

L'environnement Java EE :

Principes, états des lieux et évolutions

Guillaume Sauthier (Bull SAS)

Guillaume.Sauthier@Bull.Net

<http://jonas.ow2.org>



Plan

- **Introduction à l'environnement Java EE**
- **Composants Java EE**
- **Services fournis par l'environnement**
 - ◆ **Répartition**
 - ◆ **Transactions**
 - ◆ **Sécurité**
- **Modèles de programmation Java EE**
 - ◆ **Client/Web/EJB**
 - ◆ **Persistance**

Introduction

Objectifs de Java EE 1/2

- **Infrastructure “serveur” pour le support d'applications d'entreprise**
 - ◆ E-commerce (B2B & B2C)
 - ◆ Systèmes d'informations
 - ◆ Plates-formes de services (Audio-visuel, telco, ...)

- **Architecture multi-tiers**
 - ◆ Client léger (basée sur les navigateurs)
 - ❖ Web, WAP, vocale (synthèse, reco)
 - ❖ Artefacts XML (XHTML, WML, VoiceXML)
 - ◆ Clients lourds (GUI avancées)
 - ◆ SOA (application réparties)

Objectifs de Java EE 2/2

■ QoS (Qualité de service)

- ◆ Transactions
- ◆ Sécurité (authentification, autorisation, confidentialité)
- ◆ Gestion des ressources

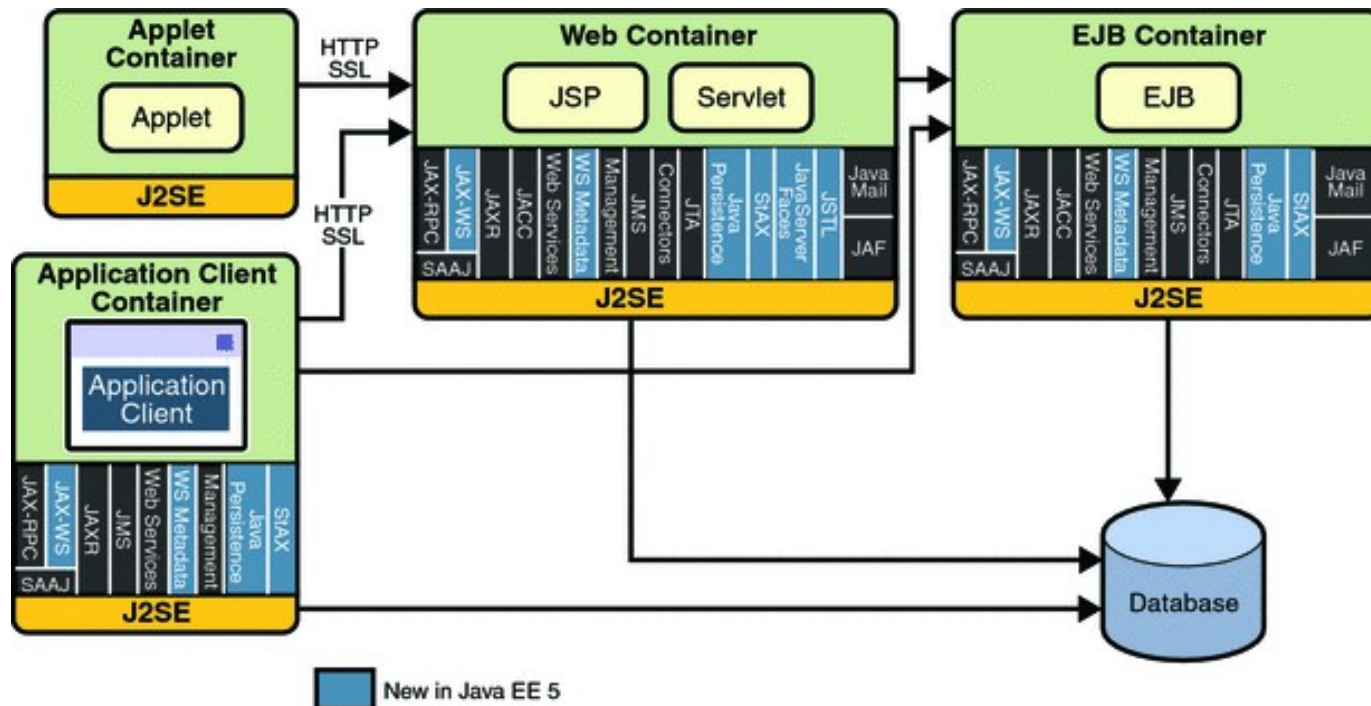
■ Connexion standard aux SI externes (legacy)

- ◆ Support de l'EAI
- ◆ Support XML

Ce que définit Java EE

- **Spécification de la plate-forme**
 - ◆ Composants, containers (programmation, assemblage, déploiement)
 - ◆ Serveur et services (exécution)
- **Implémentation de référence opérationnelle (Glassfish)**
- **Kit de verification de conformité des application (AVK)**
- **Tests de compatibilité des implémentations (TCK)**
 - ◆ Certification Sun
 - ◆ Processus payant sauf pour les produits « open source »
 - ◆ Lourd à mettre en oeuvre (+ de 20 000 tests)
- **Conseils de mise en oeuvre**
 - ◆ Java EE blueprints (pet store, adventure builder, ...)

Overview de l'architecture Java EE



Java EE 5 comme base du cours

■ Simplification du développement

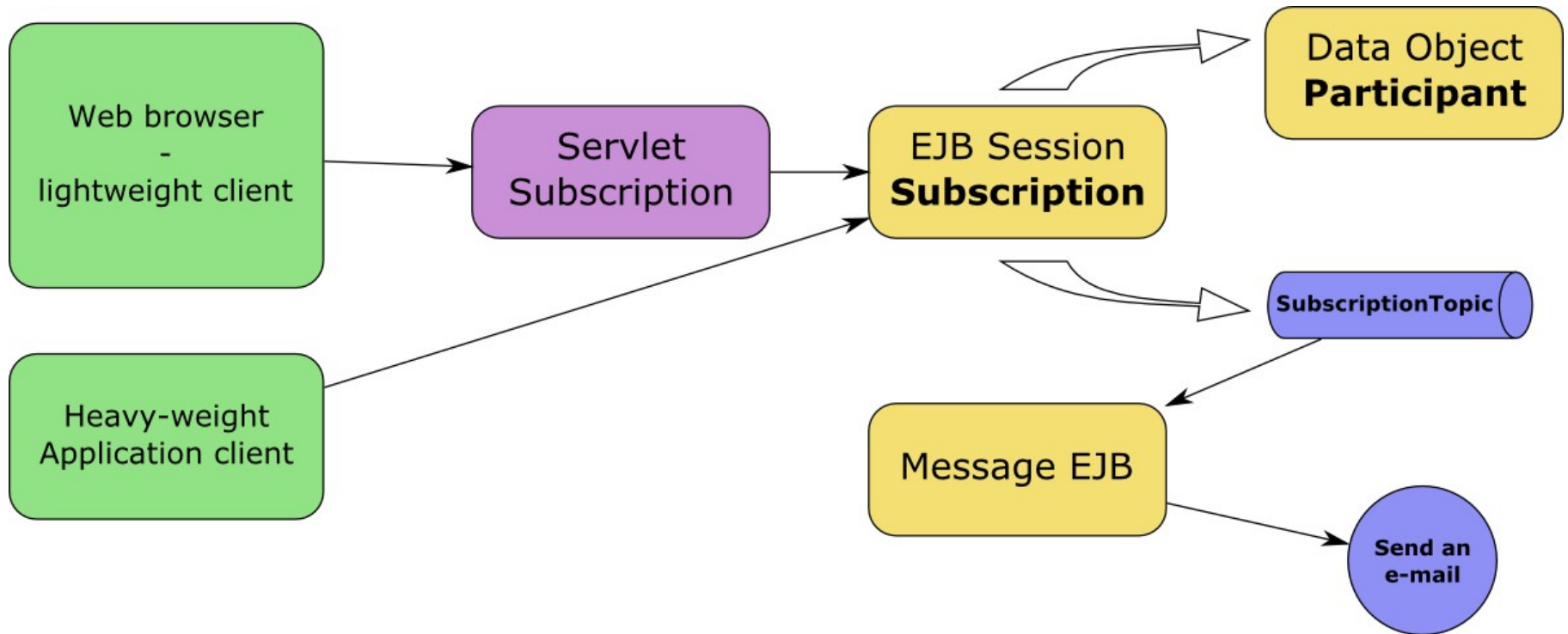
- ◆ Annotations (depuis Java SE 5)
- ◆ Contraintes de programmation des composants assouplies

