Java Enterprise Edition
Enterprise JavaBeans 3

Didier Donsez

Université Joseph Fourier - Grenoble 1
PolyTech Grenoble - LIG / ERODS
Didier.Donsez@imag.fr
Didier.Donsez@ieee.org

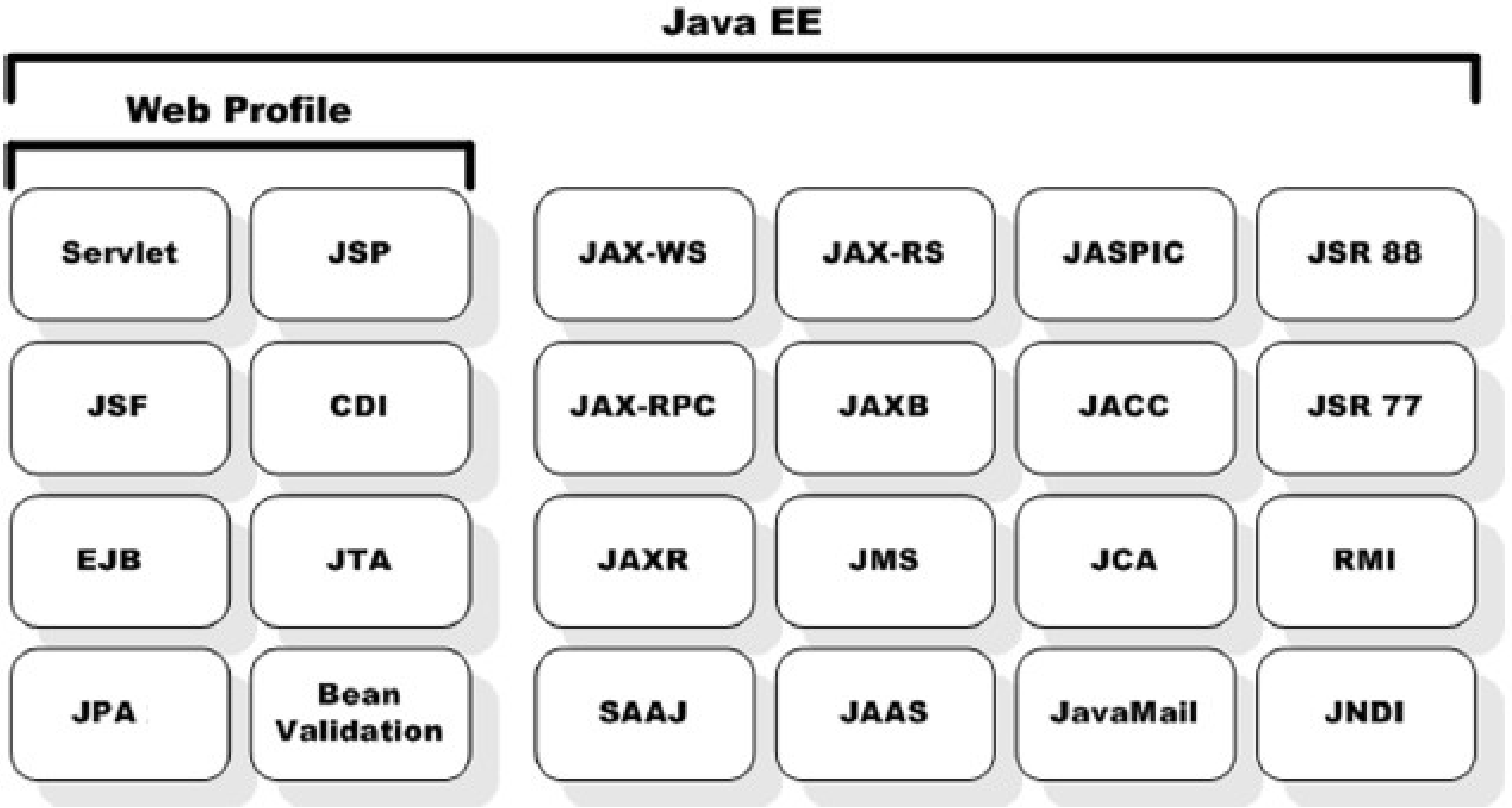
Crédits

- Présentations réalisées avec des transparents de
  - Noël de Palma
  - Fabienne Boyer
  - Pascal Déchamboux
  - Lionel Seinturier
- Et les exemples / schémas des livres cités à la fin

Motivations

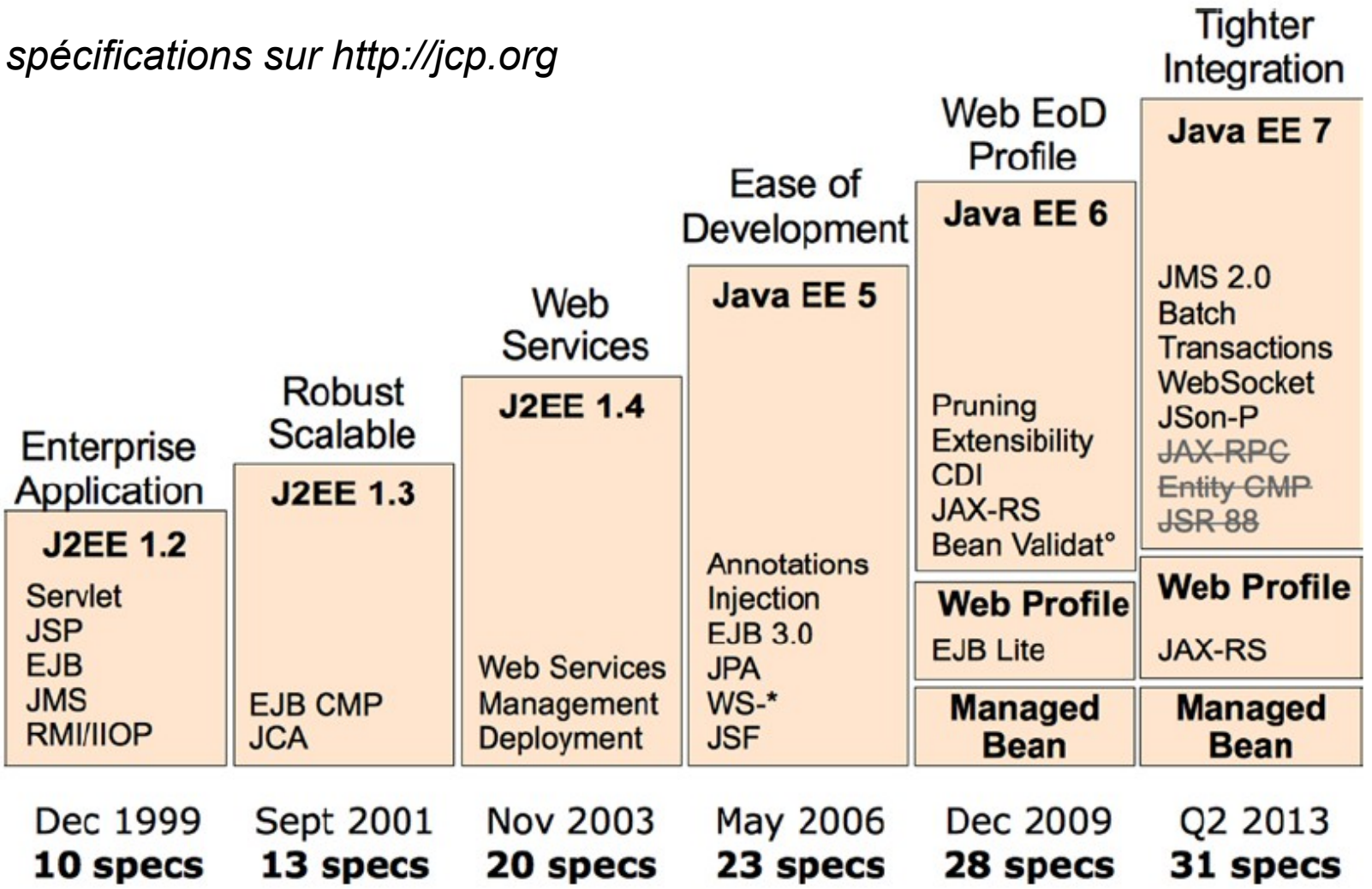
- Ensemble des spécifications
  pour les applications Java coté serveur
  - Architecture 3-tiers
    - Présentation + Métier + Persistance
  - Prise en charge des services non fonctionnels
    - Gestion des sessions utilisateurs, Persistance, Transaction ACID, Gestion de la concurrence, Sécurité, Scalabilité, …
  - Connexion à des systèmes patrimoniaux
  - Domaines variés
    - e-Commerce, Finance, Telecom, …
  - Solutions multi-vendeurs (portabilité)

