

L'environnement Java EE

Noel de palma

Remerciements

Fabienne Boyer (UJF), Pascal Déchamboux (FT R&D), Lionel Seinturier (LIFL)

Sommaire

- Introduction
- Modèles de programmation Java EE
 - ◆ Clients
 - ◆ Composants session
 - ◆ Composants entité
 - ◆ Composants mdb
 - ◆ Gestion des transactions
 - ◆ Packaging
- Conclusion

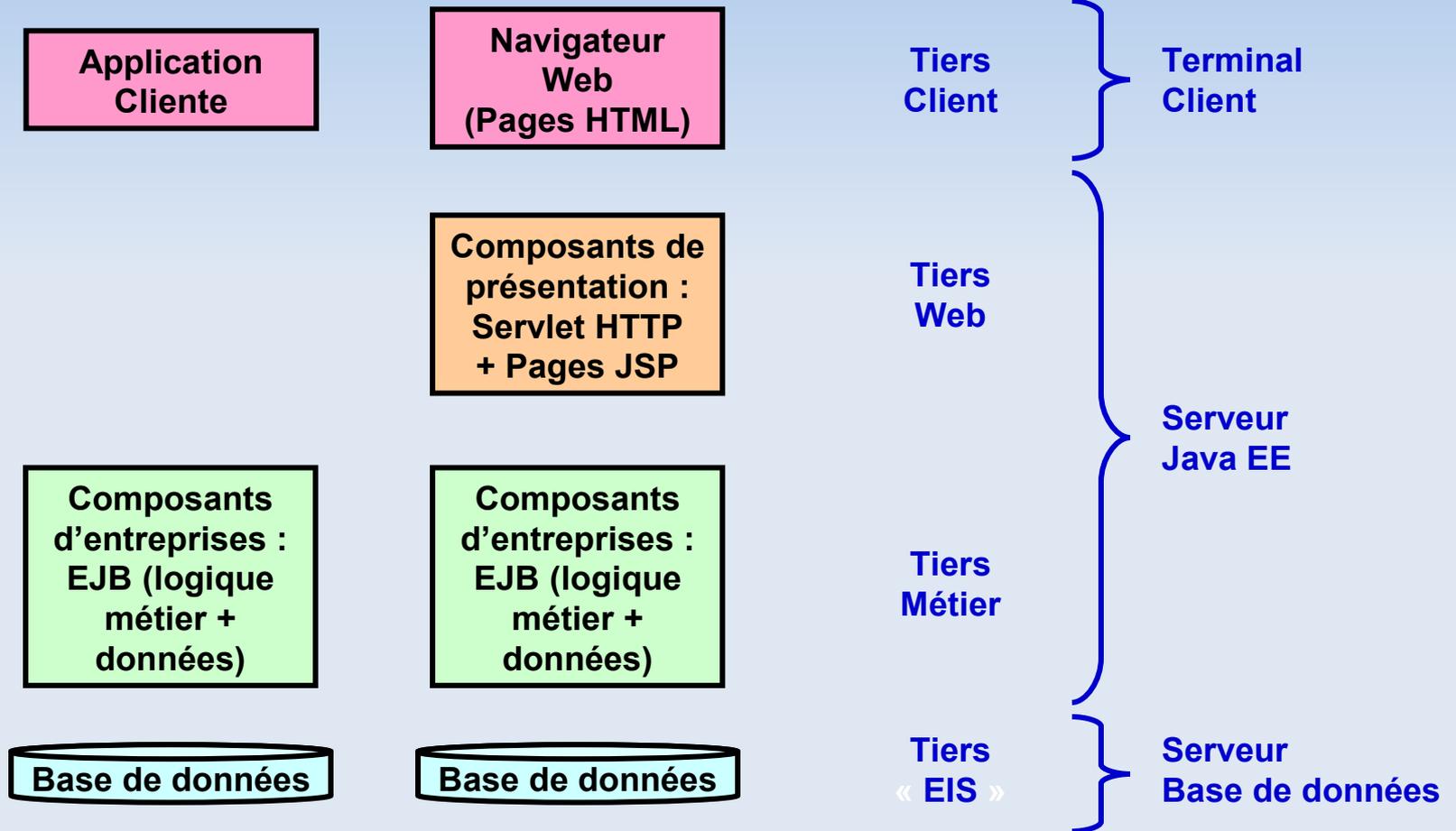
Introduction

Java EE pour quoi faire ?

