L'environnement Java EE

Principes, fonctions, utilisation

F. Boyer, Présentation pour la filière M2PGI 2009/2010,
Université Joseph Fourier

Sources

Cours de Pascal Déchamboux (FT R&D) pour l'école d'été ICAR 2003 à Autrans
Cours de Lionel Seinturier (LIFL) http://www2.lifl.fr/~seinturi/middleware/ejb3.pdf

Sommaire

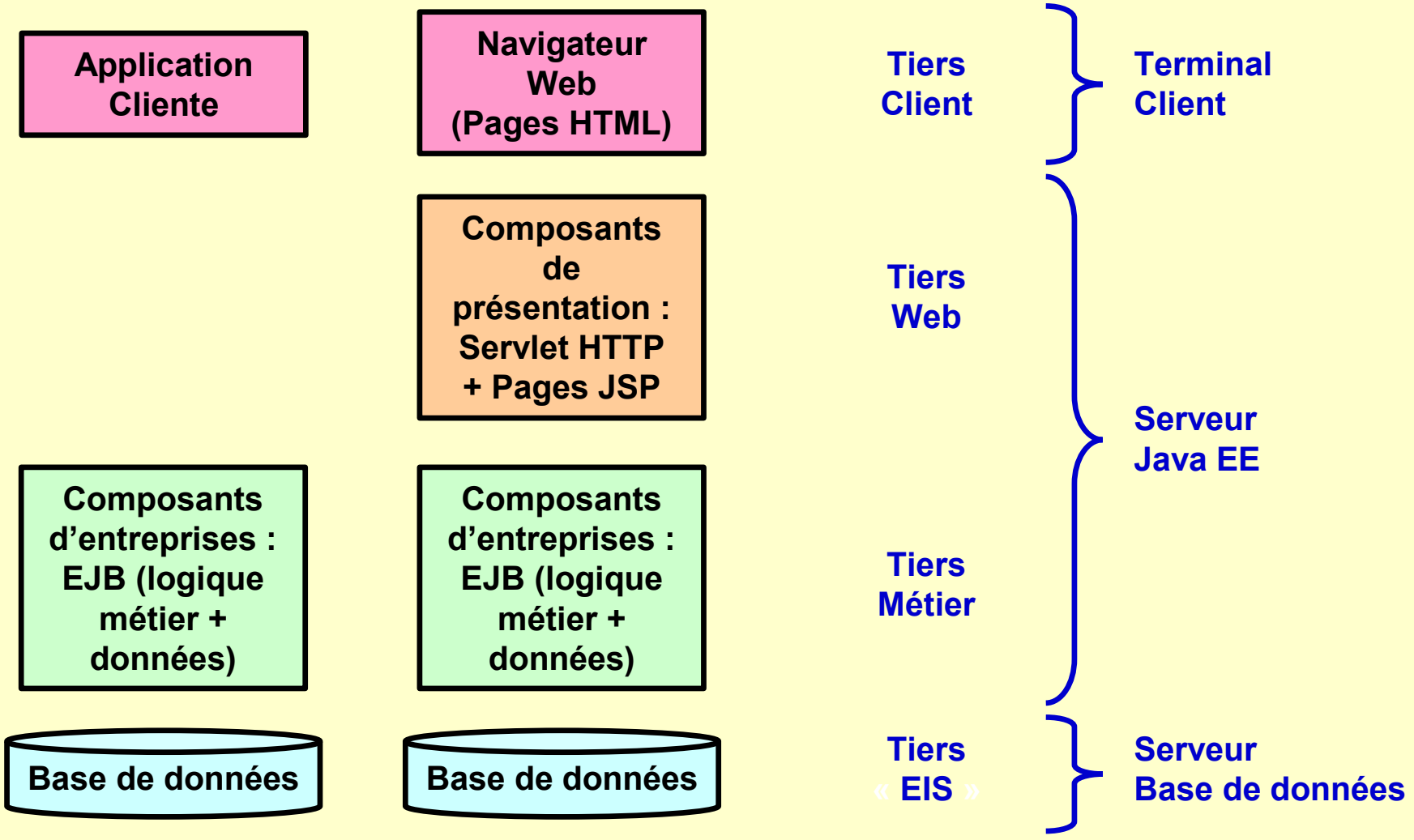
- Introduction
- Approche à composants
  - Packaging
  - Gestion externalisée des dépendances
- Modèles de programmation Java EE
  - Clients
  - Composants session
  - Composants entité
  - Composants mdb
- Fonctions avancées
  - Timers
  - Gestion des transactions
  - Gestion de la sécurité
- Conclusion

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
- Support de QoS : transaction, sécurité
- Connexion standard à des systèmes d'information externes (accès au « legacy »)
- Architecture multi-tiers
  - Architecture client léger (basée « browser »)
  - Architecture client lourd (GUI avancées)

Architectures usuelles



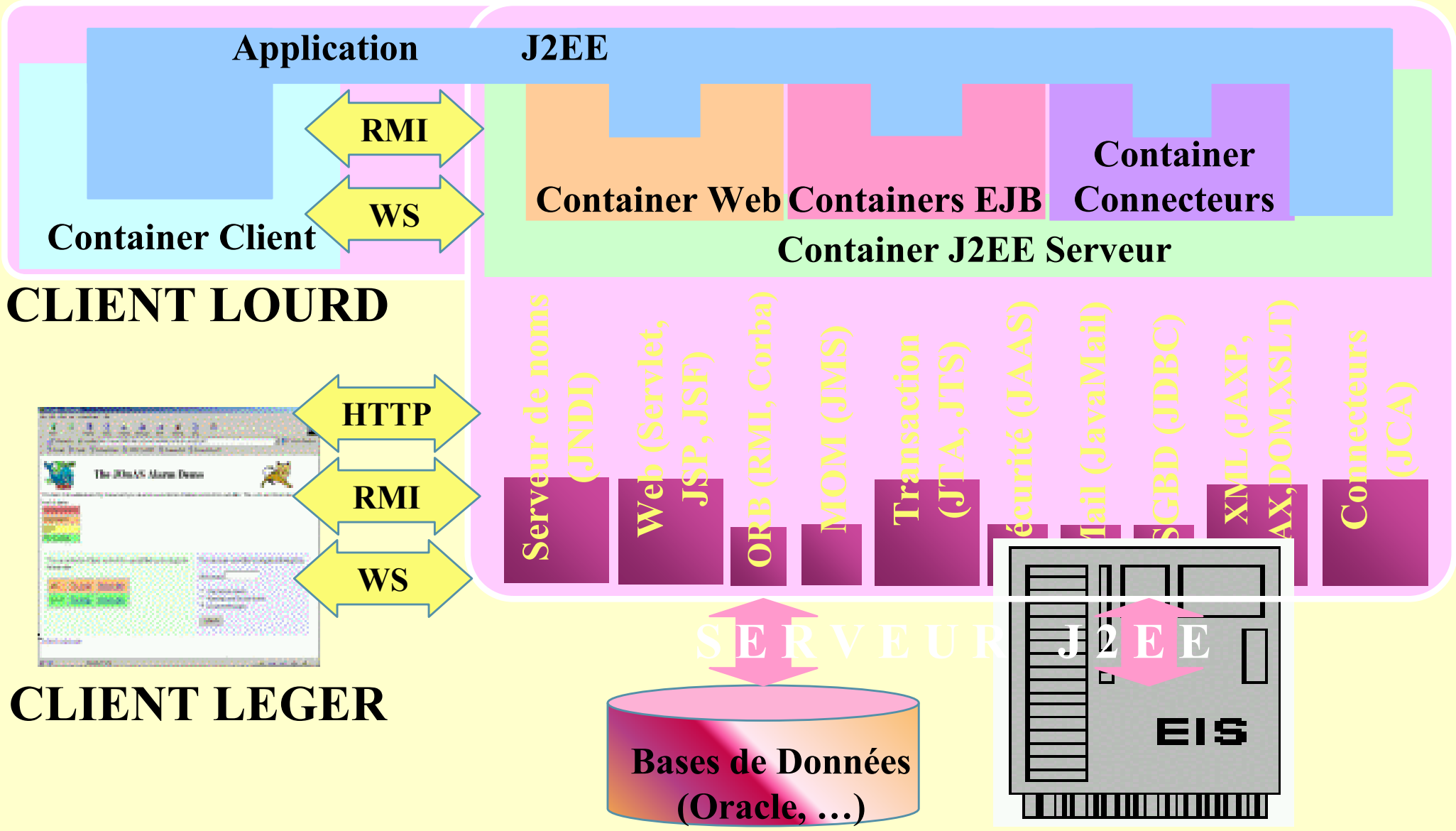
Java EE sous l'œil de Darwin !

