
L'environnement Java EE

Principes, fonctions, utilisation

F. Boyer, présentation pour la filière M2PGI 2007/2008, Université Joseph Fourier

*Ce support de cours a été conçu à partir du cours de Pascal Déchamboux (FT R&D)
pour l'école d'été ICAR 2003*

Sommaire

■ Introduction

- ◆ Architecture Java EE
- ◆ Fonctions couvertes

■ Une application Java EE

- ◆ Packaging

■ Approche à composants

- ◆ Conteneurs
- ◆ Composants
- ◆ Dépendances et assemblage

■ Modèles de programmation

- ◆ Clients
- ◆ Composants session
- ◆ Composants entité

■ Gestion des transactions

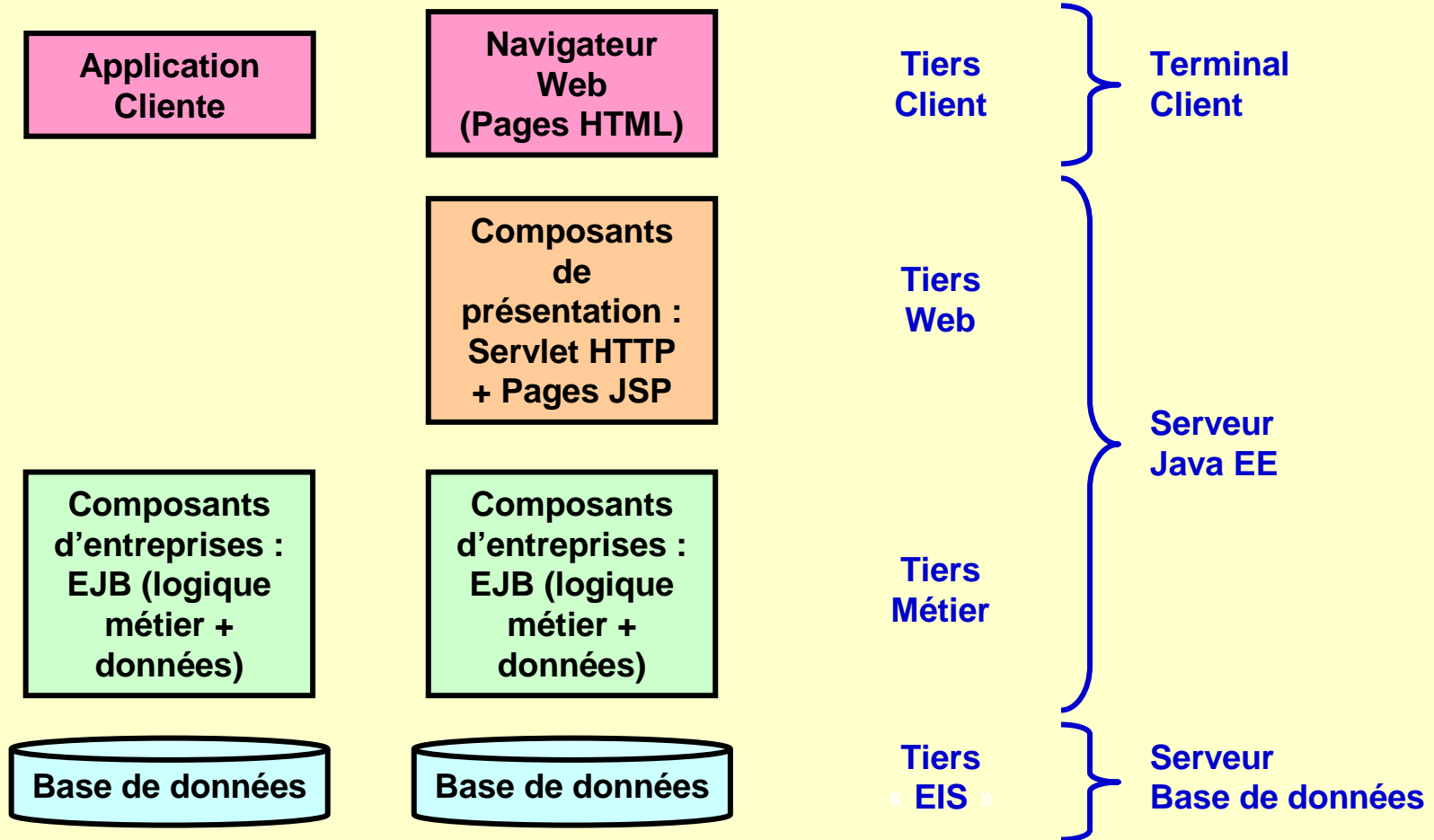
■ Gestion de la sécurité

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**
- **Connexion standard à des systèmes d'information externes (accès au « legacy »)**

Architectures usuelles (4 tiers)



Java EE sous l'œil de Darwin !

- **Standard en évolution/maturation depuis 1997/1998 (J2EE 1.0, ..., 1.3, 1.4, Java EE depuis 2006)**
- **Au départ support d'applications Web n-tiers (architecture décentralisée)**
 - ◆ **Présentation : Servlet (principalement HTTP)**
 - ◆ **Logique métier : EJB**
 - ◆ **Gestion de données : JDBC**
- **Vers une infrastructure de support standard pour EAI**
 - ◆ **Facteurs de rationalisation majeur (JTA/JTS, JMS, JCA, WS)**
 - ◆ **Évolution de produits existants vers Java EE**

Ce que définit Java EE

■ **Spécification**

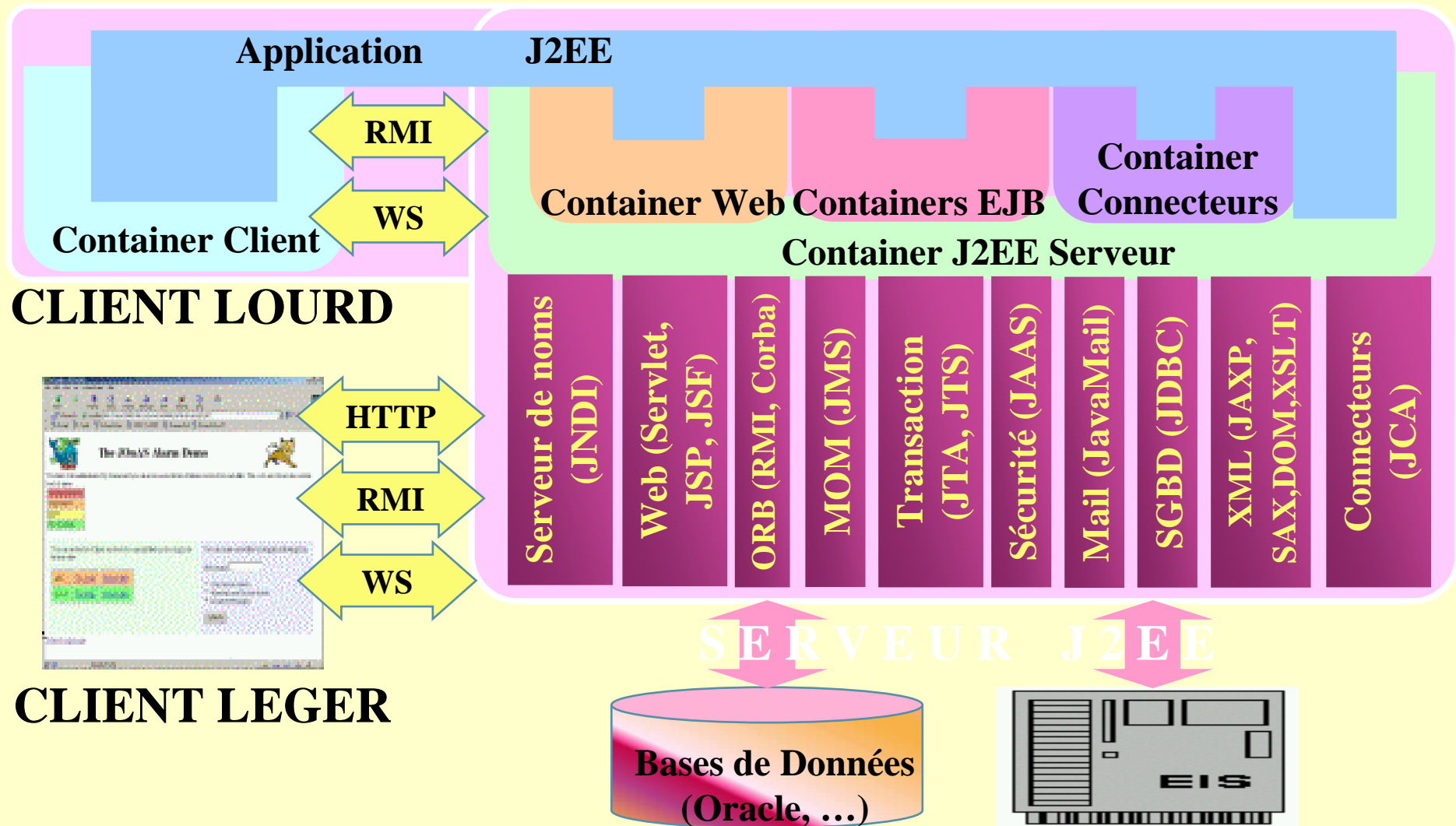
- ◆ Programmation, assemblage, déploiement
- ◆ Serveur et services