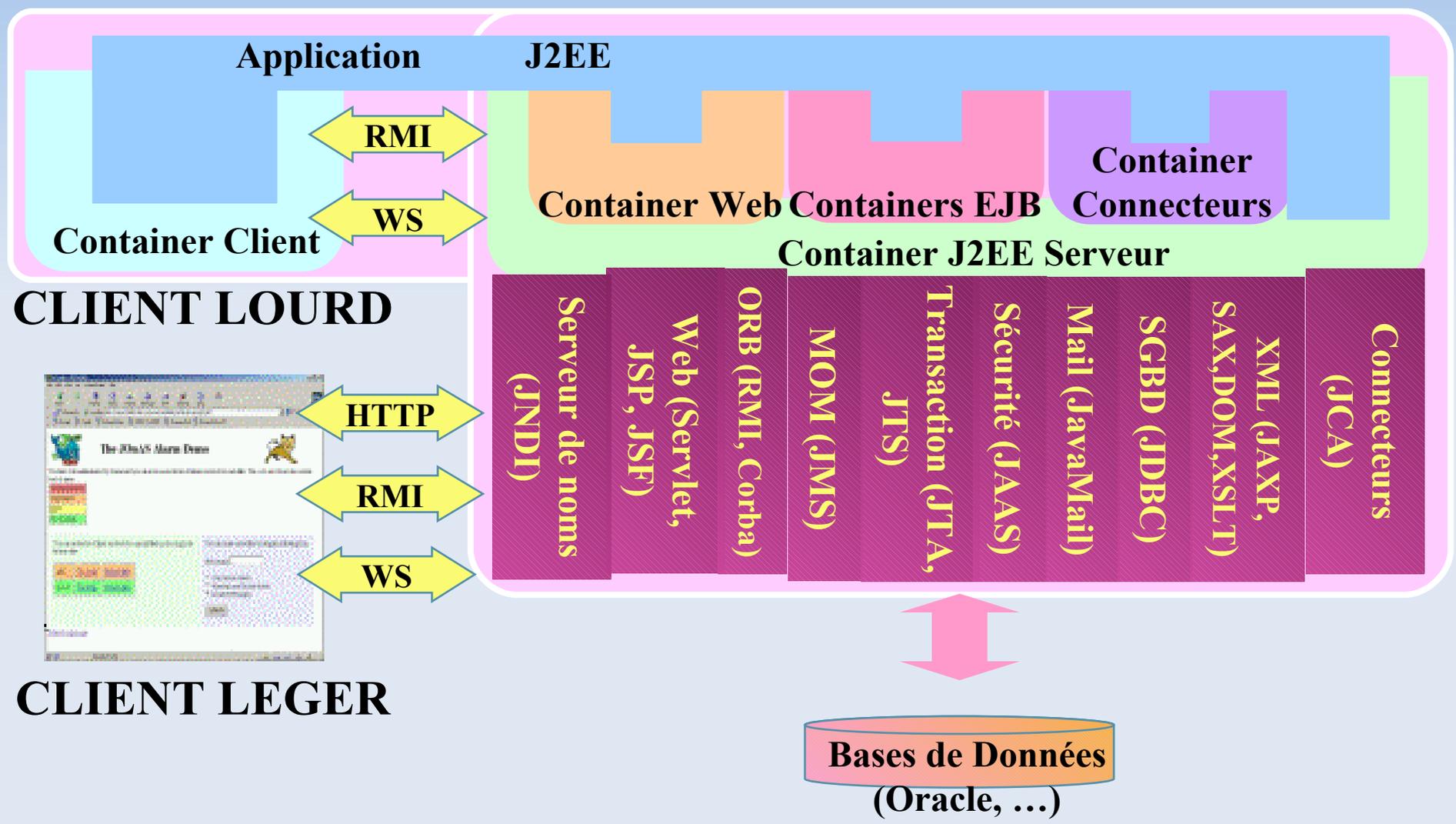
- Infrastructure « serveur » pour le support d'applications Web ou d'entreprise
 - ◆ E-commerce, SI, plateformes de services audio-visuel, telecoms, etc
- Architecture multi-tiers
 - ◆ Architecture client léger (basée « browser »)
 - ◆ Architecture client lourd (GUI avancées)
- Support de QoS : transaction, sécurité

