Université Claude Bernard Lyon 1

TI 1 : Méthodes de conception de systèmes d'information distribués

Master 2 Traitement de l'Information

Lionel Médini

Octobre 2010

Plan du cours

- Outils de programmation avancés
- Systèmes d'information distribués
- Objets transactionnels distribués
  - Exemples d'EJB 2 et de descripteurs de déploiement
  - EJB 3 : POJO et annotations
  - OSGi
- Introduction à l'urbanisation des SI

Les JavaBeans (1996)

- Définition
  - *Composants logiciels réutilisables* d'applications **non réparties** (en pratique, des classes Java)
- Structure
  - Les propriétés sont cachées et accessibles par des méthodes publiques *public String getNom()* et *public void setNom(String valeur)*
  - Les autres méthodes sont privées
- Utilisation depuis une JSP
  - Déclaration : *<jsp:useBean class="package.NomBean" id="testbean" scope="application" />*
  - Accès : *<%= testbean.getProperty0() %>*

Les JavaBeans (1996)

- Communications entre beans : le modèle événementiel
  - Principe des « écouteurs » (*listeners*) Java : un objet s'enregistre comme écouteur d'un certain événement (interface *java.util.EventListener*)
  - Un bean peut d'émettre un événement (classe *java.util.EventObject*), « capté » par le ou les écouteurs.

  ⇒ Les beans peuvent être assemblés en applications
- Persistance : un bean doit pouvoir être sauvegardé et restitué (en pratique, il doit implémenter *java.io.Serializable*)
- Interface : un bean peut être manipulé *visuellement* dans un outil d'aide à la construction d'applications (dans ce cas, étendre *java.awt.Component*)
- Introspection
  - les propriétés et événements des beans sont découverts par *introspection* par l'outil de construction d'application (classe *XXXBeanInfo* qui implémente *java.beans.BeanInfo*)
  - La découverte des beans peut être réalisée par une instance de la classe *java.beans.Introspector*

Les JavaBeans dans pages JSP

- Principe repris dans les conteneurs Web
- Définition d'un bean dans une classe simple
  - Avec un nom déclaré dans le descripteur de l'application Web
  - Avec des propriétés (attributs) et des accesseurs standards
- Utilisation avec des tags standards JSP

```
<jsp:useBean id="cart" scope="session" class="session.Carts" />
<jsp:setProperty name="cart" property="*" />

<jsp:useBean id="checking" scope="session" class="bank.Checking" >
<jsp:setProperty name="checking" property="balance" value="0.0" />
</jsp:useBean>
```

(source : http://java.sun.com/products/jsp/tags/11/syntaxref11.fm14.htm)

Les JavaBeans dans les conteneurs légers

- Principe repris dans plusieurs frameworks
  - Struts
  - Spring
- Caractéristiques
  - Identiques aux JavaBeans « POJO »
  - Pas d'interface graphique
- Exemples d'utilisation (Struts)

```
<bean:define id="coucou" value="Bonjour + <%= user.getName() %>"
    scope="session"/>
<bean:resource id="config" name="/WEB-INF/config.xml"/>
<bean:include id="menu" page="/menu.jsp?message=Welcome"/>
```

Les Enterprise JavaBeans (1998)

- Définition
  - Composant transactionnel accessible à distance
  - Représente une partie de la logique métier d'une application
- Trois types de beans
  - Beans session
    - Représentent les processus métiers
    - Ne peuvent avoir qu'un seul client à un moment donné
  - Beans entités
    - Permettent d'accéder aux données persistantes (SGBD…)
    - Fournissent une représentation de ces données sous forme d'objets
  - Beans messages
    - Peuvent échanger des messages asynchrone par l'intermédiaire de Java Message Service (JMS)

Les Enterprise JavaBeans

- Historique
  - Mars 1998 : EJB 1.0
    - EJB session uniquement
  - Novembre 1999 : EJB 1.1
    - Sécurité, première version des EJB entités
  - Août 2001 : EJB 2.0
    - Interfaces locales et distantes, EJB messages, EJB-QL
  - Novembre 2003 : EJB 2.1
    - Modification de l'EJB-QL
  - Mai 2006 : EJB 3.0
    - Simplification du développement et du déploiement
  - Encore en développement : EJB 3.1
    - Différentes fonctionnalités dont « EJB Lite »