- 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

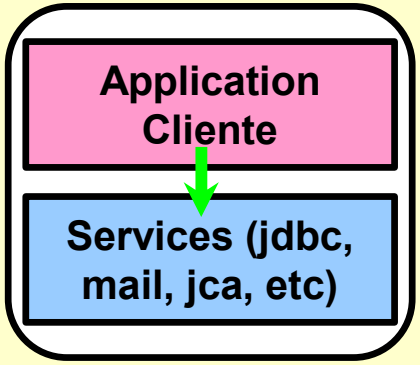
- Spécification
  - Programmation, assemblage, déploiement
  - Serveur et services
- Implantation de référence opérationnelle
- Suite de tests de conformance
  - Certification Sun
  - Accès au processus de certification payant (cher !!)
  - Lourd (> 20.000 tests)

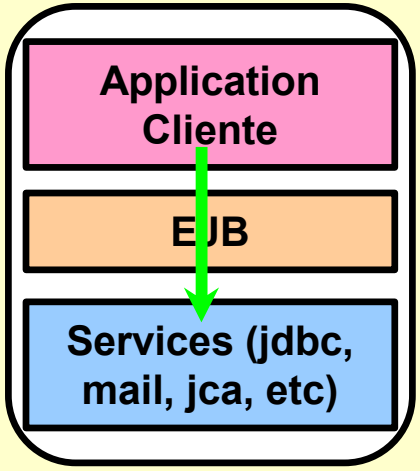
Vue générale de l'architecture Java EE



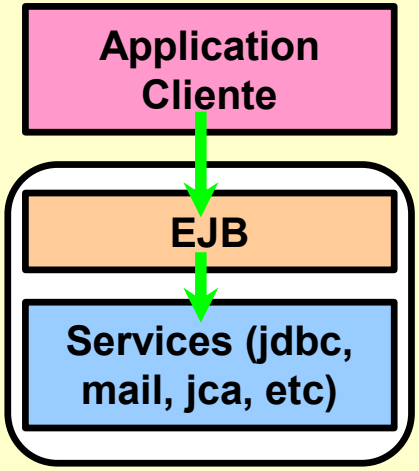
Configurations possibles d'un serveur Java EE : clients lourds

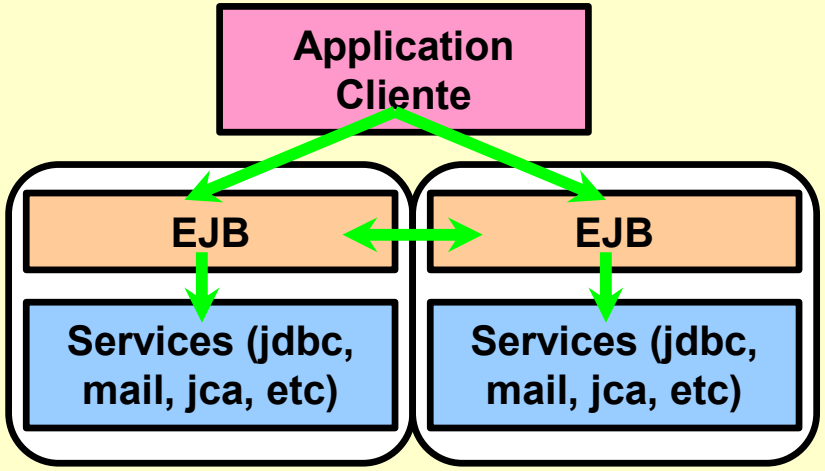
Application Java EE : Architecture centralisée





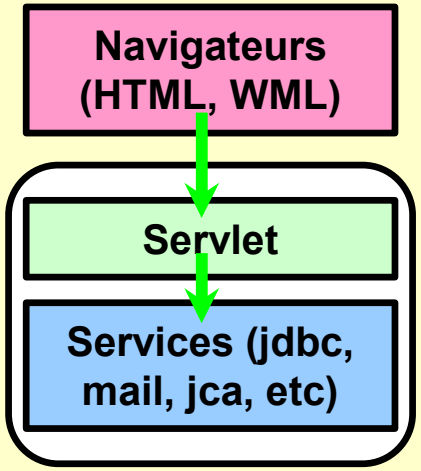
Application Java EE : Architecture client/serveur

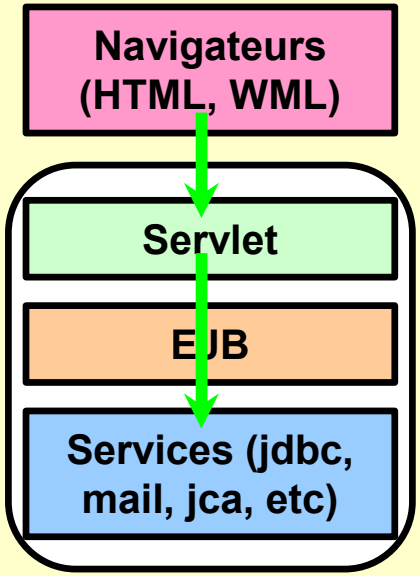




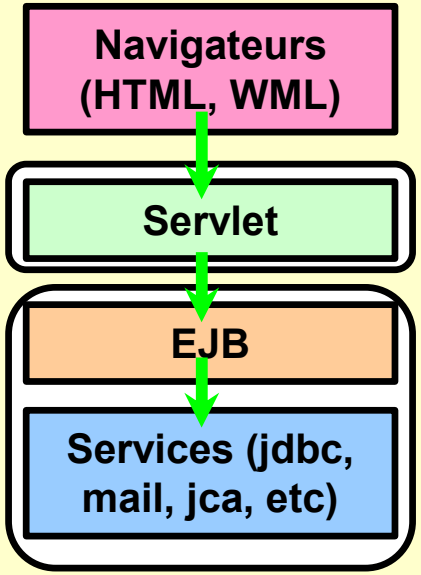
Configurations possibles d'un serveur Java EE : clients Web

