Architecture JEE

Master 2 Génie Logiciel

Présentation de l'UE

- Abla NAIT, SmartTrade Technologies, ingénieur développement logiciel (Aix-en-provence)

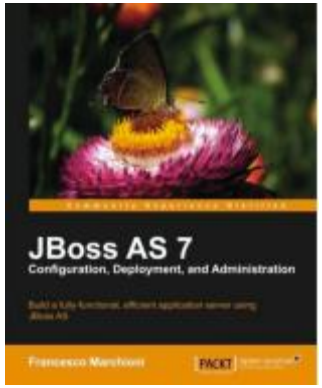
- Planning de l'UE : 5 sessions de 2h cours + 4h TP.

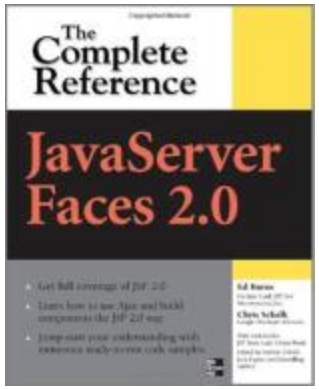
Programme du cours

- Introduction à JEE : histoire et enjeux
- Les enterprise java beans (EJB 3.1), Tp noté
- La persistance en JEE (JPA 2.0), Tp noté
- Le framework de présentation : Java Server Faces (JSF 2.1), Tp noté
- Sécurité (JAAS), mapping XML-Objet (JAXB), web services (JAWS), Tp noté

Environnement de développement

- IDE : Eclipse Juno(4.2) / Kepler(4.3).
  Plugin Eclipse : Jboss Tools 3.x/4.0, Oxygen XML
- Serveur d'applications : JBoss 7.1
- Base de données : PostgreSQL 9.x





Bibliographie

Java EE 6 et GlassFish 3, 2009. Antonio Goncalvès. Français.

JBoss AS 7 Configuration, Deployment and Administration, 2011. Francesco Marchioni. Anglais.

JavaServer Faces 2.0 : the complete reference, Chris Schalk. Anglais.

JPA
Java Persistence API

Documentation

Documentation de reference Spring

Documentation de reference Hibernate 3.6

La FAQ Hibernate par Developpez

La FAQ Maven2 par Developpez

Modalités d'évaluation

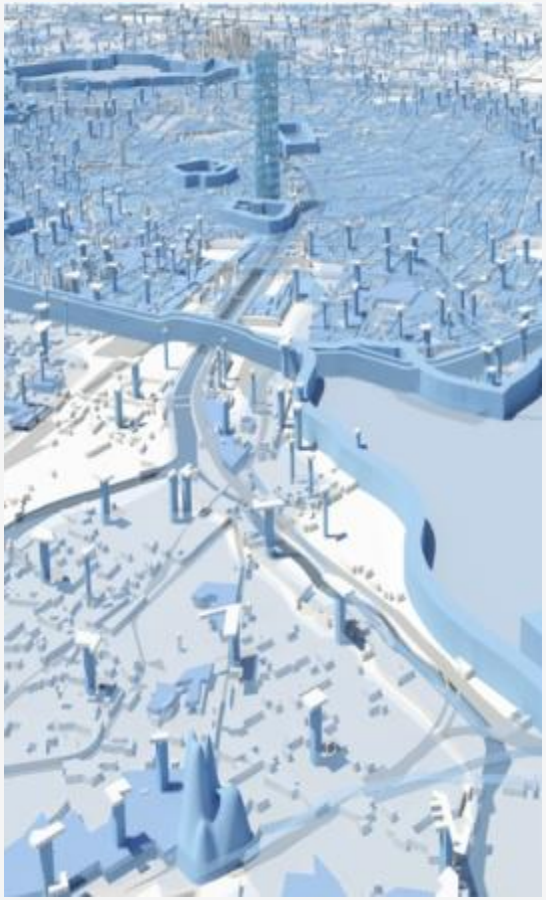
Note finale

= examen écrit en fin de semestre sur papier + 4 TPs noté sur 5

Urbanisation et SI

Organiser le SI de façon suffisamment modulaire de manière à :

- Favoriser son évolutivité, sa pérennité et son indépendance,
- Renforcer sa capacité à intégrer des solutions hétérogènes (progiciels, éléments de différentes plateformes, etc.
- Renforcer la capacité à faire interagir les composants du SI entre eux et avec d'autres SI (interopérabilité)
- Capacité à pouvoir remplacer certains composants (interchangeabilité)



LA NEBULEUSE JAVA

- Java Micro Edition (JME)
  - développement d'applications embarquées
- Java Standard Edition (JSE)
  - développement d'applications classiques
- **Java Enterprise Edition (JEE)**
  - développement d'applications d'entreprise

Applications d'entreprise

réduction des temps et coûts de développement

portables

disponibles

sécurisée

maintenables

montée en charge

sûres

extensibles

intégrables

adaptables

qualité du code

répondent aux besoins exprimés par les utilisateurs !

Architectures applicatives

- applications centralisées
- applications clients / serveurs
- **applications distribuées**

Les besoins exprimés

- Besoins de normalisation
  - Etablir un ensemble de règles ayant pour objet de simplifier et de rationaliser la production
- Besoins d'abstraction
  - Opération de désolidariser un objet de son contexte
- Besoins de communication
- Besoins de composants

Besoins de normalisation

- Pour que les applications soient
  - intégrables
  - communicantes / distribuées
  - adaptables
  - maintenables
  - portables

Besoins d'abstraction

- Pour que les applications soient
  - portables
  - maintenables
  - extensibles
  - intégrables/distribuées
  - adaptables

Besoins de communication

- Pour que les applications soient
  - intégrables
  - sécurisées
  - distribuées

Besoins de composants

- Pour que les applications soient
  - maintenables
  - sûres
  - extensibles
  - adaptables
  - portables
  - disponibles/distribuées

et surtout…

Comment réduire les temps et les coûts de développement et d'évolutions d'une application ?

Quelques principes …

Principe d'ouverture/fermeture

Les composantes d'un application doivent être ouvertes à extension mais fermées à modification !