Les Enterprise JavaBeans

- Un EJB est déployé dans un serveur d'applications J2EE / JEE5
  - Fournit différents services
    - Exemples : aspects distribués (JNDI), gestion du cycle de vie, sécurité transactions, persistence…
  - Instancie et communique avec un conteneur d'EJB
    - Fournit les services du serveur d'applications aux EJB…
    - …*via* un objet EJBContext spécialisé pour chaque type de bean
    - Appelle les méthodes exposées par la / les interfaces de l'EJB (pattern IoC)

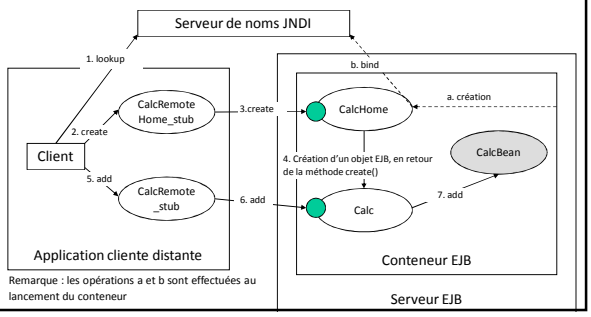
Les Enterprise JavaBeans

- Un EJB expose aux clients différents types d'interfaces
  - Fonctionnelles (EJB 1 et 2)
    - Interfaces « Home »
      - Gestion du cycle de vie (création/destruction des instances)
    - Interfaces métier
      - Méthodes mises à disposition par la classe d'implémentation

    ➔ Disparition des interfaces Home en EJB 3.0
  - En fonction du type d'accès (depuis EJB 2.0)
    - Locales
    - Distantes
- Les beans entités et messages ne sont en général pas accédés par les clients mais par d'autres beans

Les EJB 2.0

- Fonctionnement du serveur et du client



Gestion du cycle de vie (par le conteneur)

- Pooling d'instances
  - Le conteneur possède un « pool » (réserve) d'instances qui lui évitent de créer et de détruire des objets pour chaque client
  - Le conteneur crée une instance de la classe d'implémentation d'un bean et la place dans le pool
    - À l'appel de la méthode create() par un client
    - S'il n'en a aucun de disponible
    - Dans la limite du nombre fixé par l'administrateur
  - Fonctionnement pour les différents types de beans
    - Les beans session sans état, une fois créés, restent opérationnels. Après utilisation, ils sont remis dans le pool ou détruits par le conteneur
    - Les beans session avec états sont désactivés (ejbPassivate()) et réactivés (ejbActivate()) en fonction des besoins du conteneur
    - Les beans entités sont également remis dans le pool après utilisation

EJB session

- Principes de base
  - Représentent les processus métiers
  - Méthodes accessibles par le client
  - Ne peuvent avoir qu'un seul client à un moment donné
  - Deux types de Beans session
    - Avec états : c'est le même client qui réalise toutes les invocations (exemple : panier électronique)
    - Sans état : le Bean peut être utilisé successivement par plusieurs clients (exemple : calcul d'une distance)

EJB session

- Gestion du cycle de vie
  - États d'un bean session
    - Inexistant
      - Levée d'une exception à partir de n'importe quelle méthode

      ⇒Doit être créé par une méthode newInstance(), suivie de setSessionContext()
    - Prêt (avec et sans état)
      - Le bean est accessible par un client
      - Il peut exécuter des méthodes métier
    - Passivé (avec état uniquement)
      - Le bean n'est plus associé à un client
      - Il peut être réactivé par le conteneur

EJB session

- Exemples de nommage pour un EJB session sans état Calc (calculatrice)
  - Interfaces home

```
public interface CalcHome extends EJBHome {
   public Calc create() throws RemoteException, CreateException; }
public interface CalcHomeLocal extends EJBLocalHome {
   public CalcLocal create() throws CreateException; }
```

  - Interfaces métier

```
public interface Calc extends EJBObject {
   public double add(double val1, double val2) throws RemoteException;}
public interface CalcLocal extends EJBLocalObject {
   public double add(double val1, double val2); }
```