—

Architectures usuelles



Vue générale de l'architecture Java EE



Modèles de programmation des composants EJB

Enterprise Java Beans

■ EJB est la partie centrale de la plateforme J2EE

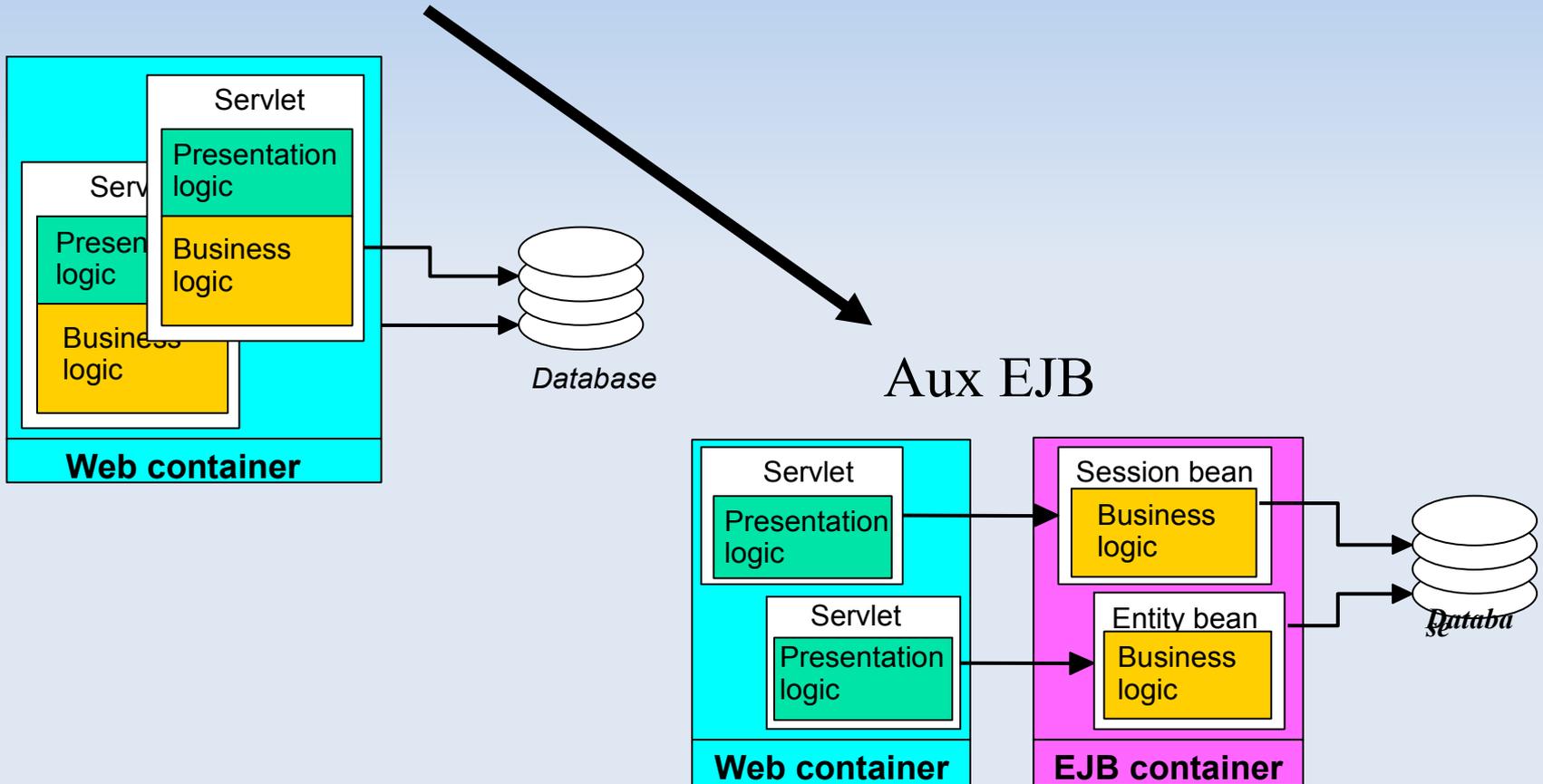
- ◆ Implanter la logique de traitement

■ Caractéristiques principales des EB

- ◆ composants "serveurs" spécialisés écrits en Java
- ◆ Support pour les aspects non fonctionnels de l'application
 - ❖ Persistance
 - ❖ Transaction
 - ❖ Sécurité
 - ❖ cycle de vie

Des servlets aux EJB

Des Servlets



principales

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

■ L'architecture EJB identifie les éléments suivants :

- ◆ composants logiciels ou *beans*,
- ◆ *conteneurs*,
- ◆ *serveurs*,
- ◆ *Clients (servlets, jsp ...)*

■ Les conteneurs

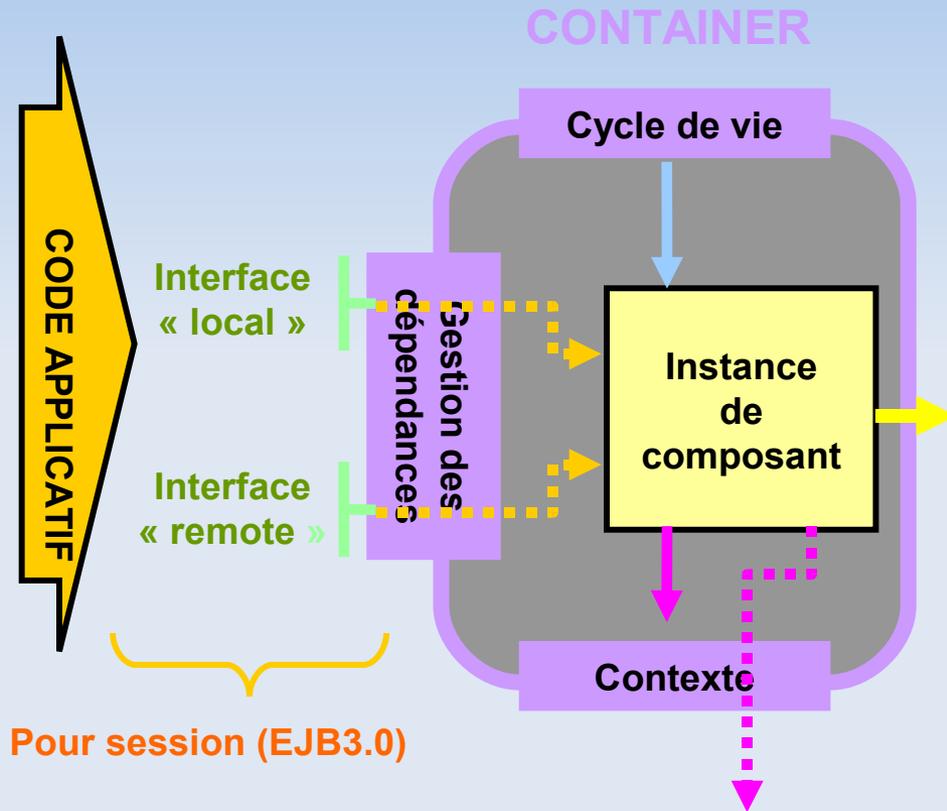
- ◆ isolent les beans du client et d'une implémentation spécifique d'un serveur
- ◆ Assurent
 - ❖ Gestion du cycle de vie
 - ❖ La liaison avec les services non-fonctionnels (persistance, transaction ...)

