Enterprise Java Beans 3

Lionel Seinturier

Université des Sciences et Technologies de Lille

Lionel.Seinturier@univ-lille1.fr

10/10/08

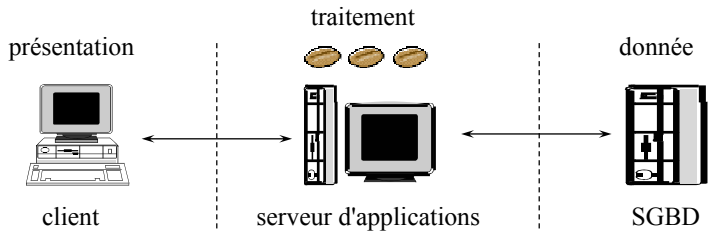
Plan

1. Composant EJB
   1.1 Session Bean
   1.2 Entity Bean
   1.3 Message Driven Bean
   1.4 Fonctionnalités avancées
2. Services
   2.1 Annuaire
   2.2 Transaction
   2.3 Sécurité
3. Design patterns EJB
4. Web Services
5. Conclusion

1. Composant EJB

Enterprise Java Bean (EJB)

Composants applicatifs pour le développement d'applications réparties



1. Composant EJB

Enterprise Java Bean (EJB)

A server-side component that encapsulates the business logic of an application

- on se focalise sur la logique applicative
- les services systèmes sont fournis par le conteneur
- la logique de présentation est du ressort du client

Vocabulaire dans ce cours : *bean* = EJB = composant

Types d'EJB

- Session : *performs a task for a client*
- Entity : *represents a business entity object that exists in persistent storage*
- Message-Driven : *listener processing messages asynchronously*

Plusieurs versions : actuellement EJB 3 (depuis 2006)

1. Composant EJB

EJB 3

- succès Java EE en général
- mais
  - trop compliqué, lourd, contraignant
  - concepts objets (héritage, typage, polymorphisme, …) difficilement exploitables
  - mapping objet/relationel limité
  - trop de fichiers XML fastidieux à écrire, maintenir, comprendre
- passage EJB 2 vers EJB 3
  - utilisation des annotations et de la généricité Java 5
  - pour simplifier l'écriture des *beans*
  - éviter autant que faire se peut l'écriture de fichiers XML

1. Composant EJB

EJB 3 `http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/`

Table 1: Summary of Findings

Application Name	Item Measured	J2EE 1.4 Platform	Java EE 5 Platform	Improvement
AdventureBuilder	Number of classes	67	43	36% fewer classes
	Lines of code	3,284	2,777	15% fewer lines of code
RosterApp	Number of classes	17	7	59% fewer classes
	Lines of code	987	716	27% fewer lines of code
	Number of XML files	9	2	78% fewer XML files
	Lines of XML code	792	26	97% fewer lines of XML code

1. Composant EJB

Annotations Java 5

- mécanisme standard dans le langage Java depuis version 5 (1.5)
- idée similaire aux commentaires Javadoc
  - informations attachées à des éléments de programme (classe, méthode, attributs, …)
  - pour ajouter de l'information sur cet élément
- `@Identificateur`
- éventuellement des paramètres : `@Identificateur(name=value,...)`
  - types autorisés
    - primitifs, String, Class, annotation
    - tableaux de primitifs, String, Class, annotation
- éventuellement plusieurs annotations par éléments

Exemple

```
@Resource(name="myDB", type=javax.sql.DataSource.class)
@Stateful
public class ShoppingCartBean implements ShoppingCart { ... }
```


1. Composant EJB

Annotations Java 5

- chaque annotation est un type (au même titre qu'une classe ou qu'une interface)
- défini dans un package (ex. : `javax.ejb.Stateless`)
- de nouvelles annotations peuvent être définies par le programmeur
  - mot clé Java 5 : `@interface`
  - 1 méthode par paramètre avec éventuellement une valeur par défaut

```
public @interface MyAnnot {
  int value() default 12;
}
@MyAnnot(value=15)
public class MyClass { ... }
```

1. Composant EJB

Annotations Java 5

- annotation = type
  - ➢ les déclarations d'annotations peuvent être annotées

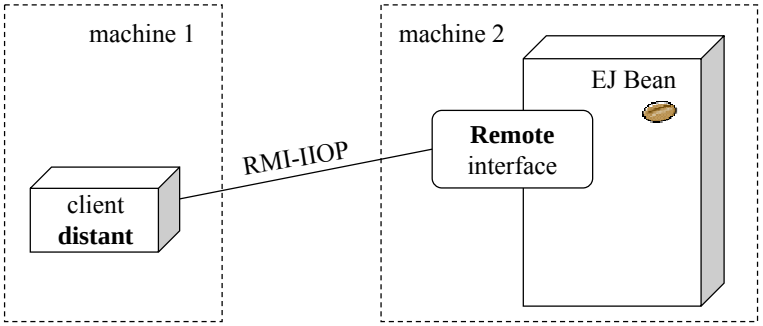
```
@Retention(value=RUNTIME)            // annotations présentes au runtime
@Target(value=CLASS)                 // annotation de classe
public @interface MyAnnot {
  int value() default 12;
}
```

- source, class, runtime
- annotation, constructeur, attribut, variable locale, méthode, package, paramètre, type

Pour + d'informations, voir :
`http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html`

1. Composant EJB

Enterprise Java Bean

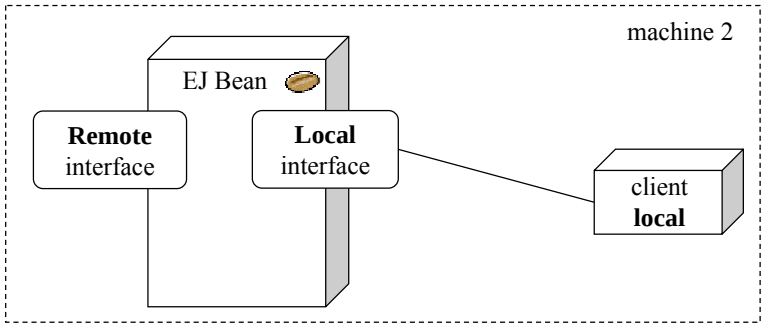


Chaque EJ Bean fournit 1 interface d'accès **distant**

- les services (méthodes) offerts par le bean à ces client

1. Composant EJB

Enterprise Java Bean



+ éventuellement 1 interface d'accès **local** (à partir EJB 2.0)

- les services offerts par le bean à ses clients locaux
- les mêmes (ou d'autres) que ceux offerts à distance