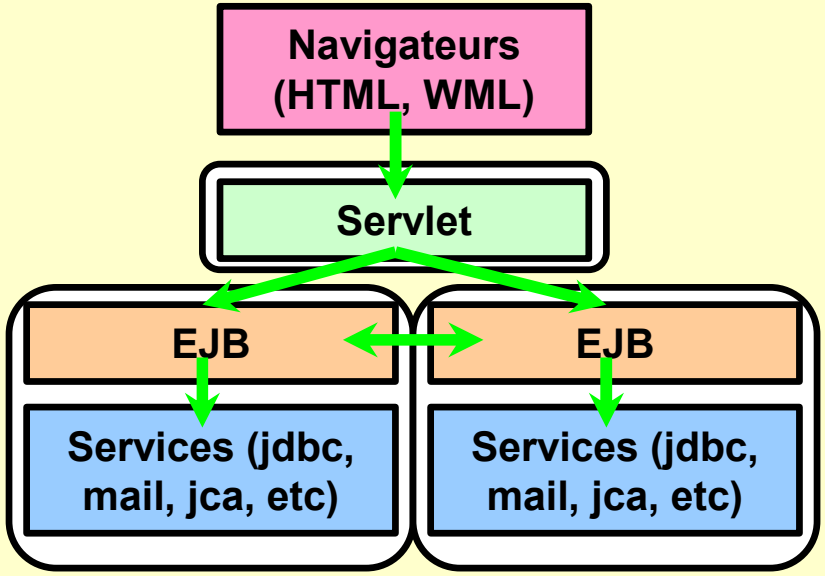
Application Java EE : Architecture Web / Serveur centralisé





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 Entreprise Server, Macromedia / Jrun, SAP Web Application Server, Iona / Orbix E2A
- Offre « open source »
  - JBoss (Red Hat)
  - JOnAS (ObjectWeb)
  - GlassFish (Sun)
  - …

Vers Java EE 5

- **Simplification du développement**
  - Relâchement des contraintes / programmation des composants
  - Utilisation des annotations amenées par Java SE 5
- **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!)

Approche à composants

Définition

Packaging

Gestion externalisée des dépendances

- Inversion de contrôle
- Conteneurs
- Définition et programmation des composants

Notion de composant

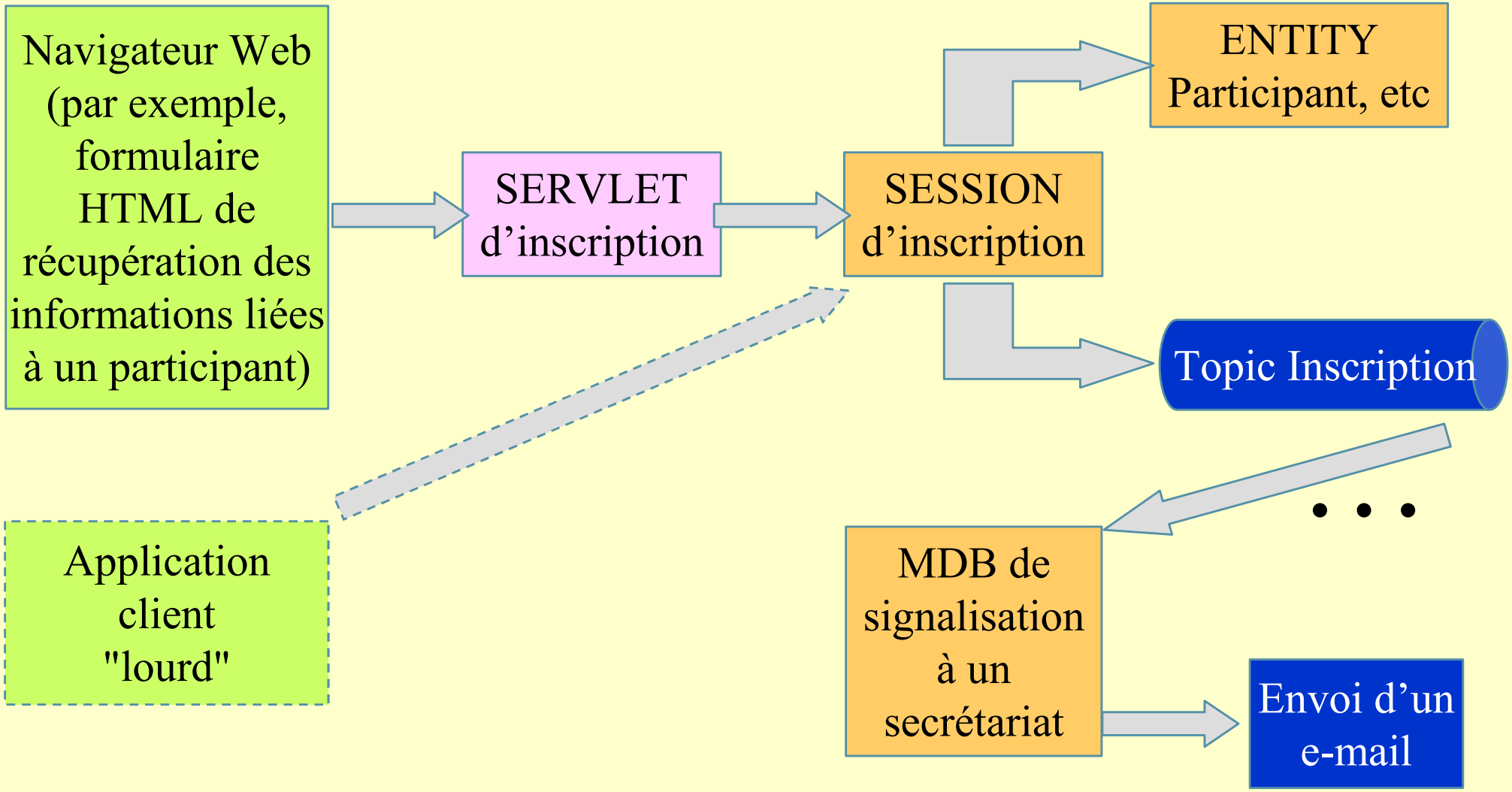
Szyperski 2002

A component is a unit of composition with contractually specified interfaces and explicit dependancies only. A software component can be deployed independantly and is subject to composition by third parties.