Rôle d'un conteneur

- Gestion du cycle de vie des composants
 - ◆ Installation (archives `.ear`, `.war`, ...)
 - ◆ Activation, passivation
 - ◆ Démarrage, arrêt

- Gestion des dépendances
 - ◆ Vers des composants
 - ◆ Vers des services de l'infrastructure
 - ◆ Vers des propriétés

Principe du conteneur Java EE



Pour session (EJB3.0)

- *EJB3.0: gestion des dépendances injectée dans les composants (e.g., injection de bytecode)*

Services non fonctionnels
JNDI :

- propriétés
- liens de composition

Composants EJB

- Composants applicatifs (code métier)
- Potentiellement répartis, transactionnels et sécurisés
- Trois profils
 - ◆ Session Beans
 - ❖ Instances dédiées à un contexte d'interaction client particulier
 - ❖ Avec / sans état
 - ◆ Entity Beans (EJB2.1) / POJOs (EJB3.0)

Définition de composants Java EE

■ Composant Java EE

◆ Code Java (POJO)

❖ Interfaces d'accès aux instances

▲ Interfaces locales et remotes

❖ Implantation des interfaces

◆ Méta-information

❖ Définition des dépendances

❖ Mapping bd

■ Expression de la méta-information

Session beans

- Non persistant (short-lived)
- Stateless session bean
 - ◆ Pas d'état
 - ◆ Peut être utilisé par tout client
- Stateful session beans
 - ◆ Maintient un état
 - ◆ Associé à un client unique (session conversationnelle)
- Détruits après un arrêt (ou une panne) de l'EJB serveur

Entity beans

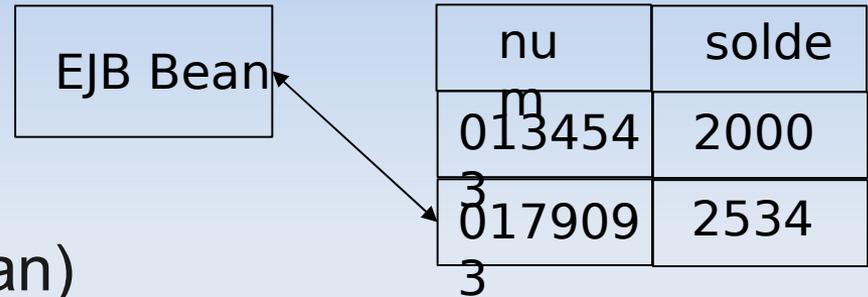
- Représentent les données d'une base de données
- Sont persistants (long-lived)
 - ◆ la gestion de la persistance via un entity manager
- Acceptent les accès multiples effectués par plusieurs clients
 - ◆ gestion de la concurrence
- Survivent aux pannes d'un serveur EJB

Entity bean

■ = tuple dans une table relationnelle

■ 3 types d'attributs

- ◆ Temporaire
- ◆ Persistant
- ◆ Primary key (= ref de bean)

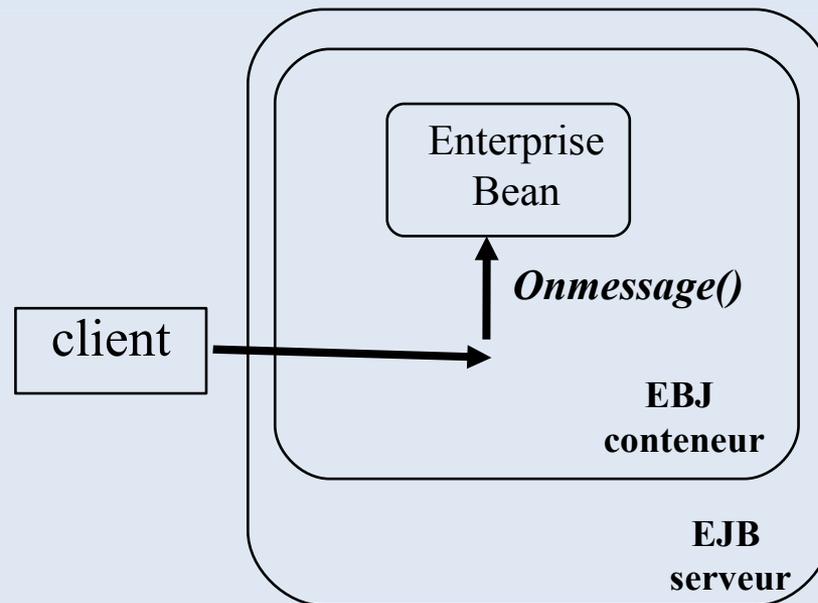


■ Relation entre bean

- ◆ 1-1
- ◆ 1-n : commande contient n produits
- ◆ n-n : un cours comporte plusieurs étudiants qui suivent plusieurs cours

Message driven Bean

- Message driven bean
 - ◆ Gestion de messages asynchrones avec JMS
 - ◆ JMS MessageListener