EJB session

- Classe d'implémentation (nom : *CalcBean*)
  - Code de la logique métier
  - Implémente une interface spécifique au type d'EJB
    - *javax.ejb.SessionBean*
    - *javax.ejb.EntityBean*
    - *javax.ejb.MessageDrivenBean*

    } Implémentent l'interface *javax.ejb.EnterpriseBean*
  - Contenu
    - Méthodes métiers : *add()*
    - Constructeur sans paramètre : *CalcBean()*
    - Méthode de création : *ejbCreate()*
    - Méthodes de gestion du cycle de vie : *ejbActivate()*, *ejbPassivate()*, *ejbRemove()*

EJB session

- Classe d'implémentation (nom : CalcBean)

```
import javax.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class CalcBean implements SessionBean {

   public double add(double val1, double val2) {
      return val1+val2; }
   public CalcBean() {}
   public void ejbCreate() {}
   public void setSessionContext(SessionContext sc) {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void ejbRemove() {}
}
```

EJB entités

- Principes généraux
  - Permettent l'accès aux données métier persistantes
    - BD, XML…
    - Système propriétaire
    - Système de stockage hétérogène
  - Un EJB entité représente un concept métier
    - Exemple : un compte en banque, un client, un achat…
  - Identification de l'instance par une classe de clé primaire (Int, String, Object…)

  ⇒Il peut y avoir autant d'instances d'EJB entités que de données archivées

  ⇒Pooling d'instances

EJB entités

- Gestion du cycle de vie
  - États d'un bean entité
    - Inexistant
      - Levée d'une exception à partir de n'importe quelle méthode
      ⇒Doit être créé par une méthode newInstance(), suivie de setEntitycontext()
    - Dans le pool
      - Le bean est desactivé et n'est associé à aucune donnée
      ⇒Peut être activé
    - Prêt
      - Le bean est associé à un objet entité déterminé (il connaît sa clé primaire)
      - Il peut exécuter des méthodes métier
      - Le conteneur appelle les méthodes *ejbLoad()* et *ejbStore()* pour gérer la synchronisation du bean avec le support de persistance

EJB entités

- Accès aux données métier d'un EJB entité
  - Identification de l'instance par sa clé primaire
  - Données accessibles
    - Par des méthodes spécifiques
      - Implémentées dans la classe d'implémentation
      - Mapping avec le support de persistance dans le descripteur de déploiement
      (utilisation d'un langage de requêtes *ad hoc* : EJB-QL)
    - Par plusieurs clients en même temps
      - Accès successifs sans transaction
      - Accès transactionnels simultanés

    ⇒Gestion des accès concurrents

EJB entités

- Gestion de la persistance
  - Il faut pouvoir sauvegarder l'état d'un bean (sérialisation)
  - La sauvegarde et la restauration des données sont des étapes critiques (gestion des transactions)
- Deux méthodes
  - CMP : Container Managed Persistance
  - BMP : Bean Managed Persistance

EJB entités


```
public interface Exemple extends EJBObject {

  public InfosExemple getInfosExemple() throws remoteException;
  public void setInfosExemple(InfosExemple ie) throws
remoteException;
  ...
}

public class InfosExemple implements java.io.Serializable {

  public final Integer champInt;
  public final String champString;

  public InfosExemple(Integer i, String s) {
    champInt = i;
    champString = s;
  }
}
```

EJB entités


```
public Collection findByCategorie(String categorie);

<entity>
 <display-name>ExempleEJB</display-name>
 <ejb-name>ExempleEJB</ejb-name>
 ...
 <abstract-schema-name>ExempleSN</abstract-schema-name>
 ...
<query>
 <query-method>
  <method-name>findByCategorie</method-name>
  <method-params>
   <method-param>java.lang.String</ method-param>
  </method-params>
 </query-method>
 <ejb-ql>
  select Object(a) from ExempleSN a where a.categorie = ?1
 </ejb-ql>
</query>
```

EJB entités

- Développement d'un bean CMP
  - L'interface home locale
    - Mêmes méthodes que l'interface distante *a priori*
    - Pas d'exception RemoteException