Principe de substitution de Liskov

Si S est un sous-type de T, alors les objets de type T peuvent être remplacés avec des objets de type S

Principe de substitution de Liskov

Programmation par contrat:

- Préconditions ne peuvent être plus fortes dans une sous-classe
- Postconditions ne peuvent être plus faibles dans une sous-classe

Une seule responsabilité

Chaque objet ne doit avoir qu'une seule responsabilité !

Comment respecter ces grands principes?

Une partie de la solution…

- Des paradigmes de programmation
- Des patrons de conception
- Des frameworks
- Des composants

Objectif

Avoir une « plateforme » pour développer des applications d'entreprise rapidement, de qualités, sûres, sécurisées, portables, performantes, disponibles, maintenables, extensibles et ce... à moindre coûts !

Définition

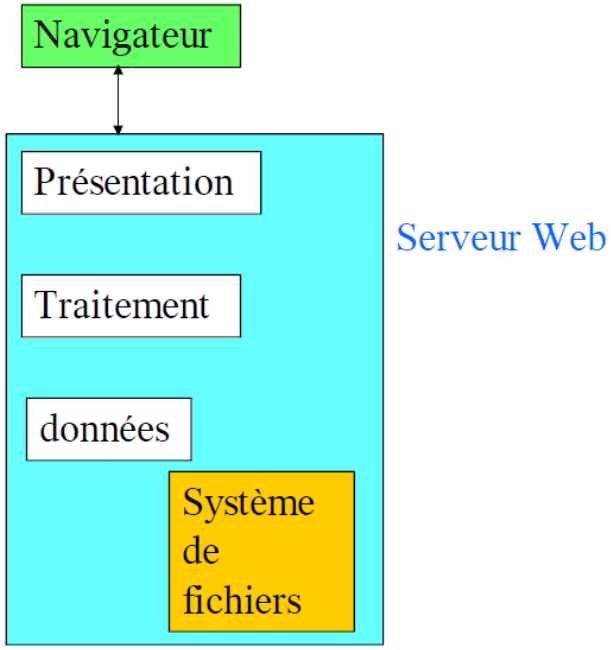
Java Enterprise Edition est une norme proposée par Sun visant à définir un standard de développement d'applications d'entreprises multi-niveaux basées sur des composants.

Principes d'architecture…

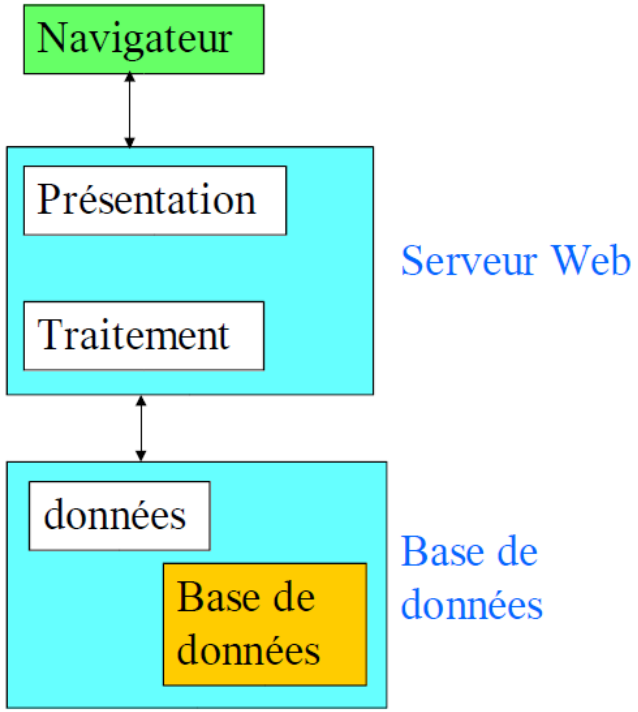
Architecture multi-niveaux

- Un niveau par besoin fonctionnel
- Augmentation de la cohésion du code
- Découplage fort entre les couches
- Code plus facilement réutilisable

Architecture 1-tiers



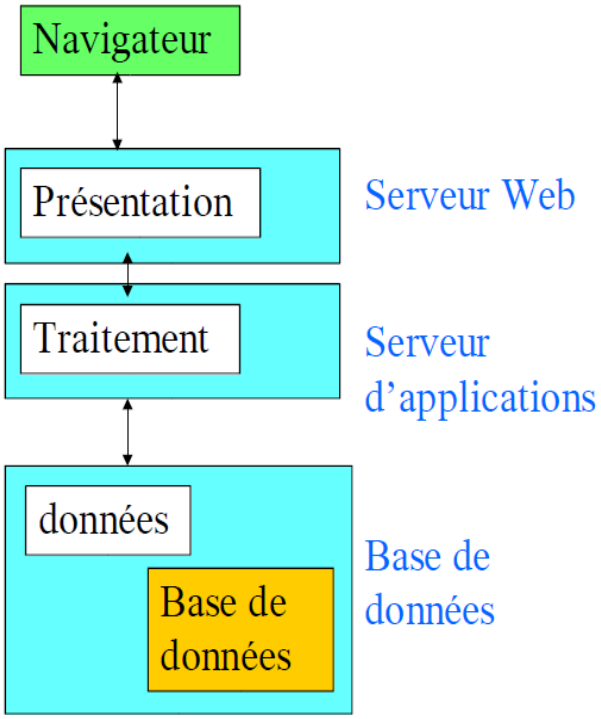
Architecture 2-tiers



Architecture 3-tiers

- \+ Dimensionnement
- \+ Maintenance
- \+ Fiabilité
- \+ Disponibilité
- \+ Extensibilité
- \+ Gestion du développement