⇨ optimisation

1.1 Session Bean

1. Définition
2. Développement
3. Client local
4. Client distant
5. Stateful session bean

1.1 Session Bean

Définition

Session Bean : représente un traitement (services fournis à un client)

1. Stateless session bean

- sans état
- ne conserve pas d'information entre 2 appels successifs
- 2 instances qqconques d'un tel *bean* sont équivalentes
- 1 instance par invocation

2. Stateful session bean

- avec un état (en mémoire)
- similaire session servlet/JSP
- même instance pendant toute la durée d'une session avec un client
- 1 instance par client

1.1 Session Bean

Développement

1 interface (éventuellement 2 : Local + Remote) + 1 classe

Interface

• annotations `@javax.ejb.Local` ou `@javax.ejb.Remote`

```
import javax.ejb.Remote;

@Remote
public interface CalculatriceItf {
  public double add(double v1,double v2);
  public double sub(double v1,double v2);
  public double mul(double v1,double v2);
  public double div(double v1,double v2);
}
```


1.1 Session Bean

Développement

Classe

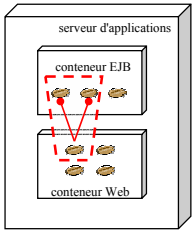
• annotation `@javax.ejb.Stateless` ou `@javax.ejb.stateful`

```
import javax.ejb.Stateless;

@Stateless
public class CalculatriceBean implements CalculatriceItf {
  public double add(double v1,double v2) {return v1+v2;}
  public double sub(double v1,double v2) {return v1-v2;}
  public double mul(double v1,double v2) {return v1*v2;}
  public double div(double v1,double v2) {return v1/v2;}
}
```

• possibilité de nommer les *beans* : `@Stateless(name="foobar")`
• par défaut, le nom de la classe

1.1 Session Bean

Client local



• typiquement une servlet ou une JSP
colocalisée sur le même serveur que le *bean*

• mécanisme dit "injection de dépendance"
  • attribut du type de l'interface
  • annoté `@EJB` éventuellement `@EJB(name="foobar")`

```
public class ClientServlet extends HttpServlet {
  @EJB(name="foobar")
  private CalculatriceItf myBean;
  public void service( HttpServletRequest req, HttpServletResponse resp ) {
     resp.setContentType("text/html");
     PrintWriter out = resp.getWriter();
     double result = myBean.add(12,4.75);
     out.println("<html><body>"+result+"</body></html>");
} }
```

1.1 Session Bean

Client distant

1. Récupération de la référence vers l'annuaire JNDI
2. Recherche du *bean* dans l'annuaire
3. Appel des méthodes du bean

```
public class Client {
  public static void main(String args[]) throws Exception {
    javax.naming.Context ic = new javax.naming.InitialContext();
    CalculatriceItf bean = (CalculatriceItf) ic.lookup("foobar");
    double res = bean.add(3,6);
} }
```


1.1 Session Bean

Stateful Session Bean

- instance du *bean* reste en mémoire tant que le client est présent
- expiration au bout d'un délai d'inactivité
- similaire session JSP/servlet
- utilisation type
  - gestion d'un panier électronique sur un site de commerce en ligne
  - rapport sur l'activité d'un client

2 annotations principales

- `@Stateful` : déclare un *bean* avec état
- `@Remove`
  - définit la méthode de fin de session
  - la session expire à l'issu de l'exécution de cette méthode

1.1 Session Bean

Stateful Session Bean

```
@Stateful
public class CartBean implements CartItf {

  private List items = new ArrayList();
  private List quantities = new ArrayList();

  public void addItem( int ref, int qte ) { ... }
  public void removeItem( int ref ) { ... }

  @Remove
  public void confirmOrder() { ... }
}
```


1.2 Entity Bean

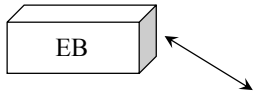
1. Définition
2. Développement
3. Gestionnaire d'entité
4. Relation
5. Autres annotations

1.2 Entity Bean

Définition

Représentation d'une donnée manipulée par l'application

- donnée typiquement stockée dans un SGBD (ou tout autre support accessible en JDBC)



Nom	Solde
John	100.00
Anne	156.00
Marcel	55.25

- correspondance objet – tuple relationnel (*mapping O/R*)
- possibilité de définir des clés, des relations, des recherches
- avantage : manipulation d'objets Java pluôt que de requêtes SQL

- mis en oeuvre à l'aide
  - d'annotations Java 5
  - de la généricité Java 5
  - de l'API JPA (Java Persistence API)

1.2 Entity Bean

Développement

- annotation `@Entity` : déclare une classe correspondant à un *entity bean* (EB)

- chaque classe de EB est mis en correspondance avec une table
  - par défaut table avec même nom que la classe
  - sauf si annotation `@Table(name="...")`

- 2 modes (exclusif) de définition des colonnes des tables
  - *property-based access* : on annote les méthodes *getter*
  - *field-based access* : on annote les attributs

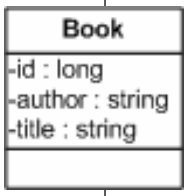
  - par défaut colonne avec même nom que *field/property*
  - sauf si annotation `@Column(name="...")`

  - annotation `@Id` : définit une clé primaire
  - types supportés
    - primitifs (et leurs types Class correspondants), String, Date

1.2 Entity Bean

Développement

```
@Entity
public class Book {

    private long id;
    private String author;
    private String title;

    public Book() {}
    public Book(String author, String title) {
        this.author = author;
        this.title = title; }

    @Id
    public long getId() { return id; }
    public void setId(long id) { this.id = id; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; } }
```



1.2 Entity Bean

Développement

Possibilité de définir des champs auto-incrémentés

- annotation `@GeneratedValue`

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
public long getId() { return id; }
```

- `GenerationType.AUTO` : les numéros de séquence sont choisis automatiquement
- `GenerationType.SEQUENCE` : un générateur de numéros de séquence est à fournir

1.2 Entity Bean

Gestionnaire d'entités

Entity Manager

- assure la correspondance entre les objets Java et les tables relationnelles
- point d'entrée principal dans le service de persistance

- permet d'ajouter des enregistrements
- permet d'exécuter des requêtes

- accessible via une injection de dépendance
  - attribut de type `javax.persistence.EntityManager`
  - annoté par `@PersistenceContext`

1.2 Entity Bean

Gestionnaire d'entités

Exemple

- création de trois enregistrements dans la table des livres