EJB entités

- Développement d'un bean CMP/BMP
  - La classe d'implémentation
    - Bean BMP : codage de la sérialisation dans cette classe
    - Bean CMP : classe déclarée abstraite pour que le conteneur puisse la sous-classer et implémenter les méthodes de gestion de la persistance
    - Contenu (BMP)
      - Déclaration des méthodes de gestion des champs
      - Méthodes métiers (déclarées dans les interfaces métier)
      - Constructeur sans paramètre (appelé par le conteneur)
      - Méthodes de création (déclarées dans les interfaces home)
      - Méthodes sans entités (déclarées dans les interfaces home)
      - Méthodes internes d'accès aux données (méthodes Select)
      - Méthodes de rappel (appelées par le conteneur)

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Déclaration des méthodes de gestion des champs

```
public abstract String getChampTexte();
public abstract void setchampTexte(String texte);
public abstract Integer getChampInt();
public abstract void setchampInt(Integer int);
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Déclaration des méthodes de gestion des champs

```
public abstract String getChampTexte();
public abstract void setchampTexte(String texte);
public abstract Integer getChampInt();
public abstract void setchampInt(Integer int);
```

    - Implémentation des méthodes métier

      Cf. beans session
    - Constructeur

      …

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes de création
      - ejbCreateXxx() : initialise la création du bean
        - » Met à jour les champs à partir des paramètres (utilise les *setters*)
        - » Implémentation des méthodes déclarées dans les interfaces home
        - » Type de la valeur de retour = type de la clé primaire
        - » Doit retourner *null* (c'est le conteneur qui retrouve la clé primaire)

```
public Integer ejbCreateAvecUnParametre(Type1 param1) throws
    CreateException {
      if (param1==null) throw new CreateException;
    else setParam1(param1);
    return null; }
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes de création
      - ejbPostCreateXxx() : appelée une fois le support de persistance (BD) mis à jour
        - » Traitements nécessaires pour finaliser la création du bean
        - » Ne doivent pas obligatoirement comporter une clause *throw CreateException*
        - » Type de retour void
        - » Mêmes noms et paramètres que la méthode create() correspondante

```
public void ejbPostCreateAvecUnParametre(Type1 param1) { }
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes sans entité : ejbHomeXxx
      - Doivent être déclarées dans les interfaces Home
      - Permettent de faire des traitements de lots

        Ex : trouver combien de clients se sont connectés à une certaine date.
      - Peuvent utiliser des méthodes Finder ou des requêtes à la base
      - Peuvent nécessiter de nombreux accès aux données

      ⇒ À ne pas utiliser

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation

```
<query>
 <query-method>
  <method-name>
    ejbSelectObjetsCommeCa
  </method-name>
  <method-params>
   < method-param>java.lang.String</ method-param>
  </method-params>
 </query-method>
 <ejb-ql>
  select a.idDonnees from DonneesSN a where a.champCa = ?1
 </ejb-ql>
</query>
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation

```
public abstract class MonEJBBean implements javax.ejb.EntityBean {
  EntityContext ec = null;
  public void setEntitycontext(javax.ejb.EntityContext param) {
   ec = param; }
  public void unsetEntitycontext( ) {
   ec = null; }
  public void ejbActivate( ) { }
  public void ejbLoad( ) { }
  public void ejbStore( ) { }
  public void ejbRemove( ) { }
  public void ejbPassivate( ) { }

  […]
}
```

EJB entités

- Développement d'un bean CMP
  - La clé primaire
    - Permet d'identifier le bean de manière unique
    - Classe sérialisable
      - Dérivant de java.lang.Object (Integer, String…)
      - Spécifique au bean MonBeanId, mais la clé doit être composée d'un seul champ
      - Classe annexe sérialisable : méthodes à implémenter
        - » Constructeur par défaut
        - » public boolean equals(Object other)
        - » public int hashcode ()
    - Le descripteur de déploiement doit indiquer
      - Le type de la classe (balise <prim-key-class>)
      - Le nom du champ (balise <primkey-field>)

Packaging

- Assembler tous les éléments dans un fichier ear
  - Interfaces métier et home (locales et/ou distantes)

Calc.class, *CalcLocal.class*, *CalcHome.class*, *CalcHomeLocal.class*

  - Classe d'implémentation

CalcBean.class

  - Classe de clé primaire (beans entités)