- \- complexité



Architecture JEE

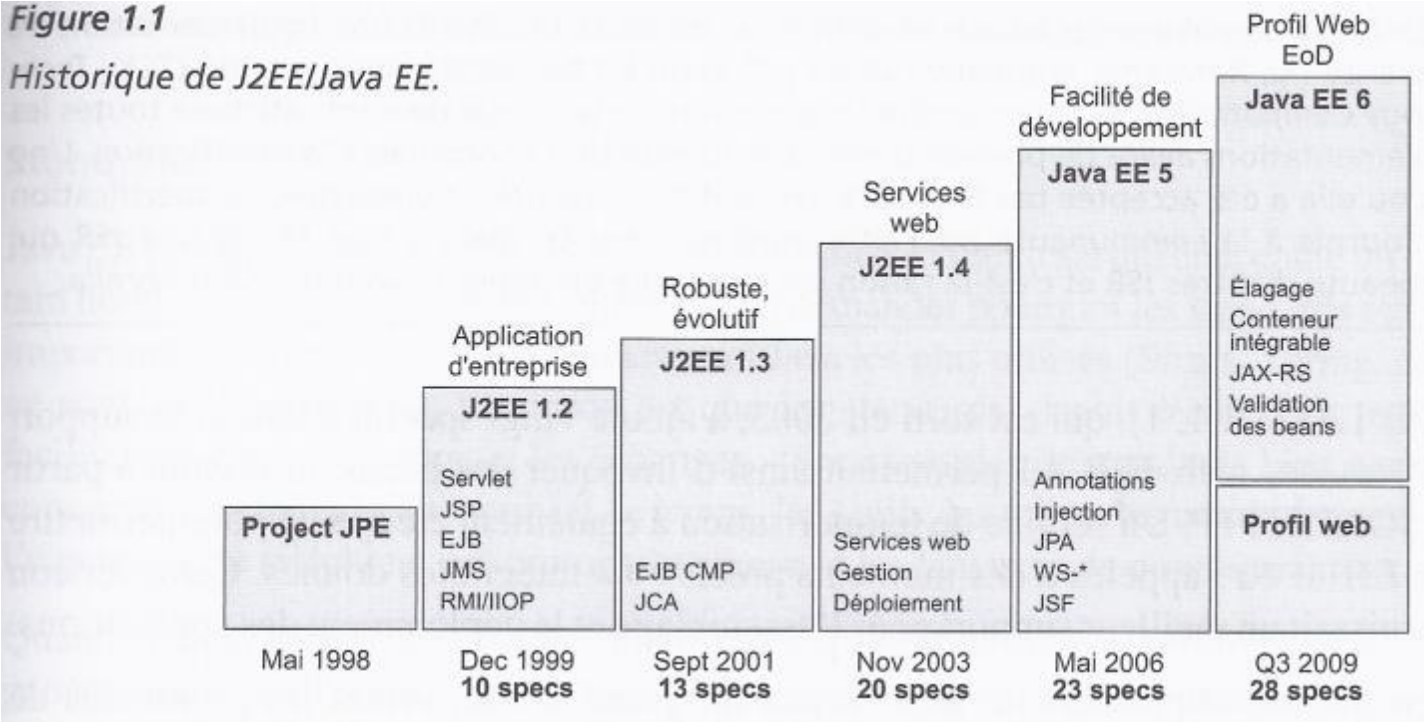
Typiquement c'est une architecture 3-tiers:

- Tiers présentation : affichage des données
- Tiers métier : gestion du métier de l'application
- Tiers donnée : persistance des données

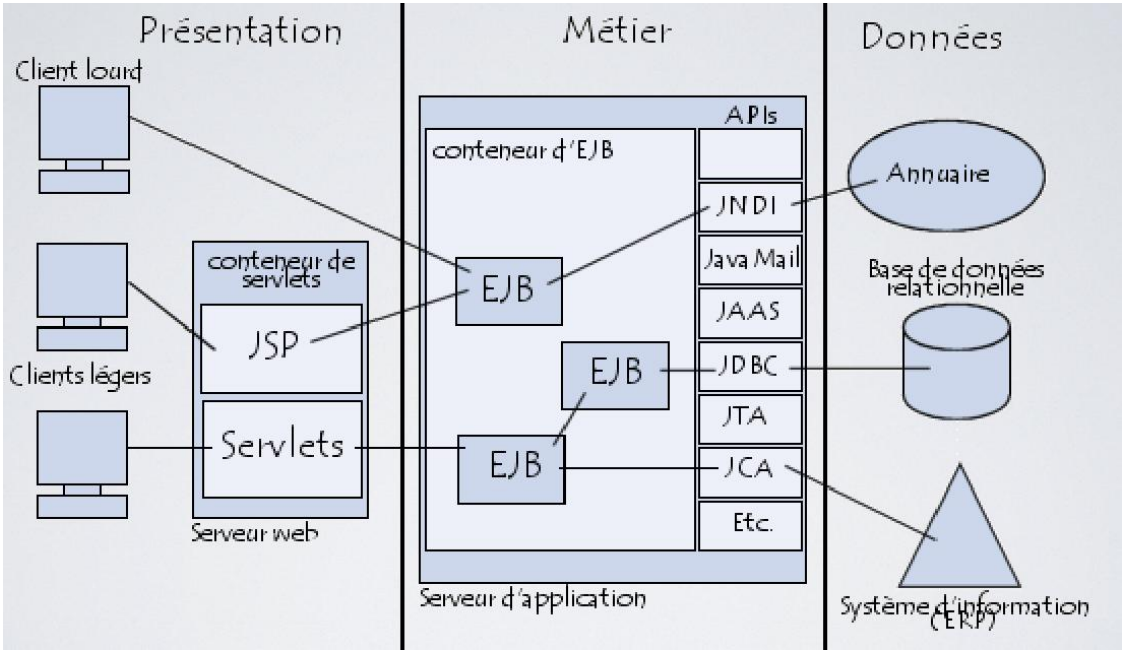
Qu'est-ce Java Enterprise Edition ?

- JEE est une norme proposée par Sun Microsystems (éditeur historique du langage Java, aujourd'hui rachetée par Oracle)
- Objectif : proposer des composants standards pour faciliter le développement et le déploiement d'applications d'entreprises multi-niveaux dans un environnement Java.