```
@Stateless
public class MyBean implements MyBeanItf {

    @PersistenceContext
    private EntityManager em;

    public void init() {
        Book b1 = new Book("Honore de Balzac","Le Pere Goriot");
        Book b2 = new Book("Honore de Balzac","Les Chouans");
        Book b3 = new Book("Victor Hugo","Les Miserables");

        em.persist(b1);
        em.persist(b2);
        em.persist(b3);
}  }
```

- de façon similaire `em.remove(b2)` retire l'enregistrement de la table

1.2 Entity Bean

Gestionnaire d'entités

Recherche par clé primaire

- méthode find du gestionnaire d'entités

```
Book myBook = em.find(Book.class,12);
```

- retourne `null` si la clé n'existe pas dans la table
- `IllegalArgumentException`
  - si 1er paramètre n'est pas une classe d'EB
  - si 2ème paramètre ne correspond pas au type de la clé primaire

1.2 Entity Bean

Gestionnaire d'entités

Recherche par requête

- requêtes SELECT dans une syntaxe dite EJB-QL étendue
- mot clé OBJECT pour désigner un résultat à retourer sous la forme d'un objet
- paramètres nommés (préfixés par :) pour configurer la requête

```
Query q =
  em.createQuery("select OBJECT(b) from Book b where b.author = :au");
String nom = "Honore de Balzac";
q.setParameter("au",nom);
List<Book> list = (List<Book>) q.getResultList();
```

- méthode `getSingleResult()` pour récupérer un résultat unique
  - `NonUniqueResultException` en cas de non unicité

1.2 Entity Bean

Gestionnaire d'entités

Recherche par requête pré-compilée

• création d'une requête nommée attachée à l'EB

```
@Entity
@NamedQuery(name="allbooks",query="select OBJECT(b) from Book b")
public class Book { ... }

Query q = em.createNamedQuery("allbooks");
List<Book> list = (List<Book>) q.getResultList();
```

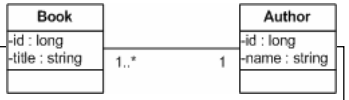
• paramètres peuvent être spécifiés (voir transparent précédent)
• plusieurs requêtes nommées peuvent être définies

```
@Entity
@NamedQueries(
  value={ @NamedQuery("q1","..."), @NamedQuery("q2","...") }
public class Book { ... }
```


1.2 Entity Bean

Relation

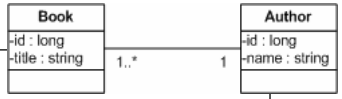
2 catégories principales : 1-n et n-n




```
@Entity
public class Author {

    private long id;
    private String name;
    private Collection<Book> books;

    public Author() { books = new ArrayList<Book>(); }
    public Author(String name) { this.name = name; }

    @OneToMany
    public Collection<Book> getBooks() { return books; }

    public void setBooks( Collection<Book> books ) { this.books=books; }
    ...
}
```


1.2 Entity Bean

Relation 1-n



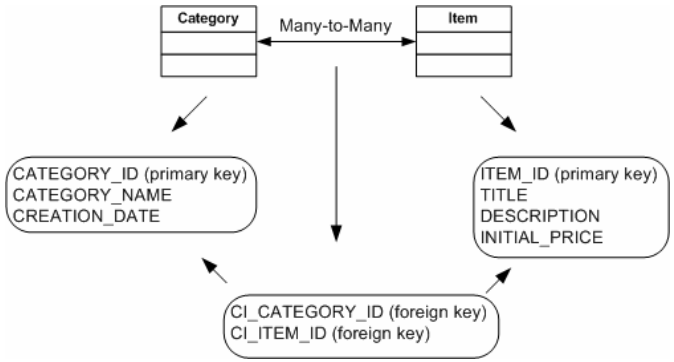

```
@Entity
public class Book {

    private long id;
    private Author author;
    private String title;

    public Book() {}
    public Book(Author author, String title) {
        this.author = author;
        this.title = title; }

    @ManyToOne
    @JoinColumn(name="Author_id")
    public Author getAuthor() { return author; }
    public void setAuthor(Author author) { this.author = author; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    ...
}
```

Nom de la colonne de jointure

1.2 Entity Bean

Relation n-n

• notion de table de jointure



1.2 Entity Bean

Relation n-n

```
@Entity
public class Category {
 @Id
 @Column(name="CATEGORY_ID")
 protected long categoryId;

 @ManyToMany
 @JoinTable(name="CATEGORIES_ITEMS",
  joinColumns=
   @JoinColumn(
     name="CI_CATEGORY_ID",
     referencedColumnName="CATEGORY_ID"),
  inverseJoinColumns=
   @JoinColumn(
     name="CI_ITEM_ID",
     referencedColumnName="ITEM_ID"))
 protected Set<Item> items;
```

```
@Entity
public class Item {
 @Id
 @Column(name="ITEM_ID")
 protected long itemId;

 @ManyToMany(mappedBy="items")
 protected Set<Category> categories;
```


1.2 Entity Bean

Autres annotations

@Enumerated : définit une colonne avec des valeurs énumérées
EnumType : ORDINAL (valeur stockée sous forme int), STRING

```
public enum UserType {STUDENT, TEACHER, SYSADMIN};

@Enumerated(value=EnumType.ORDINAL)
protected UserType userType;
```

@Lob : données binaires

```
@Lob
protected byte[] picture;
```

@Temporal : dates
TemporalType : DATE (java.sql.Date), TIME (java.sql.Time)
TIMESTAMP (java.sql.Timestamp)

```
@Temporal(TemporalType.DATE)
protected java.util.Date creationDate;
```


1.2 Entity Bean

Autres annotations

@SecondaryTable : mapping d'un EB sur plusieurs tables

```
@Entity
@Table(name="USERS")
@SecondaryTable(name="USER_PICTURES"
      pkJoinColumns=@PrimaryKeyJoinColumn(name="USER_ID"))
public class User { ... }
```

@Embeddable et @Embedded : embarque les données d'une classe dans une table

```
@Embeddable
public class Address implements Serializable {
  private String rue; private int codePostal; }

@Entity
public class User {
  private String nom;

  @Embedded
  private Address adresse;
}
```


1.2 Entity Bean

Autres annotations

@IdClass : clé composée