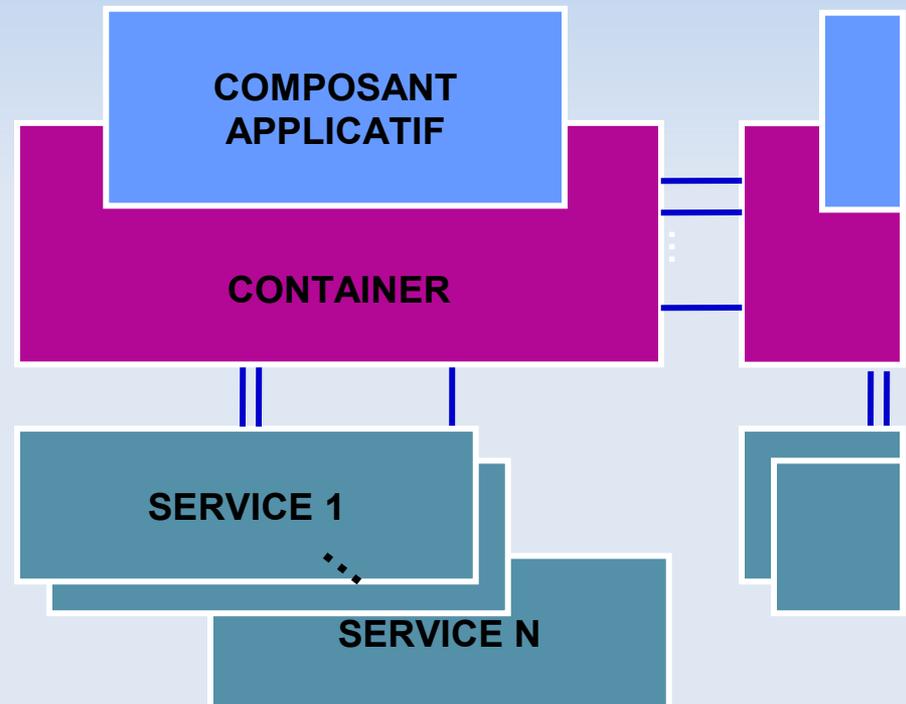


Interface et implémentation d'un composant

- Interface d'accès à un composant
 - ◆ Accès réparti à travers RMI et local dans la JVM
 - ◆ Accès réparti applicable aux profils de composants suivants
 - ❖ EJB3.0: Session
 - ◆ Définition d'une interface « remote »
 - ❖ EJB3.0: annotation @Remote
 - ◆ Définition d'une interface locale
 - ❖ EJB3.0: cas par défaut
 - ◆ Accès via SOAP pour session beans (EJB3.0)
- Implementation
 - ◆ Classe annotée par le profil en EJB3.0
 - ◆ @stateless, @statefull, @messagedriven, @entity

Gestion des dépendances

- Inversion de controle
- Gérées par les conteneurs
 - ◆ Mécanismes *d'injection* et les *callbacks applicatifs*
 - ◆ *Utilisation des annotations*
- Indépendance des modules logiciels
 - ◆ Pas de liaison statique entre modules de code applicatif (composants)
 - ◆ Pas de liaison statique entre modules de code applicatif et services plate-forme



Composants session

- Stateful / Stateless
- Stateful associés de façon unique à un client particulier
- Manipulation par méthodes métier
- Comportement transactionnel spécifique à chaque méthode

Exemple de définition d'une interface Remote d'un composant session en EJB3.0

```
package ecole_them;
import javax.ejb.Remote;

@Remote
public interface Inscription {
    public void enregistre(String nom);
    public ParticipantDTO infos(String nom);
    public void inscriptionConfirmer();
}

class ParticipantDTO { // objet de transfert de données
    public String nom;
    ...
}
```

Exemple de programmation d'un composant session en EJB3.0

```
package ecole_them;
import javax.ejb.*;

@Stateless
public class InscriptionBean implements Inscription {

    public void enregistre(String nom) {...}

    public void inscriptionConfirmer() {...}
    ...
}
..
```

Exemple de dépendance vers un composant session en EJB3.0

```
package ecole_them;
import javax.ejb.*;

public class InscriptionServlet extends HttpServlet {

    @EJB
    private Inscription inscription;

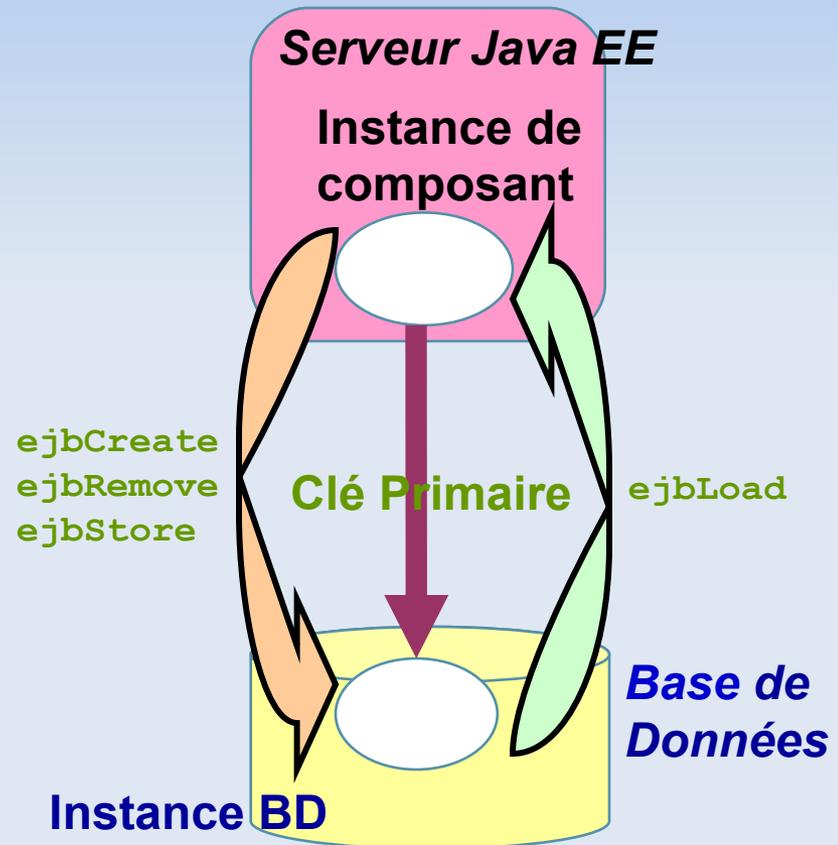
    public void service(HttpServletRequest req, HttpServletResponse resp) {
        name = ...;
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        inscription.enregistre(name);
        out.println("<html><body>Inscription enregistrée</body></html>");
    }
}
...
```