Java EE specifications



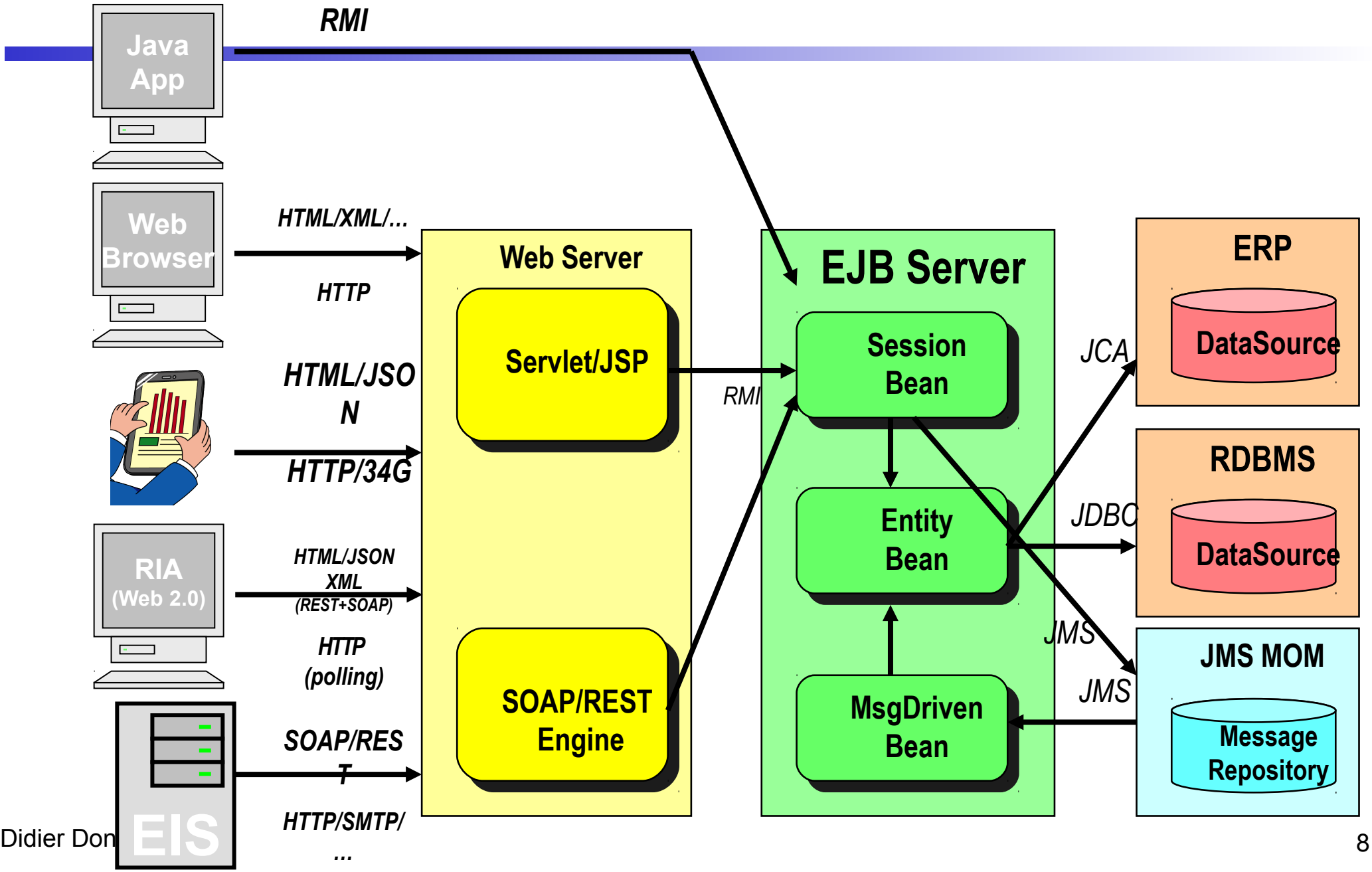
Java EE Specifications

Voir les spécifications sur http://jcp.org

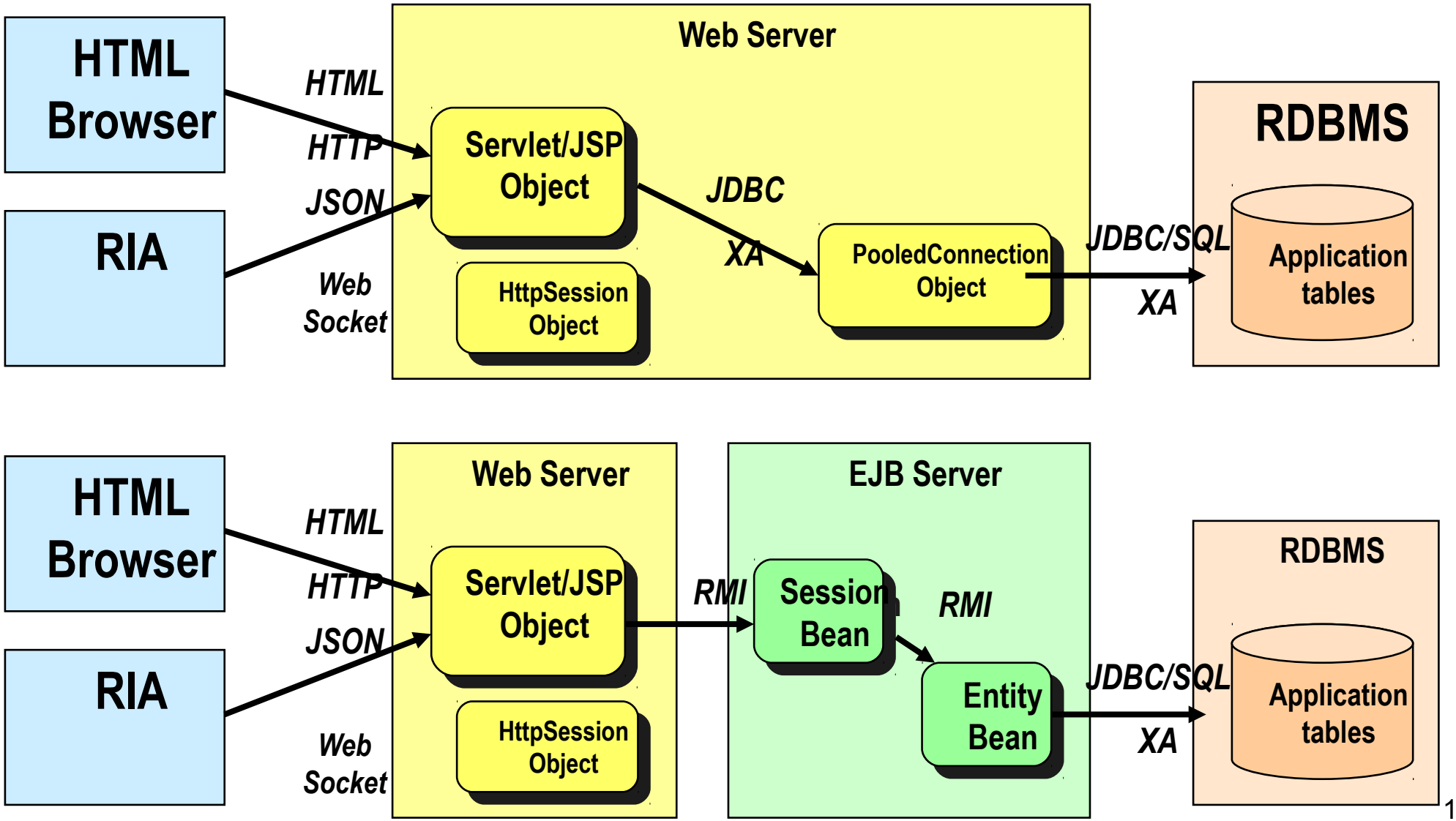


23/09/14

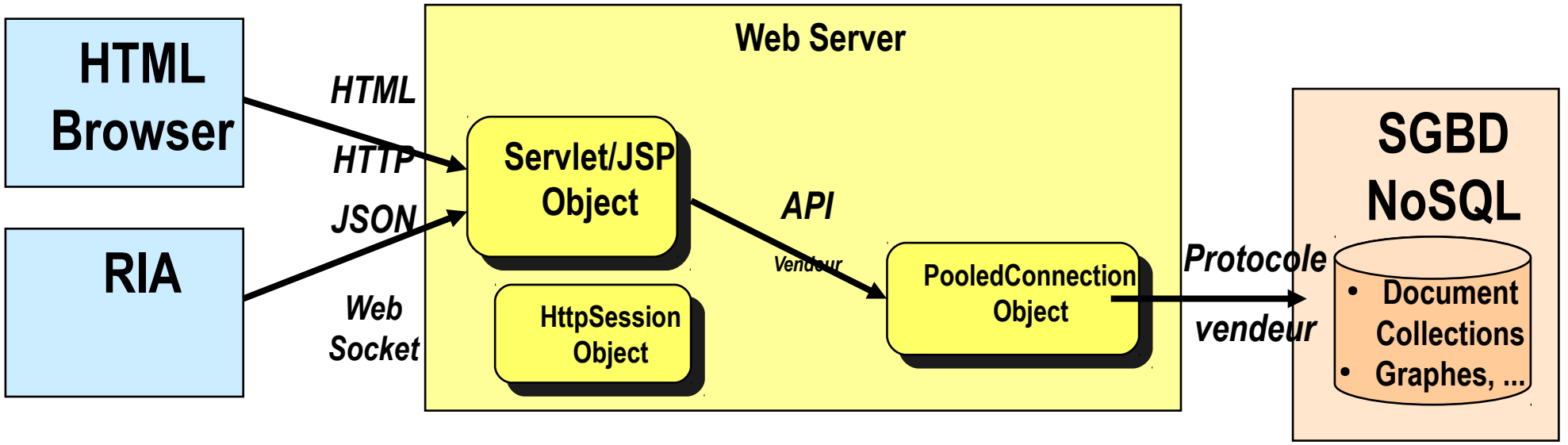
Architecture générale d'une application JavaEE (i)



Architecture générale d'une application JavaEE (ii)