```
@Entity
@IdClass(PersonnePK.class)
public class Personne {
    @Id public String getName() { return name; }
    @Id public String getFirstname() { return firstname; }
}

public class PersonnePK implements Serializable {
  private String name;
  private String firstname;
  public PersonnePK( String n, String f ) { ... }
  public boolean equals(Object other) { ... }
  public int hash() { ... }
}
```

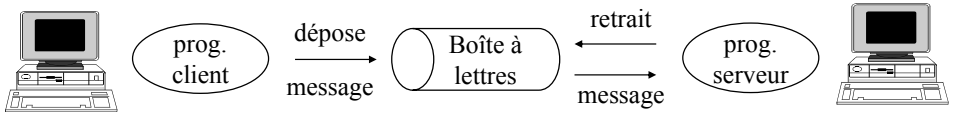
Plan

1. Composant EJB
   - 1.1 Session Bean
   - 1.2 Entity Bean
   - 1.3 Message Driven Bean
   - 1.4 Fonctionnalités avancées
2. Services
   - 2.1 Annuaire
   - 2.2 Transaction
   - 2.3 Sécurité
3. Design patterns EJB
4. Web Services
5. Conclusion

1.3 Message-driven Bean

Message-driven bean (MDB)

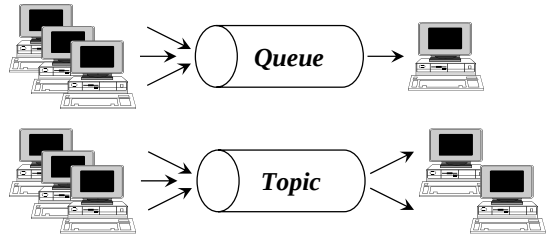
Interaction **par envoi message asynchrone** (MOM : *Message-Oriented Middleware*)



≈ CORBA COSEvent

2 modes

- n vers 1 (*queue*)
- n verrs m (*topic*)



1.3 Message-driven Bean

Caractéristiques

- consomme des messages asynchrones
- pas d'état (≡ *stateless session bean*)
- toutes les instances d'une même classe de MDB sont équivalentes
- peut traiter les messages de clients ≠

Quand utiliser un MDB

• éviter appels bloquants
• découpler clients et serveurs
• besoin de fiabilité : protection *crash* serveurs

Vocabulaire : producteur/consommateur

1.3 Message-driven Bean

Concepts

MDB basé sur Java Messaging Service (JMS) `java.sun.com/jms`

ConnectionFactory	fabrique pour créer des connexions vers une *queue/topic*
Connection	une connexion vers une *queue/topic*
Session	période de temps pour l'envoi de messages dans 1 *queue/topic*

peut être rendue transactionnelle
similitude avec les notions de sessions JDBC, Hibernate, …

Processus

1. Création d'une connexion
2. Création d'une session (* : éventuellement plusieurs sessions par connexion)
3. Création d'un message
4. Envoi du message
5. Fermeture session
6. Fermeture connexion

1.3 Message-driven Bean

Producteur

```
public class MyProducerBean {
 @Resource(name="jms/QueueConnectionFactory")  // l'id de la factory
 private ConnectionFactory connectionFactory;
 @Resource(name="jms/ShippingRequestQueue")  // l'id de la queue
 private Destination destination;
 public void produce() {
  Connection connection = connectionFactory.createConnection();
  Session session = connection.createSession(true,Session.AUTO_ACKNOWLEDGE)
  MessageProducer producer = session.createProducer(destination);
  TextMessage message = session.createTextMessage();
  message.setText("Hello World!");
  producer.send(message);
  session.close();
  connection.close();
} }
```


1.3 Message-driven Bean

Consommateur

MDB = classe
- annotée `@MessageDriven`
- implantant interface MessageListener
  - méthode `void onMessage(Message)`

```
@MessageDriven(name="jms/ShippingRequestProcessor")
public class MyConsumerBean implements MessageListener {
 public void onMessage( Message m ) {
   TextMessage message = (TextMessage) m;
   ...
} }
```


Plan

1. Composant EJB
   - 1.1 Session Bean
   - 1.2 Entity Bean
   - 1.3 Message Driven Bean
   - 1.4 Fonctionnalités avancées
2. Services
   - 2.1 Annuaire
   - 2.2 Transaction
   - 2.3 Sécurité
3. Design patterns EJB
4. Web Services
5. Conclusion

1.4 Fonctionnalités avancées

Timer bean

Déclenchement d'actions périodiques

- `@Timeout` : méthode exécutée à échéance du *timer*
  profil de méthode : `void <methodname>( javax.ejb.Timer timer )`
- `@Resource` : attribut de type `javax.ejb.TimerService`
- utilisation des méthodes de TimerService pour créer des *timers*
  - `createTimer( long initialDuration, long period, Serializable info )`

```
public class EnchereBean {
  @Resource TimerService ts;
  public void ajouterEnchere( EnchereInfo e ) {
    ts.createTimer(1000,25000,e); }
  @Timeout
  public void monitorerEnchere( Timer timer ) {
    EnchereInfo e = (EnchereInfo) timer.getInfo(); ... } }
```

1.4 Fonctionnalités avancées

Intercepteurs

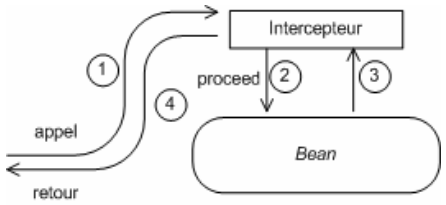
Permettent d'implanter des traitements avant/après les méthodes d'un *bean*
- influence de l'AOP (voir AspectJ, JBoss AOP, …)

- `@Interceptors` : les méthodes devant être interceptées
- `@AroundInvoke` : les méthodes d'interception

  profil de méthode
  ```
  Object <methodname>( InvocationContext ctx ) throws Exception
  ```

`javax.interceptor.InvocationContext`

- permet d'obtenir des informations (introspecter) sur les méthodes interceptées
- fournit une méthode `proceed()` pour exécuter la méthode interceptée



1.4 Fonctionnalités avancées

Intercepteurs

Plusieurs méthodes dans des classes ≠ peuvent être associées à `MyInterceptor`

```
public class EnchereBean {
  @Interceptors(MyInterceptor.class)
  public void ajouterEnchere( Bid bid ) { ... } }