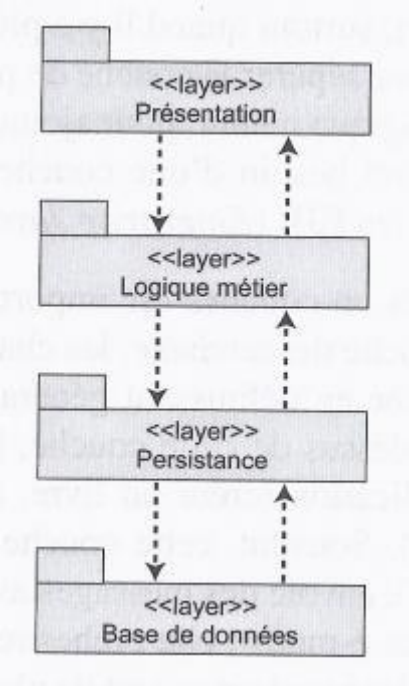
Historique des versions JEE



Architecture JEE



Architecture en couches



Un modèle d'architecture en couches standard :

- Couche « présentation »
- Couche « métier »
- Couche de persistance (vers base de données, fichiers xml, etc.)

Architecture JEE

Permet une séparation claire entre:

- L'interface homme-machine
- Les traitements métiers
- Les données

Architecture JEE

Basée sur des composants qui sont:

- distincts
- interchangeables
- distribués

Architecture JEE

De nouveaux patrons de conception (design pattern):

- Data Access Object
- Data Transfer Object
- Session Facade
- Front controller
- Modèle Vue Controleur

Composants JEE

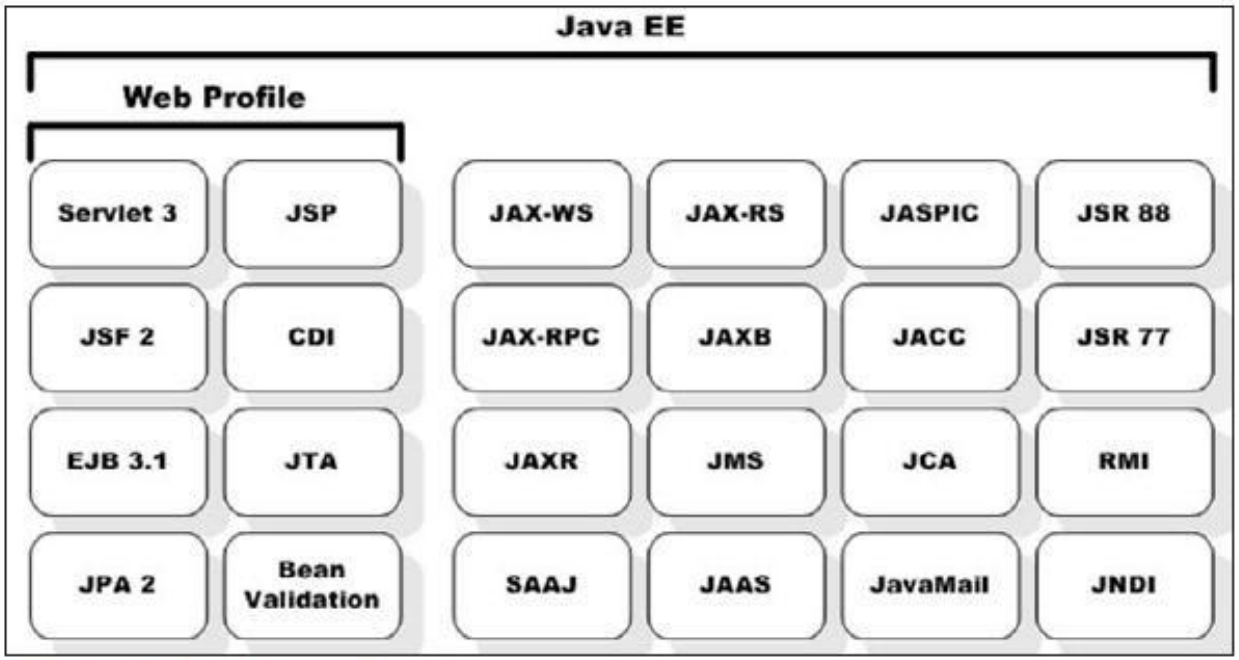


Figure 1: Java EE and the Web Profile

Services JEE

- Java Persistence API : mapping objet/relationnel (ORM), gestion de la persistance en base de données.
- Enterprise Java Beans : composants gérant les processus métiers.
- Java Transaction API (JTA) : gestion de flux transactionnels.
- JAAS : gestion de la sécurité.
- RMI : gestion des accès distants (depuis une autre JVM).
- JNDI : publication de ressources via un annuaire.
- Servlet : gère l'interface web des applications (protocole HTTP).
- Java Server Faces : framework MVC pour la conception d'UI pour des applications web.
- Java XML Binding (JAXB) : lecture/écriture de documents XML, mapping objet/xml.
- JAX-WS : gestion des web services.
- Etc.

Spécification vs Implantation

JEE

- JEE est une spécification de composants standards : chaque éditeur gère sa propre implantation

Serveurs d'applications

- Websphere (IBM)
- Weblogic (Oracle)
- Glassfish (Oracle)
- Jboss (Red Hat)
- Jonas
- …