■ Enrichissement fonctionnel

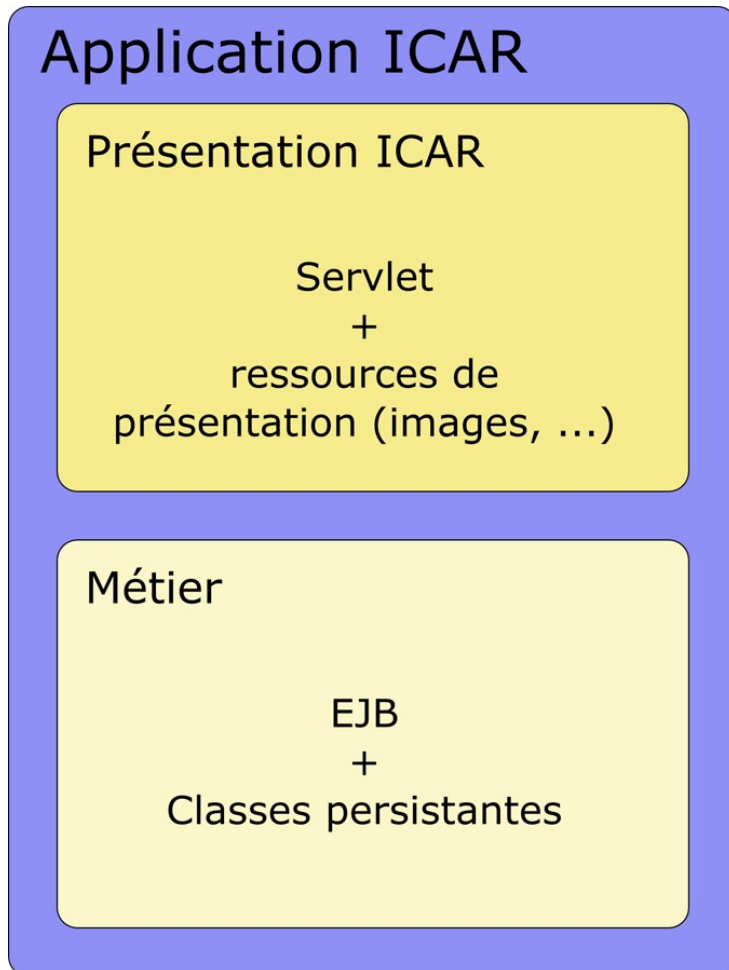
- ◆ Tiers de présentation avec Java Server Faces
- ◆ Tiers de données avec Java Persistence API
- ◆ Évolution majeure des EJB
 - ❖ Séparation gestion des composants / gestion des objets persistants

Exemple de fil conducteur

Inscription à une conférence



Décomposition de l'application



■ Application ICAR

- ◆ Application Java EE
- ◆ Packaging: icar.ear
- ◆ Contient 2 modules

■ Présentation

- ◆ Spécifique à ICAR
- ◆ Packaging: icar.war

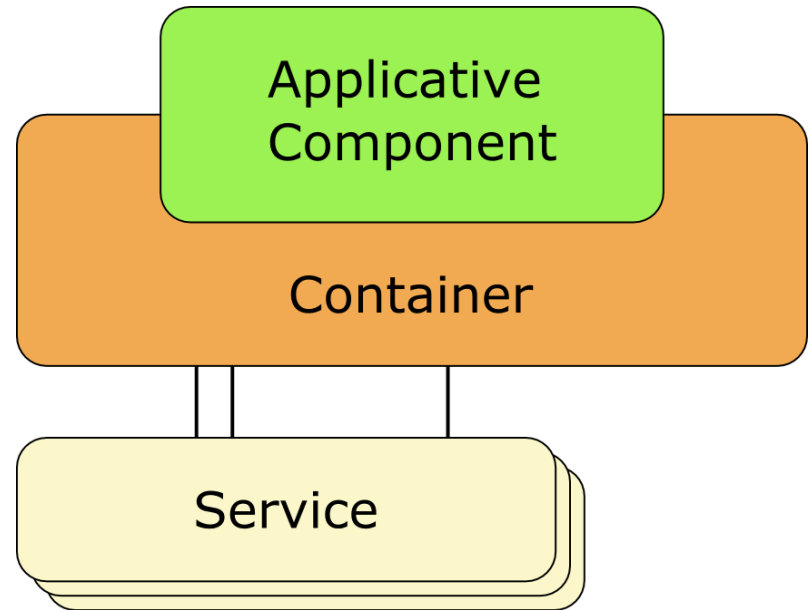
■ Métier

- ◆ Gestion d'une école thématique
- ◆ Réutilisable
- ◆ Packaging: school.jar
- ◆ Contient le code des EJB et objets persistants

Composants: Définition, Assemblage, Packaging et Déploiement

Indépendance

- **Recherche d'une indépendance maximale du code applicatif**
 - ◆ Pas de liaison statique entre éléments de code applicatif (composants)
 - ◆ Pas de liaison statique entre éléments de code applicatif et services de la plate-forme
 - ◆ Éviter l'utilisation de code « technique » dans le code applicatif (déclarativité / transparence)
- **Code applicatif plus réutilisable !**



Rôles des conteneurs

- **Support du déploiement de composants**
 - ◆ Installation d'applications et de bibliothèques (.ear, .war, .rar, .jar)
 - ◆ Activation d'applications et de bibliothèques
- **Gestion des liaisons entre composants**
- **Gestion des liaisons entre composants et services de l'environnement**
- **Support de modèle de programmation**
 - ◆ Définition et gestion des cycles de vie
 - ◆ Transparence de mise en oeuvre de services « techniques » (transactions, sécurité, ...)

Principaux conteneurs

- **Conteneur d'application clientes (clients « lourds »)**
- **Conteneur d'application serveur**
 - ◆ **Conteneur de « servlet »**
 - ◆ **Conteneur d'EJB**
 - ◆ **Conteneur de connecteurs (« ressource adapters »)**
- **Conteneur de persistance**
 - ◆ **Indépendant de Java EE**
 - ◆ **Intégré par le conteneur EJB dans le cadre de Java EE**

Composants Java EE

- **Modules de code réutilisable qu'on peut assembler**
 - ◆ **Changement de liaisons entre modules de code applicatif**
 - ❖ **Sans impact sur le code**
 - ◆ **Configuration des liaisons entre code applicatif et ressources utilisées**
 - ❖ **Sans impact sur le code**

- **Vision ad-hoc non systématique**
 - ◆ **Séparation des préoccupations pas idéale**
 - ❖ **Gestion structuration/assemblage**
 - ❖ **Modèle de programmation**
 - ◆ **Méta-informations structurelles non homogène : définition de liaison spécifique au type de liaison (composant/composant, composant/ressource, ...)**
 - ◆ **Packaging différent (META-INF ou WEB-INF ?)**

Définition d'un composant Java EE

■ Composant Java EE

- ◆ Code Java
- ◆ Information descriptive (méta information)

■ Code contraint par des contrats

- ◆ Implémentation d'interface (ou conformation à des signatures)
- ◆ Cycle de vie
- ◆ Design patterns
 - ❖ Abstract Factory
 - ❖ Accessor / Java Bean
 - ❖ ...