Composants Entity Beans (EJB2.1) ou POJOs (EJB3.0)

- Composants métier représentant des données des bases d'informations de l'entreprise
- Propriété de persistance
 - ◆ Gérée selon le standard JPA (EJB3.0)

Modèle de persistance des entités

- Notion de « clé primaire »
 - ◆ Nom persistant qui désigne 1 instance BD
- Sélection d'entités persistantes
 - ◆ Annotation *NamedQuery* en EJB3.0
 - ◆ Implémentées par une requête EJBQL (langage proche d'OQL)
 - ◆ Retourne une entité ou une collection d'entités



API du système de persistance

- `EntityManagerFactory` représente un espace de persistance (une base de données particulière)
- `EntityManager` représente un contexte d'exécution (CRUD operations)
 - ◆ SELECT
 - ◆ INSERT
 - ◆ etc
- `Query` représente une requête EBJQL (définition et exécution)

Programmation d'un composant persistant en EJB3.0

- POJO avec annotations (`javax.persistence.*`)
- Annotations de classes
 - ◆ `@Entity` pour définir une classe correspondant à un bean entité
 - ◆ `@NamedQuery` pour associer une requête nommée à ce bean
- Annotations de méthode ou d'attribut
 - ◆ `@Id` pour définir une clé primaire
 - ◆ `@GeneratedValue` pour définir un attribut dont la valeur est générée par le conteneur
- Beans rendus persistants
 - ◆ explicitement (action `persist`)
 - ◆ par attachement (liens de propagation configurables)

Les requêtes nommées en EJB3.0

- Factorisation de définition
- Pas de recompilation à chaque exécution

```
Collection<Participant> participantsFinder() {  
    Query participants =  
    entityManager.createNamedQuery("tousLesParticipants");  
    return participants.getResultList();  
}
```

Mapping avec la base de données en EJB3.0

- Chaque classe de bean entité est associée à une table
 - ◆ Par défaut, le nom de la table est le nom de la classe
 - ◆ Sauf si annotation `@Table(name=« Participant »)`
- Deux modes de définition des colonnes des tables (donc des attributs persistants)

Mapping avec la base de données (2)

■ Types supportés pour les attributs

- ◆ Types primitifs (et leurs "wrappers"), String, Date, etc.

- ◆ Données binaires

`@Lob`

```
private byte[] picture;
```

- ◆ Références d'entité (relation ?-1)

- ◆ Collections de références d'entité (relation ?-n)

■ Colonne de jointure

- ◆ `@JoinColumn(name=".. ")`

■ Table de jointure

- ◆ `@JoinTable`

Exemple de programmation d'un composant persistant en EJB3.0 (1/2)

```
import javax.persistence.Entity;
...
@Entity
@NamedQuery(
    name="tousLesParticipants",
    query="SELECT * FROM Participant p")
@Table(name = "PARTICIPANT_ICAR")

public class Participant{

    private long id;
    private String name;
    private Ecole ecole;

    public Participant() {}
    public Participant(String name) {
        setName(name);
    }
}
```

```
@Id @GeneratedValue
    (strategy=GenerationType.AUTO)
@Column(name = "PARTICIPANT_ID")

public long getId(){
    return this.id;
}

public void setId(long id){
    this.id = id;
}

public Ecole getEcole(){
    return ecole;
}

public void setEcole(Ecole ecole){
    this.ecole = ecole;
}
...
```

Exemple de programmation d'un composant persistant en EJB3.0 (2/2)

```
import javax.persistence.Entity;
@Entity
@NamedQuery(
    name="toutesLesEcoles",
    query="SELECT * FROM Ecole e")

public class Ecole {
    private long id;
    private Collection<Participant>
        participants;
    ...

    public Ecole() {}
    public Ecole(String name) {
        setName(name);
    }

    @Id @GeneratedValue
        (strategy=GenerationType.AUTO)

    public long getId() {
        return this.id;
    }
    public void setId(final long id) {
        this.id = id;
    }
    public Collection<Participant>
        getParticipants() {
        return participants;
    }
    public setParticipants(
        Collection<Participant>participants){
        this.participants = participants;
    }
}
```

Exemple de dépendance vers un composant persistant en EJB3.0

```
@Stateless
public class InscriptionBean implements Inscription {
    @PersistenceContext
    private EntityManager em;

    void enregistre(String nom) {
        Participant p = new Participant(nom);
        em.persist(p);
    }

    ParticipantDTO infos(String nom) {
        Query q = em.createQuery(
            "select OBJECT(i) from Participant p where p.nom = :np");
        q.setParameter("np", nom);
        Participant p = (Participant) q.getSingleResult();
        ...
    }
    ..
}
```

