Développement web - Java

Franck Denoël
Master 2 – ICA, UPMF
Année académique 2007-2008

J2EE - Introduction

Franck Denoël
Master 2 – ICA, UPMF
Année académique 2007-2008

Plan du chapitre

- **Présentation J2EE**
- Architecture
- Composants et services
- Développement d'une application J2EE et Frameworks

J2EE

- Rappels :
  - Les plateformes Java intègrent :
    - une JVM (Machine Virtuelle Java) et
    - une API (Application Programming Interface) :
      - Collection de composants logiciels (librairies) utilisés pour développer d'autres composants ou applications
  - Les applications java fonctionnent sur tous les systèmes compatibles
    - avantages : indépendance de la plateforme - portabilité du code, puissance, stabilité et sécurité

J2EE

- Le langage Java se décline autour de 3 briques :
  - Java 2 Standard Edition (J2SE) : vise les postes clients
  - Java 2 Micro Edition (J2ME) : cible les terminaux portables (téléphonie, PDA)
  - Java 2 Entreprise Edition (J2EE) : vise les applications d'entreprise

Problématique générale

- Les applications d'entreprise concernent aussi bien les grandes entreprises quel les petites
- Elles doivent être :
  - portables
  - fiables et sécurisées
  - maintenables et flexibles
  - performantes
- Nécessité d'intégrer ou de s'intégrer à un système d'information existant
- Toutes ces considérations rendent les applications complexes => solution = J2EE

J2EE

- Norme proposée par la société Sun depuis 1997 et portée par un consortium de sociétés internationales
- Version actuelle = EE 5.0
- J2EE fournit
  - Une plateforme standard :
    - de développement d'applications basées sur des composants
    - de déploiement des applications J2EE basée sur la notion de conteneur
  - Un ensemble de services accessibles via l'API J2EE pour concevoir les applications
  - Une documentation complète sur le site de *sun*

Plan du chapitre

- Présentation J2EE
- **Architecture**
- Composants et services
- Développement d'une application J2EE et Frameworks

Applications d'entreprise J2EE

- Une application d'entreprise J2EE est multi-niveaux (n-Tier)
- Un *Tier* est une partition (couche) logique ou fonctionnelle d'un système
- 3 grands modèles d'architecture applicative :
  - Simple Tier
  - 2-Tier
  - 3-Tier (ou n-Tier)

Applications d'entreprise J2EE

- Simple Tier
  - Client lourd
  - Avantages :
    - Développement rapide
  - Désavantages :
    - Application monolithique
    - Mises à jour et maintenance du code plus difficiles
    - Code difficilement réutilisable

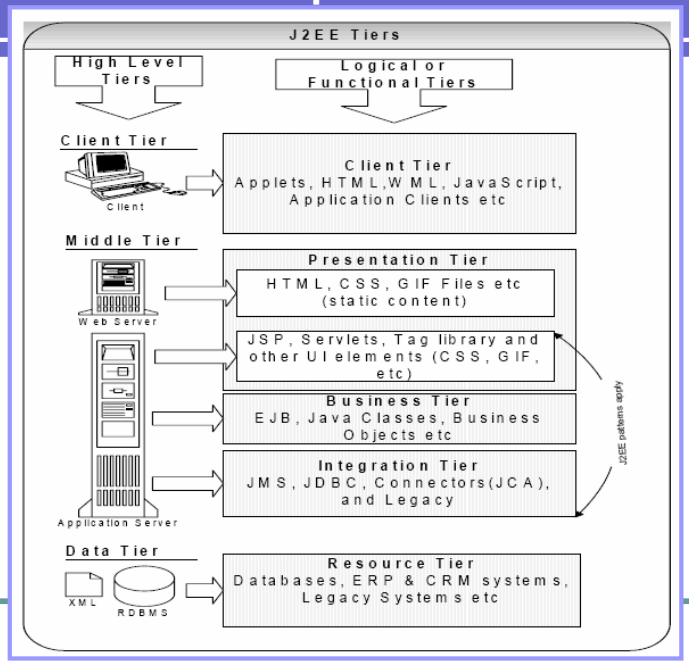
Applications d'entreprise J2EE

- 2-Tier
  - Mode de communication entre plusieurs ordinateurs clients et un serveur
  - Avantages :
    - Centralisation du code et des ressources
  - Désavantages :
    - Application serveur souvent monolithique
    - Logique métier est répartie entre l'application serveur et le client