public class MyInterceptor {
  @AroundInvoke
  public Object trace( InvocationContext ic ) throws Exception {
    // ... code avant ...
    java.lang.reflect.Method m = ic.getMethod();
    Object bean = ic.getTarget();
    Object[] params = ic.getParameters();
    // éventuellement modification paramètres avec ic.setParameters(...)
    Object ret = ic.proceed();       // Appel du bean (facultatif)
    // ... code après ...
    return ret; } }
```

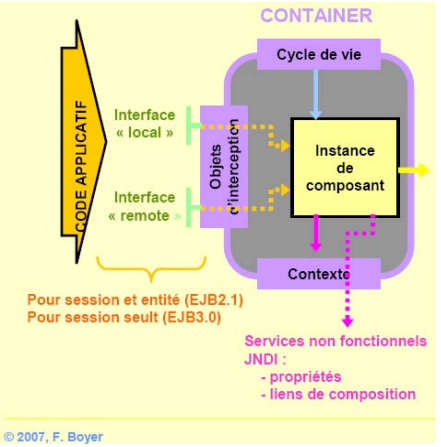

Plan

1. Composant EJB
   1.1 Session Bean
   1.2 Entity Bean
   1.3 Message Driven Bean
   1.4 Fonctionnalités avancées
2. Services
   2.1 Annuaire
   2.2 Transaction
   2.3 Sécurité
3. Design patterns EJB
4. Web Services
5. Conclusion

2. Services

Conteneur

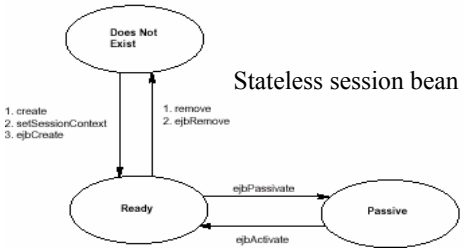
- environnement de création/gestion des instances de composants
- notion de cycle de vie
- annuaire JNDI
- fournit un contexte qui permet d'accéder
  - aux propriétés de configuration
  - aux références vers les autres comp.
  - aux références vers les services techniques (si besoin)



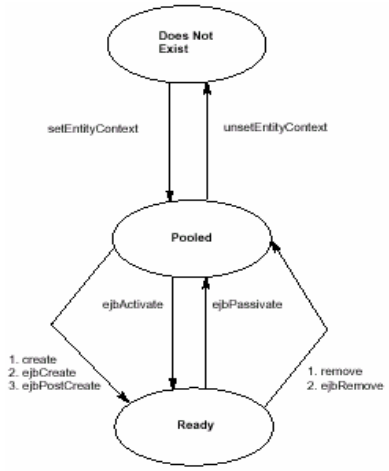
2. Services

Cycle de vie

Stateless session bean



Stateful session bean

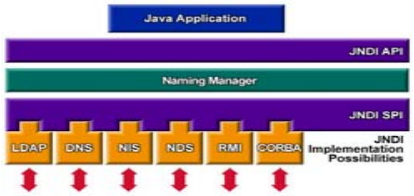


Notification sur changement état

`@PostConstruct` : après la création
`@PreDestroy` : avant la destruction
`@PrePassivate` : avant la passivation
`@PostActivate` : après l'activation

2.1 Annuaire

JNDI



Un service d'annuaire

- références vers composants
- références vers services techniques

- accès distant
- accès local `javax.naming.Context ic = new InitialContext();`
- recherche `Object o = ic.lookup("url");`

- URL JNDI type : chemin/nom
  - RMI-IIOP `iiop://myhost.com/myBean`
  - LDAP `ldap://localhost:389`

- Contexte local Java EE
  - `java:comp/env/`
  - ex. `java:comp/env/myOtherBean`, `java:comp/env/javax.user.Transaction`

2.2 Transactions

Service de transactions

Assure des propriétés **ACID** pour des transactions plates

Exemple classique : un transfert bancaire (débit, crédit)

- atomicité soit les 2 opérations s'effectuent complètement, soit aucune
- cohérence le solde d'un compte ne doit jamais être négatif
- isolation des transferts // doivent fournir le même résultat qu'en séq.
- durabilité les soldes doivent être sauvegardés sur support stable

Support complètement intégré au serveur EJB
Véritable + / aux *middlewares* style CORBA

2.2 Transactions

Granularité des transactions

Comment démarquer (délimiter) les transactions ?

Attribut transactionnel avec 6 valeurs

- REQUIRED
- REQUIRES_NEW
- SUPPORTS
- NOT_SUPPORTED
- MANDATORY
- NEVER

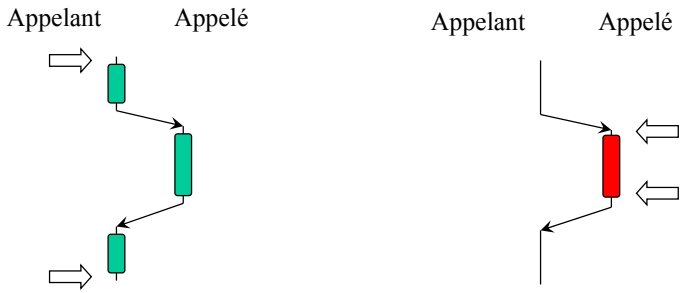
2 cas pour le *bean* appelant

- soit il s'exécute dans une transaction
- soit il s'exécute en dehors de tout contexte transactionnel

2.2 Transactions

Granularité des transactions

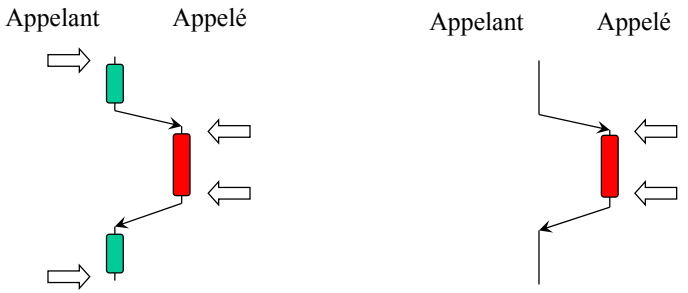
REQUIRED



2.2 Transactions

Granularité des transactions

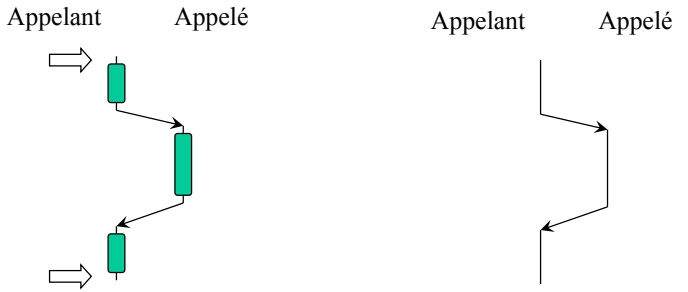
REQUIRES_NEW



2.2 Transactions

Granularité des transactions

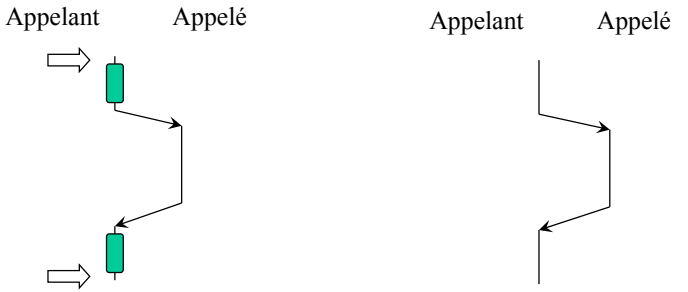
SUPPORTS



2.2 Transactions

Granularité des transactions

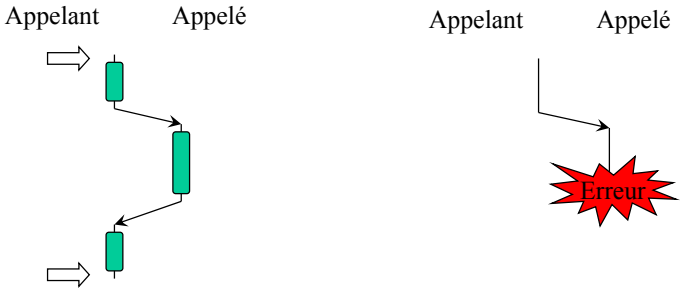
NOT_SUPPORTED



2.2 Transactions

Granularité des transactions

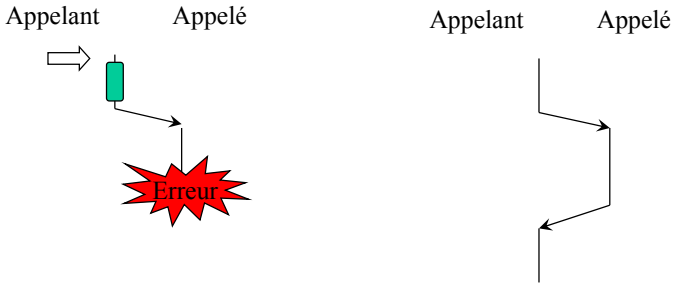
MANDATORY



2.2 Transactions

Granularité des transactions

NEVER



2.2 Transactions

Définition des transactions

2 modes

- CMT (*Container Managed Transaction*) : annotations
- BMT (*Bean Managed Transaction*) : API JTA

CMT

- `@TransactionManagement` sur la classe
- `@TransactionAttribute(TransactionAttributeType.XXX)`
  sur les méthodes transactionnelles
  où `XXX` est 1 des 6 attributs précédents
- toute la méthode est considérée comme un bloc transactionnel
- *commit* par défaut en fin de méthode
- appel `setRollbackOnly()` pour annuler

2.2 Transactions

Définitions des transactions – CMT

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class MyBean implements MyBeanItf {

  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  public void transfert() {
    try {
      Account a1 = em.find(Account.class,"Bob");
      Account a2 = em.find(Account.class,"Anne");
      a1.credit(10.5);
      a2.widthdraw(10.5);
    }
    catch( Exception e ) {
      sc.setRollbackOnly();
  } }

  @PersistenceContext
  private EntityManager em;

  @Resource
  private SessionContext sc; }
```