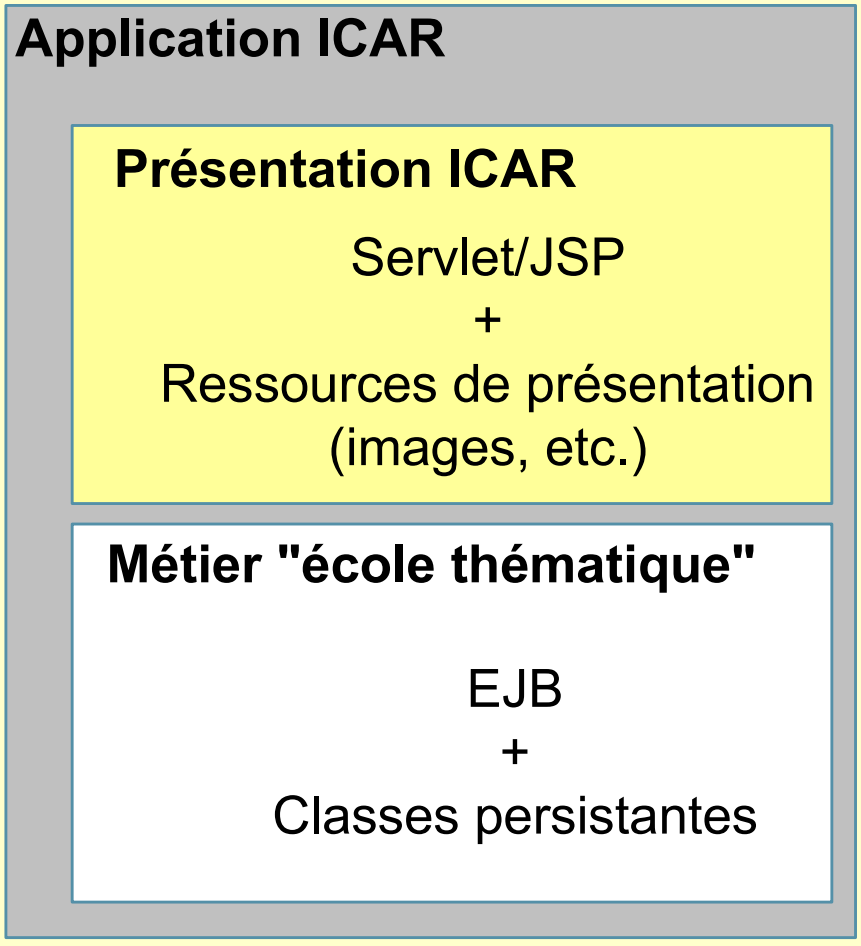
Composants Java EE

- **Composants applicatifs (code métier)**
  - Potentiellement répartis, transactionnels et sécurisés
- **Différents profils**
  - Session Beans (EJB)
    - Instances dédiées à un contexte d'interaction client particulier
    - Avec / sans état
  - Message Driven Bean (EJB)
    - Instances réagissant à l'arrivée de messages asynchrones
  - Entity Beans
    - Instances partagées représentant les données de l'entreprise
  - Servlets
    - Composants de présentation

Exemple d'application: Inscription à une conférence



Packaging de l'application



- **Application ICAR**
  - Application JEE
  - Packaging : icar.ear
  - Contient 2 archives
- **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
    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 persistance) :

```
META-INF/
   MANIFEST.MF
   ejb-jar.xml
   persistence.xml
org/
   ecole/
      api/
         *.class
      lib/
         *Bean.class
      persistance/
         *.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.IncriptionBean</ejb-class>
       <session-type>Stateless</session-type>
       <transaction-type>Container</transaction-
     type>
       ...
     </session>
  </enterprise-beans>
</ejb-jar>
```

Gestion externalisée des dépendances

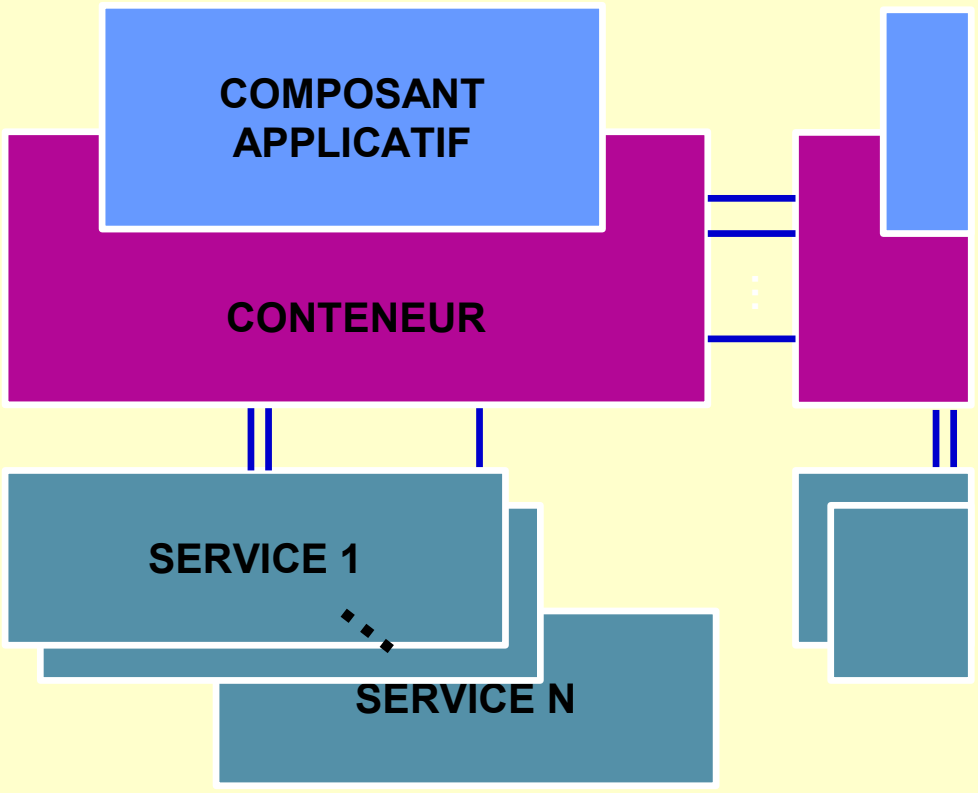
- Dépendances d'un composant (client, servlet vers :
  - un autre EJB
    - Exemple: Servlet d'inscription → EJB Inscription
  - les services / ressources de l'infrastructure
    - Exemple: service de transaction
  - les propriétés de l'application
    - Exemple: NbMaxParticipants

Inversion de contrôle

- Principe de mise en œuvre des dépendances externalisées
  - Les dépendances ne sont pas gérées par les composants
  - Elles sont gérées par l'infrastructure J2EE, au travers des conteneurs
- Recherche d'une indépendance maximale du code applicatif
  - Vision toutefois non systématique, car les définitions de dépendances sont spécifiques au type de liaison (par exemple, ejb, ressource, etc.)