■ **Implantation de référence opérationnelle**

■ **Suite de tests de conformité**

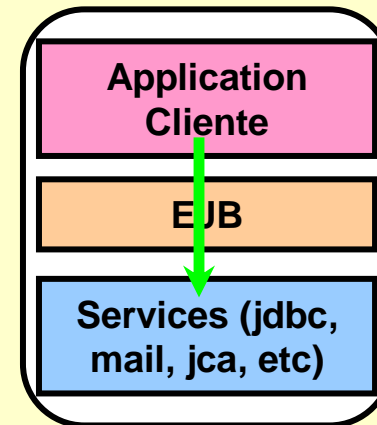
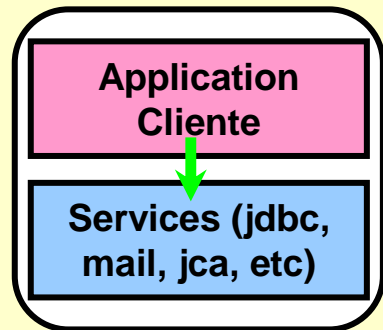
- ◆ Certification Sun
- ◆ Accès au processus de certification payant (cher !!)
- ◆ Lourd (> 20.000 tests)

Vue générale de l'architecture Java EE

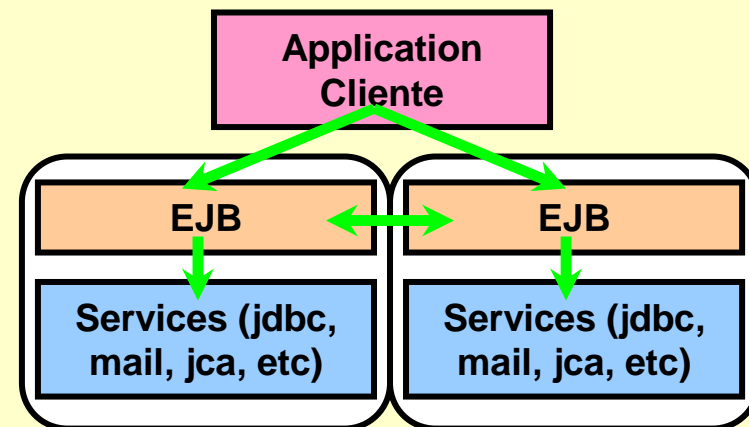
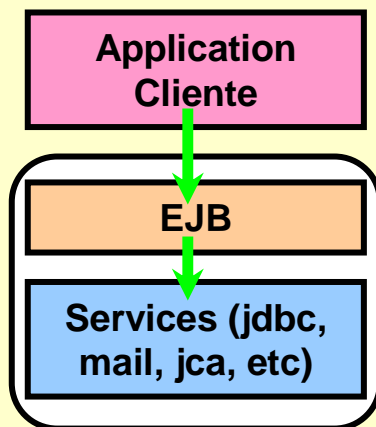


Configurations possibles d'un serveur Java EE : clients lourds

Application Java EE :
Architecture centralisée

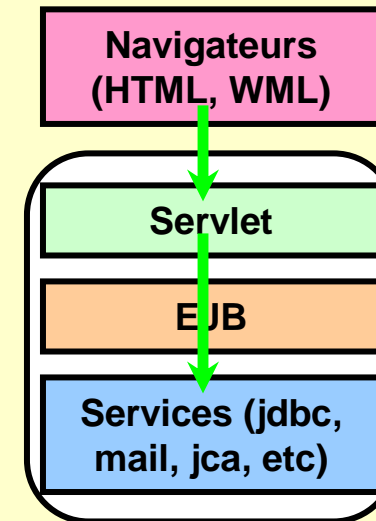
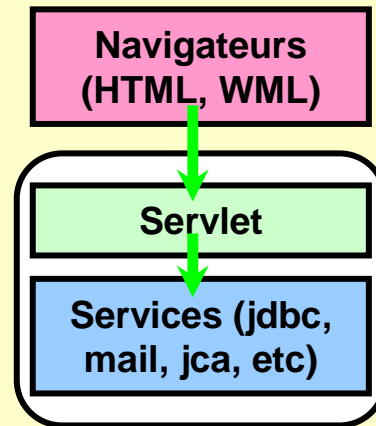


Application Java EE :
Architecture client/serveur

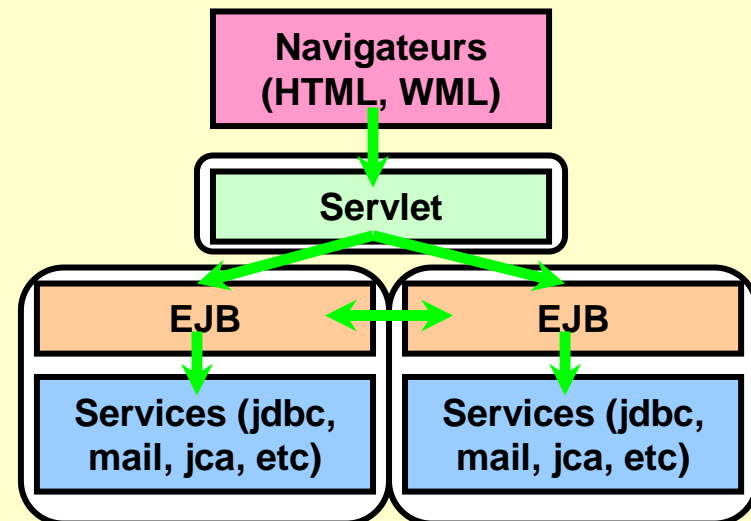
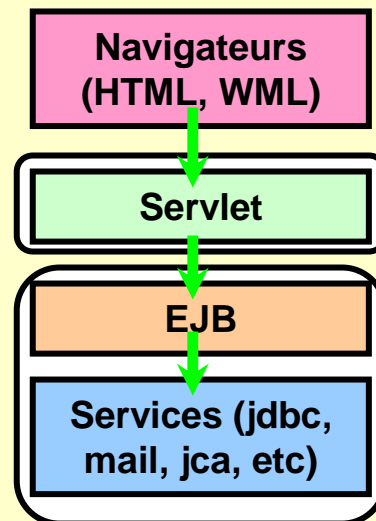


Configurations possibles d'un serveur Java EE : clients Web

Application Java EE :
Architecture Web
/
Serveur centralisé



Application Java EE :
Architecture Web
/
Serveur réparti



Offre importante

■ Offre commerciale

- ◆ IBM / WebSphere (n°1)
- ◆ BEA / WebLogic
- ◆ Sun One
- ◆ Oracle 9i Application Server
- ◆ Et aussi Borland Enterprise Server, Macromedia / Jrun, SAP Web Application Server, Iona / Orbix E2A

■ Offre « open source »

- ◆ JBoss
- ◆ JOnAS
- ◆ ...

Les fonctions couvertes par J2EE - 1

(standards connexes)

- **Java Servlets**
- **JavaServer Pages (JSP)**
- **Enterprise Java Beans**
- **Java Message Service (JMS)**
- **Data Base connectivity (JDBC)**
- **Transactions**
 - ◆ JTA
 - ◆ JTS
- **Java EE Connector Architecture (JCA)**
- **Corba (Java IDL)**
- **JavaMail**

Les fonctions couvertes par J2EE - 2 (standards connexes)

■ XML/SOAP

- ◆ Java API for XML Processing (JAXP)
- ◆ Java API for XML Registries (JAXR)
- ◆ Java API for XML-Based Remote Procedure Call (JAX-RPC)
- ◆ SOAP with Attachments API for Java (SAAJ)

■ Java EE Deployment Specification (JSR-88)

■ Java EE Management Specification (JSR-77)

■ JMX

Vers Java EE 5

■ Simplification du développement

- ◆ Utilisation des annotations amenées par Java SE 5
- ◆ Relâchement des contraintes / programmation des composants