Fichier de définition des unités de persistance en EJB3.0 : "persistence.xml"

```
<persistence>
  <persistence-unit name="DonneesEcole">
    <jta-data-source>jdbc_1</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create-drop" />
      ...
    </persistence-unit>
</persistence>
```

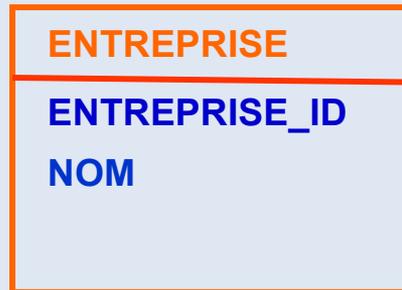
Les relations en EJB3.0

- Définition de relations entre composants persistants
 - ◆ Cardinalité 1-1, 1-n, n-1, n-n
 - ❖ @OneToOne, @OneToMany, @ManyToOne, @ManyToMany
 - ◆ Gestion optimisée par la base de donnée

Sans les relations...

```
import javax.persistence.Entity;  
@Entity  
public class Entreprise {  
  
    String nom;  
    Collection<Participant> participants;  
    ...  
}
```

```
import javax.persistence.Entity;  
@Entity  
public class Participant {  
  
    String nom;  
    Entreprise entreprise;  
    ...  
}
```



Les colonnes
ENTREPRISE_PARTICIPANTS et
ENTREPRISE_ID représentent les
mêmes informations !!

Avec les relations

(OneToMany /ManyToOne)

```
import javax.persistence.Entity;
@Entity
public class Entreprise {
    String nom;

    @OneToMany(mappedBy="entreprise")
    Collection<Participant> participants;

    public Collection<Participant>
    getParticipants() {
        return participant;
    }

    public void setParticipants
    (Collection<Participant> participants) {
        this.participants = participants;
    }

    public void addParticipant(..) {
        Participant p = new Participant(..);
        getParticipants().add(p);
    }
    ...
}
```

```
import javax.persistence.Entity;
@Entity
public class Participant {
    String nom;
    Entreprise entreprise;

    @ManyToOne
    @JoinColumn(name="Entreprise_id")
    public Entreprise getEntreprise() {
        return entreprise;
    }

    public void setEntreprise(Entreprise e) {
        this.entreprise = e;
    }
    ...
}
```

nom de la colonne
de jointure



ENTREPRISE

ENTREPRISE_ID
NOM

PARTICIPANT

PARTICIPANT_ID
NOM
ENTREPRISE_ID

Avec les relations (ManyToMany)

```
import javax.persistence.Entity;
@Entity

public class Ecole {
    String nom;

    @ManyToMany(mappedBy="ecoles")
    Collection<Participant> participants;
    ...
}
```

```
import javax.persistence.Entity;
@Entity

public class Participant {
    String nom;

    @ManyToMany
    @JoinTable(name="ECOLES_PARTICIPANTS",
        joinColumns=
            @JoinColumn(name="CI_PARTICIPANT_ID",
                referencedColumnName="PARTICIPANT_ID"),
        inverseJoinColumn=
            @JoinColumn(name="CI_ECOLE_ID",
                referencedColumnName="ECOLE_ID"))
    private Collection<Ecoles> ecoles;
    ...
}
```

ECOLE

ECOLE_ID
NOM

PARTICIPANT

PARTICIPANT_ID
NOM
ENTREPRISE_ID

ECOLES_PARTICIPANTS

CI_PARTICIPANT_ID (foreign key)
CI_ECOLE_ID (foreign key)

Composants EJB

« orientés message »

- Beans sans état
- Gestion par le conteneur
 - ◆ Réaction sur réception de message
 - ◆ Réaction transactionnelle ou non
- Dépendance vers une destination JMS
 - ◆ Agissent comme des `MessageListener`
 - ❖ Pour une `Queue` (1 récepteur pour un message)

Exemple de programmation d'un composant orienté message en EJB3.0

```
@MessageDriven(mappedName="java:/TopicIcar06")

public class NouvelleInscriptionBean implements javax.jms.MessageListener {

void onMessage(Message m) {
    System.out.println("Nom du participant : " +
        (MapMessage)m.getString("NomParticipant"));
}

}
```

Exemple d'une publication de message en EJB3.0

```
@  
@Resource(name="jms/TopicConnectionFactory")  
private TopicConnectionFactory tcf;  
  
@Resource(mappedName="java:/TopicIcar06")  
private javax.jms.Topic topic;  
  
TopicConnection topicConnection = tcf.createTopicConnection();  
TopicSession topicSession = topicConnection.createTopicSession(false,  
    Session.AUTO_ACKNOWLEDGE);  
  
MapMessage mess = topicSession.createMapMessage();  
mess.setString("NomParticipant", nom);  
  
TopicPublisher topicPublisher = topicSession.createPublisher(topic);  
topicPublisher.publish(mess);
```

Composants EJB

Timer

- Resource TimerService permet de créer des *timers*

```
@Resource TimerService ts;
```

- Permet de déclencher des actions périodiquement

```
@Timeout
```

```
public void envoiPubEcole() {...}
```