Inversion de contrôle (2)

- 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
  - Eviter l'utilisation de code « technique » dans le code applicatif



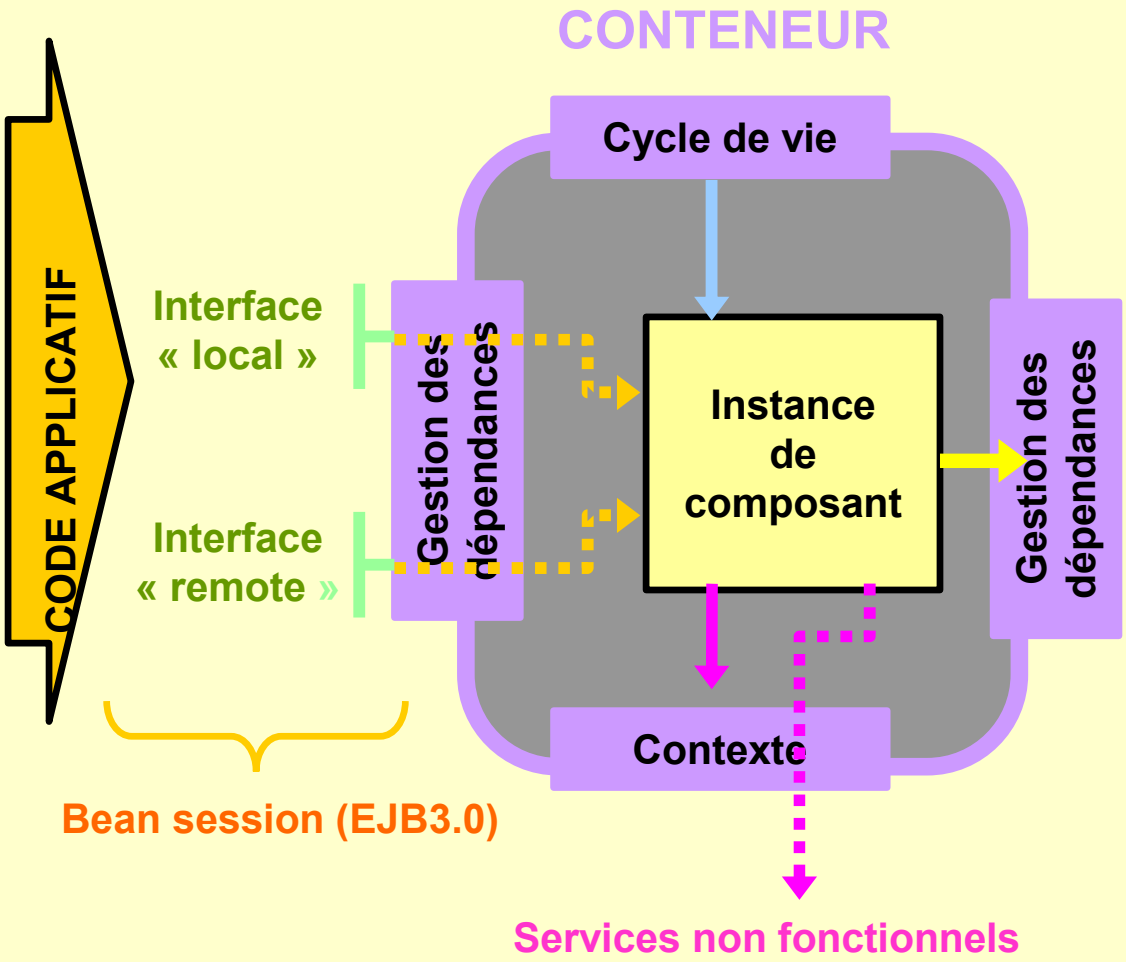
Rôle d'un conteneur

- **Gestion des dépendances**
  - Vers des EJB
  - Vers des services de l'infrastructure
  - Vers des propriétés
- **Gestion du cycle de vie des composants**
  - Démarrage, arrêt
  - Activation, passivation

Principe de mise en oeuvre des conteneurs

- **Injection**
  - Injection des propriétés
  - Injection des dépendances entre composants
  - Injection des dépendances entre composants et services de l'infrastructure
- **Interception**
  - Interception des appels vers/depuis les composants
    - Pour gérer les appels aux services non fonctionnels
- **Callbacks applicatifs**
  - Spécialisation de la gestion des dépendances et du cycle de vie (création, destruction, activation, passivation)
- **Utilisation d'un contexte propre à chaque composant (données de contrôle)**
  - Sécurité, Transactions, Contexte de Nommage

Principe du conteneur Java EE



- *EJB2.1: gestion des dépendances faite par des intercepteurs*
- *EJB3.0: gestion des dépendances injectée dans les composants*

Principaux conteneurs Java EE

- Conteneur d'applications clientes (client "lourd")
- Conteneur d'applications serveurs (déployées dans un serveur Java EE)
  - Conteneur Web (de "servlet")
  - Conteneur d'EJB
- Conteneur de persistance
  - Indépendant de JEE
  - Intégré par le conteneur EJB
  - Basé sur les principes des composants (JPA)

Gestion externalisée des dépendances

- Programmation d'un composant client, servlet, EJB :
  - Code Java (POJO)
    - Interfaces d'accès aux instances
    - Implantation des interfaces
  - Méta-information
    - Définition des dépendances
    - Sous forme d'annotations
    - Surchargées par des descripteurs XML optionnels

Dépendances vers une propriété, un EJB ou une ressource

- Dépendance entre un nom local dans le code (ex: le nom d'une variable) et une propriété, un EJB ou une ressource
  - Dépendance définie par la méta-information
- Définition d'une dépendance
  - Vers une propriété
    - Donne la valeur de la propriété
  - Vers un composant du même ear
    - Désignation à l'aide du nom local du composant
    - Vers un composant extérieur ou vers une ressource
    - Désignation à l'aide d'un nom global JNDI (*mappedName*)
- Par défaut
  - Le nom local d'un composant est celui de son interface (sauf si attribut *name* défini)
  - Le nom global d'un composant est celui de son interface (sauf si attribut *mappedName* redéfini)

Définition de dépendances vers une propriété

```
// Code du composant
…
class InscriptionBean implements Inscription {
  @Resource private int maxPart = 120; // maximum de participants

<!- descripteur optionnel →
```

Définition de dépendances vers un EJB (principe)

```
// Code du composant (par exemple un servlet) contenant la
   dépendance
@EJB(name="../ecole_them.jar#InscriptionBean")
    private Inscription inscr;

// Ou si le composant référencé et le référençant sont dans le
   même package
@EJB private Inscription inscr;
```