■ Enrichissement fonctionnel

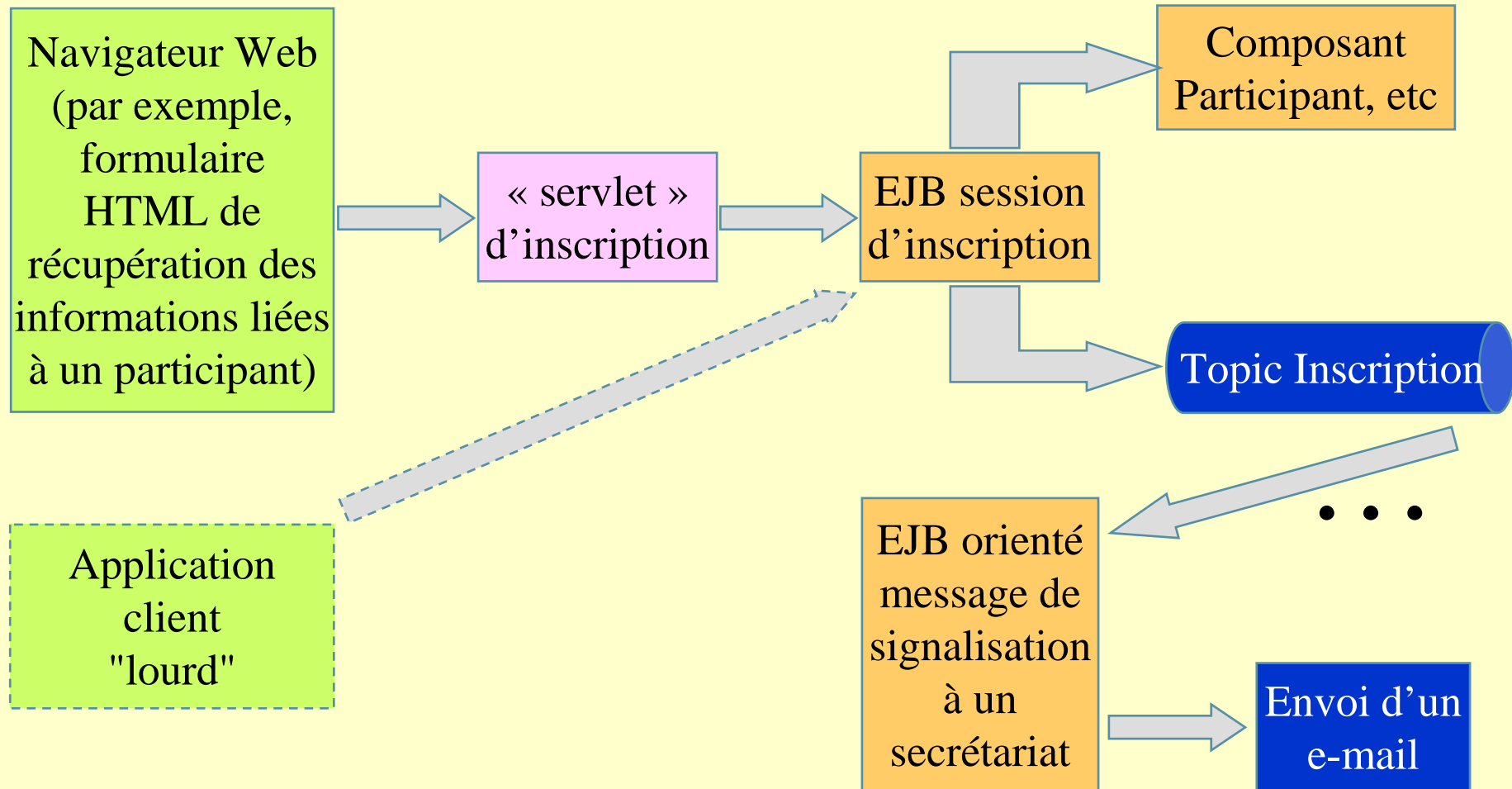
- ◆ Amélioration du support du tiers de présentation avec Ajax / Flex
- ◆ Support complet des Web Services
 - ❖ Sessions exposables avec RMI ou avec Web Services
- ◆ Amélioration du support des objets persistants avec Java Persistence API (un grand pas vers JDO!)

Plus d'information...

- <http://java.sun.com/>
- <http://www.theserverside.com/>
- <http://developer.java.sun.com/developer/technicalArticles/J2EE/>
- <http://developer.java.sun.com/developer/onlineTraining/J2EE/>
- <http://www.triveratech.com/>
- <http://jonas.objectweb.org/>

Exemple

Inscription à une conférence



Décomposition 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

Packaging

Package "icar.ear"

Contenu de l'archive "icar.ear" :

META-INF/

MANIFEST.MF

application.xml

icarweb.war

ecole_them.jar

Contenu de "application.xml" :

```
<application>
  <display-name>ICAR'06</display-name>
  <description>Gestion de l'école ICAR 06
</description>
  <module>
    <web>
      <web-uri>icarweb.war</web-uri>
      <context-root>icar06</context-root>
    </web>
  </module>
  <module>
    <ejb>ecole_them.jar</ejb>
  </module>
</application>
```

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>
```

Package "ecole_them.jar"

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

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

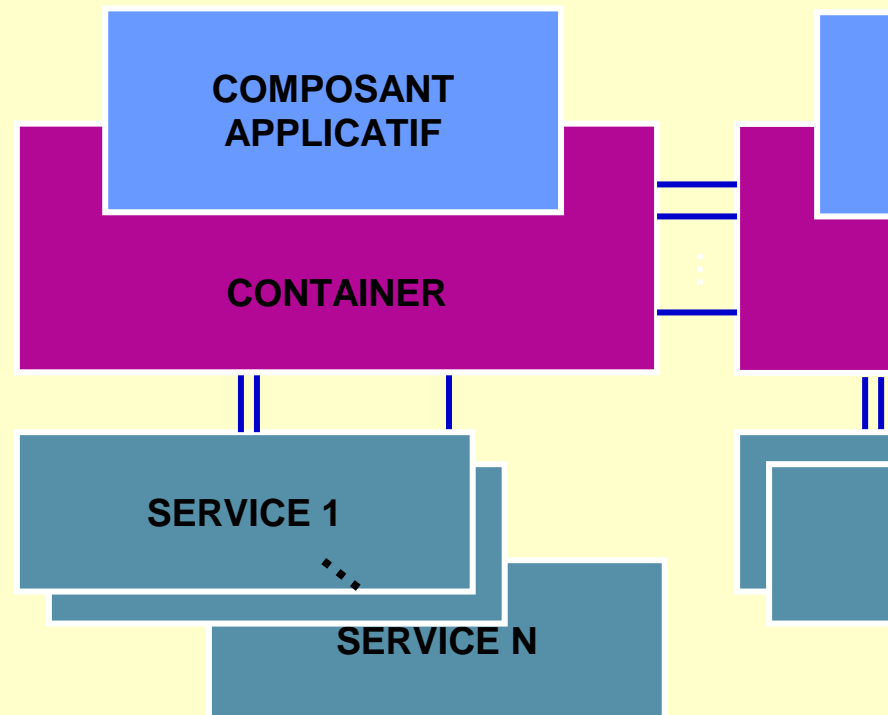
Contenu de "ejb-jar.xml" (pas nécessaire en
EJB3.0) :