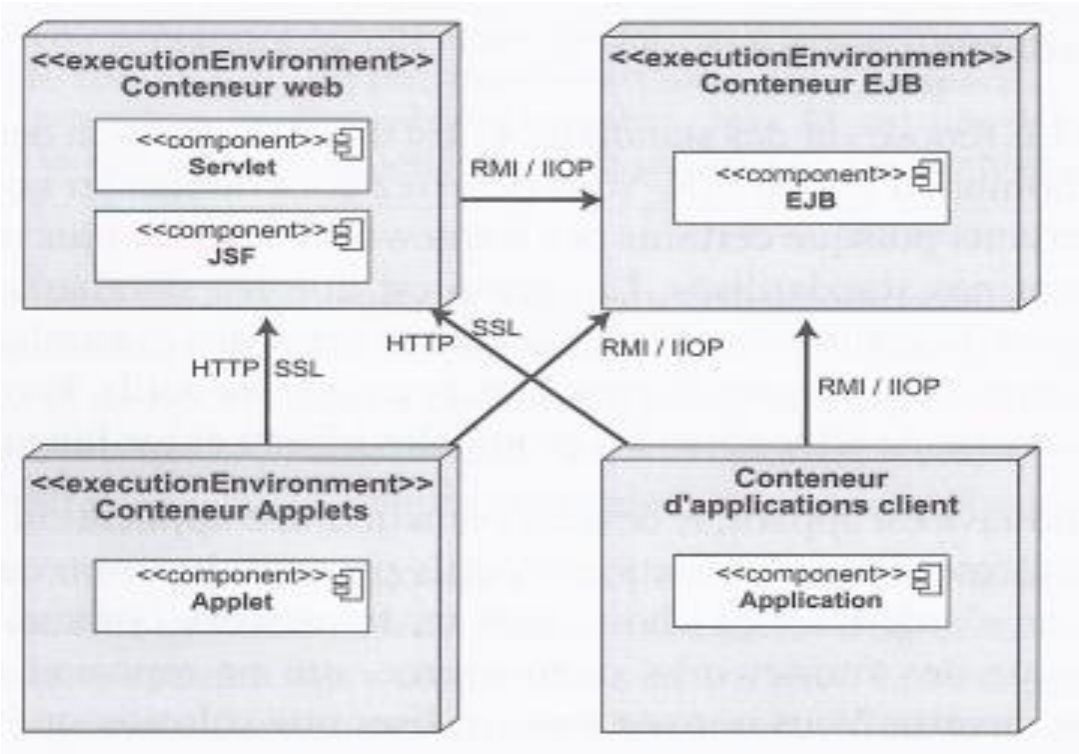
Architecture JEE : les conteneurs

Environnement d'exécution Java permettant d 'héberger des composants applicatifs et de contrôler leur exécution.

La spécification JEE décrit 4 conteneurs standards :

- Applet,
- Applications,
- Applications web: utilisés pour héberger des servlets ou des pags JSP
- EJB (Enterprise Java Beans): supportant l'exécution des composants EJB

Architecture JEE : les conteneurs



Le rôle du conteneur

Héberger des composants « métiers » et leur fournir un certain nombre de services :

- Gestion du cycle de vie,
- Injection de dépendances,
- Transaction,
- Sécurité,
- Support des protocoles d'échange.

Avantage de cette architecture :

- Les composants hébergés sont concentrés sur leur logique métier (gain de productivité, découplage),
- Leur code est délesté de toutes les problématiques techniques (accès à l'annuaire, connexion à une base de données, etc.), gérées par le conteneur standard (réutilisabilité).

Quelques exemples de conteneurs ….

Architecture JEE : les conteneurs Web

- Servlets
  Code java exécuté sur le serveur
  Equivalent du CGI
  Génération de contenu Web dynamique

- JSP: Java Server Pages
  Mélange de HTML/XML et de code java
  Librairies d 'extension (« taglibs »)
  Précompilation en servlet

Architecture JEE : RMI

- Remote Method Invocation
  - Java seulement, mais passerelles
- « RPC objet » (appels sur objets distants)
- Service de nommage (RMI registry)
- Sécurité paramétrable (SecurityManager)
- Garbage Collection distribuée
- Téléchargement de code
- Fonctions avancées
  - Activation d 'objets persistants, Réplication

Architecture JEE : JNDI

- Service de nommage / annuaire
  - Java Naming and Directory Interface
- API accès aux annuaires
  - javax.naming
  - « Service Provider » par annuaire cible (LDAP, NIS, RMI registry…)
- Utilisation avec les EJB
  - Accès à l 'interface « home » pour initialiser
  - Accès à diverses ressources (UserTransaction, Queues JMS, DataSources…)

Architecture JEE : JMS

- Java Messaging Service
- JMS Provider : inclus dans J2EE
  - Transport synchrone ou asynchrone, Garantie de livraison
  - « Messaging domains » point à point ou « publish/subscribe »
- Lien avec EJB : « message-driven bean »
  - Pour échanges asynchrones

Architecture JEE : JMX

- Java Management eXtensions
  - API unique pour applications de management
- Mbeans avec accesseurs get/set
  - Typage faible, attributs nommés
- Serveur JMX
  - Enregistrement des Mbeans
  - Les applis d 'administration dialoguent avec le serveur JMX
- Instrumenter un composant
  - Fournir un ou des Mbeans
  - Les enregistrer auprès du serveur JMX

Forces et faiblesses de JEE

- En ciblant historiquement les besoins des grandes entreprises (architectures distribuées), JEE a longtemps souffert d'une image élitiste, trop lourde à déployer.
- Beaucoup de développeurs se sont détournés de JEE, au profit de frameworks communautaires : Struts, Spring.

Forces et faiblesses de JEE

Exemple de simplification : le profil web

API	Purpose
JSF 2, Facelet, JSP, EL, JSTL, Servlet 3	Web tier
CDI, managed beans, interceptors	Dependency injection, context management, cross-cutting logic, events, extensibility, integration, testing
EJB 3.1 Lite	Business tier, declarative and programmatic transactions, declarative and programmatic security, testing
JTA	Transaction management
JPA 2	Persistence tier
Bean validation	Constraints management

Table 1: Java EE 6 Web Profile APIs

- Depuis JEE 5, gros effort de simplification :
  - ✓ Utilisation de simples POJO annotés,
  - ✓ Configuration par exceptions.
- Depuis JEE 6 :
  - ✓ Principe d'élagage (dépréciation de services) : EJB 2.x, JAX-RPC, etc.)
  - ✓ Apparition du « Web profile »