Applications d'entreprise J2EE : n-Tier

- *N-Tier* = Modèle logique d'architecture applicative qui vise à séparer nettement trois couches logicielles au sein d'une même application et à présenter l'application comme un empilement de ces couches :
  - présentation des données
  - traitement métier des données
  - accès aux données persistantes
- Les couches communiquent entre elles au travers d'un « modèle d'échange », et chacune d'entre elles propose un ensemble de services rendus
- Les services d'une couche sont mis à disposition de la couche supérieure

Applications d'entreprise J2EE : n-Tier



Applications d'entreprise J2EE : n-Tier

- Avantages :
  - Séparation forte entre les 3 niveaux
  - Chaque niveau peut être managé, dimensionné, distribué
  - Mises à jour et maintenance facilitées en minimisant l'impact sur les autres couches
  - Extensibilité : ajout de nouvelles fonctionnalités simplifié

Serveur d'application

- Les applications d'entreprise ont souvent besoin des mêmes services système :
  - Gestion de la concurrence
  - Services transactionnels entre composants
  - Sécurité
  - Gestion de la session utilisateur
  - Gestion des montées en charge
  - Ouverture sur de multiples sources de données
  - *Pools* de connexion
  - Système de tolérance aux pannes et reprise sur incident
- Le serveur d'application fournira ces services système

Serveur d'application J2EE

- Serveur d'application J2EE = implémente les API J2EE
- Il héberge des composants applicatifs
- Il fournit des services à ces composants au travers d'un conteneur
  - Un conteneur J2EE est un environnement d'exécution chargé de gérer des composants applicatifs et leur donner accès aux API J2EE

Quelques plateformes

- Les serveurs d'application intègrent l'ensemble des spécifications J2EE
- Serveurs d'application (commerciaux)
  - BEA WebLogic
  - IBM Websphere
  - Sun Java System App. Server
  - Borland Enterprise Server
  - Oracle 9i Application Server
  - SAP Web application server

Quelques plateformes

- Serveurs d'application (open-source) :
  - JBoss
  - ObjectWeb JOnAS
  - Apache Geronimo
- Les serveurs web n'implémentent que certaines spécifications liées au développement web
- Serveurs Web (open-source) :
  - Apache Tomcat
  - Jetty

Plan du chapitre

- Présentation J2EE
- Architecture
- **Composants et services**
- Développement d'une application J2EE et Frameworks

Services et composants J2EE

- La plateforme J2EE intègre :
  - des composants J2EE
  - des services
  - des protocoles de communication ...
- ... qui permettent le développement d'application n-Tier et distribuées

Services et composants J2EE

- Les applications d'entreprise J2EE sont basées sur des composants :
  - distincts
  - distribués
  - interchangeables
- Composant = unité logicielle (fonctionnelle) intégrée à une application J2EE communiquant avec d'autres composants
- Il dépend du conteneur pour le support des services système (transactions, sécurité, threading, etc.)

Services et composants J2EE

- Conteneur :
  - Les applications J2EE y sont déployées
  - Donne l'accès aux services
  - Dans l'architecture des conteneurs J2EE, les applications fournissent :
    - Composants applicatifs
    - Descripteur de déploiement (*.xml)
  - Types de conteneurs :
    - Web (Servlets, JSP, JSF)
    - EJB (EJB)