2.2 Transactions

Définition des transactions

BMT

- démarcation explicite avec begin/commit/rollback
- avantage : possibilité granularité plus fine qu'en CMT

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class MyBean implements MyBeanItf {

 public void transfert() {
  try {
   ut.begin();
   Account a1 = em.find(Account.class,"Bob");
   Account a2 = em.find(Account.class,"Anne");
   a1.credit(10.5);
   a2.widthdraw(10.5);
   ut.commit();
  }
  catch( Exception e ) { ut.rollback(); } }
```

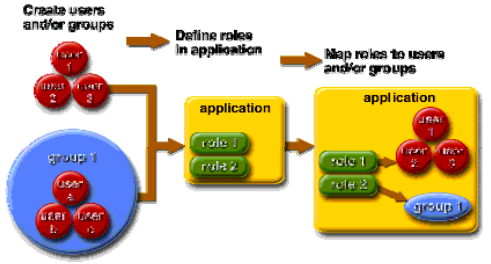
```
@PersistenceContext
private EntityManager em;

@Resource
private UserTransaction ut; }
```


2.3 Sécurité

Service de sécurité

- contrôler l'accès aux méthodes d'un bean
- authentifier l'utilisateur

À base de rôles



5 annotations

```
javax.annotation.security.PermitAll
javax.annotation.security.DenyAll
javax.annotation.security.RolesAllowed
javax.annotation.security.DeclareRoles
javax.annotation.security.RunAs
```


2.3 Sécurité

Annotations	Target		EJB or its super classes	Servlet or web libraries	Descriptions
	TYPE	METHOD			
@PermitAll	X	X	X		Indicates that the given method or all business methods of the given EJB are accessible by everyone.
@DenyAll		X	X		Indicates that the given method in the EJB cannot be accessed by anyone.
@RolesAllowed	X	X	X		Indicates that the given method or all business methods in the EJB can be accessed by users associated with the list of roles.
@DeclareRoles	X		X	X	Defines roles for security checking. To be used by EJBContext.isCallerInRole, HttpServletRequest.isUserInRole, and WebServiceContext.isUserInRole.
@RunAs	X		X (not for non-EJB super classes)	X (for servlet only)	Specifies the run-as role for the given components.

2.3 Sécurité

Service de sécurité

Exemple

```
@Stateless
@RolesAllowed("javaee")
public class HelloEJB implements Hello {

  @PermitAll
  public String hello(String msg) { return "Hello, " + msg; }

  public String bye(String msg) { return "Bye, " + msg; }
}
```

- `hello` : accessible par tout le monde
- `bye` : accessible seulement pour le rôle javaee

2.3 Sécurité

Service de sécurité

Pour les cas plus complexes : `@DeclaresRoles`

Exemple 2