MyEntityBeanKey.class

  - Descripteur de déploiement

ejb-jar.xml

Déploiement

- Réalisé par le serveur d'applications
  - Extraction du contenu du EAR/JAR
  - Lecture du fichier ejb-jar.xml
  - Génération du code déployé (code nécessaire à la communication du bean et de ses clients)
    - Objets EJB local et distant
      - Implémentent l'interface métier
      - Peuvent prendre en charge les fonctions de gestion de l'EJB (persistance, sécurité, transactions…)
    - Objets EJB Home local et distant
      - Implémentent l'interface home (méthode create())
    - Stubs, skeletons, etc. en fonction du type de serveur
  - Enregistrement d'une référence à l'objet EJB Home dans un serveur de noms accessible via JNDI

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
 <description>Descripteur de déploiement du bean Calc</description>
 <display-name>Calc</display-name>
 <enterprise-beans>
  <session>
   <description>Calculatrice simple</description>
   <display-name>Calc</display-name>
   <ejb-name>Calc</ejb-name>
   <home>calcul.CalcHome</home>
   <remote>calcul.Calc</remote>
   <ejb-class>calcul.CalcBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
 </enterprise-beans>
 <assembly-descriptor>
  <container-transaction>
    <method>
     <ejb-name>Calc</ejb-name>
     <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
 </assembly-descriptor>
</ejb-jar>
```

```
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
<ejb-jar>
 <enterprise-beans>
  <entity>
   <ejb-name>ExempleBean</ejb-name>
   <home>com.demo.ExempleHome</home>
   <remote>com.demo.ExempleObject</remote>
   <ejb-class>com.demo.ExempleBean</ejb-class>
   <persistence-type>Container</persistence-type>
   <prim-key-class>java.lang.String</prim-key-class>
   <reentrant>False</reentrant>
   <abstract-schema-name>ExempleSN</abstract-schema-name>
   <cmp-field>
    <field-name>ExempleId</field-name>
   </cmp-field>
   <cmp-field>
    <field-name>ExempleData</field-name>
   </cmp-field>
   <primkey-field>ExempleId</primkey-field>
   <query>
    <query-method>
     <method-name>findByCategorie</method-name>
     <method-params>
      <method-param>java.lang.String</ method-param>
     </method-params>
    </query-method>
    <ejb-ql>select Object(a) from ExempleSN a where a.categorie = ?1</ejb-ql>
   </query>
   <query>...</query>
  </entity>
 </enterprise-beans>
 <assembly-descriptor>
  ...
```

Programmation du client

- Tâches à effectuer
  - Contact du serveur de noms (*cf*. RMI)
  - Récupération d'une référence sur l'interface home distante du bean
  - Création d'un objet EJB pour accéder au bean
  - Invocation des méthodes métier

```
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.rmi.PortableRemoteObject;

public class CalcClient {
 public static void main(String args[]) {
  try {
   // Obtention du contexte initial par défaut
   Context initialContext = new InitialContext();

   //Recherche de l'interface home de distante de l'EJB
   Object objref = initialContext.lookup("Calc");
   CalcHome home = (CalcHome)PortableRemoteObject.narrow(objref, CalcHome.class);

   // Création du bean
   Calc myCalc = home.create();

   // Utilisation du bean
   System.out.println("3 + 2 = " + myCalc.add(3., 2.));

  } catch (Exception e) {
    e.printStackTrace(); }
 }
}
```

Conclusion

- Les EJB 2 sont des composants
  - Sûrs
  - Pratiques
  - Puissants
  - Extensibles
  - … mais pas simples

⇒ Nécessitent une modélisation approfondie de l'application

⇒Ne pas « bypasser » l'IDE de génération

Les EJB 3.0

- Principe
  - Conserver/augmenter la puissance des EJB 2
    - Mêmes mécanismes sous-jacents
    - Intégration de fonctionnalités JEE5
  - Simplifier l'utilisation pour le développeur
    - Utilisation de POJOs
    - Annotations Java
    - Injection de dépendances
    - Utilisation de l'API Persistence 1.0 (mapping objet/relationnel)

Les EJB 3.0

- Simplifications
  - Disparition des interfaces Home et de leurs méthodes dans la classe d'implémentation
  - Utilisation d'annotations dans des classes Java standard (POJO)
  - Injection de dépendances (par *setters*) par le conteneur
  - Méthodes *callback interceptors* (appelées par des annotations) pour la gestion du cycle de vie
  - Classes *interceptor* permettant de regrouper tous les traitements liés au cycle de vie (POA)

Les EJB 3.0

- Classe d'implémentation
  - Classe Java standard (POJO)
  - Déclaration : `public` (mais pas `final` ni `static`)
  - Constructeur sans argument
  - Annotations permettant
    - L'injection de dépendances
    - L'appel des méthodes *callbacks interceptors*