```
<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>
```

Approche à composants

Indépendance des modules logiciels comme principe directeur

- Recherche d'une indépendance maximale du code applicatif
 - ◆ Pas de liaison statique entre modules de code applicatif (composants)
 - ◆ Pas de liaison statique entre modules de code applicatif et services plate-forme
 - ◆ Eviter l'utilisation de code « technique » dans le code applicatif
- Pas important vers du code applicatif plus réutilisable !!



Composants Java EE

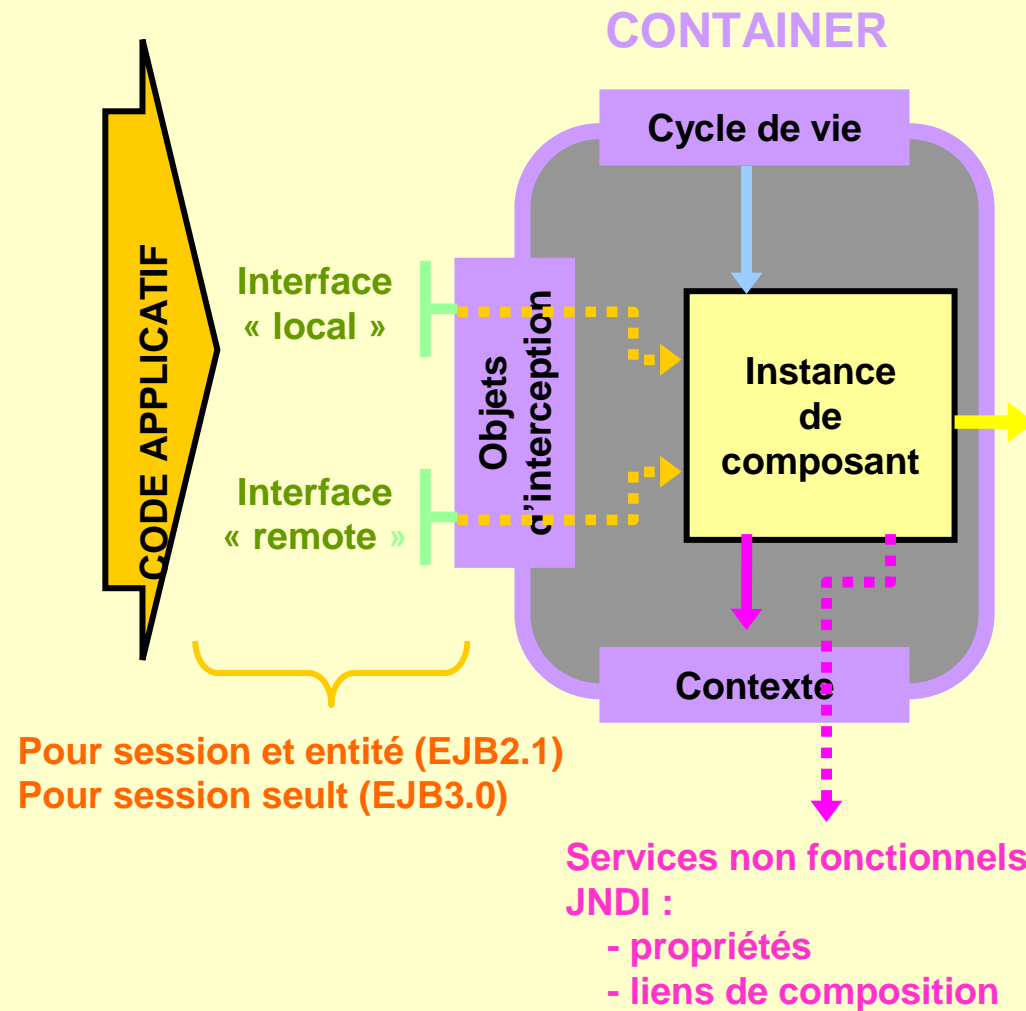
■ Modules de code réutilisable qu'on peut assembler

- ◆ Changement de liaisons entre modules de code application
⇒ pas d'impact sur le code !
- ◆ Configuration des liaisons entre code applicatif et ressources utilisées
⇒ pas d'impact sur le code !

■ Vision toutefois non systématique...

- ◆ Définitions de liaison spécifiques au type de liaison (par exemple, composant/composant, composant/ressource, etc.)

Le principe du « container » dans Java EE



- *EJB2.1: le code d'interception est généré dans des classes spécifiques*
- *EJB3.0: le code d'interception est intégré aux composants par injection de bytecode*

Principe de mise en oeuvre des containers

■ Interception

- ◆ Interception des appels applicatifs aux composants
 - ❖ Pour gérer les liaisons entre composants
 - ❖ Pour gérer les liaisons entre composants et services de l'infrastructure (mise en œuvre des propriétés non fonctionnelles)

■ Callbacks applicatifs

- ◆ Spécialisation de la gestion du cycle de vie (création, destruction, activation, passivation)

Utilisation d'un container

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

- **Gestion des dépendances entre composants**

- **Gestion des dépendances entre composants et services de l'infrastructure**

- **Gestion d'un contexte propre à chaque composant**

Principaux containers

- **Container d'applications clientes (client "lourd")**
- **Container d'applications serveurs (déployées dans un serveur Java EE)**
 - ◆ Container Web (de "servlet")
 - ◆ Container d'EJB
 - ◆ Container de Connecteurs ("resource adapters")
- **Container de persistance**
 - ◆ Indépendant de JEE
 - ◆ Intégré par le conteneur EJB

Définition de composants Java EE

■ Composant Java EE = code Java (POJO) + méta-information

- ◆ Interfaces d'accès aux instances
 - ❖ Interfaces locales et remotes
 - ❖ Interface home (EJB2.1)
- ◆ Implantation des interfaces
- ◆ Définition des dépendances

■ Expression de la méta-information

- ◆ Annotations Java SE5 + descripteurs XML (EJB3.0)
- ◆ Descripteurs XML (EJB2.1)

Définition des dépendances vers une propriété, un composant ou une ressource

- Définition d'un nom local (utilisé dans le code)
- La méta-information associe un lien effectif au nom local
- Le contexte contient l'association (nom local, lien effectif)

- Définition du lien effectif
 - ◆ Vers une propriété
 - ❖ Donne la valeur de la propriété
 - ◆ Vers un composant du même package
 - ❖ Désignation à l'aide du nom du composant
 - ◆ Vers un composant extérieur ou vers une ressource
 - ❖ Désignation à l'aide d'un nom global JNDI

- Conventions de nommage
 - ◆ Nom de propriété ⇔ nom de variables de classe Java
 - ◆ Nom de ressource ⇔ nom de classe Java préfixé par type de ressource (ex : ejb, jms, jdbc, mail, ...)

Manipulation du contexte d'un composant

```
// Accès au contexte d'un composant en EJB3.0
...
class InscriptionBean implements Inscription {
@Resource private SessionContext ctxt; // le contexte du composant
...
}
```

```
// Accès au contexte d'un composant en EJB2.1

class InscriptionBean implements Inscription, ... {
    private SessionContext ctxt; // le contexte du composant

    void setSessionContext(SessionContext c) {
        ctxt = c;
    }
...
}
```

Définition de propriétés en EJB 3.0

```
// Code du composant
```

```
...
```

```
class InscriptionBean implements Inscription {
```

```
    @Resource private int maxPart = 120; // maximum de participants
```

```
<!-- descripteur optionnel →
```

```
<session>
```

```
    <ejb-name>InscriptionBean</ejb-name>
```

```
    ...
```

```
    <env-entry>
```

```
        <description>Nombre maximum de participants</description>
```

```
        </env-entry-value>150</env-entry-value>
```

```
        <injection-target>maxPart</injection-target>
```

```
    </env-entry>
```

```
    ...
```

```
</session>
```

Définition de propriétés en EJB 2.1

```
// Code du composant contenant la dépendance en EJB2.1
```

```
class InscriptionBean implements Inscription {
```

```
...
```

```
    int maxParticipants = (int)  
        ctxt.lookup("java:comp/env/maxParticipants");
```

```
...
```

```
<!-- descripteur obligatoire →
```

```
<session>
```

```
    <ejb-name>InscriptionBean</ejb-name>
```

```
...
```

```
    <env-entry>
```

```
        <description>Nombre maximum de participants</description>
```

```
        <env-entry-name>maxParticipants</env-entry-name>
```

```
        <env-entry-type>java.lang.Integer</env-entry-type>
```

```
        </env-entry-value>150</env-entry-value>
```

```
    </env-entry>
```

```
...
```

Définition de dépendances vers un composant en EJB 3.0

```
// Code du composant (par exemple un client lourd ou un servlet)
// contenant la dépendance
```

```
@EJB(beanName="../ecole_them.jar#Inscription")
    private Inscription inscr;
```

```
// Ou si le composant référencé et le référençant sont dans le
// même package
```

```
@EJB    private Inscription inscr;
```

```
<!-- descripteur optionnel →
```

```
<ejb-ref>
```

```
...
```

```
    <ejb-link>../ecole_them.jar#InscriptionTest</ejb-link>
```

```
        <injection-target>inscr</injection-target>
```

```
</ejb-ref>
```

Définition de dépendances vers un composant en EJB 2.1

```
// Code du composant contenant une dépendance vers une interface distante
InscriptionHome inscrHome = (InscriptionHome)PortableRemoteObject.narrow(
    ctxt.lookup("java:comp/env/ejb/InscriptionHome", InscriptionHome.class));

<!-- descripteur obligatoire ->
<ejb-ref>
  <ejb-ref-name>ejb/InscriptionHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>org.icar.api.InscriptionHome</home>
  <remote>org.icar.api.Inscription</remote>
  <ejb-link>../ecole_them.jar#Inscription</ejb-link>
</ejb-ref>
```

```
// Code du composant contenant une dépendance vers une interface locale
InscriptionLocalHome inscrLocalHome =
    (InscriptionLocalHome)ctxt.lookup("java:comp/env/ejb/InscriptionlocalHome");

<!-- descripteur obligatoire ->
<ejb-local-ref>
  <ejb-ref-name>ejb/InscriptionLocalHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local>org.icar.api.Inscription</home>
  <local-home>org.icar.api.InscriptionLocalHome</local-home>
  <ejb-link>../ecole_them.jar#Inscription</ejb-link>
</ejb-local-ref>
```

Définition de dépendances vers une usine à connexions JDBC en EJB3.0

```
// Code du composant contenant la dépendance
@Resource (mappedName="java:/JdbcIcar")
    javax.sql.DataSource maBdIcar;
```

```
<!-- descripteur optionnel →
<resource-ref>
    <description>BD pour ICAR</description>
    ...
    <mapped-name>java:/JdbcIcarTest</mapped-name>
    <injection-target> maBdIcar </injection-target>
</resource-ref>
```

Définition de dépendances vers une usine à connexions JDBC en EJB2.1

```
// Code du composant contenant la dépendance en EJB2.1
```

```
DataSource maBdIcar = (DataSource)  
    ctxt.lookup("java:comp/env/jdbc/bdIcar");
```

```
<!-- obligatoire en EJB2.1 →
```

```
<resource-ref>
```

```
    <description>BD pour ICAR</description>
```

```
    <res-ref-name>jdbc/bdIcar</res-ref-name>
```

```
    <res-type>javax.sql.DataSource</res-type>
```

```
    <res-auth>Container</res-auth>
```

```
</resource-ref>
```

Définition de dépendances vers une ressource JMS en EJB3.0

```
// Code du composant contenant la dépendance en EJB3.0
```

```
@Resource(mappedName="java:/TopicIcar06")
    javax.jms.Topic topicInscription;
```

```
<!-- descripteur optionnel →
```

```
<resource-env-ref>
```

```
    <description>Topic pour processus d'inscription</description>
```

```
    <mapped-name>java:/TopicIcar07</mapped-name>
```

```
    <injection-target>topicInscription</injection-target>
```

```
</resource-env-ref>
```

Définition de dépendances vers une ressource JMS en EJB2.1

```
// Code du composant contenant la dépendance en EJB2.1
```

```
Topic signalisationInscription = (Topic)  
    ctxt.lookup("java:comp/env/jms/TopicInscription");
```