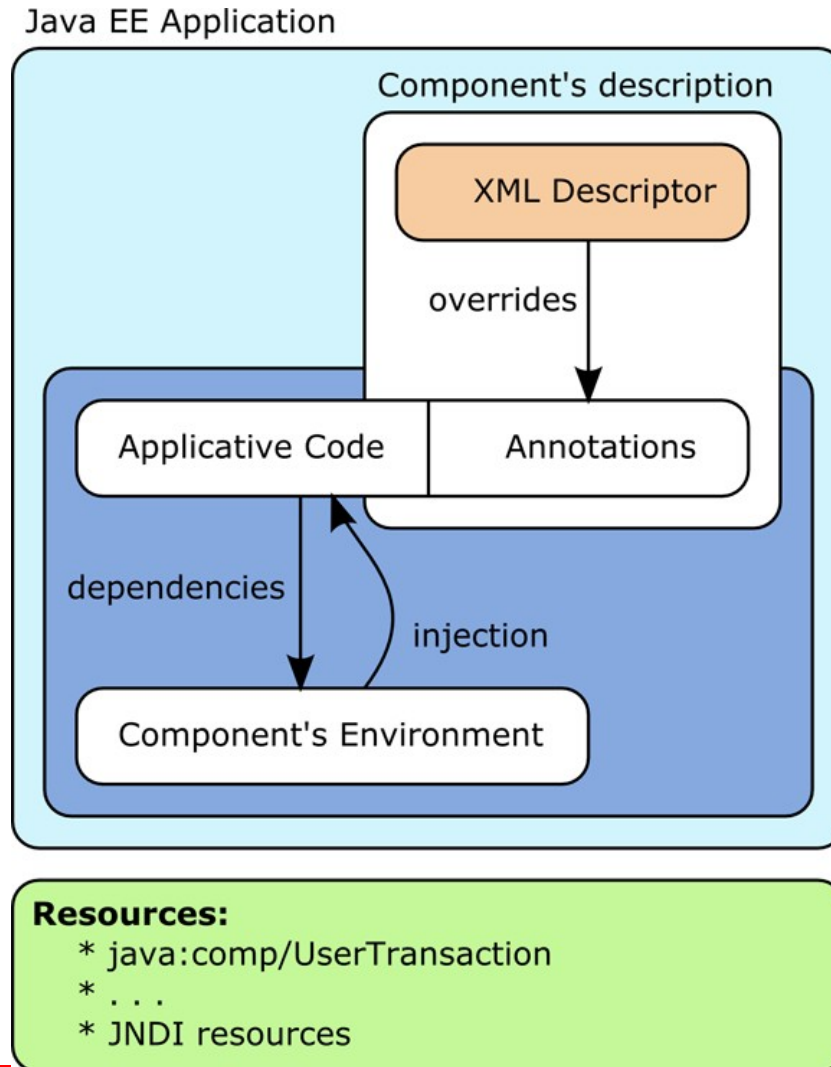
Méta-information d'un composant Java EE

- **Expression de la méta-information**
 - ◆ **Annotation Java SE 5**
 - ◆ **Descripteurs XML**
- **Types de méta-information**
 - ◆ **Interfaces d'instances ou de factory (« home »)**
 - ❖ **Interfaces fournies**
 - ◆ **Définition des dépendances : interfaces requises**
 - ❖ **Composant/Composant (Servlets, EJB, ...)**
 - ❖ **Composant/Ressource (Services)**
 - ▲ **Factory: JDBC, JMS, JCA, JavaMail, ...**
 - ▲ **Resource: une destination JMS, un service web**
 - ▲ **Cas particulier: UserTransaction, TimerService, ORB, ...**
 - ◆ **Définition de propriétés paramétrables (nom, type, valeur)**
 - ◆ **Définition de propriétés non fonctionnelle (transaction, persistance, ...)**

Utilisation des annotations

- **Java EE 5 nécessite Java SE 1.5 pour le support des annotations**
 - ◆ Manipulations au niveau source ou à l'exécution (réflexion)
 - ◆ Associé aux concepts Java (Type, Field, Method, Parameter, ...)
- **Description des composants dans le code**
 - ◆ Toutes les informations des descripteurs peuvent être exprimés sous la forme d'annotations dans le code
 - ◆ Mise en oeuvre plus légère : simplification du développement
 - ◆ 2 types d'informations peuvent être utilisable indistinctement ou simultanément (les descripteurs surchargent les annotations)
- **Principe d'utilisation**
 - ◆ Certaines informations descriptives relève de propriétés liées au code (par exemple le comportement transactionnel, attributs persistants, ...)
 - ◆ Annotations utilisées pour exprimer des informations par défaut ou pour le développement (propriétés paramétrables, liaisons, ...)

Définition d'un composant



Exemple de définition d'un composant

```
@Stateful(name="Subscription")
public class SubscriptionBean implements
ISubscription {
```

```
    @Resource
    private int maxParticipants = 1;
```

```
    @Resource
    private SessionContext sc;
```

```
    /* or
    @Resource
    void setSessionContext(SessionContext c) {
        sc = c;
    }
    */
```

Surcharge les
annotations

```
    <session>
    <ejb-name>Subscription</ejb-name>
    <business-local>org.ow2.icar.bean.ISubscription</business-local>
    <ejb-class>org.ow2.icar.bean.SubscriptionBean</ejb-class>
    <env-entry>
    <env-entry-name>property/maxParticipants</env-entry-name>
    <env-entry-type>java.lang.Integer</env-entry-type>
    <env-entry-value>120</env-entry-value>
    <injection-target>
    <injection-target-name>maxParticipants</injection-target-name>
    </injection-target>
    </env-entry>
    </session>
```

Nom du bean dans l'espace
de nommage de l'ejbjar.

Nom de la propriété dans
l'espace de nommage du bean
(java:comp/env)

Définition de la
variable injectée

Définition des dépendances (références)

■ Références entre composants « packagée » ensemble

- ◆ Définition d'un nom local
- ◆ Définition du lien effectif
 - ❖ Désignation à l'aide du nom local du composant (dans l'espace applicatif)
 - ❖ Désignation à l'aide d'un nom JNDI pour référencer un composant distant (attribut `mappedName`)

■ Référence vers une ressource du serveur

- ◆ Définition d'un nom local
- ◆ Définition du lien effectif: spécifique en J2EE 1.4, définissable en Java EE (attribut `mappedName`)

■ Convention de nommage

- ◆ Nom de propriété == nom de variable de classe Java
- ◆ Nom de ressource (ex `ejb`) == nom de classe Java
- ◆ Nom de référence préfixé par type de ressource (ex: `mail`, `ejb`, ...)

