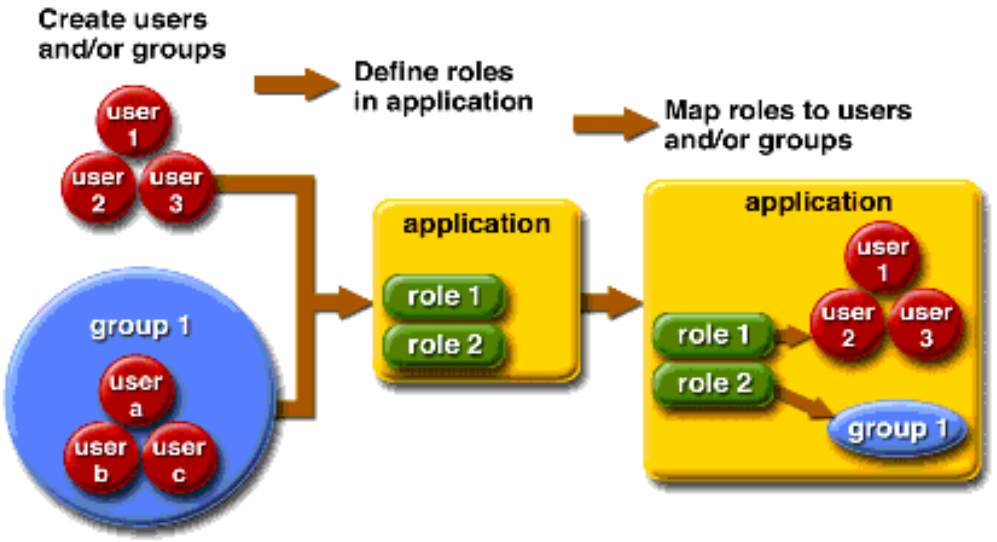
```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class MyBean implements MyBeanItf {

 public void transfert() {
  try {
   ut.begin();
   Account a1 = em.find(Account.class,"Bob");
   Account a2 = em.find(Account.class,"Anne");
   a1.credit(10.5);
   a2.widthdraw(10.5);
   ut.commit();
  }
  catch( Exception e ) { ut.rollback(); } }
```

```
@PersistenceContext
private EntityManager em;

@Resource
private UserTransaction ut; }
```

Service de sécurité

- contrôler l'accès aux méthodes d'un bean
- authentifier l'utilisateur

À base de rôles



5 annotations

```
javax.annotation.security.PermitAll
javax.annotation.security.DenyAll
javax.annotation.security.RolesAllowed
javax.annotation.security.DeclareRoles
javax.annotation.security.RunAs
```

2.3 Sécurité

Annotations	Target		EJB or its super classes	Servlet or web libraries	Descriptions
	TYPE	METHOD			
@PermitAll	X	X	X		Indicates that the given method or all business methods of the given EJB are accessible by everyone.
@DenyAll		X	X		Indicates that the given method in the EJB cannot be accessed by anyone.
@RolesAllowed	X	X	X		Indicates that the given method or all business methods in the EJB can be accessed by users associated with the list of roles.
@DeclareRoles	X		X	X	Defines roles for security checking. To be used by EJBContext.isCallerInRole, HttpServletRequest.isUserInRole, and WebServiceContext.isUserInRole.
@RunAs	X		X (not for non-EJB super classes)	X (for servlet only)	Specifies the run-as role for the given components.

Service de sécurité

Exemple

```
@Stateless
@RolesAllowed("javaee")
public class HelloEJB implements Hello {

  @PermitAll
  public String hello(String msg) { return "Hello, " + msg; }

  public String bye(String msg) { return "Bye, " + msg; }
}
```

- `hello` : accessible par tout le monde
- `bye` : accessible seulement pour le rôle javaee

Service de sécurité

Pour les cas plus complexes : `@DeclaresRoles`

Exemple 2

```
@Stateless
@DeclaresRoles({"A", "B"})
public class HelloEJB implements Hello {

  @Resource private SessionContext sc;

  public String hello(String msg) {
    if (sc.isCallerInRole("A") && !sc.isCallerInRole("B")) {
      ...
    } else {
      ...
    } } }
```

- `hello` : accessible par ceux qui sont dans le rôle A mais pas dans le rôle B

Plan

1. Composant EJB
   - 1.1 Session Bean
   - 1.2 Entity Bean
   - 1.3 Message Driven Bean
   - 1.4 Fonctionnalités avancées

2. Services
   - 2.1 Annuaire
   - 2.2 Transaction
   - 2.3 Sécurité

3. Conclusion

4. Conclusion

Java EE

Apports

- prise en charge des services techniques par la plate-forme
- le composant se focalise sur le métier
- *packaging*& déploiement

Mais

- uniquement Java
- nombreux domaines applicatifs concernés par la répartition non couverts par Java EE
  - ➢ temps-réel, embarqué, mobilité, CAO, infographie, …
- déploiement sur un seul serveur à la fois

4. Conclusion

EJB Lite : Uneversion allégée des EJB

Feature	EJB Lite	EJB
Stateless beans	✔	✔
Stateful beans	✔	✔
Singleton beans	✔	✔
Message driven beans		✔
No interfaces	✔	✔
Local interfaces	✔	✔
Remote interfaces		✔
Web service interfaces		✔
Asynchronous invocation		✔
Interceptors	✔	✔
Declarative security	✔	✔
Declarative transactions	✔	✔
Programmatic transactions	✔	✔
Timer service		✔
EJB 2.x support		✔
CORBA interoperability		✔

Table 1: EJB and EJB Lite Feature Comparison

4. Conclusion

Commerciaux

•WebSphere	IBM	www-01.ibm.com/software/webservers/appserv/wasproductline/
•WebLogic	BEA	www.weblogic.com
• App Server	Borland	www.borland.com/appserver
•iPortal	IONA	www.iona.com/products

• Oracle, Sybase, HP, Fujitsu, ATG, Hitachi, Macromedia, SilverStream, ...

Voir http://www.oracle.com/technetwork/java/javaee/community/

Open source

• Java EE 6 (Sun)	http://download.oracle.com/javaee/6/
•JOnAS (ObjectWeb)	jonas.ow2.org
•JBoss (Red Hat)	www.jboss.org
• Geronimo (Apache)	geronimo.apache.org
•Jfox	www.huihoo.org/jfox

4. Conclusion

Ressources

API Java EE 6 : `http://download.oracle.com/javaee/6/api/`

Tutorial Java EE 6 : `http://download.oracle.com/javaee/6/tutorial/doc/`

Livre Mastering Enterprise Java Beans, 3rd edition

`http://www.theserverside.com/tt/books/wiley/masteringEJB/index.tss`