```
<!-- obligatoire en EJB2.1 →
```

```
<resource-env-ref>
```

```
    <description>Topic pour processus d'inscription</description>
```

```
    <resource-env-ref-name>jms/TopicInscription
```

```
</resource-env-ref-name>
```

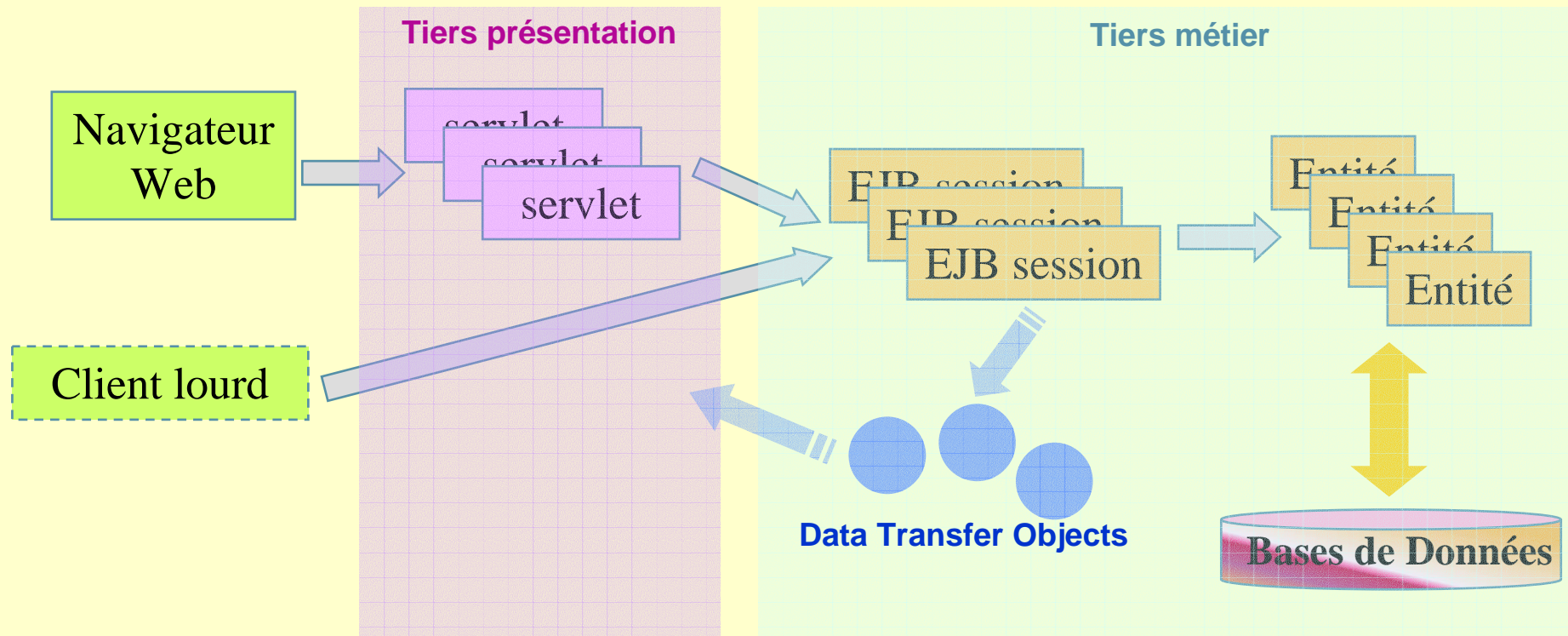
```
    <resource-env-ref-type>javax.jms.Topic</resource-env-ref-type>
```

```
</resource-env-ref>
```

Modèles de programmation des composants Java EE

- **EJBs**
- **Clients lourds**
- **Servlets**

Une conception classique J2EE



*Exemple: EJB Session ProductInfos fournit
getProduct(productName,..) → ProductDTO*

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)**
 - ❖ **Instances partagées représentant les données de l'entreprise**
 - ◆ **Message Driven Bean**
 - ❖ **Instances réagissant à l'arrivée de messages asynchrones**

Interfaces d'accès

- **Accès réparti à travers RMI et local dans la JVM**
- **Accès réparti applicable aux profils de composants suivants**
 - ◆ EJB3.0: Session
 - ◆ EJB2.1: Session, Entité, Session Home, Entité Home
- **Définition d'une interface « remote »**
 - ◆ EJB3.0: annotation `@Remote`
 - ◆ EJB2.1: étend `javax.ejb.EJBObject/EJBHome`
- **Définition d'une interface locale**
 - ◆ EJB3.0: cas par défaut
 - ◆ EJB2.1: étend `javax.ejb.EJBLocalObject/EJBLocalHome`
- **Accès via SOAP pour session beans (EJB3.0)**

Implantation d'un composant

■ Annoté par le profil en EJB3.0

- ◆ @stateless, @statefull, @messagedriven

■ En EJB2.1:

- ◆ Implante l'interface du profil concerné (javax.ejb.SessionBean, etc)
- ◆ Pour les sessions et entités
 - ❖ Implante les méthodes des interfaces « local » et « remote » (pas les interfaces elles-mêmes !)
 - ❖ Implante une version renommée des méthodes des interfaces « local home » et « remote home »

<code>interface InscriptionHome</code>	<code>implantation Bean</code>
<code>Inscription create(...)</code>	<code>void ejbCreate(...)</code>

- ◆ Implante un constructeur public sans paramètre

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);
}

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

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

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

@WebService // Only for stateless beans
public interface InscriptionWS {
    public void enregistre(String nom);
    public ParticipantDTO infos(String nom);
}

...
}
```

Exemple de définition de l'interface d'un composant session en EJB2.1

```
...
public interface Inscription extends EJBObject {
    public void enregistre(String nom) throws RemoteException;
    public InfoParticipant infos(String nom) throws RemoteException;
    ...
}

public interface InscriptionHome extends EJBHome {
    Inscription create() throws CreateException,
    RemoteException;
}

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 Incription {

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

Exemple de dépendance vers un composant session en EJB3.0

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

@Stateless
public class GoldInscriptionBean implements ... {

    @EJB
    private Inscription inscription;

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

Exemple de programmation d'un composant session en EJB2.1

```
...
class InscriptionBean implements SessionBean {
    private Context ctx;
    private ParticipantHome partH;

    void ejbActivate() {}
    void ejbPassivate() {}
    void ejbRemove() {}
    void ejbCreate() {}

    void setSessionContext(SessionContext c) {
        ctx = c;
        ...
    }

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

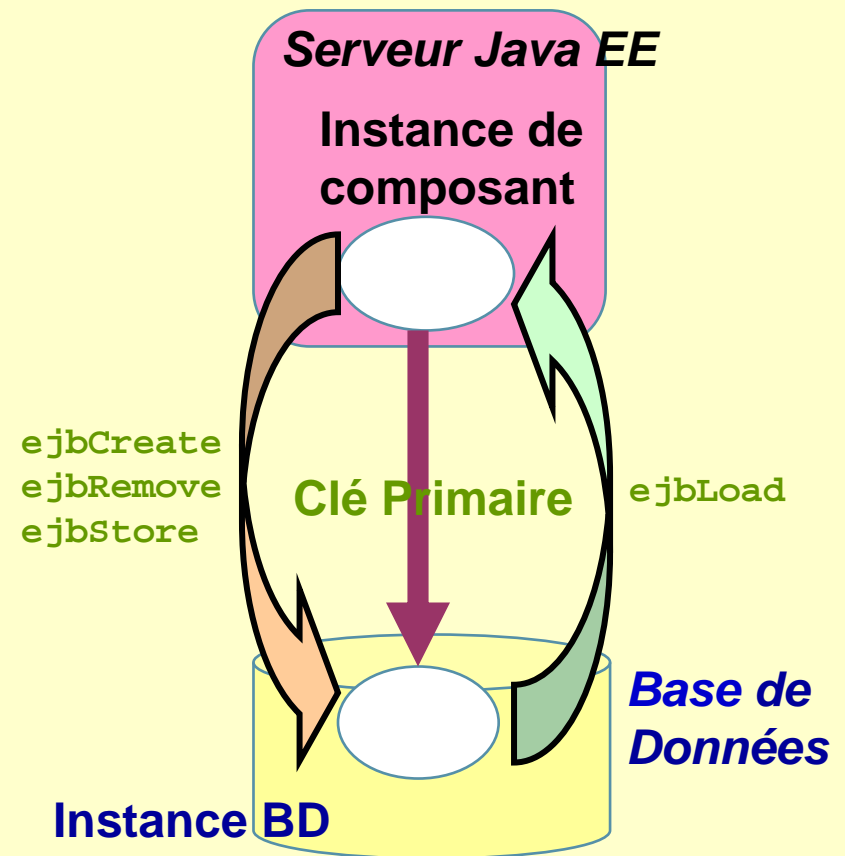
Composants Entity Beans (EJB2.1) ou POJOs en EJB3.0

- **Composants métier représentant des données des bases d'informations de l'entreprise**

- **Propriété de persistance**
 - ◆ Gérée par le « container » (EJB2.1 / CMP 1.1 ou 2.0)
 - ◆ Gérée par l'application (EJB2.1 / BMP)
 - ◆ 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 persistentes**
 - ◆ Méthode *finder* définie dans une interface « home » en EJB2.1
 - ◆ Annotation *NamedQuery* en EJB3.0
 - ◆ Implémentées par une requête EJBQL (langage proche d'OQL)
 - ◆ Retourne une entité ou une collection d'entités
- **Champs persistents**