```
@Stateless
@DeclaresRoles({"A", "B"})
public class HelloEJB implements Hello {
  @Resource private SessionContext sc;
  public String hello(String msg) {
    if (sc.isCallerInRole("A") && !sc.isCallerInRole("B")) {
      ...
    } else {
      ...
    } } }
```

• `hello` : accessible par ceux qui sont dans le rôle A mais pas dans le rôle B

Plan

1. Composant EJB
   1.1 Session Bean
   1.2 Entity Bean
   1.3 Message Driven Bean
   1.4 Fonctionnalités avancées
2. Services
   2.1 Annuaire
   2.2 Transaction
   2.3 Sécurité
3. Design patterns EJB
4. Web Services
5. Conclusion

3. Design Patterns EJB

Gabarit de conception (*design pattern*)

Problèmes de codage récurrents

- parcourir un arbre de données dont les noeuds sont typés
- maj une fenêtre en fonction de modifications sur des données (et vice-versa)
- …

⇨ design pattern (DP) : solutions reconnues d'organisation du code

But

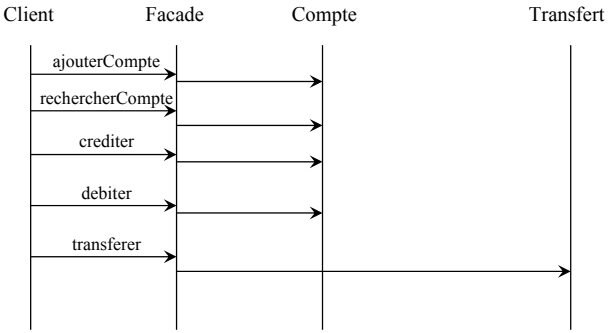
- améliorer la clarté, la compréhension du code
- mettre en avant des éléments d'architecture logicielle

3. Design Patterns EJB

DP Session facade

Pb : nombreuses dépendances entre les clients et les beans

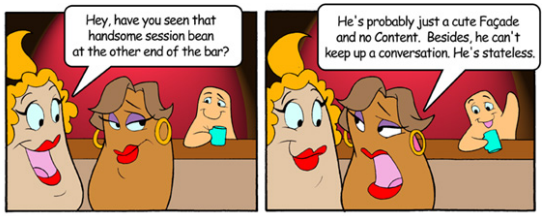
Solution : présenter aux clients **une seule interface façade** (*stateless session bean*)



3. Design Patterns EJB

DP Session facade

- traitements effectués par la façade minimaux
  essentiellement **délégation**
- éventuellement plusieurs façades sur un même ensemble de beans
  ⇨ différentes vues

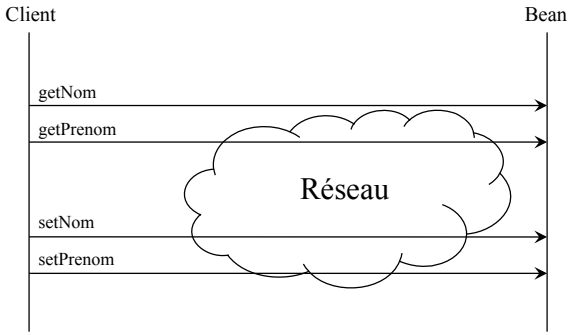


`http://www.theserverside.com/cartoons/TalesFromTheServerSide.tss`

3. Design Patterns EJB

DP Data Transfert Object

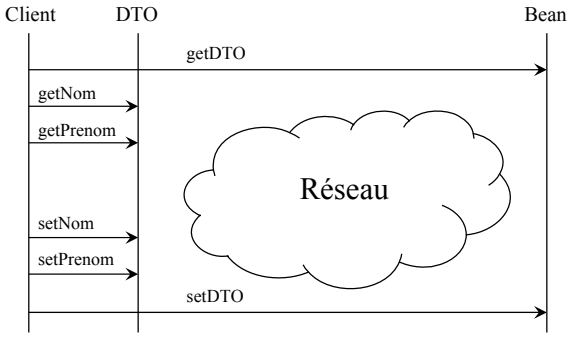
Aussi connu sous le terme : Value Object
Pb : nombreux échanges réseaux pour de simples get/set



3. Design Patterns EJB

DP Data Transfert Object

Solution : transmettre une instance par valeur



3. Design Patterns EJB

Autres DP

- application service
  centraliser un processus métier s'étendant sur plusieurs *beans*
- composite entity
  rassembler dans 1 seul EB les données persistantes de plusieurs EB
- transfert object assembler
  aggregation de plusieurs DTO
- service activator
  invoquer des services de façon asynchrone

Plan

1. Composant EJB
   - 1.1 Session Bean
   - 1.2 Entity Bean
   - 1.3 Message Driven Bean
   - 1.4 Fonctionnalités avancées
2. Services
   - 2.1 Annuaire
   - 2.2 Transaction
   - 2.3 Sécurité
3. Design patterns EJB
4. Web Services
5. Conclusion

4. Web Services

Web Services (WS)

Invocation de traitements à distance via le Web
Solution d'interopérabilité entre systèmes hétérogènes (par ex. Java EE ↔ .NET)
Repose sur HTTP / SOAP / XML / WSDL

• les méthodes fournies par les beans peuvent être exposées comme des WS

`@WebService` : annotation pour la classe d'un bean
`@WebMethod` : annotation pour une méthode d'un bean accessible via un WS
`@WebResult` et `@WebParam` : annotations pour les paramètres d'une WebMethod

• la plate-forme génére le code (WSDL, *stub*, *skeleton*)
nécessaire au fonctionnement WS

4. Web Services

Exemple

```
@WebService(targetNamespace="urn:UserService")
@Stateless
public class MyBean implements MyBeanItf {

  @WebMethod
  @WebResult(name="UserId")
  public long addUser(
    @WebParam(name="UserName") String name,
    @WebParam(name="UserAge") int age )
 { ... }
}
```

Accès (ex. myhost.com 8080) :
`http://myhost.com:8080/UserService/MyBean/addUser?UserName=Bob&UserAge=15`

4. Web Services

Utilisation de WS

@WebServiceRef : référence un web service

```
public class MyClientBean {

  @WebServiceRef(
    wsdlLocation="http://localhost:8080/UserService/MyBean?WSDL")
  private MyBeanItf service;

  public void foo() {
    long id = service.addUser("Bob",15);
} }
```

Plan

1. Composant EJB
   - 1.1 Session Bean
   - 1.2 Entity Bean
   - 1.3 Message Driven Bean
   - 1.4 Fonctionnalités avancées
2. Services
   - 2.1 Annuaire
   - 2.2 Transaction
   - 2.3 Sécurité
3. Design patterns EJB
4. Web Services
5. Conclusion

5. Conclusion

Java EE

Apports

- prise en charge des services techniques par la plate-forme
- le composant se focalise sur le métier
- *packaging* & déploiement

Mais

- uniquement Java
- nombreux domaines applicatifs concernés par la répartition non couvert par Java EE
  - temps-réel, embarqué, mobilité, CAO, infographie, …
- déploiement sur un seul serveur à la fois

Évolution (permanente)

- EJB 3.1 : *bean singleton*, invocation de méthode asynchrone

5. Conclusion

Perspective : EJB 3.1

Évolution permanente depuis EJB 1 (1996/97)

Fonctionnalités en proposition/discussion actuellement
- pas encore (10/2008) de garantie de mise en oeuvre
- JSR 318 : Enterprise JavaBeans 3.1

- interface optionnelle pour *session bean*
  - véritable POJO (*Plain Old Java Object*)
  - interface déduite des méthodes publiques de la classe
  - `Local` vs `Remote` ?

- 5è type de *bean* : singleton (annotation `@Singleton`)
  - conserver des données en mémoire (évite l'usage d'attribut `static`)
  - par défaut toutes méthodes du *bean* singleton sont *thread-safe* et transactionnelles

5. Conclusion

Perspective : EJB 3.1

- simplification du timer bean
  - annotation `@Schedule` pour définir la périodicité