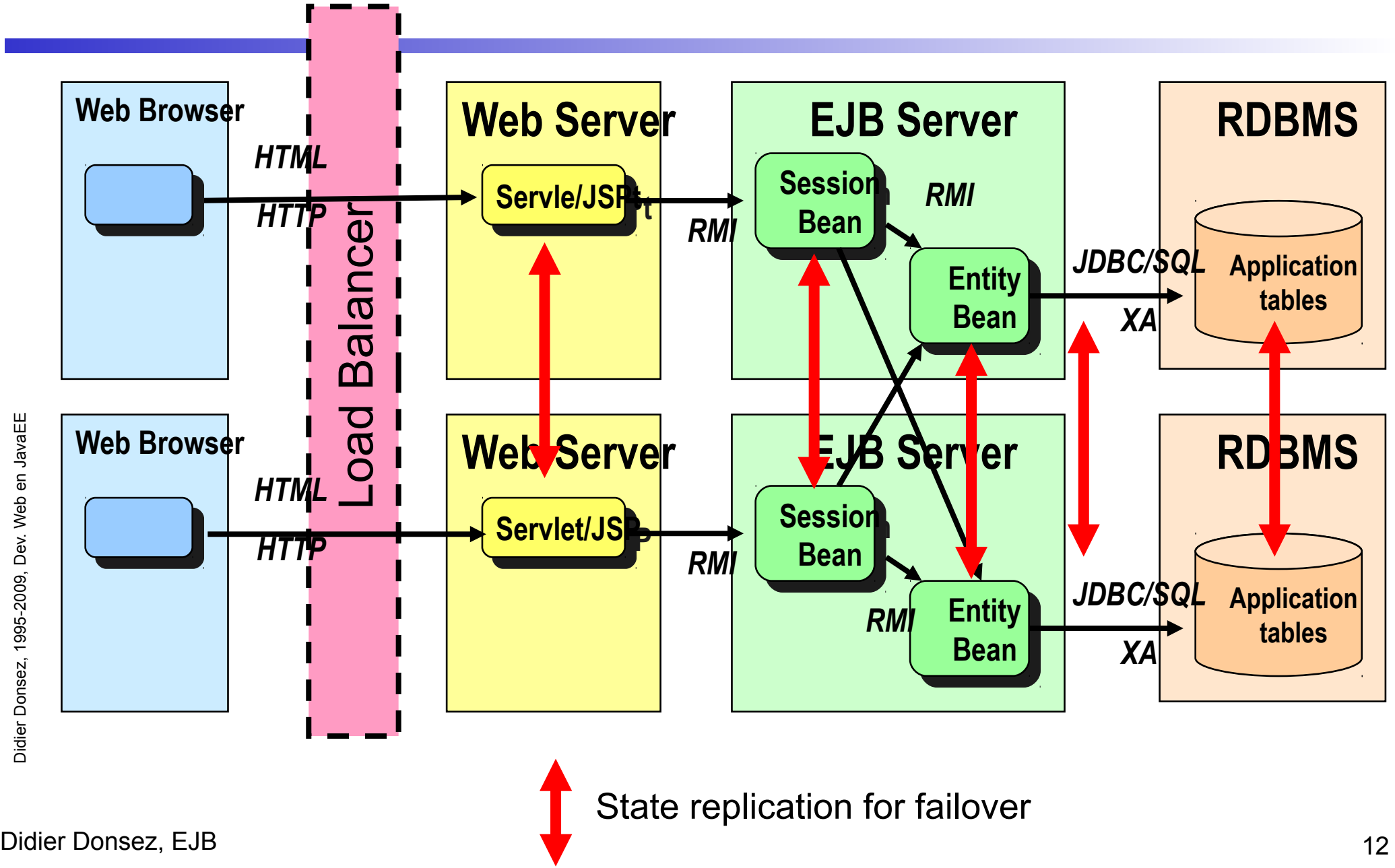


Architecture générale d'une application JavaEE (ii)



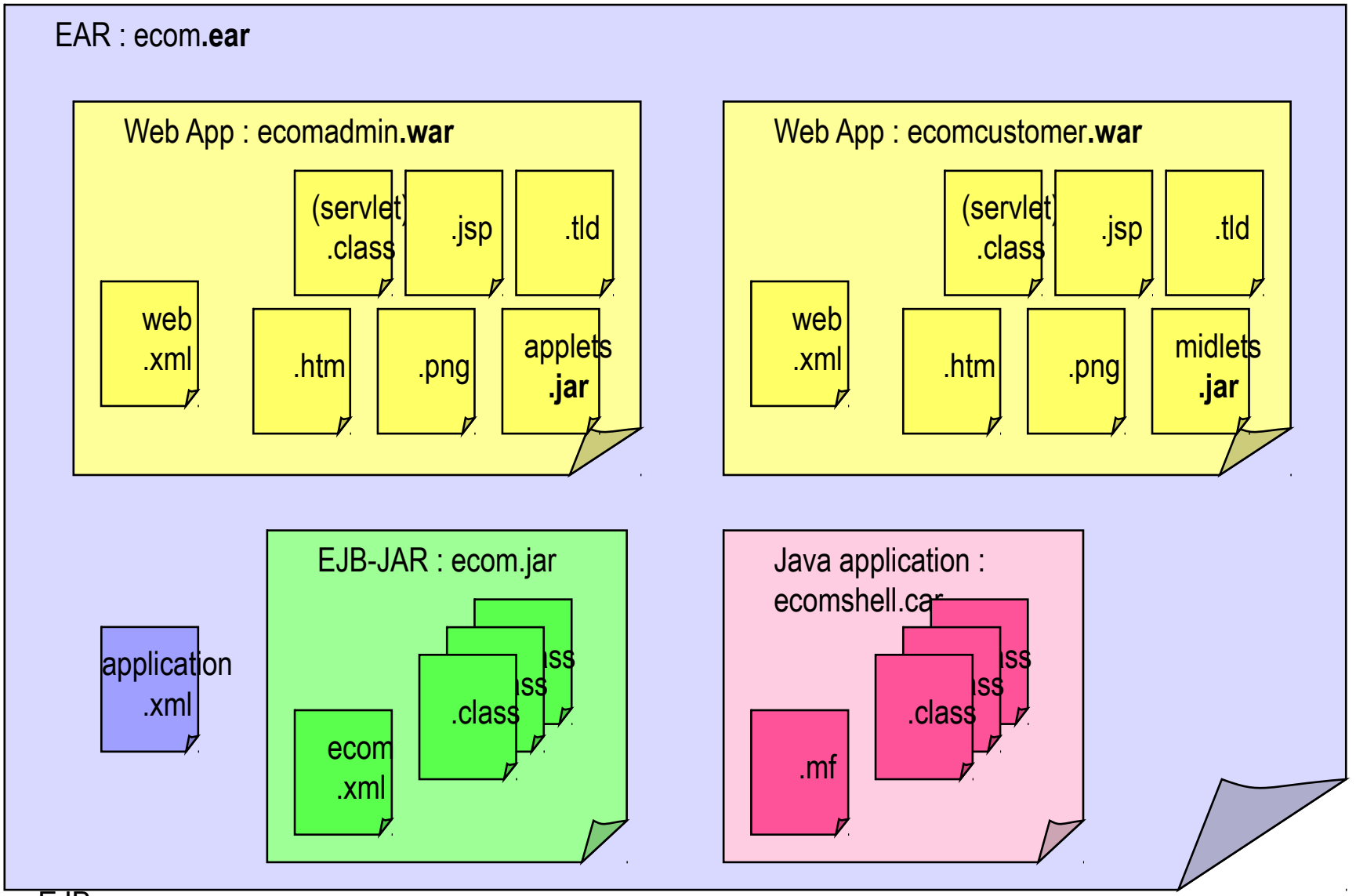
23/09/14

JavaEE en cluster + failover



23/09/14

JSR 88
Packaging d'une application JavaEE



Enterprise JavaBeans

- Composants *métier* coté serveur
  - → Couche métier de l'architecture 3-tiers
  - → Présentation réalisée par les Servlets ou côté client
- Différents types
  - Entity Bean : données persistantes (ORM)
  - Session Bean
    - Stateless : traitement métier
    - Stateful : associé à une session utilisateur
  - Message-Driven Bean : traitant de messages JMS
  - Timed Object : traitant déclenché à date fixe (et/ou périodique)
  - EQL, JPQL : requête directe de la base relationnelle
- Interfaces distantes / locales
  - RMI, CORBA, WebService, RESTFul services, JMS

Session Bean
Définition

- Représente un traitement de la logique métier
- Stateless SB (sans état)
  - ne conserve pas d'information entre 2 appels successifs
  - 2 instances qqconques d'un tel bean sont équivalentes
  - Annotation @Stateless
- Stateful SB (avec état)
  - État conservé attaché à la session d'un client
    - 1 instance par client (pendant la durée de la session)
  - Souvent attaché à la HttpSession de la webapp (Servlet/JSP)
  - Annotation @Stateful

Interfaces
@Remote vs @Local

- @Remote
  - Remote Business Method invoked via RMI, WS, ..
  - Parameters/Result passed **by value**
    - need to be serializable or DTO.
- @Local
  - Local business interface
    - Same JVM, Same Class Loader tree
  - Parameters/Result passed **by reference** from the client to the bean.