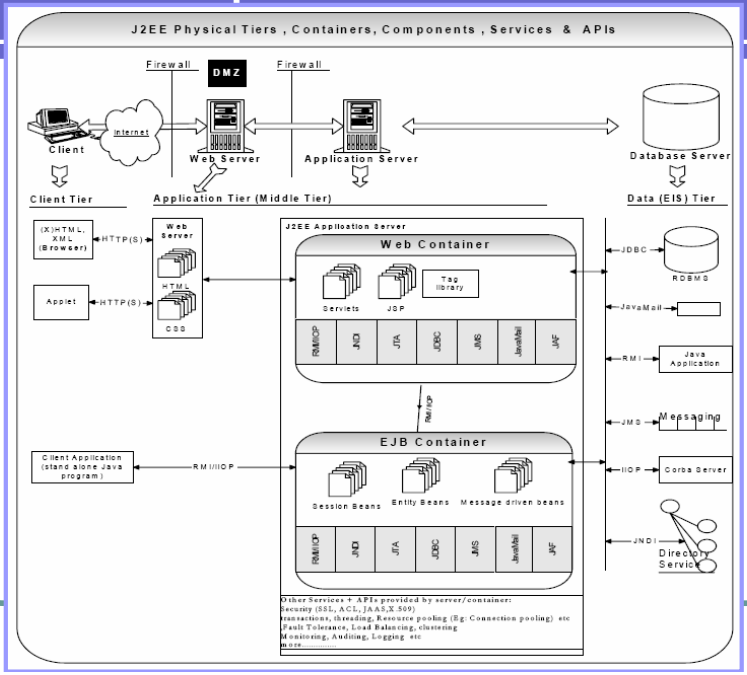
Services et composants J2EE

- Service J2EE :
  - Composant technique pouvant être utilisé de manière distante à travers une interface de manière synchrone ou asynchrone
  - Types : JDBC, JTS/JTA, JMS, JavaMail, JAAS, Web service, etc.

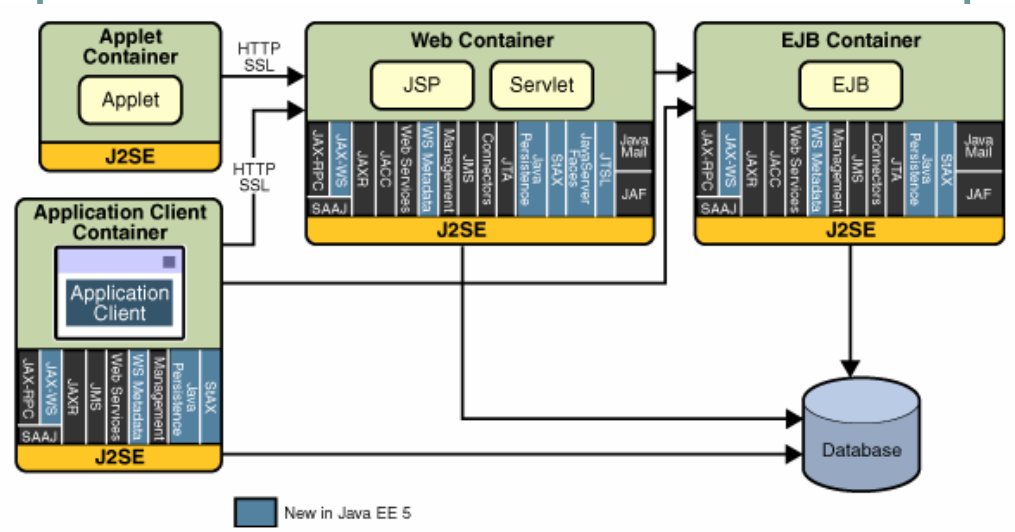
Services et composants J2EE

- Protocoles :
  - Accéder à des services Internet
  - Types (principaux) :
    - HTTP (*HyperText Transfer Protocol*)
    - TCP/IP (*Transmission Control Protocol / Internet Protocol*)
    - RMI (*Remote Method Invocation*)
    - SOAP (*Simple Object Access Protocol*)
    - SSL (*Secured Socket Layer*)

Services et composants J2EE



Architecture J2EE



JDBC

- JDBC = *Java DataBase Connectivity*
- API permettant l'accès à des sources de données compatibles (SGBDR)
- Elle gère les 3 étapes indispensables à l'accès aux données :
  - la création d'une connexion à la base
  - l'envoi d'instructions SQL
  - l'exploitation des résultats provenant de la base
- Pour accéder à la base de données, JDBC s'appuie sur des pilotes (*drivers*) spécifiques à un fournisseur de SGBDR ou sur des pilotes génériques