```
@Stateless
public class NewsLetterGeneratorBean implements NewsLetterGenerator {
    @Schedule(second="0", minute="0", hour="0",
              dayOfMonth="1", month="*", year="*")
    public void generateMonthlyNewsLetter() {
        // ...
} }
```

  ➢ déclenchement tous les 1ers du mois à 00h00
  - déclenchement ?
  - délai avant le 1er déclenchement ?

- simplification du *packaging* des .ear en autorisant des *beans* directement dans des .war

5. Conclusion

Perspective : EJB 3.1

- invocation asynchrone des méthodes d'un *bean*

```
@Stateless
public class OrderBillingServiceBean implements OrderBillingService {
    @Asynchronous
    public void billOrder(Order order) { ... }
}
```

- ne remplace *message-driven bean*
- s'implifie le développement pour beaucoup de cas asynchrones simples
- définition d'un objet dit futur en cas de résultat

```
@Asynchronous
public Future<OrderStatus> billOrder(Order order) { ... }
```

- interface `java.util.concurrent.Future` (depuis JDK 6)
- méthodes `isDone()`, `get()`

5. Conclusion

Perspective : EJB 3.1

EJB Lite
Une version allégée des EJB

- *entity bean* ?
- définition de profiles au niveau Java EE ?

Feature	EJB Lite	EJB
Stateless beans	✔	✔
Stateful beans	✔	✔
Singleton beans	✔	✔
Message driven beans		✔
No interfaces	✔	✔
Local interfaces	✔	✔
Remote interfaces		✔
Web service interfaces		✔
Asynchronous invocation		✔
Interceptors	✔	✔
Declarative security	✔	✔
Declarative transactions	✔	✔
Programmatic transactions	✔	✔
Timer service		✔
EJB 2.x support		✔
CORBA interoperability		✔

Table 1: EJB and EJB Lite Feature Comparison

5. Conclusion

Perspective : EJB 3.1

- intégration des EJB et des Web Beans
  - alimentation directe d'un *entity bean* à partir des données d'un formulaire Web

```
<h:form>
  <h:inputText value="#{bid.bidder}"/>
  <h:inputText value="#{bid.item}"/>
  <h:inputText value="#{bid.bidPrice}"/>
  <h:commandButton type="submit" action="#{placeBid.addBid}"/>
</h:form>
```

```
@Stateless
@Named("placeBid")
public class PlaceBidBean {
  @In
  private Bid bid;
  public void addBid() {
    entityManager.persist(bid);
} }
```

```
@Entity
@Named("bid")
public class Bid {
    private Long bidId;
    private String bidder;
    private String item;
    private Double bidPrice;
    ...
}
```


5. Conclusion

Perspective : EJB 3.1

- simplification des intercepteurs
  - intercepteur associé à une annotation métier (**`@Audited`**)
  - méthodes interceptées peuvent être déclarées avec une annotation métier

```
@InterceptorBindingType
public @interface Audited {}

@Audited @Interceptor
public class AuditInterceptor {
  @AroundInvoke
  public Object audit(InvocationContext context) throws Exception {
} }

@Stateless
@Named("placeBid")
public class PlaceBidBean {
  @Audited
  public void addBid() { ... }}
```

5. Conclusion

Perspective : EJB 3.1

- standardisation du format des noms JNDI pour les beans

JBoss global JNDI name	action-bazaar/PlaceBidBean/remote
GlassFish global JNDI name	PlaceBid
WebSphere Community Edition global JNDI name	action-bazaar-ejb/PlaceBidBean/PlaceBid
Oracle Application Server (OC4J) global JNDI name	PlaceBidBean

Table 1: Vendor-specific global JNDI names

```
java:global[/<application-name>]/<module-name>/<bean-name>#<interface-name>
```

• améliore la portabilité des applications

5. Conclusion

Perspective : EJB 3.1

- conteneur EJB embarquable
  - par ex. directement dans un JDK
  - éviter d'avoir à utiliser une plate-forme Java EE complète
  - faciliter, promouvoir l'utilisation des EJB
  - lien avec la notion de profile

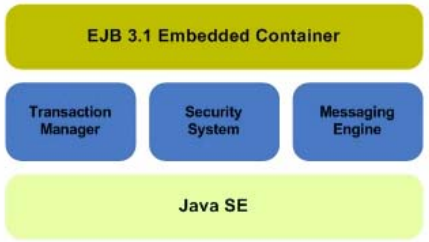


Figure 1: EJB 3.1 Embeddable Containers Concept

```
static void main(String[] args) {
  EJBContainer container =
    EJBContainerFactory.
      createEJBContainer();
  Context context =
    container.getContext();
  PlaceBid placeBid =
    (PlaceBid) context.lookup
    ("java:global/action-bazaar/
      PlaceBid");
  placeBid.addBid(new Bid("rrahman",
    10059, 200.50));
  container.close(); }
```


5. Conclusion

Perspective : EJB 3.1

Encore d'autres pistes

- intégration EJB et Spring
- intégration de JAX-RS (API Java EE pour REST Web Services)
- intéropérabilité avec OSGi
- support *Rich Internet Application* (RIA)
  - faire en sorte que les EJB soit "conscient" de l'utilisation sous-jacente de HTTP dans la partie présentation

5. Conclusion

Commerciaux

• WebSphere	IBM	`www-01.ibm.com/software/webservers/appserv/wasproductline/`
• WebLogic	BEA	`www.weblogic.com`
• App Server	Borland	`www.borland.com/appserver`
• iPortal	IONA	`www.iona.com/products`

• Oracle, Sybase, HP, Fujitsu, ATG, Hitachi, Macromedia, SilverStream, ...

voir `java.sun.com/javaee/overview/compatibility.jsp`

Open source

• Java EE 5 (Sun)	`java.sun.com/javaee`
• JOnAS (ObjectWeb)	`jonas.ow2.org`
• JBoss (Red Hat)	`www.jboss.org`
• Geronimo (Apache)	`geronimo.apache.org`
• JFox	`www.huihoo.org/jfox`

5. Conclusion

Ressources

API Java EE 5 : `http://java.sun.com/javaee/5/docs/api/`
Tutorial Java EE 5 Sun : `http://java.sun.com/javaee/5/docs/tutorial/doc/`

Livre Mastering Enterprise Java Beans, 3rd edition
`http://www.theserverside.com/tt/books/wiley/masteringEJB/index.tss`