Session Bean Développement

- 1 ou n interfaces (@Local, @Remote, WS, REST)

```
@javax.ejb.Remote
interface IShippingCalculator {
   Price getShippingPrice(Address from, Address to,
         double weight, double volume) ;
}

@javax.ejb.Stateless(name="fedex-shipping")
class FedexShippingCalculator implements IShippingCalculator {
   Price getShippingPrice(Address from, Address to,
         double weight, double volume)  { … }
}
```

Opetionel :
Nom unique dans l'ejbjar

Session Bean Développement

```
public interface ItemLocal {
  List<Book> findBooks();
  List<CD> findCDs();
}

public interface ItemRemote {
  List<Book> findBooks();
  List<CD> findCDs();
  Book createBook(Book book);
  CD createCD(CD cd);
}

@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
  // ...
}
```

Session Bean Développement

```
@Local
public interface ItemLocal {
  List<Book> findBooks();
  List<CD> findCDs();
}
@WebService
public interface ItemSOAP {
  List<Book> findBooks();
  List<CD> findCDs();
  Book createBook(Book book);
  CD createCD(CD cd);
}
@Path("/items")
public interface ItemRest {
  List<Book> findBooks();
}
@Stateless
public class ItemEJB implements ItemLocal, ItemSOAP, ItemRest {
  // ...
}
```

Injection de la référence

- Injection du bean dans un client (Servlet, ...
  - @EJB(name="*jndiname*") ou @EJB *nom par défaut*

```
public class ShippingServlet extends HttpServlet {
        @EJB(name="fedex-shipping")
        private IShippingCalculator fedexCalc;
        @EJB(name="dhl-shipping")
        private IShippingCalculator dhlCalc;

        public void doGet(HttpServletRequest req, HttpServletResponse resp ) {
          resp.setContentType("text/html");
          PrintWriter out = resp.getWriter();
          ...
          out.println("<tr><td>Fedex</td><td>"+
              + fedexCalc.getShippingPrice(from,to,weight,volume) ;
              +"</td></tr>");
          out.println("<tr><td>DHL</td><td>"+
              + dhlCalc.getShippingPrice(from,to,weight,volume) ;
              +"</td></tr>");
          … }}
```

Stateful Session Bean

- Une instance par session client
  - Reste en mémoire tant que le client est présent
  - Expiration au bout d'un délai d'inactivité
  - Attaché à la session JSP/servlet
- Utilisation type
  - gestion d'un panier sur un site de commerce en ligne
  - rapport sur l'activité d'un client
- 2 annotations principales
  - @Stateful : déclare un EB avec état
  - @Remove : définit la méthode de fin de session
    - la session expire à l'issu de l'exécution de cette méthode

Stateful Session Bean

```
@Stateful
@StatefulTimeout(value = 15, unit = TimeUnit.MINUTS)
public class ShoppingCartBean implements IShoppingCart {
private List<Item> cartItems = new ArrayList<>();
public void addItem(Item item) { if (!cartItems.contains(item)) cartItems.add(item); }
public void removeItem(Item item) {  if (cartItems.contains(item)) cartItems.remove(item); }
public Integer getNumberOfItems() {  if (cartItems == null || cartItems.isEmpty()) return 0;
return cartItems.size(); }
public Float getTotal() {
  if (cartItems == null || cartItems.isEmpty()) return 0f;
  Float total = 0f; for (Item cartItem : cartItems) { total += (cartItem.getPrice()); } return total;
}
public void empty() { cartItems.clear(); }
@Remove
public void checkout() { cartItems.clear(); }
}
```

Singleton Session Bean

Pattern Singleton (1 instance par application → Partagé par tous)

```
@Singleton
@Lock(LockType.WRITE) // contrôle de concurrence
                      // pour toutes les méthodes sauf getFromCache
@AccessTimeout(value = 20, unit = TimeUnit.SECONDS)
public class CacheBean {
  private Map<Long, Object> cache = new HashMap<>();
  public void addToCache(Long id, Object object) {
    if (!cache.containsKey(id)) cache.put(id, object);
  }
  public void removeFromCache(Long id) {
    if (cache.containsKey(id)) cache.remove(id);
  }
  @Lock(LockType.READ)
  public Object getFromCache(Long id) {
    if (cache.containsKey(id)) return cache.get(id); else return null;
  }
}
```

Méthodes Asynchrones

- Gestion des traitements (longs/lents)

```
@Stateless
public class OrderBean {
@Resource
SessionContext ctx;
@Asynchronous
  public void sendEmailOrderComplete(Order order, Customer customer) { /* Very Long task */}
@Asynchronous
  public Future<Integer> sendOrderToWorkflow(Order order) {
     Integer status = 0;
     // processing
     status = 1;
     if (ctx.wasCancelCalled()) {
       return new AsyncResult<>(2);
     }
     // processing
    return new AsyncResult<>(status);
 }}}
```

Timed Object (i)

- Motivation
  - déclenchement d'actions périodiques
- Usage
  - ...
- Méthodes
  - Par Injection @Schedule
  - Programmatique via la ressource TimeService

Timed Object (ii)
Par Injection

```
public class NewsletterBean {
   @Schedule(second = "0", minute = "0", hour = "0", dayOfMonth = "1", month = "*", year = "*")
   public void sendMonthlyNewsletter() {
      // sends out the newsleter
   }
   @Schedules({
      @Schedule(second = "0", minute = "0", hour = "12",
      dayOfMonth = "Last Thu", month = "Nov", year = "*"),
      @Schedule(second = "0", minute = "0", hour = "12",
      dayOfMonth = "18", month = "Dec", year = "*")
   })
   @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
   public void sendHolidayNewsletter() {
      // Sends out a holiday newsletter...
   }
}
```

Timed Object (iii) via TimeService

```
@Stateless
public class BirthDayBean {
  @Resource
  TimerService timerService;
  @PersistenceContext(unitName = "bookshop")
  private EntityManager em;
  public void createCustomer(Customer customer) {
     em.persist(customer);
    ScheduleExpression birthDay = new ScheduleExpression()
        .dayOfMonth(customer.getBirthDay())
        .month(customer.getBirthMonth());
    timerService.createCalendarTimer(birthDay, new TimerConfig(customer, true));
  }
  @Timeout
  public void sendBirthdayEmail(Timer timer) {
    Customer customer = (Customer) timer.getInfo();
    // ...
    }
}
```

Entity Bean
Définition

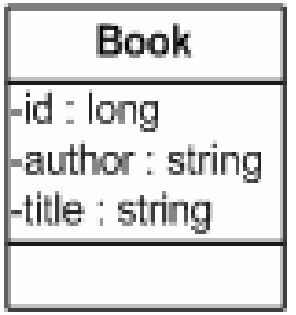
- Représentation d'une donnée persistante manipulée par l'applicati
  - stockée dans un SGBD (via l'API JDBC, API JPA)
  - Gestion automatique de chargement en mémoire et de la sauvegarde
  - Gestion des transactions ACID
  - Gestion de la concurrence (forte, faible)
- Correspondance objet – tuple relationnel (mapping O/R)
  - Notion d'@Id (clés primaires)
  - Notion de relations (1-1, 1-N, N-M)
    unidirectionnelles / bidirectionnelles entre des Entity Beans
  - Requêtage via JPQL

Entity Bean Développement

- POJO avec getter/setter + annotations
  - @Entity : déclare une classe correspondant à un EB
  - @Table(name="...") : optionnel, désigne la table
  - @Id : définit de la clé primaire
  - @Column(name="...") optionnel, désigne la colonne
- 2 modes (exclusif) de définition des colonnes des tables
  - property-based access : annotation des getters
  - field-based access : annotation des champs

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; } }
```



Entity Bean
Clés (i)

- Identifie la donnée (ligne) dans la table
- Clé primaire simple
  - @Id private String sku
- Clé générée (unique)
  - @Id @GeneratedValue(strategy=GenerationType.AUTO) // par le conteneur
  - @Id @GeneratedValue(strategy=GenerationType.SEQUENCE) // séquence à définir
- Clé Composite
  - @Id private String newsId ;
  - @Id private String lang ;

Entity Bean
Clés (ii)

- Clé Composite (suite)

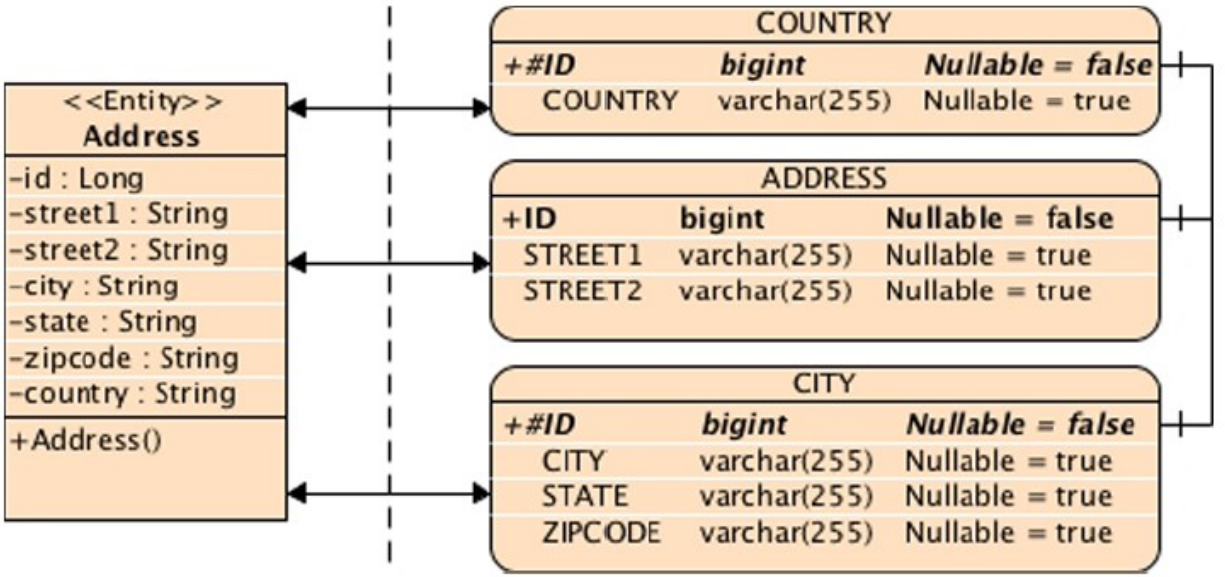
```
@Embeddable
public class NewsId {
  private String title;
  private String language;
}
@Entity
public class News {
  @EmbeddedId private NewsId id;
  private String content;
  ...
}