JDBC

- Exemple :

```
...
try {

Class.forName("org.postgresql.Driver");
Connection con = DriverManager.getConnection
          ( "jdbc:postgresql://localhost:5432/myDb", "myLogin","myPassword");

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT age, nom, prenom FROM client");
while (rs.next()) {
    int age = rs.getInt("age");
    String nom = rs.getString("nom");
    String prenom = rs.getFloat("prenom");
    }

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

JNDI

- JNDI = *Java Naming and Directory Interface*
- API unique pour obtenir (*look up*) des ressources et les relier (*binding*) à un service de nommage ou à un annuaire
  - Un service de nommage permet d'associer un nom unique à un objet (information) et faciliter ainsi l'obtention de cet objet
    Exemple : DNS, NIS
  - Un annuaire est un service de nommage qui possède en plus une représentation hiérarchique des objets qu'il contient et un mécanisme de recherche
    Exemple : LDAP

JNDI

- L'accès à un service est possible au moyen d'un pilote qui assure le dialogue avec l'API
- JNDI est notamment utilisé par les API :
  - JDBC
  - JMS
  - EJB

JNDI

- Exemple :

```
...
import javax.naming.*;
...
  public void createName() throws NamingException {
    Context context = new InitialContext();
    // bind an Object
    context.bind("/config/myApplication", "une valeur");
  }
...
```