Référence d'un composant EJB

```
@EJB(name="ejb/Subscription", beanName="icar.jar#Subscription")  
private ISubscription sessionSubscription;
```

```
<ejb-ref>  
  <ejb-ref-name>ejb/Subscription</ejb-ref-name>  
  <ejb-ref-type>org.ow2.icar.bean.ISubscription</ejb-ref-type>  
  <ejb-link>icar.jar#Subscription</ejb-link>  
  <injection-target>  
    <injection-target-name>sessionSubscription</injection-target-name>  
  </injection-target>  
</ejb-ref>
```

Référence vers une DataSource JDBC

```
@Resource(name="jdbc/DBIcar",  
          mappedName="jdbc_1")  
private DataSource dbIcar;
```

```
<resource-ref>  
  <res-ref-name>jdbc/DBIcar</res-ref-name>  
  <res-type>javax.sql.DataSource</res-type>  
  <res-auth>Container</res-auth>  
  <injection-target>  
    <injection-target-name>dbIcar</injection-target-name>  
  </injection-target>  
</resource-ref>
```

Référence vers une ressource JMS

```
@Resource(name="jms/SubscriptionTopic",  
          mappedName="IcarTopic")  
private Topic topic;
```

```
<resource-env-ref>  
  <resource-env-ref-name>jms/SubscriptionTopic</resource-env-ref-name>  
  <res-type>javax.jms.Topic</res-type>  
  <injection-target>  
    <injection-target-name>topic</injection-target-name>  
  </injection-target>  
</resource-env-ref>
```

Référence vers une unité de persistance JPA

```
@PersistenceUnit(name="persistence/DBSchool",
                  unitName="icar.jar#SchoolData")
private EntityManagerFactory emf;
```

```
<persistence-unit-ref>
  <persistence-unit-ref-name>persistence/DBSchool</persistence-unit-ref-name>
  <persistence-unit-name>icar.jar#DataSchool</persistence-unit-name>
  <injection-target>
    <injection-target-name>emf</injection-target-name>
  </injection-target>
</persistence-unit-ref>
```

Package icar.ear

■ Contenu de l'archive:

```
META-INF/  
  MANIFEST.MF  
  application.xml  
icar.war  
icar.jar
```

■ Contenu de application.xml

```
<application>  
  <display-name>ICAR'08</display-name>  
  <description></description>  
  <module>  
    <web>  
      <web-uri>icar.war</web-uri>  
      <context-root>icar08</context-root>  
    </web>  
  </module>  
  <module>  
    <ejb>icar.jar</ejb>  
  </module>  
</application>
```

Package icar.jar

■ Contenu de l'archive:

- ◆ `persistence.xml`, cf section persistence

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

■ Contenu de `ejb-jar.xml`

- ◆ Pas nécessaire en EJB 3.0

```
<ejb-jar>  
  <enterprise-beans>  
    <session>  
      <ejb-name>Subscription</ejb-name>  
      <ejb-class>  
org.ow2.icar.bean.SubscriptionBean  
      </ejb-class>  
      <session-type>Stateful</session-type>  
      [ . . . ]  
    </session>  
  </enterprise-beans>  
  [ . . . ]  
</application>
```

Support de l'exécution répartie

Appel de méthode à distance

■ RMI

- ◆ Transport Java (JRMP) ou CORBA (IIOP)
- ◆ Standard d'interopérabilité inter-serveurs Java EE (IIOP)
- ◆ Propagation de contextes optionnelle en Java EE 5 (transaction, sécurité, ...)

■ Web Services

- ◆ Deux modèles
 - ❖ JAX-RPC 1.1
 - ❖ JAX-WS 2.x
- ◆ Basée sur une pile SOAP configurable
 - ❖ Support de différents type de transport (SAAJ)

Communication asynchrone

■ Standard Java Messaging Service (JMS)

◆ Clients de publication et clients de consommation

- ❖ Mode file de messages (queues): message consommé par un unique consommateur
- ❖ Mode topic: message consommé par tous les consommateurs inscrits

◆ Niveaux de QoS sur la délivrance et sur le traitement

- ❖ Sémantique « au moins une fois », « au plus une fois », « exactement une fois »
- ❖ Persistance
- ❖ Support transactionnel

Systeme des connecteurs JCA

- **Intégration des systèmes « legacy »**
 - ◆ Support de protocoles de communication dédiés
 - ◆ Support de messages sortant et entrants
- **Contrats d'intégration « systèmes »**
 - ◆ Gestion de réserve de connexions (dimensionnement de pool)
 - ◆ Gestion des transactions
 - ◆ Gestion de la sécurité
- **Utilisé aussi pour intégrer des services Java EE**
 - ◆ Connecteurs JDBC
 - ◆ Connecteurs JMS

Support des transactions

Fonctions du support transactionnel (JTA)

■ Gestion d'un contexte transactionnel

- ◆ Démarrage: `begin()`
- ◆ Validation: `commit()`
- ◆ Annulation: `rollback()`

■ Association du contexte transactionnel au « thread » courant

- ◆ Évite le passage explicite d'un représentant de la transaction dans le code
- ◆ Attention aux changements de contexte: passage d'un « thread » à l'autre

Rôle du contrat client de gestion des transactions

- **Interface `javax.transaction.UserTransaction`**
 - ◆ Gestionnaire de transactions
 - ◆ Vue client
- **Contrôle de la transaction**
 - ◆ Démarcation (`begin`, `commit`, `rollback`)
 - ◆ Gestion de « timeout »
 - ◆ Demande d'annulation différée à la terminaison (« `rollback-only` »)
 - ◆ Consultation de l'état

Mise en oeuvre du contrat client de gestion des transactions

■ Accès au gestionnaire de transactions

- ◆ Standardisé dans l'environnement Java EE
- ◆ Accès à travers JNDI
- ◆ Nom standard: `java:comp/UserTransaction`

```
UserTransaction utx = (UserTransaction) new
    InitialContext().lookup(«java:comp/UserTransaction»);
utx.begin();
// . . .
utx.commit();
// ou utx.rollback();
```

Rôle du contrat système « standard » de gestion des transactions

■ Utilisé par les composants plutôt techniques

- ◆ Connecteurs / ressource adapters (JMS / JDO / ...)
- ◆ Systèmes de persistance JPA
- ◆ Etc

■ Gestionnaire de transactions évolué

- ◆ Permet l'interception des événements transactionnels: enregistrement d'objets callbacks `javax.transaction.Synchronisation` (par exemple une transaction JDO)
- ◆ Permet de définir des paramètres associés au contexte transactionnel
- ◆ Utilisation de l'interface `javax.transaction.TransactionSynchronisationRegistry`

Mise en oeuvre du contrat système « standard » de gestion des transactions

- **Accès au gestionnaire de transactions évolué**
 - ◆ **Standardisé dans l'environnement Java EE**
 - ◆ **Accès à travers JNDI**
 - ◆ **Nom standard: `java:comp/TransactionSynchronisationRegistry`**

```
// Exemple : connecteur JCA Speedo
TransactionSynchronisationRegistry tsr =
    (TransactionSynchronisationRegistry) new
    InitialContext().lookup(«java:comp/
                            TransactionSynchronisationRegistry»);
```

```
JDOPOManager persistenceManager; // Speedo PersistenceManager
tsr.registerInterposedSynchronization(persistenceManager);
```

Support de la sécurité

Approche pour l'authentification et l'autorisation

■ Notions de base

- ◆ Principal = identifiant
- ◆ Credential = principal authentifié
- ◆ Rôle = profil d'autorisation