Callbacks applicatives : spécialisation du cycle de vie

- Permet d'ajouter des traitements métier à certains moments du cycle de vie d'un bean
- Session bean
 - ◆ @PostConstruct : après la création
 - ◆ @PreDestroy : avant la destruction
 - ◆ @PrePassivate : avant la passivation (swapping out), seulement pour statefull
 - ◆ @PostActivate : après l'activation (swapping in)
- Entity bean
 - ◆ @ejbCreate
 - ◆ @ejbPostCreate
 - ◆ @ejbRemove
 - ◆ @ejbPostRemove
 - ◆ @PrePassivate : avant la passivation (swapping out)
 - ◆ @PostActivate : après l'activation (swapping in)

Gestion des transactions

- Applicables aux profils de composants

- ◆ Session
- ◆ Entity (EJB2.1 seulement)
- ◆ Message Driven Bean

- Propriétés ACID

- ◆ Atomicity, Consistency, Isolation, Durability

- Gestion déclarative ou programmatique (JTA)

des transactions (« container- managed »)

- Méthodes transactionnelles
- Descripteur de déploiement / Annotations
 - ◆ @TransactionAttribute(TransactionAttributeType.REQUIRED)
 - ◆ Annotation pour une méthode

- Comportements possibles
 - ◆ « NotSupported » : si transaction courante, elle est suspendue
 - ◆ « Required » : si pas de transaction, nouvelle transaction
 - ◆ « RequiresNew » : nouvelle transaction (si tx courante, suspendue)
 - ◆ « Mandatory » : exception si pas de transaction courante
 - ◆ « Supports » : si transaction courante, l'utiliser
 - ◆ « Never » : exception si transaction courante

- Seuls « Required » et « NotSupported » valables pour les MDB

Exemple de gestion transactionnelle déclarative

...

```
@Stateless
```

```
@TransactionManagement(TransactionManagementType.CONTAINER)
```

```
public class InscriptionBean implements Inscription {
```

```
    @Resource
```

```
    private SessionContext context;
```

```
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
```

```
    public void enregistre(String nom) {
```

```
        ...
```

```
    } catch (EnregistrementException e) {
```

```
        context.setRollbackOnly();
```

```
    }
```

```
        ...
```

```
    }
```

...

Packaging de l'application

Application ICAR

Présentation ICAR

Servlet/JSP
+
Ressources de présentation
(images, etc.)

Métier "école thématique"

EJB
+
Classes persistantes

- Application ICAR
 - ◆ Application JEE
 - ◆ Packaging : icar.ear
 - ◆ Contient 2 modules
- Module de présentation
 - ◆ Spécifique à ICAR
 - ◆ Packaging : icarweb.war
- Module métier
 - ◆ Gestion d'une école thématique
 - ◆ Non spécifique à l'école ICAR (réutilisable)
 - ◆ Packaging : ecole_them.jar
 - ◆ Contient code des EJB et des composants persistants

Package "icar.ear"

Contenu de l'archive

"icar.ear" :

META-INF/

 MANIFEST.MF

icarweb.war

ecole_them.jar

Package "webicar.war"

Contenu de l'archive
"webicar.war" :

```
META-INF/  
  MANIFEST.MF  
WEB-INF/  
  web.xml  
  classes/  
    org/  
      icar/  
        servlet/  
          *.class
```

Contenu de "web.xml" :

```
<web-app>  
  <servlet>  
    <servlet-name>ICAR'06</servlet-name>  
    <servlet-class>  
      org.icar.servlet.TraiteFormulaireInscr  
    </servlet-class>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>ICAR'06</servlet-name>  
    <url-pattern>/inscr/traitform</url-pattern>  
  </servlet-mapping>  
  <ejb-ref>  
    <ejb-ref-name>ejb/Inscription</ejb-ref-  
name>  
    ...  
  </ejb-ref>  
</web-app>
```

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Package

"ecole_them.jar"

Contenu de l'archive
"ecole_them.jar"
(persistence.xml, cf.
section persistance) :

```
META-INF/  
  MANIFEST.MF  
  ejb-jar.xml  
  persistence.xml  
org/  
  ecole/  
    api/  
      *.class  
    lib/  
      *Bean.class  
  persistence/  
    *.class
```

Contenu de "ejb-jar.xml" (pas obligatoire en EJB3.0) :

```
<ejb-jar>  
  <enterprise-beans>  
    <session>  
      <ejb-name>Inscription</ejb-name>  
      <ejb-class>  
        org.ecole.lib.InscriptionBean</ejb-class>  
      <session-type>Stateless</session-type>  
      <transaction-type>Container</transaction-type>  
      ...  
    </session>  
  </enterprise-beans>  
</ejb-jar>
```

Conclusion

- Une des solutions industrielles les plus abouties
 - ◆ Prise en charge des contraintes techniques
 - ◆ Gestion du packaging et du déploiement
 - ◆ Spectre fonctionnel large

- Vers une stabilisation des spécifications
 - ◆ Capitalisation importante sur Corba
 - ◆ Support XML / Web Services (Java EE 1.4 = 1000 pages de « tutorial »)
 - ❖ L'interface fournie par un session bean peut être exposée comme un Web Service

- Problèmes de maturité
 - ◆ Persistance (co-existence avec JDO)
 - ◆ Modèle de composants (manque d'homogénéité, problèmes avec gestion de l'héritage)