API du système de persistance

- `EntityManagerFactory` représente un espace de persistance (une base de données particulière)
- `EntityManager` représente un contexte d'exécution (CRUD operations)
- `Query` représente une requête EBJQL (définition et exécution)
- `EntityTransaction` représente la transaction associée à un contexte d'exécution

Programmation d'un composant persistant en EJB3.0 (1/2)

- **POJO avec annotation @Entity (javax.persistence.Entity)**
- **Entités rendues persistantes et détruites**
 - ◆ explicitement (action `persist`)
 - ◆ par attachement (liens de propagation configurables)

Programmation d'un composant persistant en EJB3.0 (2/2)

- **Deux modes de définition d'attributs persistants**
 - ◆ Mode "field" → simple variable de classe
 - ◆ Mode "property" → méthodes *getField* / *setField*
- **Types supportés pour les attributs**
 - ◆ Types primitifs (et leurs "wrappers"), String, Date, etc.
 - ◆ Références d'entité (relation ?-1)
 - ◆ Collections de références d'entité (relation ?-n)
- **Support de l'héritage**
- **Support des classes abstraites**
- **Support des classes incluses**

Exemple de programmation d'un composant entité en EJB2.1 (1/3)

```
interface Participant extends EJBLocalObject {  
    int  getNbInscriptions();  
    ...  
}
```

```
interface ParticipantHome extends EJBLocalHome {  
    Participant create(String nom, String coord) throws  
    CreateException;  
    Participant findByPrimaryKey(String nom) throws  
    FinderException;  
    java.util.Collection findSomeParticipants(int statut)  
    throws FinderException;  
}
```

```
// Les requêtes associées aux méthodes finder sont exprimées  
// dans le descripteur de déploiement
```

Exemple de programmation d'un composant entité en EJB2.1 (2/3)

```
abstract class ParticipantBean implements EntityBean {
    void setEntityContext(EntityContext c) {}
    void ejbActivate() {}
    void ejbPassivate() {}
    void ejbLoad() {}
    void ejbStore() {}
    void ejbRemove() {}
    abstract String getNom();
    abstract void setNom(String val);
    abstract String getCoordonnees();
    abstract void setCoordonnees(String val);
    abstract int getStatut();
    abstract void setStatut(int val);
    static final int PREINSCRIT = 1;
    static final int FACTURE = 2;
    static final int INSCRIT = 3;
```

} Définition de l'état
persistent :
nom, coordonnees, statut

Exemple de programmation d'un composant entité en EJB2.1 (3/3)

```
void ejbCreate(String nom, String coord) {  
    setNom(nom);  
    setCoordonnees(coord);  
    setStatut(PREINSCRIT);  
}
```

Implantation des méthodes
« local home »

```
void ejbPostCreate(String nom) {}
```

```
int getNbInscriptions() {  
    ...  
}  
}
```

Implantation des
méthodes « local »

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

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

```
@Entity
@NamedQuery(
    name="toutesLesEcoles",
    query="SELECT e 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;
    }
}
```

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>
```

Exemple de dépendance vers un composant persistant en EJB2.1

```
class InscriptionBean implements SessionBean {
    private Context ctx;
    private ParticipantHome partH;

    void ejbActivate() {}
    void ejbPassivate() {}
    ...

    void setSessionContext(SessionContext c) {
        ctx = c;
        partH = ctx.lookup("comp:env/ejb/ParticipantHome");
    }

    void enregistre(String nom) {
        Participant p = partH.create(nom); // création + sauveg. dans DB
    }
    ...
}
```

Exemple de dépendance vers un composant persistant en EJB3.0

```
@Remote(Inscription.class)
@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();
        ...
    }
    ..
}
```

Les requêtes en EJB3.0

- Requêtes ensemblistes
 - ◆ Support à base d'EJBQL étendu
- Requêtes nommées
 - ◆ Factorisation de définition
 - ◆ Pas de recompilation à chaque exécution

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

Les relations en EJB3.0

■ Définition de relations entre entités

- ◆ Cardinalité 1-1, 1-n, n-1, n-n
- ◆ Gestion optimisée par la base de donnée
- ◆ Possibilité de détruire de façon transitive (« cascade delete »)

Sans les relations...

```
Class Auteur {
```

```
...
```

```
String nom;
```

```
Collection<Livre> livres;
```

```
Class Livre {
```

```
...
```

```
String titre;
```

```
Auteur auteur;
```

id	nom
----	-----

Auteur table

id	Auteur_id	Auteur_livre
----	-----------	--------------

Livre table

Les colonnes Auteur_id et Auteur_livre
représentent les mêmes informations !!

Avec les relations...

```
Class Auteur {  
    ...  
    String nom;  
    Collection<Livre> livres;  
}
```

`@OneToMany(mappedBy="auteur")`

```
public Collection<Livre> getLivres()  
{...}
```

```
Class Livre {  
    ...  
    String titre;  
    Auteur auteur;  
}
```

`@ManyToOne`

```
@JoinColumn(name="Auteur_id")  
public Auteur getAuteur() {...}
```

id	nom	...
----	-----	-----

Auteur table

id	titre	Auteur_id
----	-------	-----------

Livre table

Donne le nom `Auteur_id`
à la colonne de jointure

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

```
import javax.persistence.Entity;
...
@Entity
@NamedQuery(
    name="tousLesParticipants",
    query="SELECT p FROM Participant p")
public class Participant {
    private long id;
    private String name;
    private Ecole ecole;
    ...
    public Participant() {}
    ...
}

@ManyToOne
@JoinColumn(name="Ecole_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)

```
@Entity
@NamedQuery(
    name="toutesLesEcoles",
    query="SELECT e FROM Ecole e")
public class Ecole {
    private long id;
    private Collection<Participant>
        participants;
    ...

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

@OneToMany(mappedBy="ecole")
public Collection<Participant>
    getParticipants() {
    return participants;
}

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

Composants EJB « orientés message »

■ Manipulation par le « container »

- ◆ Réaction sur réception de message
- ◆ Transactionnelle ou non

■ Lien avec une destination JMS

- ◆ Agissent comme des `MessageListener`
 - ❖ Pour une `Queue`
 - ❖ Pour un `Topic`

Exemple de programmation d'un composant orienté message en EJB2.1

```
class NouvelleInscription implements MessageDrivenBean, MessageListener {
    private InetAddress[] adrSecretariatICAR;
    private javax.mail.Session mailSession;
    void setMessageDrivenContext(MessageDrivenContext c) {}
    void ejbRemove() {}

    void ejbPostCreate() {
        mailSession =
            (Session)initialContext.lookup("java:comp/env/mail/MailSession");
        String adr =
            (String)initialContext.lookup("java:comp/env/AdrSecretariat");
        adrSecretariatICAR = new InetAddress[] {new
            InetAddress(adr)};
    }
    void onMessage(Message msg) {
        javax.mail.Message message = new MimeMessage(mailSession);
        message.setRecipients(Message.RecipientType.TO, adrSecretariatICAR);
        message.setSubject("EffectuerInscription");
        message.setContent(((TextMessage) msg).getText(), "text/plain");
        javax.mail.Transport.send(message);
    }
}
```

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

```
@MessageDriven(mappedName="java:/TopicIcar06")
class NouvelleInscriptionBean implements javax.jms.MessageListener {

private String ecole = "Ecole thematique";
@Resource private String secretaireEcole = "ecole.secretaire@monorg.fr";
@Resource private InetAddress[] destinataires;

@PostConstruct
void init() {
    destinataires = new InetAddress[]{new
        InetAddress(secretaireEcole)};
}
void onMessage(Message m) {
    javax.mail.Message mailmess = new MimeMessage(session);
    mailmess.setRecipients(Message.RecipientType.TO, destinataire);
    mailmess.setSubject("Nouvelle inscription - " + ecole);
    mailmess.setContent("Nom du participant : " +
        (MapMessage)m.getString("NomParticipant"), "text/plain");
    Transport.send(mailmess);
}
```

Exemple d'une publication de message en EJB3.0

```
Resource(mappedName="java:/TopicIcar06")
```

```
private javax.jms.Topic topic;
```

```
TopicConnectionFactory tcf ...;
```

```
TopicConnection topicConnection = tcf.createTopicConnection();
```

```
TopicSession topicSession = topicConnection.createTopicSession(false,  
    Session.AUTO_ACKNOWLEDGE);
```