NewsId pk = new NewsId("Richard Wright ...", "EN")
News news = em.find(News.class, pk);
```

Entity Beans - DB Mapping (i)

```
@Entity
@Table(name="SHIPPING_REQUEST")
public class ExpressShippingRequest implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id @GeneratedValue(strategy = GenerationType.AUTO) private Long id ;
    @Column(name = "ITEM_ID") private String item;
    @Column(name = "SHIP_ADDRESS") private String shippingAddress;
    @Column(name = "SHIP_METHOD")  private String shippingMethod;
    @Column(name = "SHIP_INSURANCE_DOLLARS")  private double insuranceAmount;

    public Long getId() { return id; }
    public void setId(Long id) {  this.id = id; }
    public String getItem() {    return item; }
```

Entity Beans - DB Mapping (ii)

```
@Entity
@SecondaryTables({
  @SecondaryTable(name = "city"),
  @SecondaryTable(name = "country")
})
public class Address {
  @Id
  private Long id;
  private String street1;
  private String street2;
  @Column(table = "city")
  private String city;
  @Column(table = "city")
  private String state;
  @Column(table = "city")
  private String zipcode;
  @Column(table = "country")
  private String country;
  // Constructors, getters, setters
}
```



Entity Beans - DB Mapping (iii)

- Stratégies de chargement
  - EAGER et LAZY
- Exemple

```
@Entity public class CDTrack {
  @Id private String title;
  private String description ; private int duration ;
  @Basic(fetch = FetchType.LAZY) // chargement à l'invocation de getWav()
  @Lob // SQL Long OBject
  private byte[] wav;
  ...
}
```

Entity Manager

Point d'entrée principal du service de persistance

- permet de faire persister les beans
- permet d'exécuter des requêtes

- Accessible via une injection de dépendance
  - attribut de type javax.persistence.EntityManager
  - annoté par @PersistenceContext
- Méthode de javax.persistence.EntityManager
  - `void persist(Object o)`
  - `void remove(Object o)`
  - `<T> T find(Class<T> aClass, Object o)`
  - `Query createQuery(String query)`
  - `<T> T merge(T t)` // pour rattacher l'entity au container ejb (s'il a été sérializé)

Entity Manager

```
@Stateless
public class LibraryBean implements ILibrary {
    @PersistenceContext
     private EntityManager em;
     public int createBook(String author, String title, int year) {
          Book b = new Book(author, title, year);
           em.persist(b);
          return b.getId() ;
     }
     public Book findBook(int id) {
          Book b = en.find(Book.class, id) ;
         if (b==null) ….
         return b ;
     }
    ….
}
```

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

Entity Manager

```
@PersistenceContext
private EntityManager em;
 ...
public void updateBook(int id, String author, String title) {
   Book b = en.find(Book.class, id) ;
   if (b==null) return ;
   if (author!=null) b.setAuthor(author);
if (title!=null) b.setTitle(title);
 }
 public void updateBook(Book b) {
    em.merge(b);
 }
```

Entity Bean (Gestionnaire d'entités)

- Recherche par requête
  - ➢ requêtes SELECT dans une syntaxe dite EJB-QL étendue
  - ➢ 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

Entity Bean (Gestionnaire d'entités)

- Recherche par requête pré-compilée
  - ➢ création d'une requête nommée attachée à l'EB

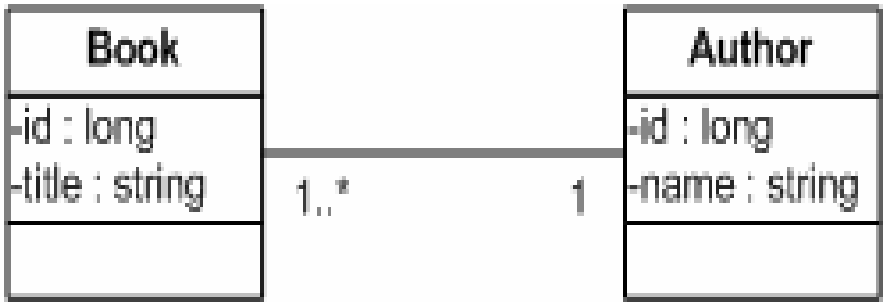
```
@Entity
@NamedQuery(name="allbooks",query="select OBJECT(b) from Book b")
public class Book { ... }

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 { ... }
```

Entity Bean (Relation)

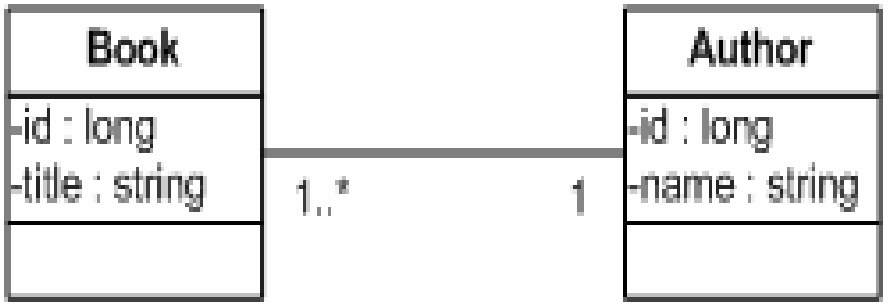
- Représente les relations entre les EBs
- @OneToOne, @OneToMany, @ManyToOne, @ManyToMany




```
@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(mappedBy="author")
        public Collection<Book> getBooks() { return books; }
        public void setBooks(Collection<Book> books) { this.books=books; }
        ...
}
```

Entity Bean (Relation 1-n)

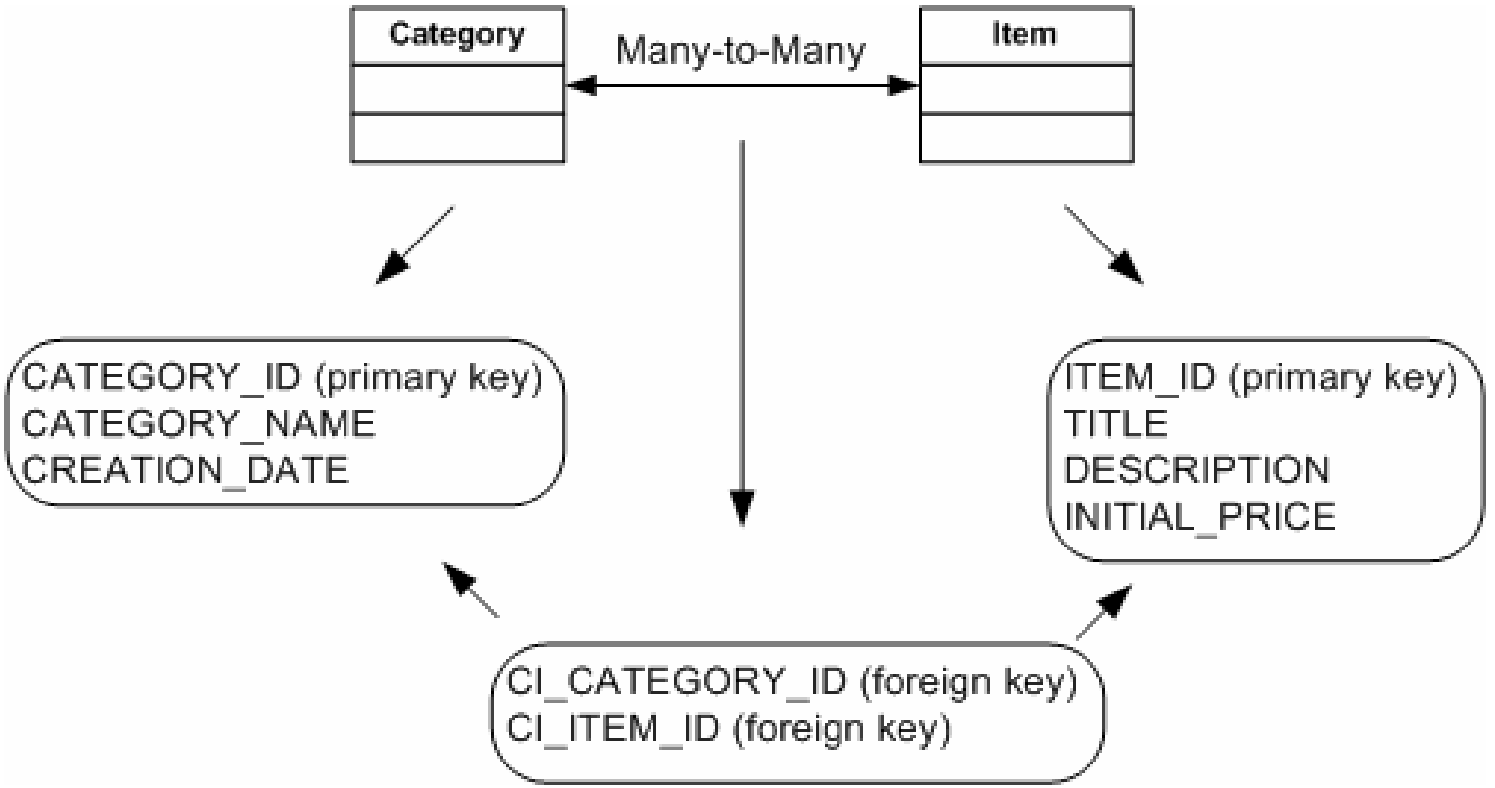
```
@Entity
public class Book {
      private long id;
      private Author author;
      private String title;
      public Book() {}
      public Book(Author author, String title) {
            this.author = author;
            this.title = title; }
      @ManyToOne
      @JoinColumn(name="Author_id")
      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