Définition de dépendances vers une usine à connexions JDBC

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

Définition de dépendances vers une ressource JMS

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

@Resource(mappedName="java:/TopicInscriptions09")
                    javax.jms.Topic topicInscription;
```

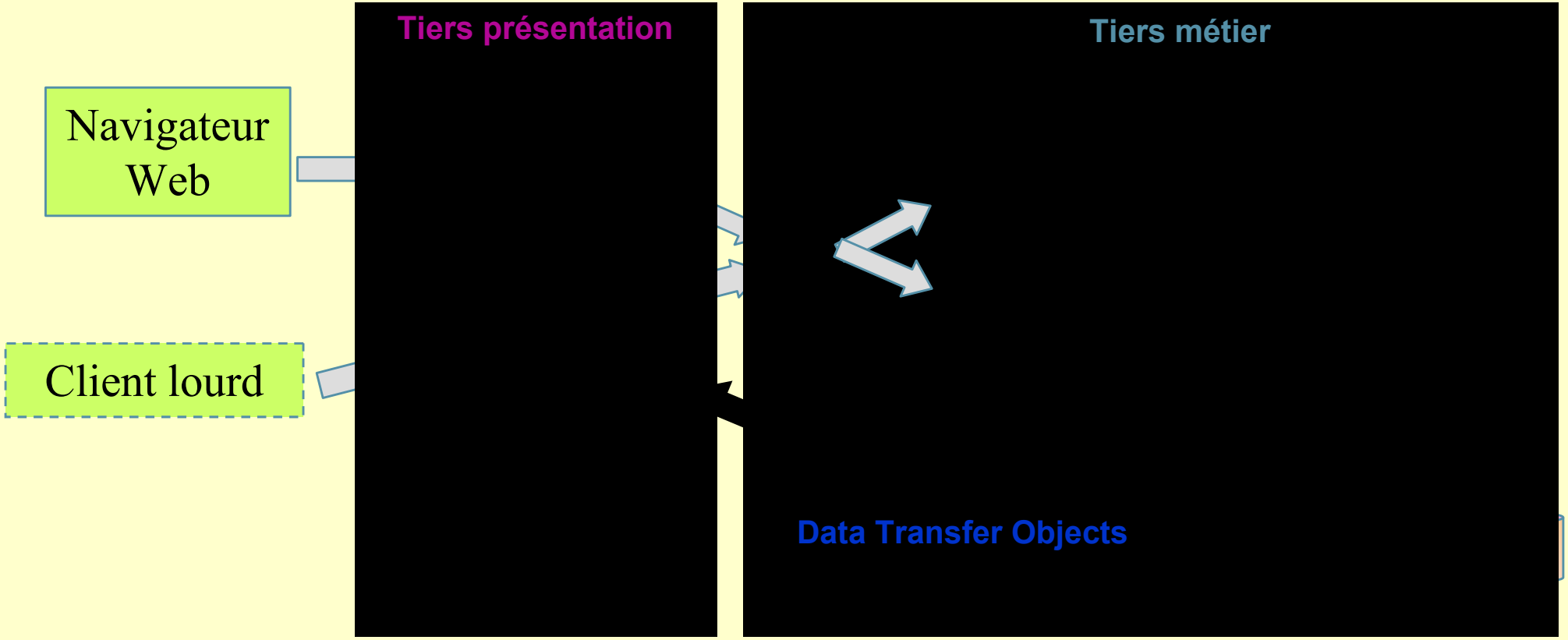
Accès au contexte d'un composant

```
// 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;
  }
…
}
```

Modèles de programmation des composants Java EE

- Composants EJB
  - Session
  - Orienté message
- Composants Entity

Une conception classique J2EE



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

Bean Session

- Stateful / Stateless
  - Annotation @Stateful / @Stateless (javax.ejb)
- Interface locale / distante / Web service
  - Annotation @Local / @Remote / @WebService
- Stateful associés de façon unique à un client particulier
  - Typiquement : caddie
  - @Remove définit la méthode de fin du bean (EJB3.0)
  - Sinon destruction automatique au bout d'un délai d'inactivité
- Manipulation par méthodes métier
- Comportement transactionnel spécifique à chaque méthode

Bean Session (interface Remote)

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

@Remote
 public interface Inscription {
        public void enregistre(String nom, …);
        public void confirme();
        public … infos(String nom);
 }
```

Bean Session (interface WebService)

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

// Only for stateless beans
@WebService (targetNameSpace=« urn:InscService »);
@Stateless
public class InscriptionBean implements Inscription {
    @WebMethod
    @WebResult(name=« UserId »)
    public void enregistre(
         @WebParam(name=« UserName ») {
         ...
}
..
```

Accès (ex): http://host:8080/InscService/InscriptionBean/enregistre?UserName=Paul

Bean Session (Implantation)

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

@Stateless
public class InscriptionBean implements Inscription {

   public void enregistre(String nom) {...}
   @Remove
   public void confirme() {...}
   ...
}
..
```

Dépendance vers un bean session depuis un composant local (même ear)

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

public class InscriptionServlet extends HttpServlet {

    @EJB
    private Inscription inscription;

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

Dépendance vers un bean session depuis un client distant

- « 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`)

```
import javax.ejb.*;

public class ClientDistant {
        ...
         Context ctx = new javax.naming.InitialContext();
         Inscription inscription = (Inscription)
            ctx.lookup(Inscription.class.getName());
         inscription.enregistre(name);
         ...
}
...
```

Patrons de programmation des Beans Session

- **Bean Session Façade**
  - Présenter aux clients une unique interface (limite les dépendances des clients vers le tiers métier)
  - Le bean façade délègue les traitements
- **Data Transfer Objects (DTO)**
  - Limiter les dépendances entre les tiers métier et présentation
  - Clarifier le contrat entre ces deux tiers

Bean Session (DTO)

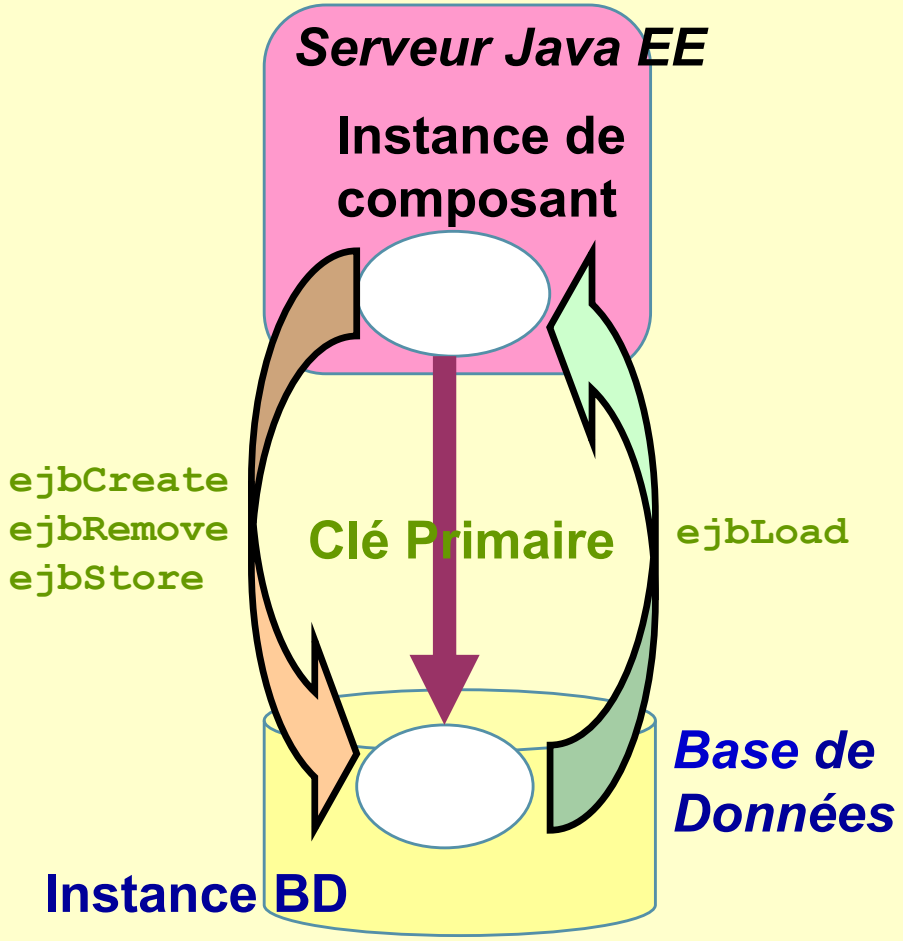
```
package ecole_them;
import javax.ejb.Remote;

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

class ParticipantDTO { // objet de transfert de données entre
    public String nom;  // tiers métier et client
    public Date date;
    ...
}
```