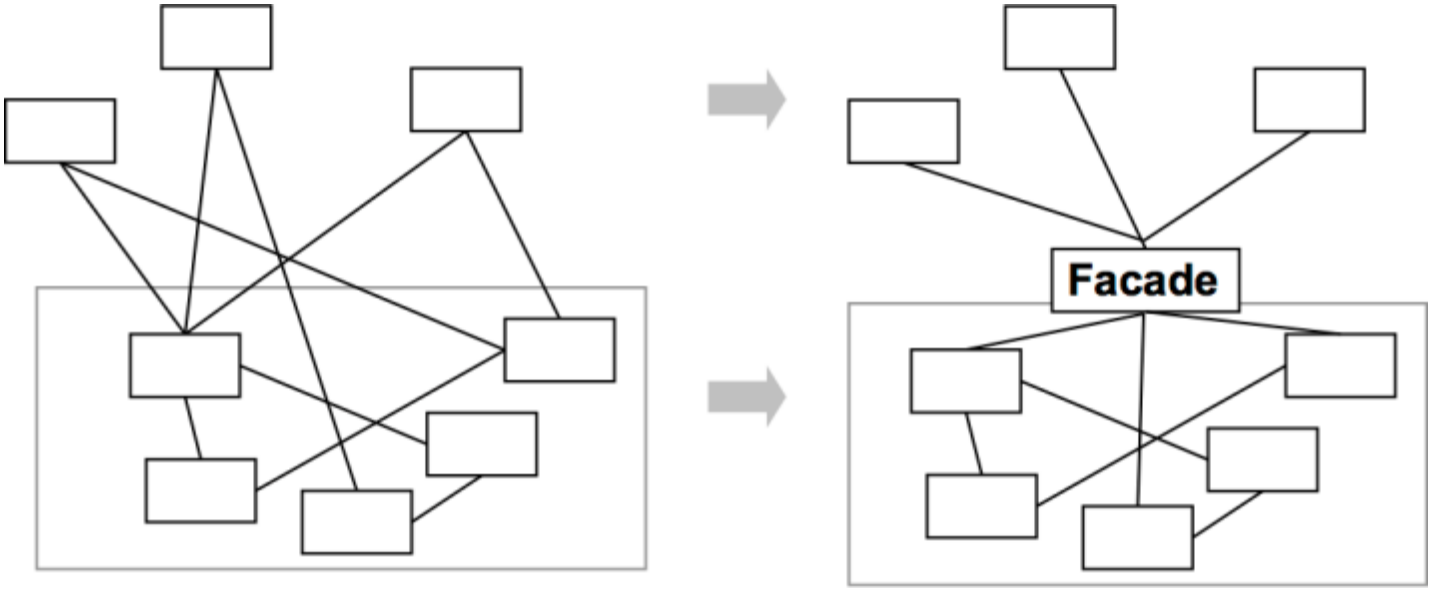
Une architecture basée sur les « design pattern »

La spécification JEE ne se limite pas à décrire des services à offrir pour le développement de solutions d'entreprises.

Elle vise aussi à proposer un guide de bonnes pratiques pour le développement de ces solutions. Pour cela, elle s'appuie sur les design patterns, définis par le Gang of Four (GoF) au milieu des années 90 :

- Facade Pattern,
- Singleton Pattern,
- Delegate Pattern,
- Factory pattern,
- etc.

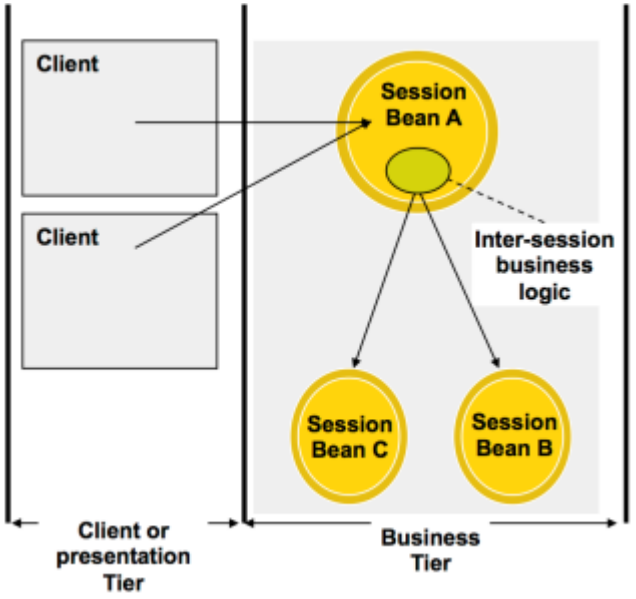
Facade pattern



Permet de limiter le couplage entre les éléments de deux composants

« Facade pattern » dans le contexte EJB

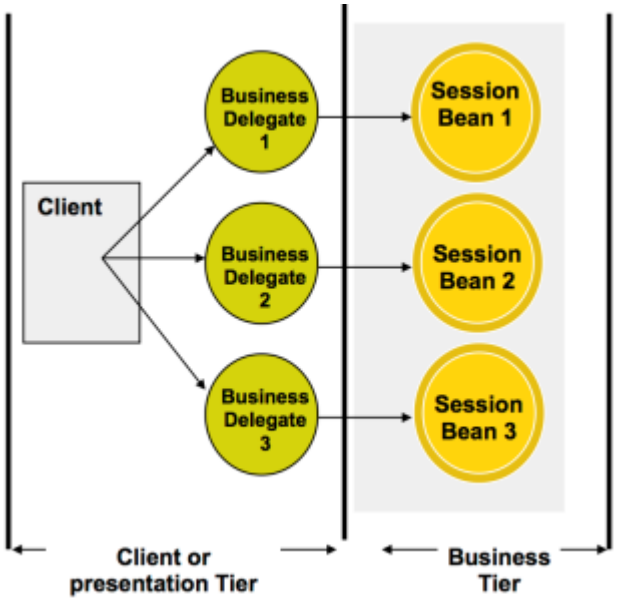
Répond à 3 objectifs :



- Regrouper des traitements dans un même guichet (limiter le nombre d'objets exposés, facilite la manipulation des transactions impliquant les beans)
- Masquer la complexité des traitements pour le client
- Masquer l'accès aux données du système (encapsulation)

« Delegate pattern »

Dans le contexte JEE, pour la couche présentation

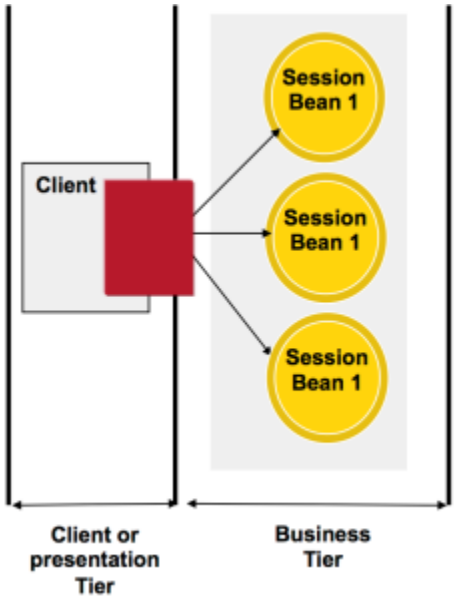


On délègue à un bean managé l'accès aux EJB :

- Eviter le couplage entre le code du tiers de présentation et du tiers métier
- Le « business delegate » est chargé du traitement des exceptions émanant de la couche métier (notification à l'utilisateur d'un message intelligible, localisé etc.)