Entity Bean (Relation n-n)

- Notion de table de jointure



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;
```

Entity Bean (annotations liées aux relations)

- **Mode de chargement d'une relation**
  - Attribut Fetch sur l'annotation d'une relation
  - EAGER ou LAZY
- **Persistance ou suppression en cascade pour une relation**
  - Attribut Cascade sur l'annotation d'une relation
  - CascadeType.ALL, CascadeType.PERSIST ...

ex :

```
@OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
```

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;
```

- ...

Message-Driven Bean

- Rappel : Java Messaging Service
  - API asynchrone d'envoi et de réception de message entre applications via un MOM
  - 2 Modèles
    - Topics : Publish / Subscribe
      - Un message publié est recu par tous les soucripteurs
    - Queue : Producer / Consumer
      - Un message produit n'est recu que par un des consommateurs
- Usages
  - Désynchroniser des traitements dans l'application
    - Performance
  - EAI Patterns (ESB, ...)

Message-Driven Bean

- Représente un consommateur / souscripteur à une Queue / Topic dans l'application (ejb-jar)
  - Annotation @MessageDriven
  - Implante Interface MessageListener { void onMessage(Message m); }
- Exemple de Consommateur

```
@MessageDriven(ActivationConfig={
 @ActivationConfigProperty(
   propertyName= "destination", propertyValue="ShippingRequestQueue"),
 @ActivationConfigProperty(
   propertyName= "destinationType", propertyValue="javax.jms.Queue"),
})
public class ShippingProcessBean implements MessageListener {
  @Resource
  private MessageDrivenContext mdc;
  public void onMessage( Message m ) {
    TextMessage message = (TextMessage) m;
    ...
  }
}
```

Producteur JMS

- Injection des Ressources JMS
- Exemple

```
@Stateless
public class OrderProcessBean{
  @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 ship(Order order) {
   Connection connection = connectionFactory.createConnection();
   Session session =
      connection.createSession(true,Session.AUTO_ACKNOWLEDGE)
   MessageProducer producer = session.createProducer(destination);

   TextMessage message = session.createTextMessage();
   message.setText(order.toJSON());
   producer.send(message);

   session.close();
   connection.close();
} }
```

Service de Nommage JNDI

- Références vers composants ou 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/
    - java:comp/env/myOtherBean
    - java:comp/env/javax.user.Transaction

EJB References

- JNDI Lookup

```
Context context = new InitialContext();
HelloUserBean helloUser = (HelloUserBean) context.lookup("java:module/HelloUserBean");
helloUser.sayHello("George") ;
```

- EJB Injection

```
@EJB
private HelloUserBean helloUser;
void hello(){
  helloUser.sayHello("George");
}
```

- CDI Dependency Injection

```
@Inject
private HelloUserBean helloUser;
void hello(){
  helloUser.sayHello("George");
}
```

Transaction Gestion Declarative

- Support de l'architecture XA (X/Open DTP)
  - extensions standards à JDBC 2.0, JCA
  - Java Transaction API (JTA) sur des ressources XA
    - SGBD-R, Queue et Topic JMS, ...
- Modèle de transaction plat ACID
  - pas de transactions imbriquées (nested)
- Gestion de la démarcation des transactions
  - Container-Managed Transaction (CMT)
    - Prise en charge par le container en fonction du contexte transaction de l'appelant
  - Bean-Managed Transaction (BMT)
    - Contrôle de la démarcation par le développeur via JTA
- Ressources XA
  - Entity Beans, les Stateful Session Beans, Message-Driven Beans

Transaction

Container-Managed Transaction (CMT)

- Démarcation prise en charge par le conteneur
- Exemple

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class BankBean implements IBank {
  @PersistenceContext private EntityManager em;
  @Resource private UserTransaction ut;

 @TransactionAttribute(TransactionAttributeType.SUPPORTS)
  public int getBalance(String iban) {
        Account a = em.find(Account.class, iban);
        return a.getBalance();
  }
 @TransactionAttribute(TransactionAttributeType.REQUIRED)
  public void transfert(String ibanFrom, String ibanTo,int amount) {
    try {
        Account af = em.find(Account.class, ibanFrom);
        Account at = em.find(Account.class, ibanTo);
        at.credit(amont);
        af.widthdraw(amont);
    } catch( Exception e ) {
        sc.setRollbackOnly();
  } } }
```

Transaction CMT

Démarcation @TransactionAttribute(TransactionAttributeType.XXX)

Beans Transaction Attribute TransactionAttributeType.	Client's Transaction	Transaction associated bean's method
TX_NOT_SUPPORTED	None T1	None None
TX_NEVER	None T1	None Error
TX_REQUIRED	None T1	T2 T1
TX_SUPPORTS	None T1	None T1
TX_REQUIRES_NEW	None T1	T2 T2
TX_MANDATORY	None T1	Error T1
TX_BEAN_MANAGED	None T1	

Transaction

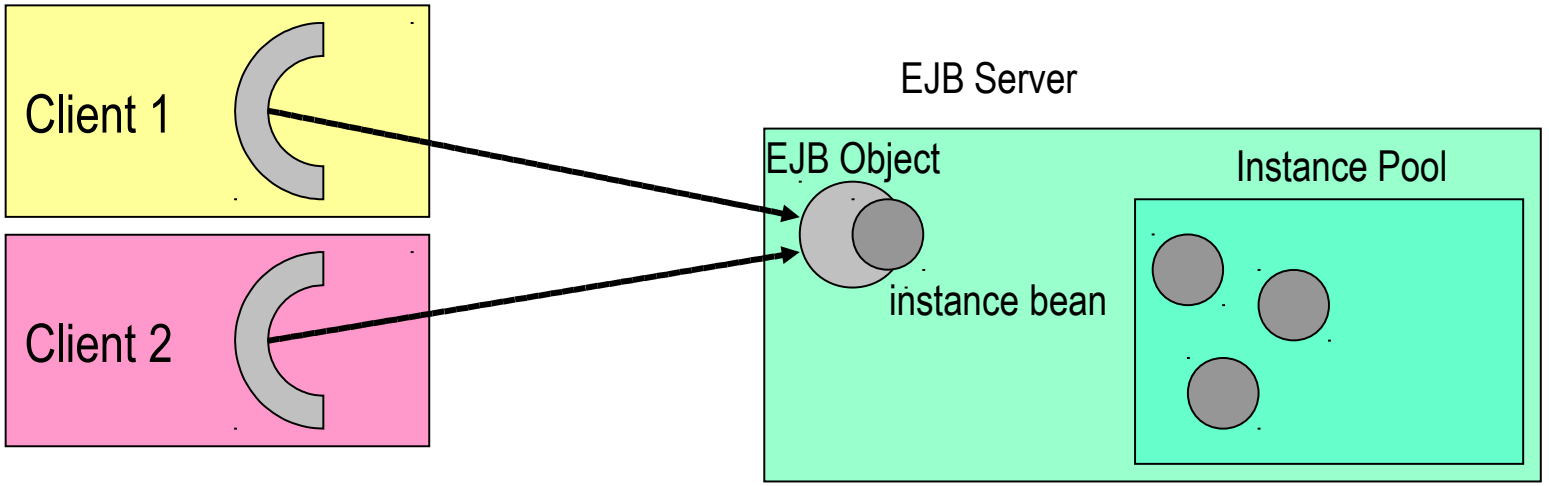
Bean-Managed Transaction (BMT)

- Démarcation explicite l'API JTA begin/commit/rollbac
  - granularité plus fine qu'en CMT
- Exemple

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class BankBean implements IBank {
  @PersistenceContext private EntityManager em;
  @Resource private SessionContext sc;
  public void transfert(String ibanFrom, String ibanTo,int amount) {
    try {
      ut.begin();
        Account af = em.find(Account.class, ibanFrom);
        Account at = em.find(Account.class, ibanTo);
        at.credit(amont);
        af.widthdraw(amont);
        ut.commit();
    } catch( Exception e ) {
        ut.rollback();
  } } }
```

Gestion de la Concurrence

- Accès de plusieurs clients à un Entity Bean



- Plusieurs problèmes
  - Lecture Sale (Dirty Read)
    - lecture des modifications d ’une autre transaction qui avortera
  - Lecture non répétable (Non repeatable read)
    - la transaction lit successivement deux valeurs différentes d ’une même donnée
  - Lecture Fantôme (Phantom Read)
    - une transaction ne lit pas des données insérées précédenment