Entity Beans

- Composants métier représentant des données persistantes, en base relationnelle
- Accès via jdbc
- Gestion du mapping objet-relationnel (standard JPA)

Mapping objet-relationnel

- **Notion de « clé primaire »**
  - Nom persistant qui désigne 1 instance BD
- **Sélection d'entités persistantes**
  - Requêtes (nommées ou dynamiques)
  - Implémentées en EJBQL (~OQL)
  - Retourne une entité ou une collection d'entités
- **Dépendances entre les données en base → injectées dans les beans entités**



Mapping avec la base de données

- 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=…)
- Deux modes de définition des colonnes des tables (donc des attributs persistants)
  - Mode "field" → annote les attributs
  - Mode "property" → annote les méthodes `getter/setter`
  - Par défaut, le nom de la colonne est le nom de l'attribut
  - Sauf si annotation @Column(name=…)

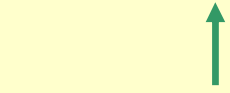
Mapping avec la base de données

- 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=".. ", referencedColumnName=" .. ")
- Table de jointure
  - @JoinTable



Nom de l'attribut dans la classe

Fichier de définition des unités de persistance : "persistence.xml"

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

API du système de persistance

- `EntityManager` représente le gestionnairedu mapping (CRUD operations)
  - Create, Read, Update, Delete
  - Select (renvoie une entité ou une List / Collection Java)
  - Attribut de type javax.persistence.EntityManager annoté par @PersistenceContext
- `Query` représente une requête EBJQL (définition et exécution)
  - Pré-compilée si nommée

Programmation d'un Entity Bean

- POJO avec annotations (javax.persistence.*)
  - Capacités d'héritage
- Annotations de classes
  - @Entity pour définir une classe correspondant à un bean entité
  - @NameQuery 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)

Programmation d'un Entity Bean

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

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)

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;
}
 …
```

Programmation d'un Entity Bean

```
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;
 }
..
```

Dépendance vers des Entity Beans

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

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

   Collection<Participant> participantsFinder() {
    Query q = em.createNamedQuery("tousLesParticipants");
    return q.getResultList();
   }

   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();
        ...
```

Dépendance vers des Entity Beans

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

     // recherche par la primary key
    Participant participantFinder(long participantId) {
     return em.find(Participant.class, Integer.valueOf(participantId));
    }

    ...
```

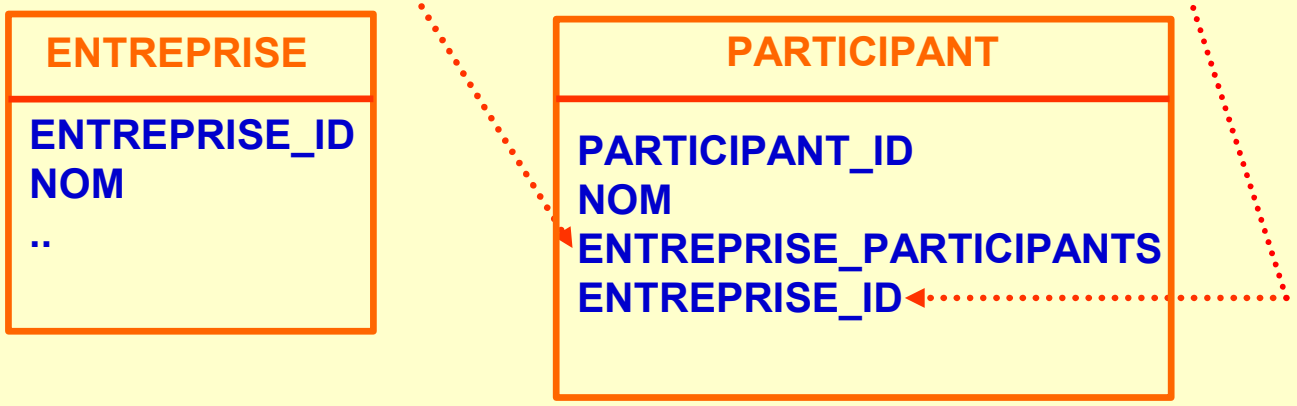
Les relations

- **Définition de relations entre composants persistants**
  - Gestion optimisée
  - Cardinalité 1-1, 1-n, n-1, n-n
    - @OneToOne, @OneToMany, @ManyToOne, @ManyToMany
  - Principe
    - Annote un champ d'une classe
    - Attribut *mappedBy* permet de désigner le champ qui définit la relation dans l'autre classe

Sans les relations…

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

```
import javax.persistence.Entity;
@Entity
public class Participant {
  long id;
  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 {
long id; ...

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

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

 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 {
  long id; ...
  Entreprise entreprise;

@ManyToOne
@JoinColumn(name="MonEntreprise_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 MONENTREPRISE_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"),
  inverseJoinColumns=
    @joinColumn(name="CI_ECOLE_ID",
    referencedColumnName="ECOLE_ID"))
 private Collection<Ecoles> ecoles;
...
```

ECOLE
ECOLE_ID
NOM

PARTICIPANT
PARTICIPANT_ID
NOM
MONENTREPRISE_ID

ECOLES_PARTICIPANTS
CI_PARTICIPANT_ID (foreign key)
CI_ECOLE_ID (foreign key)

Message Driven Beans (MDB)

- Bean sans état
- Réagit à des messages envoyés sur une destination JMS
- Permet de désynchroniser et fiabiliser l'échange avec le client
- Dépendance vers une destination JMS
  - Un MDB est agit comme un `MessageListener`
    - Pour une `Queue (1 récepteur pour un message)`
    - Pour un `Topic (n récepteurs pour un message)`
- `Base JMS`
  - `ConnectionFactory` pour créer des queues/topics
  - `Connection` pour créer une connexion vers une queue / un topic

Bean Message Driven