« Delegate pattern »

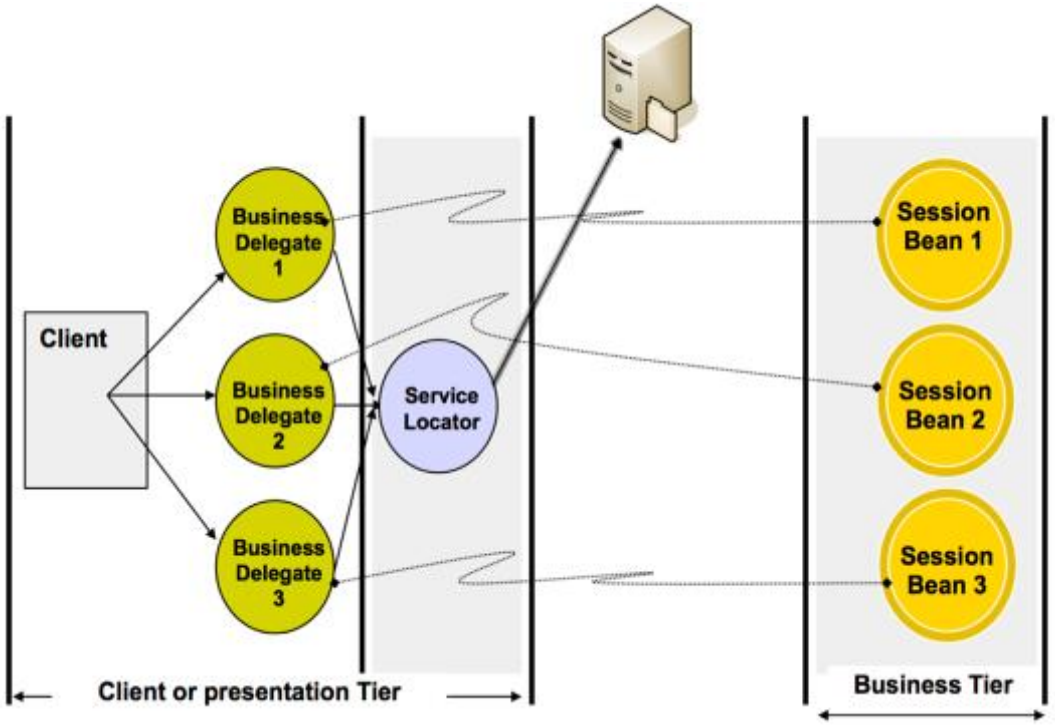
Dans le contexte JEE, pour connecter une application tierce



On délègue à un connecteur (driver) l'accès aux EJB par une application tierce :

- On n'expose que les services réellement nécessaires à l'application tierce
- L'accès à l'application est indirect: l'application tierce ne voit que les services exposés par le « business delegate »
- Le protocole d'échange est adapté à l'application tierce : EJB, messages, web services, flux

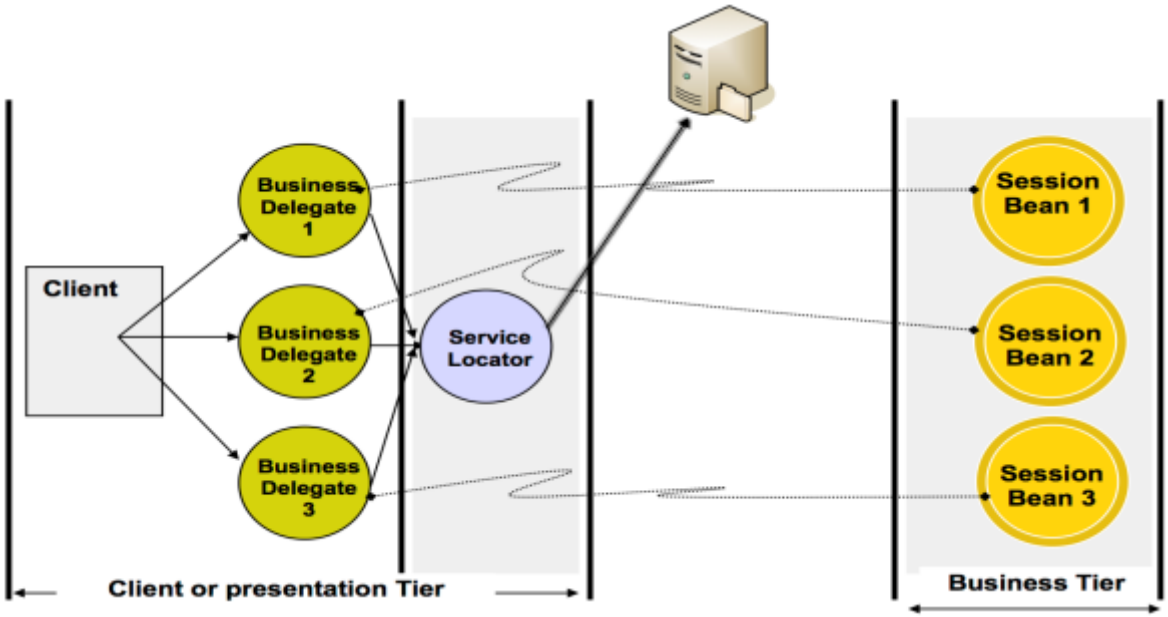
« Service locator »



- On masque au client l'adresse des objets à appeler.
- On décharge le client de la complexité technique pour appeler les objets distants (protocole RMI, lookup dans un annuaire).