■ Authentification

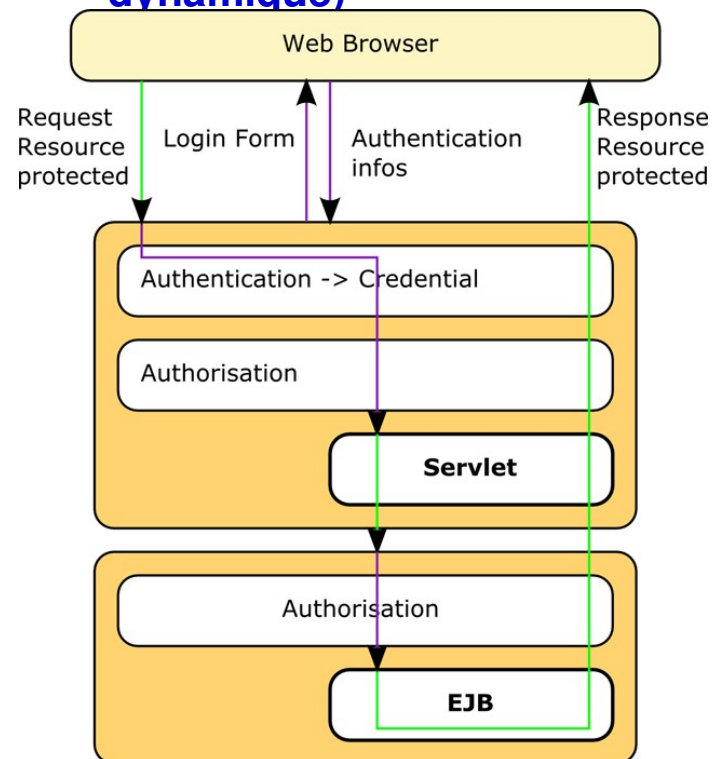
- ◆ Valide un principal et associe un credential à sa session
- ◆ Credential passé au système d'autorisation

■ Autorisation

- ◆ Vérification de droits d'utilisation au d'accès
- ◆ Basé sur les rôles
- ◆ Règle: accès autorisé si un credential est dans un rôle autorisant l'accès à la ressource

■ Gestion de la sécurité

- ◆ Méta-information (minimal)
- ◆ De façon programmatique (externalisation, gestion dynamique)



Composants client applicatif (conteneur client)

Mise en oeuvre client « lourd » 1/2

- **Deux mode d'accès au serveur**
 - ◆ Appel synchrone/RPC: RMI (JRMP/CORBA) ou Web Services
 - ◆ Appel asynchrone / publication de messages: JMS
- **Utilisation de RMI (transparence de la mise en liaison par rapport à un accès local ou distant)**

```
@EJB(name=«ejb/Subscription»,
      mappedName=«Subscription»)
private ISubscription sub;

// . . .

InfoParticipant sub.infos(« Bill Gates »);
```

Mise en oeuvre client « lourd » 2/2

■ Utilisation des Web Services

```
@WebServiceRef(name=«ws/SubscriptionService»,
               org.school.ws.ISubscriptionService.class)
private ISubscriptionPort port;

// . . .

InfoParticipant port.infos(« Bill Gates »);
```

Composants de présentation (conteneur de Servlets)

Service offert par la conteneur web

■ Notion de Servlet

- ◆ Génération de contenu web dynamique (page HTML ou XML)
- ◆ Activation de « thread » pour traitement d'une requête par un objet Servlet
- ◆ Projection d'URLs vers un objet

■ Fonctions de plus haut niveau

- ◆ JSP (Java Server Pages)
 - ❖ Mélange Java/HTML/XML (contenu statique + tags + directives)
 - ❖ Compilation en Servlet au déploiement
- ◆ JSF (Java Server Faces) (utilise JSP)
 - ❖ Meilleure séparation présentation/contrôle (Servlet de contrôle)
 - ❖ Définition d'un graphe d'enchaînement de pages

Servlet en action

```
@EJB(name=«ejb/Subscription»,
      beanName=«icar.jar#Subscription»)
public class ProcessFormServlet extends HttpServlet {

    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) {
        InfoParticipant info = new InfoParticipant();
        info.nom = req.getParameter(« nom »);
        // . . .
        ISubscription sub =
ictx.lookup(« java:comp/env/ejb/Subscription »);
        sub.register(info);
        resp.setContentType(« text/html »);
        PrintWriter pw = resp.getWriter();
        pr.println(« html »);
        // . . .
    }
}
```

Composants métier non persistants (conteneur EJB)

Support de l'étage logique métier 1/2

- **EJB = Composants définissant la logique applicative**
- **Version 3 représente une évolution non négligeable**
 - ◆ Assure la compatibilité ascendante / 2.1
 - ◆ Reste conceptuellement compatible avec la version 2.1
 - ◆ Relâche certaines contraintes au niveau du modèle de programmation
- **Principal objectif: Simplifier la vie du développeur**
 - ◆ Logique POJO
 - ◆ Pas d'implémentation de contrats programmatique (SessionBean), mais respect de signature
 - ◆ Pas nécessaire de définir et de manipuler des factory: mise en liaison automatique (injection)
 - ◆ Plus de composants persistants: seulement des POJOs sans dépendances

Support de l'étage logique métier 2/2

- **Support de la persistance séparé en 2 spécifications**
 - ◆ **EJB 3 core: Intégration de fournisseur de persistance indépendant**
 - ◆ **EJB 3 Persistence: JPA (`javax.persistence` et non dans `javax.ejb`)**
- **Pour l'essentiel des composants gérant des sessions de communication avec un client de l'application activé**
 - ◆ **Par l'arrivée d'un message d'un client**
 - ❖ **Communication synchrone type RPC**
 - ❖ **Communication asynchrone à travers des composants JCA**
 - ◆ **Par des événements temporels (échéanciers/timers)**

Les types de composants métiers

■ Deux modes de communication supportés

- ◆ Appel de procédure à distance (RPC): RMI/CORBA ou Web Services
- ◆ Envoie de message asynchrone (pas d'attente bloquante côté client)