```
@MessageDriven(
activationConfig =
{ @ActivationConfigProperty(
    propertyName="destination", propertyValue="InscriptionsQueue"),
  @ActivationConfigProperty(
    propertyName="destinationType", propertyValue="javax.jms.Queue")
})
public class NouvelleInscriptionBean implements
   javax.jms.MessageListener{
@resource private String secretaireEcole = "ecole.secretaire@monorg.fr";


public void onMessage(Message m) { // toujours appelé par le conteneur
   javax.mail.Message mailmess = new MimeMessage(...);
   mailmess.setRecipients(secretaireEcole);
   mailmess.setSubject("Nouvelle inscription - " + ecole);
   mailmess.setContent(...)
   Transport.send(mailmess);
   }
   ...
```

Publication de message (depuis un client par exemple)

```
@Resource(mappedName="jms/ConnectionFactory")
 private static ConnectionFactory connectionFactory;

@Resource(mappedName=”jms/InscriptionQueue”)
 private static Queue queue;

        Connection connection = factory.createConnection();
         Session session = connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer= session.createProducer(queue);

        // on peut envoyer k messages
        Message message = session.createTextMessage("...");
          messageProducer.send(message);
        ...
}
```

Fonctionnalités avancées

- Timer Beans
- Callbacks applicatives
- Transactions
- Sécurité

Composants EJB Timer

- Pour les beans sans état seulement
- Resource TimerService permet de créer des *timers*
- Timer permet de déclencher des actions périodiquement

```
@Stateless
public class InscriptionBean implements Inscription, TimerSession {
@Resource  TimerService timerService;

public void createTimer(long delai) {
   Timer timer = TimerService.createTimer(delai, "");
}

@timeout public void envoiPubEcole(Timer timer){..}
...
```

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)
  - @afterBegin : après le démarrage d'une transaction, avant invocation de toute méthode métier
  - @beforeCompletion : juste avant la terminaison de la transaction
  - @afterCompletion : juste avant la terminaison de la transaction
- Entity bean
  - @ejbCreate
  - @ejbPostCreate
  - @ejbRemove
  - @ejbPostRemove
  - ...

Callbacks applicatives : intercepteurs

- Permet d'ajouter des traitements métier avant/après les méthodes des beans
  - @Interceptors : méthodes devant être interceptées
  - @AroundInvoke : méthodes d'interception
  - InvocationContext: contexte de l'interception
    - Méthode interceptée
    - Permet de continuer la méthode (proceed())

Gestion des transactions

- Applicables aux profils de composants
  - Session
  - Message Driven Bean
- Propriétés ACID
  - Atomicity, Consistency, Isolation, Durability
- Gestion déclarative ou programmative (JTA API)
- Utilisation de `javax.transaction.UserTransaction`
- Contrôle de la transaction `(timeout, « rollbackonly »)`

Exemple de gestion transactionnelle programmative (client lourd)

```
public final class Client {
  ...
  Inscription inscr;

  ...
   UserTransaction utx
    =ctxt.lookup("javax.transaction.UserTransaction");
   utx.begin();

   Context ctx = new javax.naming.InitialContext();
   Inscription inscription = (Inscription)
   ctx.lookup(Inscription.class.getName());
   inscription.enregistre(name);
   inscr.enregistre(…);
   ...

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

Gestion déclarative des transactions (« container-managed »)

- Méthodes transactionnelles définies dans les EJB (meilleure conception)
- Annotations de méthode @TransactionAttribute
  - « 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
- L'infrastructure gère la mise à jour transactionnelle des données persistantes, mais c'est au programmeur de gérer si nécessaire le rafraîchissement de l'état de ses EJB.

Exemple de gestion transactionnelle déclarative

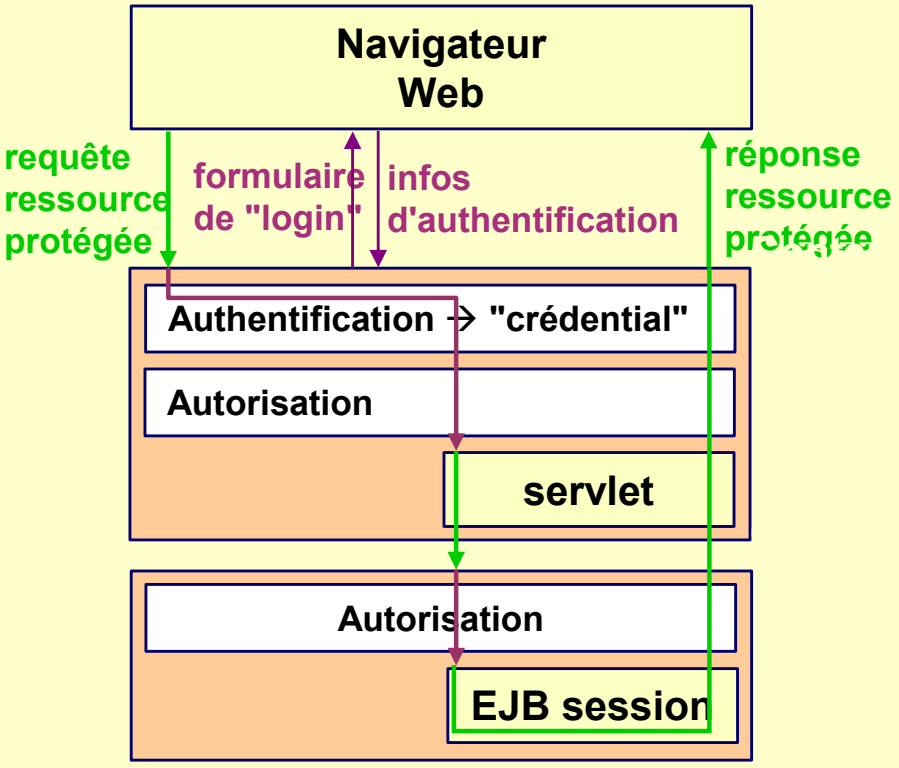
```
…
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class InscriptionBean implements Inscription {

    @Resource
    private SessionContext context;

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void enregistre(String nom) {
        ...
        } catch(EnregistrementException e) {
                context.setRollbackOnly();
        }
        ...
    }
...
```

Exemple de gestion transactionnelle déclarative

- **Pour les EJB Statefull**
  - Utiliser les callbacks *afterBegin(), beforeCompletion() et afterCompletion()* pour gérer l'état de l'EJB
  - beforeCompletion() est la dernière occasion de faire aborter la transaction, en positionnant la propriété setRollbackOnly
  - afterCompletion (boolean comitted) permet de rafraîchir l'état de l'EJB en cas de rollback, si l'on a sauvegardé cet état lors de l'appel à afterBegin().

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



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 resources doivent être sécurisées et comment

```
web.xml :
<login-config>
            <auth-method>BASIC </auth-method>
            <realm-name>InscriptionsRealm </realm-name>
</login-config>
 <security-constraint>
            <web-resource-collection>
               <web-resource-name> Inscription Admin </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 de sécurité déclarative

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

    @RolesAllowed(“ADMIN”)
    public void desenregistre(String nom) {
        ...
    }
    @PermitAll
    public void enregistre(String nom) {
        ...
    }

}
```

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-existance avec JDO)
  - Modèle de composants (manque d'homogénéité, problèmes avec gestion de l'héritage)

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/