```
MapMessage mess = topicSession.createMapMessage();
```

```
mess.setString("NomParticipant", ip.nom);
```

```
TopicPublisher topicPublisher = topicSession.createPublisher(topic);
```

```
topicPublisher.publish(mess);
```

Programmation des clients lourds

Java EE

Composants « client »

- **Classes Java ayant accès à tous les services Java EE**
- **« Container » client Java EE**
 - ◆ Un client a un environnement (contexte)
 - ◆ Il peut référencer des composants et ressources Java EE
 - ◆ Il a accès aux transactions
(`javax.Transaction.UserTransaction`)
 - ◆ Il dispose de mécanismes de gestion de la sécurité

Mise en œuvre client lourd

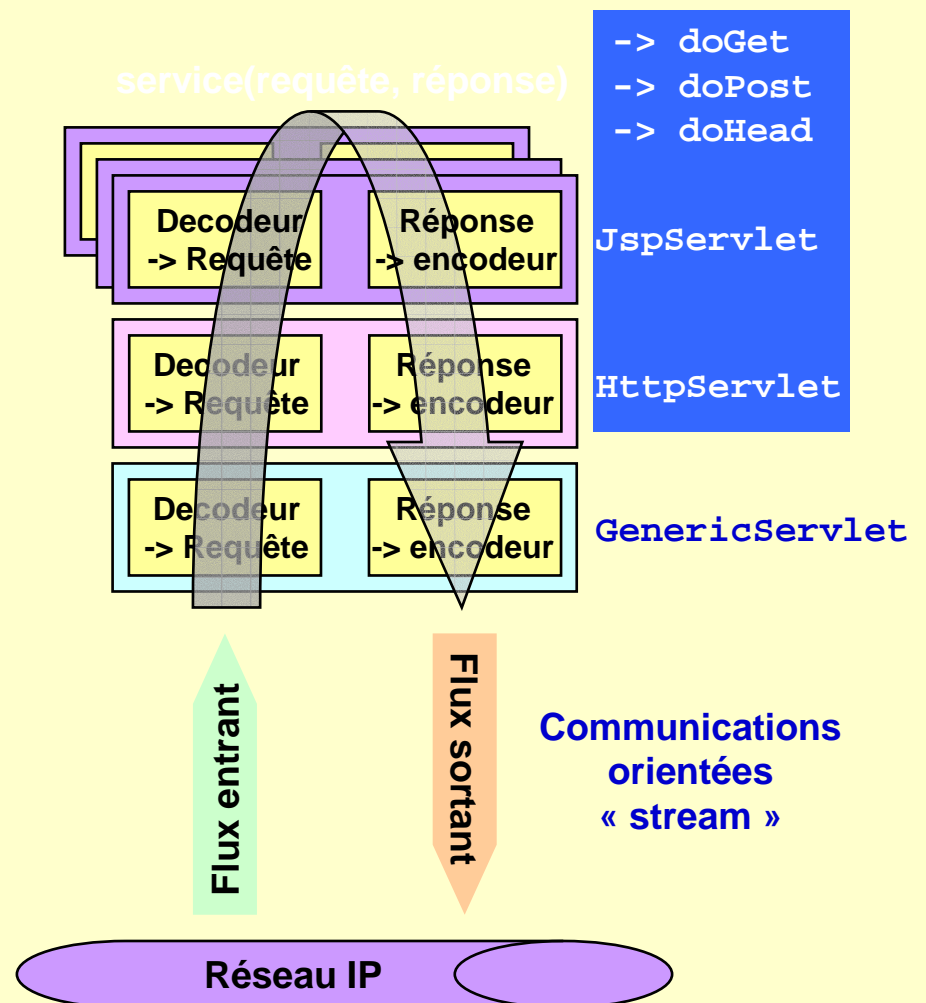
```
public final class Client { // EJB2.1
    ...
    Context ctxt = new InitialContext();
    InscriptionHome inscrHome = (InscriptionHome)
        PortableRemoteObject.narrow(
            ctxt.lookup("java:comp/env/ejb/InscriptionHome"), InscriptionHome.class)
    Inscription inscr = InscriptionHome.create();
    inscr.enregistre(..);
    ...
}
```

```
public final class Client { // EJB3.0
    ...
    @EJB (beanName="../ecole_them.jar#Inscription") // Instanciation automatique
    private Inscription inscr;
    inscr.enregistre(...);
    ...
}
```

Programmation des « Servlets »

Composants « Servlet »

- Composants protocolaires de Java EE (principalement utilisé pour présentation Web)
- Traitement d'interactions réseau de type « requête/réponse »
- Architecture de gestion d'une pile protocolaire
- Encapsulation des flux entrant/sortant par un appel procédural (`service`)
- Hypothèse de base = réseau IP
- Composition de protocole par héritage



Service offert par le Container Web

■ Gestion de `HttpServlet`

- ◆ Equivalent des « `cgi` » mais en Java
- ◆ Activation de « `thread` » plutôt que de « `process` »
- ◆ Génération de contenu Web dynamique

■ JSP (JavaServer Pages)

- ◆ Mélange de HTML et de code java
- ◆ Pré-compilation en Servlet au déploiement de web-app (ensemble de Servlet)

■ Autres technologies de production de contenu Web dynamique

- ◆ Pages « `Enhydra/XMLC` » : pas de mélange HTML / Java !!
- ◆ Templates « `Velocity` »

Déploiement d'application Web (web-app)

■ web-app

- ◆ ensemble de `Servlet`
- ◆ packagés dans un fichier « `jar` » (e.g., `webapp.war`)
- ◆ décrits par un fichier « `web.xml` »

■ Définition des propriétés (`env-entry`) et des dépendances (`resource-ref`, `resource-env-ref`, `ejb-ref`) au niveau de la web-app

-> pas au niveau de la `Servlet` !!!

■ Accès à l'environnement dans le code de la `Servlet` par "`java:comp/env/...`"

« Servlet » en action

■ Page Web (traitement par doPost ou doGet selon la méthode)

```
<form method="get" action="infos">
  <input type="text"
    name="nom"/>
  <input type="submit"/>
</form>
<a href="infos?nom=Dupont">
Mr Dupont</a>
```

■ Descripteur web.xml

```
<servlet>
  <servlet-name>s_infos</...>
  <servlet-class>ServletInfos</...>
</servlet>
```

■ Classe ServletInfos

```
doGet(HttpServletRequest rq,
  HttpServletResponse rs) {
  String nom =
    rq.getParameter("nom");
  rs.setContentType("text/html");
  PrintWriter pr = rs.getWriter();
  pr.println("html");
  ...
}
```

```
<servlet-mapping>
  <servlet-name>s_infos</...>
  <url-pattern>/infos</...>
</servlet-mapping>
```

Gestion de sessions

■ Identifier un utilisateur

- ◆ Géré par le client
- ◆ Transmission de l'identité à chaque requête

■ Les méthodes

- ◆ Cookies (méthode par défaut)
- ◆ Réécriture d'URL et champs HIDDEN

■ Classe `javax.servlet.http.HttpSession`

`HttpSession getSession(boolean create)`
dans `HttpServletRequest`

Classe HttpSession

- Lire et mémoriser des données

Object getAttribute(String nom)

void setAttribute(String nom, Object val)

- Timeout d'inactivité

int getMaxInactiveInterval()

void setMaxInactiveInterval(int sec)

- Gestion par réécriture d'URL (sans cookie)

String getRequestedSessionId()
dans HttpServletRequest

String encodeURL(String url)
dans HttpServletResponse

Utilisation des « Cookies »

- Classe `javax.servlet.http.Cookie`

- Envoyer un « Cookie »

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws
    ServletException {
    Cookie c = new Cookie("icar", "Dupond03");
    c.setMaxAge(365*24*60*60); // valable un an
    response.addCookie(c);
    // ...
}
```

- Lire les « Cookies » d'une requête

```
public Cookie[] getCookies() dans HttpServletRequest
```

Utilisation de « Cookies »

■ Cookie = {attribut}

- ◆ Obligatoires : Name, Value
- ◆ Optionnels : Domain, Path, MaxAge (sec), Secure (si transmis par SSL)

■ Transport par HTTP ou HTTPS

- ◆ Dans les requêtes et les réponses
- ◆ {(nom,valeur)} dans l'en-tête

■ Mémorisé par le navigateur de façon permanente

■ Renvoyé par le navigateur

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 programmatic (JTA API)

Gestion programmatique des transactions

- Utilisation de `javax.transaction.UserTransaction`
- Contrôle de la transaction (timeout, « rollbackonly »)

```
UserTransaction utx = (UserTransaction)
    new InitalContext().lookup("javax.transaction.UserTransaction");
utx.begin();

...
utx.commit(); // ou utx.rollback();
```

Exemple de gestion transactionnelle programmative

```
public final class Client {           // EJB2.1
    ...
    Context ctxt = new InitialContext();
    InscriptionHome inscrHome = ...;
    UserTransaction utx = ctxt.lookup("javax.transaction.UserTransaction");
    utx.begin();
    Inscription inscr = InscriptionHome.create();
    inscr.enregistre(..);
    ...
}
```

```
public final class Client {           // EJB3.0
    ...
    @EJB (beanName="../ecole_them.jar#Inscription") Inscription inscr;
    UserTransaction utx = ctxt.lookup("javax.transaction.UserTransaction");
    utx.begin();
    inscr.enregistre(...);
    ...
}
```

Gestion déclarative des transactions (« container-managed »)

■ Méthodes transactionnelles

■ Descripteur de déploiement / Annotations

- ◆ @TransactionAttribute(TransactionAttributeType.REQUIRED)