Typiquement, le service locator est un composant technique et réutilisable : fourni par le serveur d'applications :
D'une manière générale avec JEE, l'initialisation des dépendances entre composants se fait par injection (IoC pattern)

Synthèse des 3 patterns

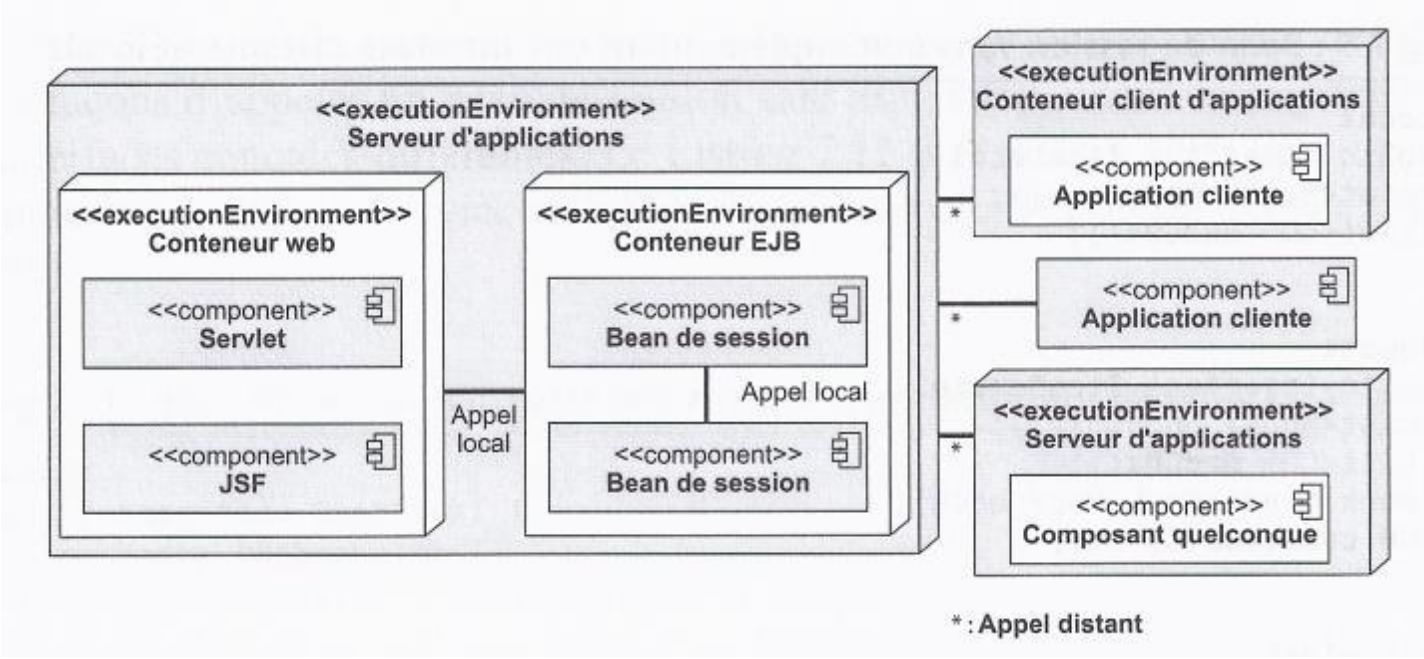


Les EJB sont exposés en façade : ils assurent les traitements métiers et la persistance des données

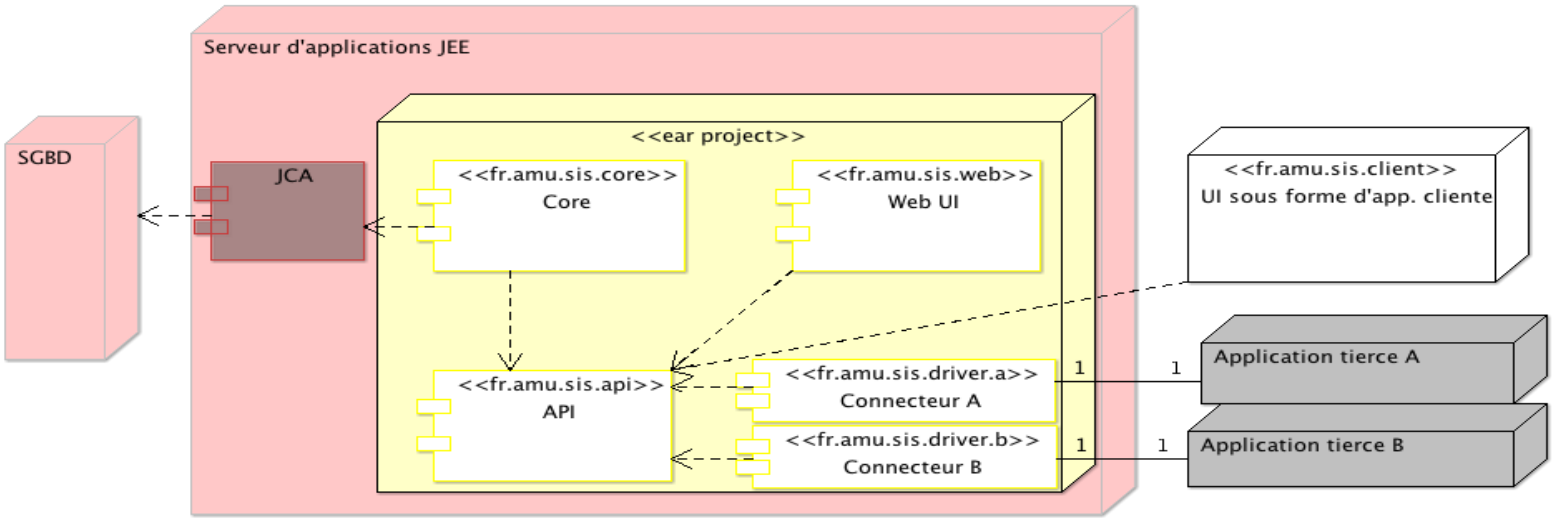
L'application cliente respecte une architecture Modèle-Vue-Contrôleur (MVC). Les beans managés assurent l'interface entre la couche présentation et la couche métier, en dialoguant avec les EJB.

L'adresse des EJB est fournie aux contrôleurs clients par le service locator (directement injectée par le conteneur pour une application web)

Architecture de déploiement d'une application JEE



Déploiement JEE : le projet « ear »



- Contenu du projet ear : les entités, les EJB, l'application web, les drivers.
- Bonne pratique : programmer par interfaces afin de limiter le couplage entre composants