Gestion de la Concurrence

- La source de données « verrouille » la donnée correspondant en fonction du type d ’accès
  - Read Lock
    - la valeur n ’est pas changée par les autres transactions
  - Write Lock
    - la valeur n ’est pas changée par d ’autres transactions
  - Exclusive Write Lock
    - les autres transactions sont bloquées jusqu ’au relachement
  - Snapshot
    - chaque transaction possède une valeur figée
- Une transaction sera bloquée jusqu ’au relâchement du verro en fonction du niveau d ’isolation souhaité

Gestion de la Concurrence

- Définition des niveaux d ’isolation de la transaction
  - ISOLATION LEVEL (même définition que SQL et JDBC)

Isolation Level	Sale	Non Répétable	Fantômes
TRANSACTION_READ_UNCOMMITED	oui	oui	oui
TRANSACTION_READ_COMMITED	non	oui	oui
TRANSACTION_REPEATABLE_READ	non	non	oui
TRANSACTION_SERIALIZABLE	non	non	non

Bean Validation

- Motivation
  - Contraintes sur les valeurs de champs de *beans*
- Contraintes génériques *(builtins)*
  - **@NotNull @Size @Min @ Max @Past ...**
- Contraintes applicatives
  - **@Email @ISBN10 @CreditCard @VISACreditCard, @ChronologicalDates**

Bean Validation Builtin Constraints

Constraint	Accepted Types	Description
`AssertFalse` `AssertTrue`	`Boolean, boolean`	The annotated element must be either false or true
`DecimalMax` `DecimalMin`	`BigDecimal, BigInteger, CharSequence, byte, short, int, long, and respective wrappers`	The element must be greater or lower than the specified value
`Future` `Past`	`Calendar, Date`	The annotated element must be a date in the future or in the past
`Max` `Min`	`BigDecimal, BigInteger, byte, short, int, long, and their wrappers`	The element must be greater or lower than the specified value
`Null` `NotNull`	`Object`	The annotated element must be null or not
`Pattern`	`CharSequence`	The element must match the specified regular expression
`Digits`	`BigDecimal, BigInteger, CharSequence, byte, short, int, long, and respective wrappers`	The annotated element must be a number within accepted range
`Size`	`Object[], CharSequence, Collection<?>, Map<?, ?>`	The element size must be between the specified boundaries

Bean Validation Constraints examples

```
public class Customer {
  @NotNull @Pattern(regexp = "^[A-Za-z0-9]+$")
  private String userId;
  @NotNull @Size(min = 4, max = 50, message = "Firstname should be between {min} and {max}")
  private String firstName;
  private String lastName;
  @Email(message = "Recovery email is not a valid email address")
  private String recoveryEmail;
  @Pattern(regexp="^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$", message="{invalid.phonenumber}")
  private String phoneNumber;
  @Min(value = 18, message = "Customer is too young. Should be older that {value}")
  private Integer age;
  @CreditCard(message = "{value} is an incorrect card number")
  private String creditCardNumber;
  // Constructors, getters, setters
}
```

Bean Validation
Date Contraints

```
@ChronologicalDates
public class Order {
  ...
  private Date creationDate;
  private Date paymentDate;
  private Date deliveryDate;
  public Order(@Past Date creationDate) {
    this.creationDate = creationDate;
  }
}
```

Bean Validation
Custom Validator

```
@Pattern.List({
  @Pattern(regexp = "[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\."
    +"[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*"
    +"@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?")
})
@Constraint(validatedBy = {})
@Documented
@Target({ElementType.METHOD,
    ElementType.FIELD,
    ElementType.ANNOTATION_TYPE,
    ElementType.CONSTRUCTOR,
    ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
public @interface Email {
    String message() default "{invalid.email}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
    @Target({ElementType.METHOD,
        ElementType.FIELD,
        ElementType.ANNOTATION_TYPE,
        ElementType.CONSTRUCTOR,
        ElementType.PARAMETER})
    @Retention(RetentionPolicy.RUNTIME)
    @Documented
    @interface List {
        Email[] value();
    }
}
```

http://docs.oracle.com/javaee/6/tutorial/doc/gkfgx.html

Design Pattern (DP)

Problèmes de conception récurrents

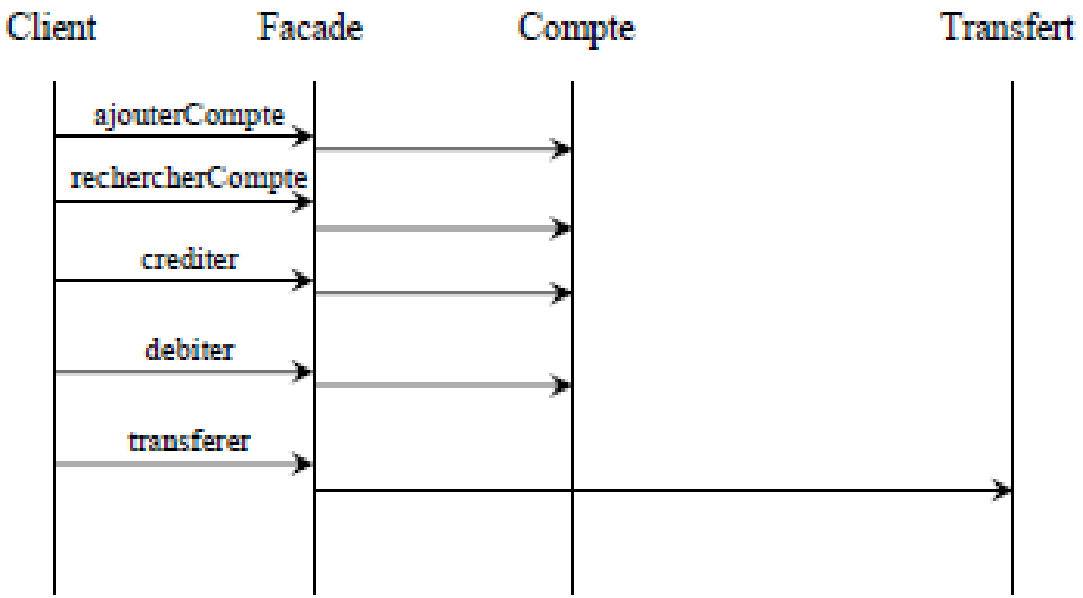
→ Design pattern (DP)

- solutions reconnues (et cataloguées) d'organisation du code

- Objectifs
  - améliorer la clarté et la compréhension du code
  - mettre en avant des éléments d'architecture logicielle
- DP pour Java EE
  - Facade (stateless vs stateful)
  - Data Tranfert Object (DTO)

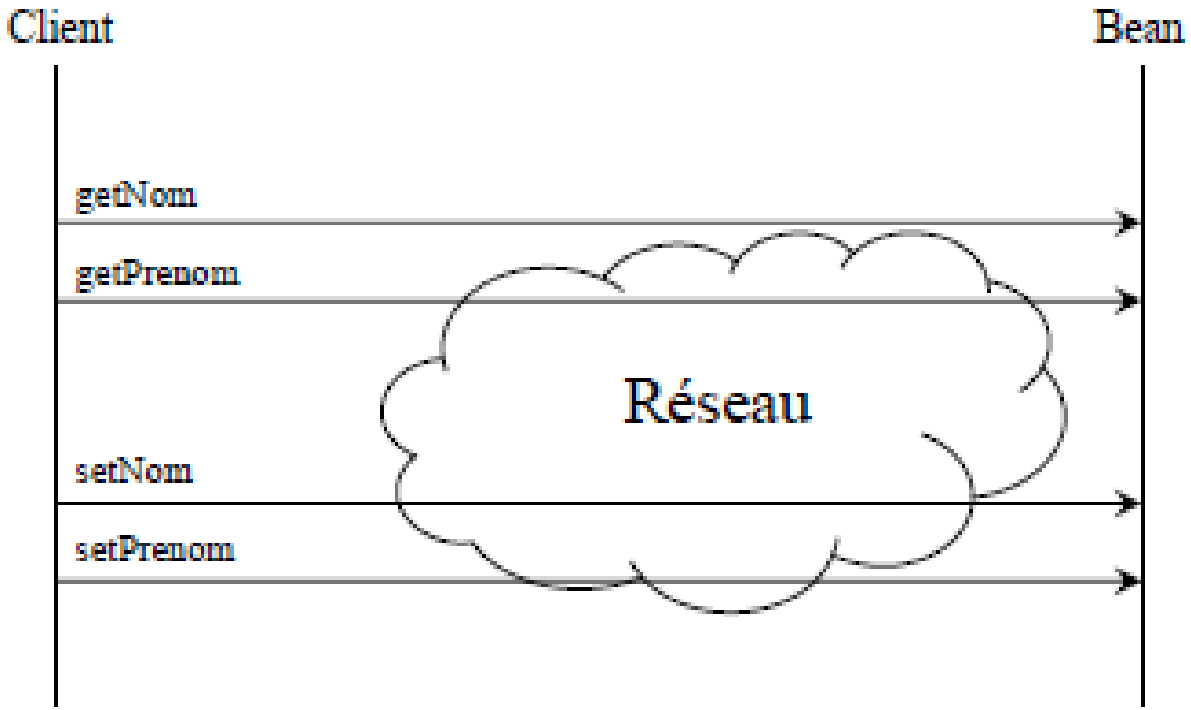
Design Pattern
DP Session facade

- Pb : nombreuses dépendances entre les clients et les beans
- Solution : présenter aux clients **une seule interface façade** (*stateless session bean*)
- Expose les traitements minimaux
- Eventuellement plusieurs façades (multiple rôles)