■ 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();
        }
        ...
    }
...

```

Gestion de l'authentification et des autorisations

■ Notions de base

- ◆ "principal"= identifiant (utilisateur)
- ◆ "credential"="principal" authentifié
- ◆ "role"=profil d'autorisation

■ Authentification

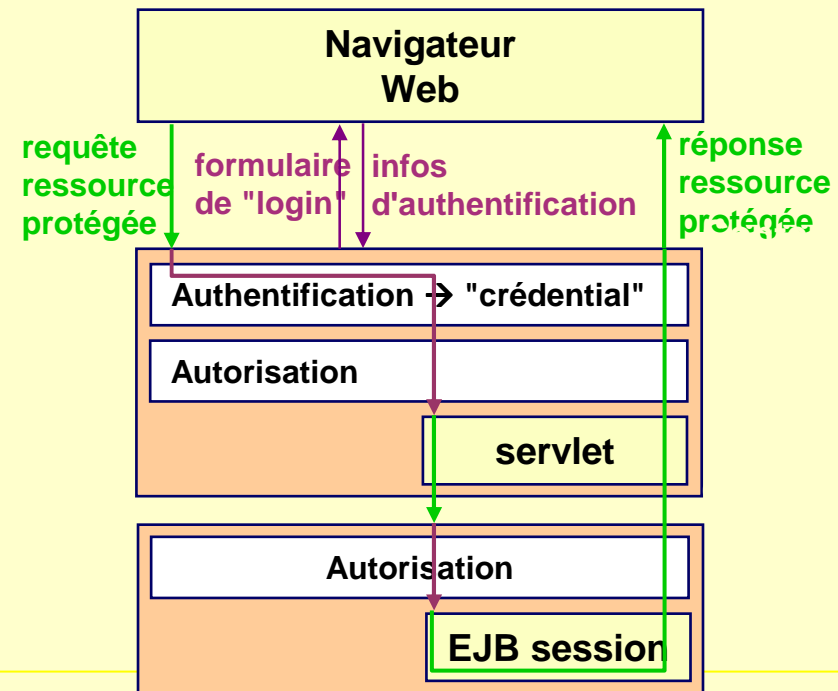
- ◆ Valide un "principal" et associe un "credential" à sa session
- ◆ "credential" passé au système d'autorisation

■ Autorisation

- ◆ Vérification de droits d'utilisation ou d'accès
- ◆ Basé sur les "roles"
- ◆ Règle : accès autorisé si un "credential" est dans un "role" autorisant l'accès à la ressource

■ Gestion de la sécurité

- ◆ Méta-information (pas terrible, minimal)
- ◆ De façon programmatique)



Principes de Sécurité

- Authentification et autorisation peuvent être gérés au niveau de chaque tiers (Web, EJB, etc), et peuvent être déclaratifs / programmatiques.
- Web tiers: indiquer au servlet container quelles ressources doivent être sécurisées et comment

web.xml :

```
<login-config>
    <auth-method>BASIC </auth-method>
    <realm-name>IcarRealm </realm-name>
</login-config>
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            Icar Administrative Component
        </web-resource-name>
        <url-pattern>/admin/* </url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>ADMIN</role-name>
    </auth-constraint>
</security-constraint>
```

Exemple de gestion transactionnelle déclarative

```
...
@DeclareRoles("BUYER", "ADMIN")
@Stateless
...
public class InscriptionBean implements Inscription {

    @RolesAllowed("ADMIN")
    public void desenregistre(String nom) {
        ...
    }
    @PermitAll
    public void enregistre(String nom) {
        ...
    }
}
```

Java EE avancé

■ Intercepteurs applicatifs

- ◆ Logging
- ◆ Auditing

Conclusion

■ 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
- ◆ Quelques extensions aux EJB : échéanciers (« batchs »)

■ Problèmes de maturité

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

■ La solution industrielle la plus aboutie

- ◆ Niveaux d'indépendance
- ◆ Spectre fonctionnel très large

Annexes

Descripteur du composant session "Inscription" en EJB2.1

```
<ejb-jar>
| <description>Sessions ICAR</description>
| <display-name>Sessions ICAR</display-name>
| <enterprise-beans>
| | <session>
| | | <ejb-name>Inscription</ejb-name>
| | | <home>org.icar.api.InscriptionHome
| | | </home>
| | | <remote>org.icar.api.Inscription
| | | </remote>
| | | <ejb-class>org.icar.lib.InscriptionBean
| | | </ejb-class>
| | | <session-type>Stateless</session-type>
| | | <transaction-type>Container
| | | </transaction-type>
| | | <ejb-ref>
| | | | <description>Entité
| | | | Participant</description>
| | | | <ejb-ref-name>ejb/Participant
| | | | </ejb-ref-name>
| | | | <ejb-ref-type>Entity</ejb-ref-type>
| | | | <home>org.icar.api.ParticipantHome
| | | | </home>
| | | | <remote>org.icar.api.Participant
| | | | </remote>
| | | | <ejb-link>donneesIcar.jar#Participant
| | | | </ejb-link>
| | | </ejb-ref>
| | </session>
| | <resource-ref>
| | | <resource-ref-name>jms/ConnFactory
| | | </resource-ref-name>
| | | <res-type>javax.jms.TopicConnectionFactory
| | | </res-type>
| | | <res-auth>Container</res-auth>
| | </resource-ref>
| | <resource-env-ref>
| | | <resource-env-ref-name>jms/InscriptionTopic
| | | </resource-env-ref-name>
| | | <resource-env-ref-type>javax.jms.Topic
| | | </resource-env-ref-type>
| | </resource-env-ref>
| </session>
| </enterprise-beans>
| <assembly-descriptor>
| | <container-transaction>
| | | <method>
| | | | <ejb-name>Inscription</ejb-name>
| | | | <method-name>enregistre</method-name>
| | | </method>
| | | <trans-attribute>Required</trans-attribute>
| | </container-transaction>
| </assembly-descriptor>
</ejb-jar>
```

Description du composant entité « Participant » en EJB2.1

```
<ejb-jar>
| <description>Entites ICAR</description>
| <display-name>Entites ICAR</display-name>
| <enterprise-beans>
| | <entity>
| | | <ejb-name>Participant</ejb-name>
| | | <local-home>org.icar.api.ParticipantHome
| | | </local-home>
| | | <local>org.icar.api.Participant
| | | </local>
| | | <ejb-class>org.icar.lib.ParticipantBean
| | | </ejb-class>
| | | <persistence-type>Container
| | | </persistence-type>
| | | <primary-key-class>java.lang.String
| | | </primary-key-class>
| | | <reentrant>True<reentrant>
| | | <cmp-version>2.x<cmp-version>
| | | <abstract-schema-name>participantsIcar
| | | </abstract-schema-name>
| | | <cmp-field>
| | | | <field-name>nom</field-name>
| | | </cmp-field>
| | | <cmp-field>
| | | | <field-name>coordonnees</field-name>
| | | </cmp-field>
| | | <cmp-field>
| | | | <field-name>statut</field-name>
| | | </cmp-field>
| | | <primkey-field>nom</primkey-field>
| | | <query>
| | | | <query-method>
| | | | | <method-name>findSomeParticipants
| | | | | </method-name>
| | | | | <method-params>
| | | | | | <method-param>int
| | | | | | </method-param>
| | | | | </method-params>
| | | | </query-method>
| | | | <ejb-ql>SELECT OBJECT(p)
| | | | | FROM participantsIcar p
| | | | | WHERE p.statut = ?1
| | | | </ejb-ql>
| | | </query>
| | </entity>
| </enterprise-beans>
</ejb-jar>
```

Description du composant orienté message « nouvelleInscription » en EJB2.1

```
<ejb-jar>
| <description>Sessions ICAR</description>
| <display-name>Sessions ICAR</display-name>
| <enterprise-beans>
| | <message-driven>
| | | <ejb-name>NouvelleInscription</ejb-
| | | name>
| | | <ejb-
| | | class>org.icar.lib.NouvelleInscriptionBean
| | | </ejb-class>
| | | <session-type>Stateful</session-type>
| | | <transaction-type>Container
| | | </transaction-type>
| | | <acknowledge-mode>Auto-acknowledge
| | | </acknowledge-mode>
| | | <message-driven-destination>
| | | | <subscription-durability>NonDurable
| | | | </subscription-durability>
| | | | <destination-type>javax.jms.Topic
| | | | </destination-type>
| | | </message-driven-destination>
| | | <env-entry>
| | | | <env-entry-name>adrSecretariat
| | | | </env-entry-name>
| | | | <env-entry-type>java.lang.String
| | | | </env-entry-type>
| | | </env-entry-type>
| | | <env-entry-value>icar03@inrialpes.fr
| | | </env-entry-value>
| | | </env-entry>
| | | <resource-ref>
| | | | <resource-ref-name>mail/MailSession
| | | | </resource-ref-name>
| | | | <res-type>javax.mail.Session
| | | | </res-type>
| | | | <res-auth>Container</res-auth>
| | | </resource-ref>
| | </message-driven>
| </enterprise-beans>
| <assembly-descriptor>
| | <container-transaction>
| | | <method>
| | | | <ejb-name>NouvelleInscription</ejb-name>
| | | | <method-name>*</method-name>
| | | </method>
| | | <trans-attribute>Required</trans-attribute>
| | </container-transaction>
| </assembly-descriptor>
</ejb-jar>
```