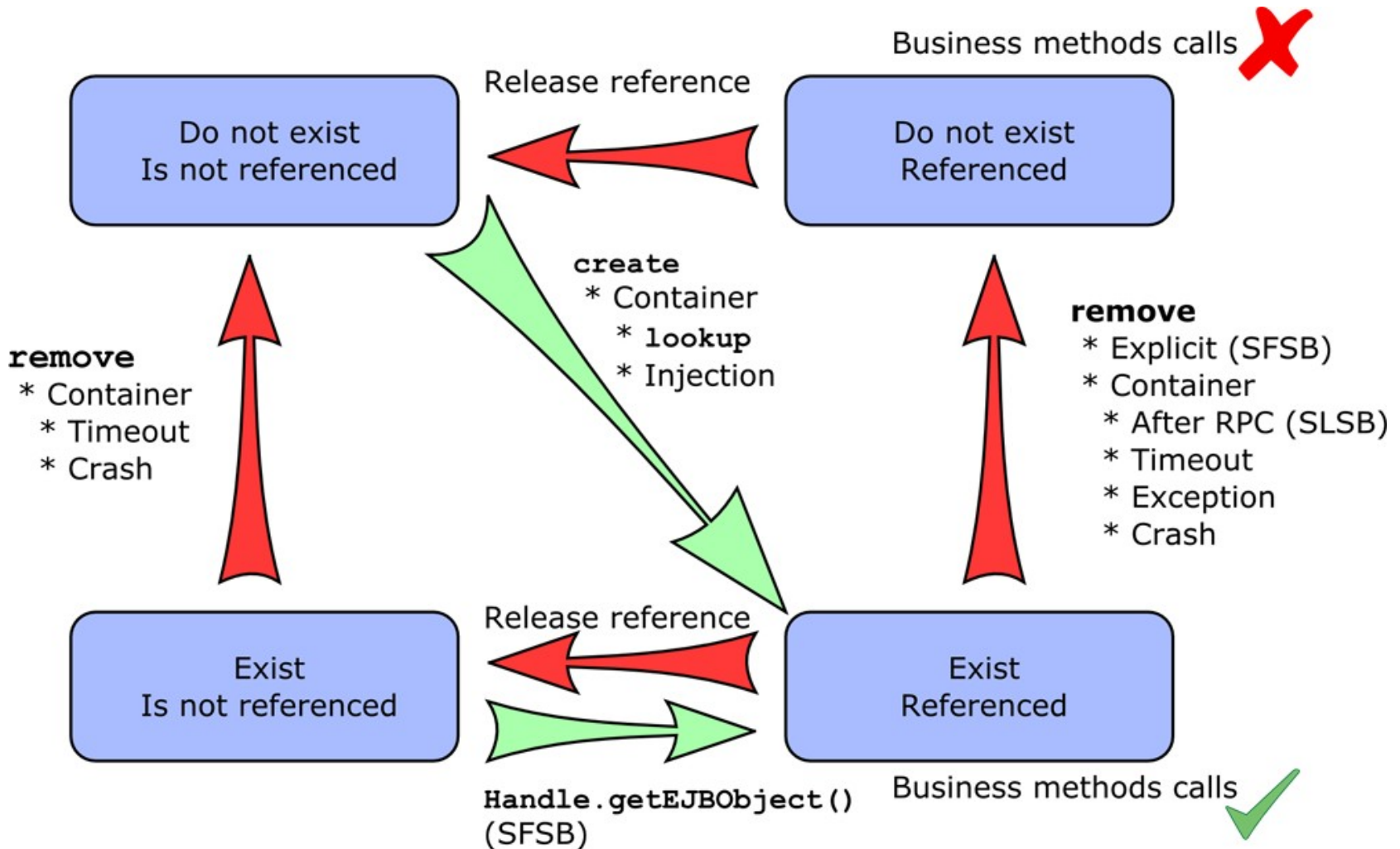
■ Possibilité de gérer un dialogue évolué entre un client et l'application serveur

- ◆ Session traitant plusieurs appels consécutifs
- ◆ Gestion d'informations représentant l'état du dialogue

■ Trois profils de session

- ◆ Composant sans état
 - ❖ Session sans état (StateLess Session Bean, SLSB)
 - ❖ Composants réactifs (Message Driven Bean, MDB)
- ◆ Session avec état, seulement RPC (StateFul Session Bean, SFSB)

Cycle de vie vue du client d'un EJB session (SLSB ou SFSB)



Vue client d'un composant métier

EJB 3.0

- **Vue client pour les composants réactifs**
 - ◆ **API JMS: publication de messages sur une queue ou un topic**
 - ◆ **Format de message accordé entre le client et le serveur (API métier)**
- **Vue client pour les sessions (SLSB et SFSB)**
 - ◆ **API Java métier**
 - ❖ **Interface Java**
 - ❖ **Pas une interface Remote**
 - ◆ **Transparence de la mise en relation**
 - ❖ **Vis-à-vis de la répartition**
 - ❖ **Vis-à-vis du type de RPC (RMI ou WS)**
- **Injection d'une dépendance vers un EJB**
 - ◆ **Utilisation de l'annotation @EJB**

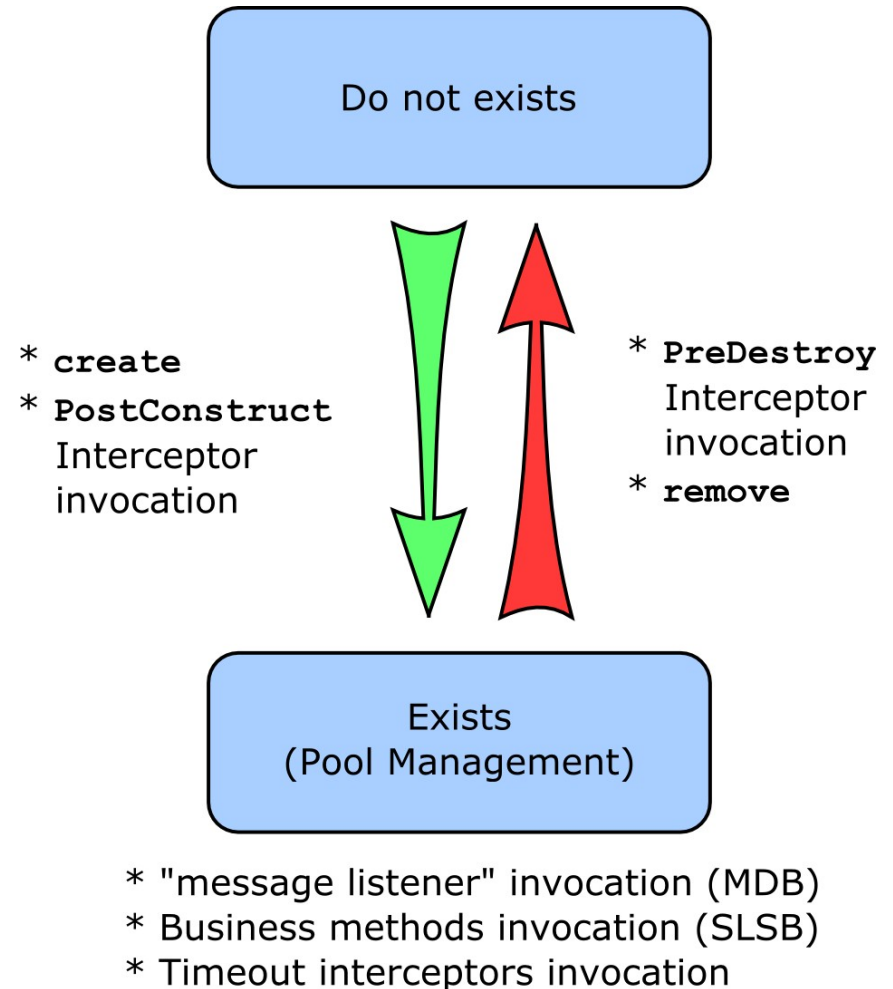
Cycle de vie d'un composant sans état (SLSB ou MDB)