```
import javax.ejb.Remote;
@Remote
public interface PremierEJB3 {                                  Interface
  public String ditBonjour(String aQui);                        distante
}
-----------------------------------------------------------------------------------------------------------------------
import javax.ejb.Stateless;
@Stateless
public class PremierEJB3Bean implements PremierEJB3 {          Classe
  public String ditBonjour(String aQui) {                       d'implémentation
    return "Bonjour " + aQui + " !!!";
  }
}
-----------------------------------------------------------------------------------------------------------------------
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class ClientPremierEJB3 {
  public static void main(String[] args) {
    try {
      Context context = new InitialContext();                   Client
      PremierEJB3 beanRemote = (PremierEJB3)                    (appelle directement l'interface métier)
      context.lookup("PremierEJB3Bean/remote");
      System.out.println(beanRemote.ditBonjour("ClientPremierEJB3"));
    } catch (NamingException e) {
      e.printStackTrace();
    }
  }
}
```

(source : http://www.eclipsetotale.com/articles/Introduction_EJB3_avec_Eclipse.html)

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Produit implements Serializable {                  EJB entité

  @Id
  private String id;
  private String libelle;
  private int quantiteEnStock;

  public Produit() { super(); }
  public Produit(String id) { this.id = id; }
  public Produit(String id, String libelle, int quantiteEnStock) {
    this.id = id;
    this.libelle = libelle;
    this.quantiteEnStock = quantiteEnStock; }
  public String getLibelle() { return libelle; }
  public void setLibelle(String libelle) { this.libelle = libelle; }
  public int getQuantiteEnStock() { return quantiteEnStock; }
  public void setQuantiteEnStock(int quantiteEnStock) { this.quantiteEnStock = quantiteEnStock; }
  public String getId() { return id; }
  public String toString() {return "Produit n°" + id + " - " + libelle + " - quantité disponible : " + quantiteEnStock; }
}
```

(source : http://www.eclipsetotale.com/articles/Introduction_EJB3_avec_Eclipse.html)

```
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface GestionDeStock {                               Interface de l'EJB client
  public void ajouter(Produit produit);                         (session sans état)
  public Produit rechercherProduit(String id);
  public List<Produit> listerTousLesProduits();
}
```

```
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class GestionDeStockBean implements GestionDeStock {

  @PersistenceContext                                           Classe d'implémentation
  EntityManager em;                                             de l'EJB client

  public void ajouter(Produit produit) { em.persist(produit); }
  public Produit rechercherProduit(String id) { return em.find(Produit.class, id); }
  public List<Produit> listerTousLesProduits() {
    return em.createQuery("SELECT p FROM Produit p ORDER BY p.quantiteEnStock").getResultList(); }
}
```

```
<persistence>
  <persistence-unit name="IntroEJB3">                           Fichier persistence.xml (JPA)
    <jta-data-source>java:/DefaultDS</jta-data-source>
    <properties><property name="hibernate.hbm2ddl.auto" value="update"/></properties>
  </persistence-unit>
</persistence>
```

(source : http://www.eclipsetotale.com/articles/Introduction_EJB3_avec_Eclipse.html)

Le framework OSGi

- Généralités
  - À l'origine (1998), Open Services Gateway Initiative
  - Framework open source supporté par l'OSGi Alliance
  - Fondation de l'OSGi Alliance : 1999
  - Spécifications en « Releases »
  - Nombreuses applications industrielles
  - Nombreux frameworks applicatifs

Le framework OSGi

- Objectifs du framework
  - Instancier des objets dynamiquement en dehors de l'initialisation des classes
  - Pouvoir obtenir une nouvelle implémentation
    - à chaque invocation de méthode
    - À chaque mise à jour distante de la classe
  - Instanciation en Java « classique »

```
exemple1.QuelqueChose objet = new exemple2.AutreChose();
```