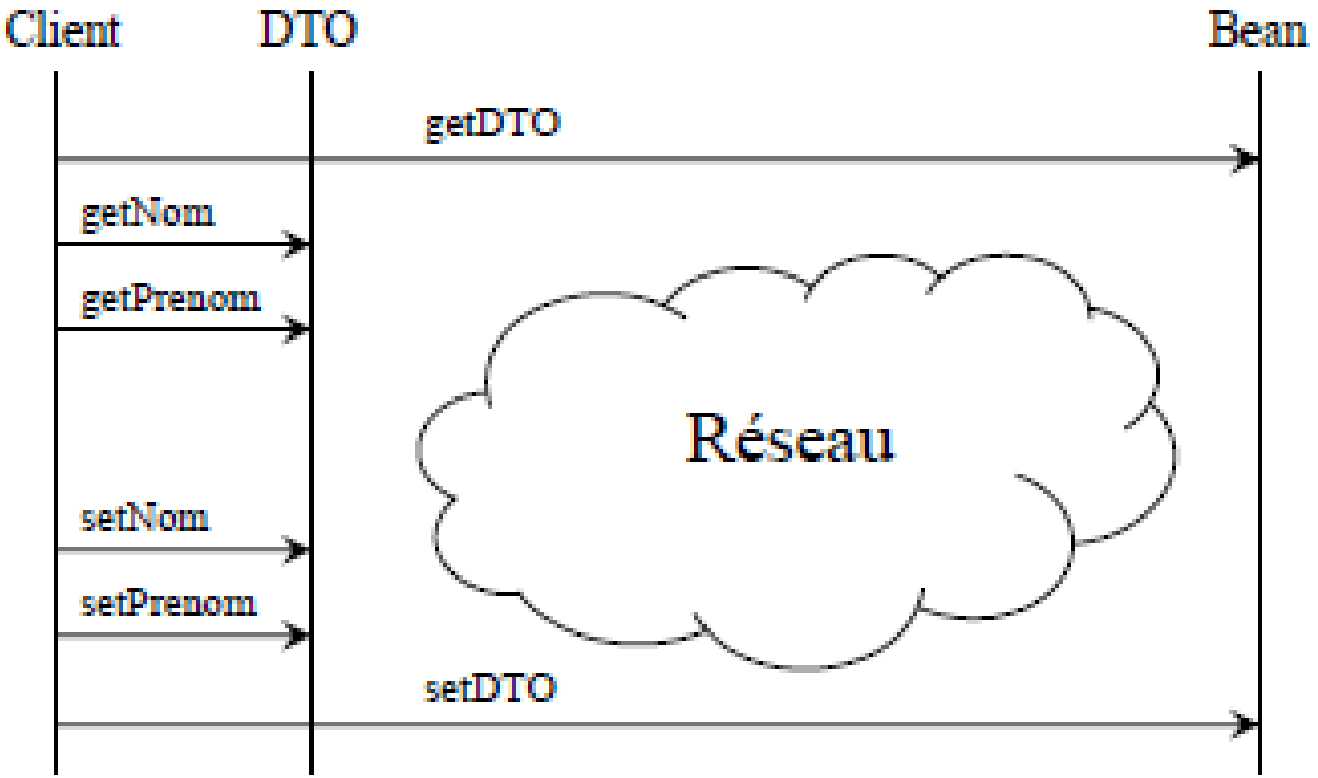
Design Patterns : DTO

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



Design Patterns : DTO

Solution : transmettre une instance par valeur



Testable POJO components

```
@RunWith(Arquillian.class)
public class HelloUserBeanTest {
@EJB
private HelloUser helloUser;
@Deployment
public static Archive<?> createDeployment() {
  return ShrinkWrap.create(JavaArchive.class, "hello.jar")
  .addClasses(HelloUserBean.class);
}
@Test
public void testSayHello() {
  String helloMessage = helloUser.sayHello("George");
  Assert.assertEquals("Hello George",helloMessage);
}
}
```

Conclusion

- Applications complexes “facile” à écrire
  - gestion de la persistance
  - gestion des transactions de manière déclarative
  - gestion intégré de la sécurité
  - gestion de la répartition
- Indépendance entre applications et plate-formes
- Mais
  - Evolution permanente ...

Livres EJB

- Debu Panda, Reza Rahman, Ryan Cuprak, and Michael Remijan, EJB 3 in Action, Second Edition, March 2014, ISBN: 9781935182993,
- Andrew Lee Rubinger, Bill Burke, Enterprise JavaBeans 3.1, 6th Edition, O'Reilly Media, Final Release Date: September 2010, Pages

Livres Java EE

- Arun Gupta, Java EE 7 Essentials, Enterprise Developer Handbook, O'Reilly Media, August 2013
- Antonio Goncalves, Beginning JavaEE 7, APress, 2013
- Mick Knutson, Java EE6 Cookbook for securing, tuning, and extending enterprise applications, Packt Publishing, June 2012

Extra

Annotations vs XML

Annotation	Type	Annotation Element	Corresponding Descriptor Element
`@Stateless`	EJB type		`<session-type>Stateless`
		`name`	`ejb-name`
`@Stateful`	EJB type		`<session-type>Stateful`
			`ejb-name`
`@MessageDriven`	EJB type		`message-driven`
		`name`	`ejb-name`
`@Remote`	Interface type		`remote`
`@Local`	Interface type		`local`
`@Transaction-Management`	Transaction management type at bean level		`transaction-type`

Annotations vs XML

Annotation	Type	Annotation Element	Corresponding Descriptor Element
`@Transaction-Attribute`	Transaction settings method		`container-transaction trans-attribute`
`@Interceptors`	Interceptors		`interceptor-binding interceptor-class`
`@ExcludeClass-Interceptors`	Interceptors		`exclude-class-interceptor`
`@ExcludeDefault-Interceptors`	Interceptors		`exclude-default-interceptors`
`@AroundInvoke`	Custom interceptor		`around-invoke`
`@PreConstruct`	Lifecycle method		`pre-construct`
`@PostDestroy`	Lifecycle method		`post-destroy`
`@PostActivate`	Lifecycle method		`post-activate`
`@PrePassivate`	Lifecycle method		`pre-passivate`

Annotations vs XML

Annotation	Type	Annotation Element	Corresponding Descriptor Element
`@DeclareRoles`	Security setting		`security-role`
`@RolesAllowed`	Security setting		`method-permission`
`@PermitAll`	Security setting		`unchecked`
`@DenyAll`	Security setting		`exclude-list`
`@RunAs`	Security setting		`security-identity` `run-as`

`@Resource`	Resource references (`DataSource`, JMS, Environment, mail, etc.)		`resource-ref`
			`resource-env-ref`
			`message-destination-ref`
			`env-ref`

	Resource injection	Setter/field injection	`injection-target`
`@EJB`	EJB references		`ejb-ref` `ejb-local-ref`
`@Persistence-Context`	Persistence context reference		`persistence-context-ref`
`@PresistenceUnit`	Persistence unit reference		`persistence-unit-ref`

Annotations vs XML

Annotations Grouped by Type	XML Element
Object type	
`@Entity`	`entity`
`@MappedSuperClass`	`mapped-superclass`
`@Embedded`	`embedded`
`@Embeddable`	`embeddable`
Table mapping	
`@Table`	`table`
`@SecondaryTable`	`secondary-table`

Annotations vs XML

Annotations Grouped by Type	XML Element
Query	
`@NamedQuery`	`named-query`
`@NamedNativeQuery`	`named-native-query`
`@SqlResultsetMapping`	`sql-result-set-mapping`
Primary key and column mapping	
`@Id`	`id`
`@IdClass`	`id-class`
`@EmbeddedId`	`embedded-id`
`@TableGenerator`	`table-generator`
`@SequenceGenerator`	`sequence-generator`
`@Column`	`column`
`@PrimaryKeyJoinColumn`	`primary-key-join-column`
`@GeneratedValue`	`generated-value`

Annotations vs XML

Annotations Grouped by Type	XML Element
Relationship mapping	
@ManyToMany	many-to-many
@OneToOne	one-to-one
@OneToMany	one-to-many
@ManyToOne	many-to-one
@JoinTable	join-table
@JoinColumn	join-column
@InverseColumn	inverse-join-column
Listeners	
@ExcludeDefaultListeners	exclude-default-listeners
@ExcludeSuperClassListeners	exclude-superclass-listeners
@PreUpdate	pre-update

Annotations vs XML

Annotations Grouped by Type	XML Element
Listeners *(continued)*	
`@PostUpdate`	`post-update`
`@PrePersist`	`pre-persist`
`@PostPersist`	`post-persist`
`@PreRemove`	`pre-remove`
`@PostRemove`	`post-remove`
`@PostLoad`	`post-load`

EJB Lite vs EJB

Feature	EJB Lite	EJB
Stateless beans	☑	☑
Stateful beans	☑	☑
Singleton beans	☑	☑
Message-driven beans		☑
No interfaces	☑	☑
Local interfaces	☑	☑
Remote interfaces		☑
Web service interfaces		☑

Feature	EJB Lite	EJB
Asynchronous invocation	☑	☑
Interceptors	☑	☑
Declarative security	☑	☑
Declarative transactions	☑	☑
Programmatic transactions	☑	☑
Timer service	☑	☑
EJB 2.x support		☑
CORBA interoperability		☑