■ Allocation d'instances

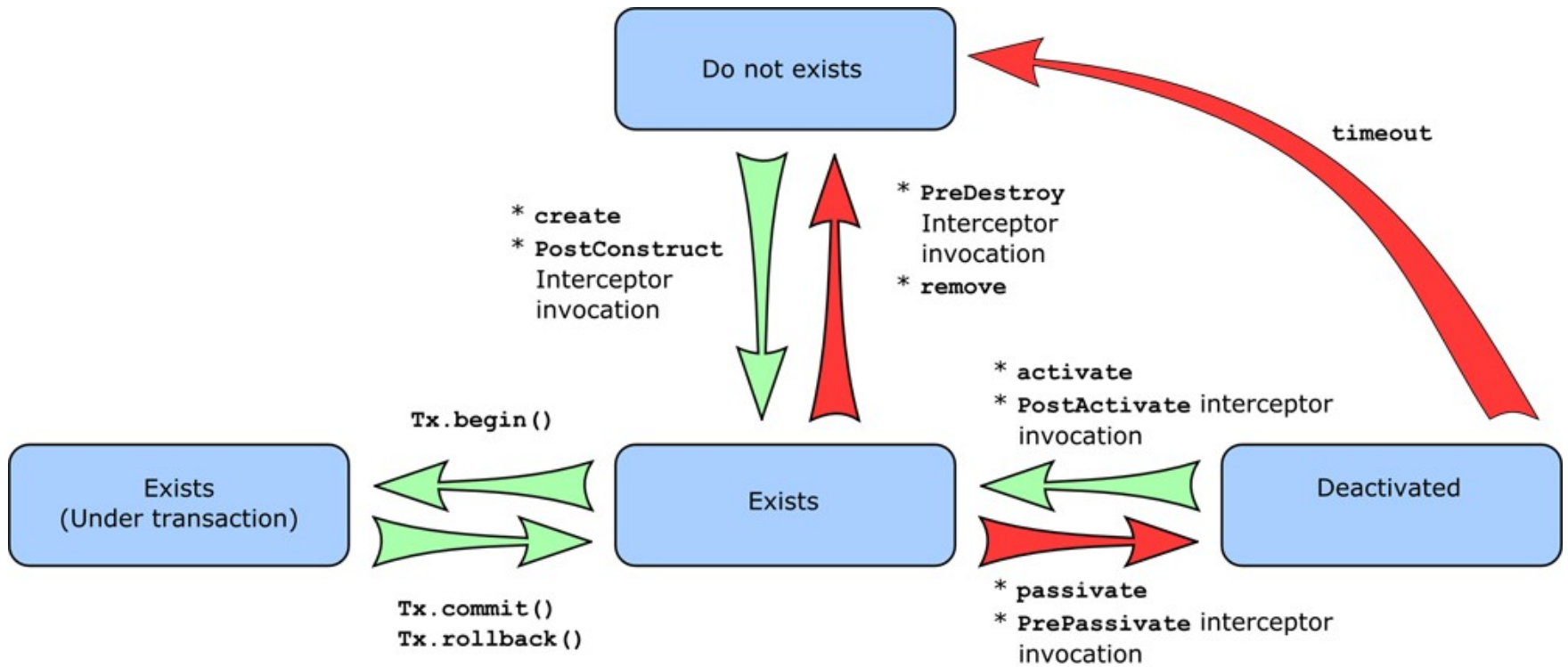
- ◆ Allocation à la discrétion du conteneur (pool)
- ◆ Invocation des intercepteurs `PostConstruct/PreDestroy`

■ Utilisation d'instances

- ◆ Invocation des méthodes métiers
 - ❖ Méthodes des interfaces SLSB
 - ❖ Méthodes des interfaces « message listener »
 - ❖ Sérialisation de l'exécution garantie par le conteneur
- ◆ Définition d'échéanciers (service timer)



Cycle de vie d'une session état (SFSB)



Business methods calls

Be careful: reentrant call **X**

Gestion déclarative des transactions (« container managed »)

- **Comportement associé aux méthodes**
 - ◆ Démarcation == appel de méthode
- **Comportements possibles (« required » par défaut)**
 - ◆ **NotSupported**: si transaction active, transaction suspendue
 - ◆ **Required**: si pas de transaction active, nouvelle transaction
 - ◆ **RequiresNew**: nouvelle transaction (si transaction active, transaction suspendue et réactivée à la fin de la nouvelle)
 - ◆ **Mandatory**: exception si pas de transaction active
 - ◆ **Supports**: rien à faire (si transaction active, l'utiliser)
 - ◆ **Never**: exception si transaction active
- **Seuls Required et NotSupported valables pour les composants réactifs**

Définition de l'interface du composant session « SubscriptionBean » - EJB 2.1

```
public interface IEJB2Subscription extends EJBObject {  
  
    void register(InfoParticipant participant) throws RemoteException;  
    InfoParticipant infos(String name) throws RemoteException;  
}  
  
public interface IEJB2SubscriptionHome extends EJBHome {  
  
    IEJB2Subscription create() throws CreateException, RemoteException;  
}  
  
public class InfoParticipant { // Data Transfert Object (DTO)  
  
    public String nom;  
    public String address;  
  
    // ...  
}
```

Définition de l'interface du composant session « SubscriptionBean » - EJB 3.0

```
@Remote
public interface IEJB3Subscription {

    void register(InfoParticipant participant);
    InfoParticipant infos(String name);

}

public class InfoParticipant {

    public String nom;
    public String address;

    // ...
}
```

Implémentation du composant session « SubscriptionBean » EJB 2.1

```
public class EJB2SubscriptionBean implements SessionBean {

    private ParticipantHome participantHome;

    public void ejbActivate() throws EJBException, RemoteException {}
    public void ejbPassivate() throws EJBException, RemoteException {}
    public void ejbRemove() throws EJBException, RemoteException {}

    public void setSessionContext(SessionContext arg0) throws EJBException,
        RemoteException {
        Object ph = new InitialContext().lookup("java:comp/env/ejb/Participant");
        participantHome = PortableRemoteObject.narrow(ph, ParticipantHome.class);
    }

    public void register(InfoParticipant info) {
        Participant p = participantHome.create(info.nom, info.address);
        /* JMS publication: next slide */
    }

    public InfoParticipant infos(String name) {
        Participant p = participantHome.findByPK(name);
        return p.getInfoParticipant();
    }
}
```

Implémentation du composant session « SubscriptionBean » EJB 3.0

```
@PersistenceContext(name="persistence/DBSchool")
@Stateless
public class EJB3SubscriptionBean implements IEJB3Subscription {

    @Resource
    private SessionContext sc;

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void register(InfoParticipant info) {
        Participant p = new Participant(info.nom, info.address);
        EntityManager em = (EntityManager) sc.lookup("persistence/DBSchool");
        em.persist(p);
        em.close();
        // JMS publish: see MDB slides
    }

    public InfoParticipant infos(String name) {
        EntityManager em = (EntityManager) sc.lookup("persistence/DBSchool");
        Query q = em.createQuery("SELECT OBJECT(i) from Subscription i WHERE i.nom = :pn");
        q.setParameter("pn", name);
        Participant p = (Participant) q.getSingleResult();
        InfoParticipant ip = p.getInfoParticipant();
        em.close();
        return ip;
    }
}
```

Description du composant session

« Subscription » - EJB2.1

```
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1">
  <display-name>ICAR Sessions</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>Subscription</ejb-name>
      <home>org.school.api.IEJB2SubscriptionHome</home>
      <remote>org.school.api.IEJB2Subscription</remote>
      <ejb-class>org.school.lib.EJB2SubscriptionBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <ejb-ref>
        <ejb-ref-name>ejb/Participant</ejb-ref-name>
        <ejb-ref-type>Entity</ejb-ref-type>
        <home>org.school.api.ParticipantHome</home>
        <remote>org.school.api.ParticipantHome</remote>
        <ejb-link>icar.jar#Participant</ejb-link>
      </ejb-ref>
      <resource-ref>
        <res-ref-name>jms/ConnFactory</res-ref-name>
        <res-type>javax.jms.TopicConnectionFactory</res-type>
        <res-auth>Container</res-auth>
      </resource-ref>
      .
      .
      .
    </session>
  </enterprise-beans>
</ejb-jar>
```

Description du composant session

« Subscription » - EJB2.1

. . .

```
<resource-env-ref>
  <resource-env-ref-name>jms/SubscriptionTopic</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>Subscription</ejb-name>
      <method-name>register</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

```
</ejb-jar>
```

Exemple d'une publication de message à partir d'un SLSB

```
@Resource(name="jms/SubscriptionTopic",
    mappedName="SubscriptionTopic")
private Topic topic;
// In EJB 2.1: topic = (Topic) new
InitialContext().lookup("java:comp/env/jms/SubscriptionTopic")