➔Approche par interface

Le framework OSGi

- Fonctionnement du chargement dynamique

```
URLClassLoader myCL = new URLClassLoader("http://www.somewhere.biz");
Class myClass = myCl.load("foo.AServiceImpl");
test.Service  exemple = (test.Service)myClass.newInstance();
```

  - Couplage léger entre demandeur et fournisseur de la classe
    - Seule l'interface du service est couplée au client
    - L'implémentation est chargée à l'exécution
- Remarques
  - Plus de constructeur → injection de dépendances
  - Nécessite de savoir où trouver les implémentations

Le framework OSGi

- Caractéristiques du framework
  - Orienté composants (services)
  - Léger
  - Dynamique : déploiement (chargement / déchargement) d'applications « à chaud »
  - S'appuie sur un annuaire de composants
    - Capable de localiser les composants
    - Capable d'envoyer le code au client
  - Résolution des dépendances versionnées de code

Le framework OSGi

- Caractéristiques d'un composant
  - Nom OSGi : « bundle »
  - Téléchargé à partir d'une URL
  - Packagé dans un jar avec
    - Un point de lancement : interface Activator

```
public void start(BundleContext bc) throws BundleException;

public void stop(BundleContext bc) throws BundleException;
```

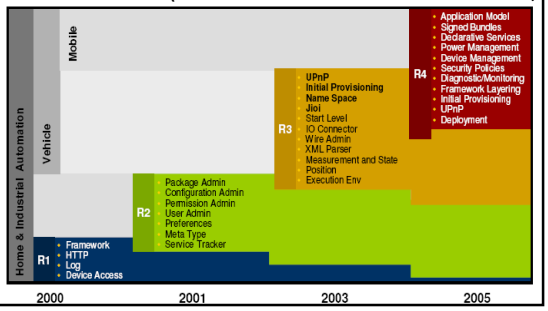
    - Des bibliothèques de code
    - Des ressources (code natif, documents…)

Le framework OSGi

- Types d'applications ciblées
  - À l'origine : mise à jour de modems / passerelles
  - Depuis : *a priori*
    - Toute application décomposable en Java
    - Tout système multi-applications
- Exemples d'utilisation
  - Plateformes dynamiques (API UPnP)
  - Plateformes mobiles (J2ME), embarquées (mBedded)
  - Frameworks : Jonas, Eclipse Equinox, Fuse ESB 4…

Le framework OSGi

- Fonctionnalités (©2008 S. Frénot, D. Donsez, N. Le Sommer)



Le framework OSGi

- Références sur OSGi
  - Cours de Stéphane Frénot (CITI, INSA Lyon), donné à l'école Intergiciel et Construction d'Applications Réparties
  - Projet Apache Felix : http://felix.apache.org/
  - Site officiel OSGi : http://www.osgi.org/
- Remarque :
  - Référence sur les bundles disponibles : http://www.osgi.org/Repository/HomePage

Conclusion

- La réutilisation comme principe général de conception
  - Sélectionner les outils disponibles…
    - Un framework
    - Des bibliothèques
  - …en fonction de vos besoins
    - Nécessite d'avoir correctement spécifié les besoins et réalisé le travail d'analyse

➔Objectif : limiter le plus possible les développements à la logique métier

Conclusion

- Choix d'un framework
  - Identifier le gain : services proposés / lourdeur de l'outil
  - S'attacher à la finalité d'un framework et non à ce que l'on peut faire avec
    - Les utilisateurs peuvent être perdus par une utilisation non standard d'un outil
  - Évolutivité des solutions proposées
    - Penser à l'évolution de votre application
      - Passage à l'échelle
      - Nouveaux services
      - Intégration de technologies futures

Conclusion

- Modularité : penser composants dès les spécifications
  - Précision de la phase de conception et d'analyse (cahier des charges)
  - Utiliser des solutions standard
    - Surtout si vos applications s'insèrent dans un SI existant et si d'autres peuvent devoir s'interfacer avec
    - Prévoir la possibilité de changer radicalement d'interface
      - RIA / RDA
      - Terminaux mobiles
      - Services Web