// in register(InfoParticipant)
TopicConnectionFactory tcf = null;
TopicConnection tc = tcf.createTopicConnection();
TopicSession session = tc.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
TopicPublisher tp = session.createPublisher(topic);

MapMessage mm = session.createMapMessage();
mm.setString("participant.name", info.nom);
tp.publish(mm);
```

Exposer « SubscriptionBean » comme un Web Service

```
@WebService(name="Subscription",
            wsdlLocation="META-INF/wsdl/subscription.wsdl",
            serviceName="SubscriptionService",
            portName="SubscriptionPort")
@Stateless
public class EJB3SubscriptionBean implements IEJB3Subscription {

    @Resource
    private SessionContext sc;

    @WebMethod
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void register(InfoParticipant info) {
        Participant p = new Participant(info.nom, info.address);
        EntityManager em = (EntityManager) sc.lookup("persistence/DBSchool");
        em.persist(p);
        em.close();
    }
}
```

Exemple de composant réactif - MDB

```
@MessageDriven(messageListenerInterface=MessageListener.class,
                mappedName="TopicIcar08")
public class NewSubscriptionBean implements MessageListener {

    @Resource private String school = "Thematic School";
    @Resource private String secretary = "school.secretary@myorg.fr";
    @Resource private Session session;

    private InetAddress[] addresses;

    @PostConstruct
    void init() {
        addresses = new InetAddress[] {new InetAddress(secretary)};
    }

    public void onMessage(Message m) {
        MapMessage mm = (MapMessage) m;
        javax.mail.Message mail = new MimeMessage(session);
        mail.setRecipients(javax.mail.Message.RecipientType.TO, addresses);
        mail.setSubject("New subscription - " + school);
        mail.setContent("Participant name: " +
                        mm.getString("participant.name"), "text/plain");
        Transport.send(mail);
    }
}
```

Support des échéanciers 1/2

■ Définition d'évènements temporels

- ◆ Échéance unique ou périodique
- ◆ Définitions persistantes

■ Composant les supportant

- ◆ SLSL et MDB
- ◆ EJB 2.1 Entities

■ Événements applicatifs

- ◆ Activation d'une instance du composant
- ◆ Différents des « timeout » associés à l'exécution d'instance particulières (événements furtifs)

Support des échéanciers 2/2

■ Définition d'un intercepteur d'échéances

```
@Timeout
public void process(Timer t) {
    // ...
}
```

■ Définition d'un échéancier

```
@Resource
private TimerService timerService;

// Fire timeout every hours starting from now
timerService.createTimer(System.currentTimeMillis(),
    3600000, "+1 heure");
```

Composants métiers persistants (Conteneur JPA)

Support de l'étage données métiers

■ Standard JPA 1.0

- ◆ Partie d'EJB 3.0 (JSR 220)
- ◆ Compatible Java SE et Java EE
- ◆ Proche JDO (objectif et approche)

■ Données métier : POJO

- ◆ Au choix de l'utilisateur d'abstraire (ou pas) l'accès au données/sessions
- ◆ Plutôt un problème d'organisation de l'étage session

■ Ce ne sont pas des composants

- ◆ Pas comme les Entities EJB 2.1

■ Standardisation du mapping Objet/Relationnel

Modèles de persistance

■ Objets persistants (entities)

- ◆ Instances de classes persistantes
- ◆ Rendus persistants explicitement (action `persist`)
- ◆ Rendus persistants par attachement (lien de propagation configurables)
- ◆ Détruit explicitement (action `remove`)
- ◆ Détruit par propagation (cascade configurable)

■ API du systeme de persistance

- ◆ `EntityManagerFactory`: représente un espace de persistance (une BD)
- ◆ `EntityManager`: représente un contexte d'exécution (une transaction)
- ◆ `Query`: représente une requête EJBQL (définition et exécution)
- ◆ `EntityTransaction`: représente la transaction associée à un contexte d'exécution (mêmes fonctions que `UserTransaction`)

Modèles de données

- **Entité = ensemble d'attributs persistants**
 - ◆ 2 modes de définition d'attributs
 - ❖ **Field**: simple variable de classe
 - ❖ **Property (Java Bean like)**: méthodes setter et getter
- **Types supportés pour les attributs**
 - ◆ Types primitifs (et leurs wrappers)
 - ◆ Références d'entités (relation ?-1)
 - ◆ Collection de références d'entités (relation ?-n)
- **Support de l'héritage**
- **Support des classes abstraites**
- **Support des classes incluses (lien composite)**

Exemples de classes entities

```
@Entity
@Table(name="PARTICIPANT")
public class Participant {

    @Id
    @GeneratedValue
    @Column(name="PID")
    private long id;

    @Basic
    private String name;

    @Embedded
    private Address address;

    @ManyToMany(mappedBy="participants")
    private Set<School> schools;

}
```

```
@Entity
@Table(name="SCHOOL")
public class School {

    @Id
    @GeneratedValue
    private long id;

    @Embedded
    private Address place;

    @ManyToMany(cascade=CascadeType.PERSIST)
    @JoinTable(name="PARTICIP_SCHOOL",
        joinColumns={
            @JoinColumn(referencedColumnName="id")},
        inverseJoinColumns={
            @JoinColumn(referencedColumnName="PID")})
    private Set<Participant> participants;

}
```

Définition des unités de persistance

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
  <persistence-unit name="DataSchool" transaction-type="JTA">
    <jta-data-source>jdbc_1</jta-data-source>
    <class>org.school.persistence.Address</class>
    <class>org.school.persistence.School</class>
    <class>org.school.persistence.Participant</class>
    <!-- . . . -->
  </persistence-unit>
</persistence>
```

Les requêtes

■ Requêtes ensemblistes

- ◆ Support à base d'EJBQL étendu
- ◆ Support de requêtes SQL natives (attention à la portabilité)

■ Requetes nommées

- ◆ Factorisation de définition
- ◆ Pas de re-compilation EJB à chaque exécution

```
@NamedQuery(name="faithful", query="SELECT OBJECT(p) FROM Participant p, IN "
      + "(p.participations) e GROUP BY p HAVING COUNT(e) > :nbSchools")
public class Participant { /* . . . */ }
```

```
Query q = em.createNamedQuery("faithful");
q.setParameter("nbSchools", 1);
List<Participant> p = q.getResultList();
```

Future

Future

■ Évolution des spécifications

◆ EJB 3.1:

- ❖ Appels de méthode asynchrone
- ❖ Singleton

◆ JCA 1.6:

- ❖ Relâchement des contraintes de ClassLoader (visibilité des librairies)

◆ JPA 2.0:

- ❖ Prise en compte des problèmes relevés avec JPA 1.0

◆ Servlet 3.0:

- ❖ Web framework pluggability + dynamic changes
- ❖ Asynch support

Conclusion

Conclusion

- **Solution industrielle parmi les plus abouties**
 - ◆ Niveaux d'indépendance (HW, OS, DB, RPC, ...)
 - ◆ Spectre fonctionnel très large
- **Evolution des spécifications**
 - ◆ Maturation et stabilisation
 - ◆ Modèles de composants (non homogène)
 - ◆ Modèles de programmation simplifiant le développement (code plus propre)
- **Mise en oeuvre**
 - ◆ Environnement puissant mais complexe
 - ◆ Outil de productivité pour utilisateurs aguerris
 - ◆ Outillage important (Eclipse, NetBeans, IDEA)

Questions / Réponses