```
...
import javax.naming.*;
...
  public String getValue() throws NamingException {
    Context context = new InitialContext();
    return (String) context.lookup("/config/myApplication");
  }
...
```

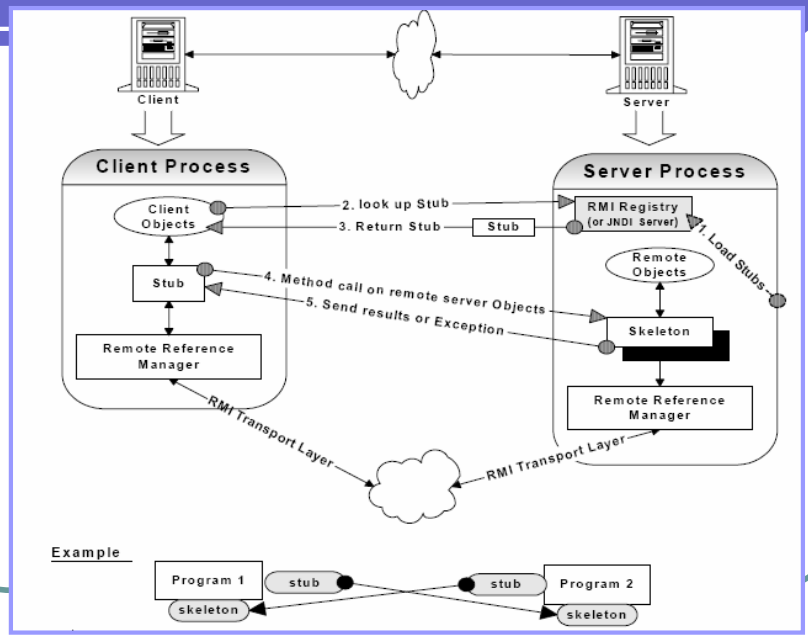
RMI

- RMI = *Remote Method Invocation*
- API permettant la communication synchrone entre des objets Java s'exécutant sur des JVM distinctes
  - Implémentation du *design pattern* "proxy"
- Les objets distants sont manipulés comme s'ils étaient locaux
- Utilisation du protocole de transport IIOP (*Internet Inter-Orb Protocol*)

RMI

- Facilite le développement des applications distribuées en masquant la communication client/serveur
- API souvent utilisée en parallèle avec JNDI ou avec les EJB
- Nécessite l'emploi d'un registre RMI :
  - localisé sur la machine distante
  - héberge les objets (préalablement enregistrés) à appeler

RMI



JDBC-JNDI-RMI

- En conclusion :
  - JDBC = accès à des SGBDR
  - JNDI = localisation de composants disribués
  - RMI = permet à des composants distribués d'être utilisés comme s'ils étaient locaux

JCA

- JCA = *Java Connector Architecture*
- API utilisée pour créer des modules qui permettent à un serveur J2EE (où ils sont déployés) de se connecter à un système propriétaire (*legacy system*) et de normaliser les accès et interactions avec ce système
  - Avant JCA, chaque serveur d'application devait fournir un adaptateur spécifique pour chaque EIS (liaison point-à-point)
  - Avec JCA, un même adaptateur fonctionne avec tous les conteneurs J2EE compatibles
- JCA = communication et utilisation de ressources

JCA

- L'api JCA définit un contrat entre le serveur d'application et le systeme, via des interfaces comme le fait jdbc
  - Connecteur = fichier .rar contenant les classes d'implémentation des interfaces standard et le descripteur de déploiement
- JCA est une généralisation de JDBC
- JCA, à partir de sa version 1.5, permet d'envoyer des messages vers le système propriétaire et de recevoir des messages qui seront traités par un *Message Listener*, généralement un *Message Driven Bean* (EJB)

JMX

- JMX = *Java Management eXtension*
- API unique utilisée par les applications de management
- Fonctionnalités :
  - Modifier dynamiquement le comportement d'une application Java en *runtime*
  - Générer des statistiques sur le fonctionnement de l'application et de les rendre disponibles
  - Notifier des dysfonctionnements

JMX

- Ces fonctionnalités sont accessibles en local, mais également à distance
- Les applications de management dialoguent avec le serveur JMX où sont enregistrés des MBeans
  - Accès aux composants ressources via des MBeans

JTS/JTA

- JTS(A) = *Java Transaction Service (Api)*
- JTA fournit des interfaces Java standards pour permettre la communication entre un gestionnaire de transaction (JTS) et les différentes parties impliquées dans un système de transactions distribuées : les ressources (DB, ...), le serveur d'application et les applications transactionnelles
- JTS utilise JTA pour gérer et accéder à toutes les transactions sur un système donné

JMS

- JMS = *Java Message Service*
- API permettant à une application J2EE de créer, d'envoyer, de recevoir et de lire des données sous forme de messages (MOM) de manière asynchrone
- Avantages :
  - fiable (garantie de livraison)
  - gestion des messages centralisée
  - communication faiblement couplée

JMS

- Deux modèles :
  - *Publish and subscribe* : des entités s'inscrivent pour recevoir des messages sur un certain sujet. Celui qui publie les messages et ceux qui les reçoivent ne se connaissent pas
  - Point à point : le producteur publie les messages dans une file et le client lit les messages de la file. Dans ce cas le producteur connaît la destination des messages et poste les messages directement dans la file du client

JAAS

- JAAS = *Java Authentification and Authorization Service*
- API permettant de gérer l'authentification et les droits d'accès d'un utilisateur ou d'un groupe d'utilisateur pour une application J2EE
- L'authentification de JAAS se fait par branchement (*pluggable*)
  - D'où indépendance des technologies d'authentification

JavaMail

- API standard de gestion de courriers électroniques
- Il permet d'envoyer et de recevoir du courrier électronique et de manipuler les messages (en-tête, sujet, corps, pièces jointes...)
- Note : Ce n'est pas un serveur de courrier mais un outil pour interagir avec ce type de serveur

JavaMail

- JavaMail utilise différents protocoles comme SMTP, IMAP ou POP
- Il possède :
  - une interface applicative pour envoyer des mails et
  - un *service provider* qui permet d'envoyer des mails par Internet
- La plateforme J2EE embarque un Javamail *service provider*

JavaMail

- JAF = *JavaBeans Activation Framework*
  - Utilisé par JavaMail
  - Il fournit les services standards pour déterminer des types de données (MIME), y accéder, découvrir les opérations possibles dessus et créer les composants JavaBeans appropriés pour réaliser ces opérations

XML et Web services

- Web service = Application permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués
- Permet une communication de type application/application utilisant XML
- Ensemble de fonctionnalités exposées par un serveur et exécutables à distance en temps réel par des clients applicatifs
- Web service ≠ Sites webs !

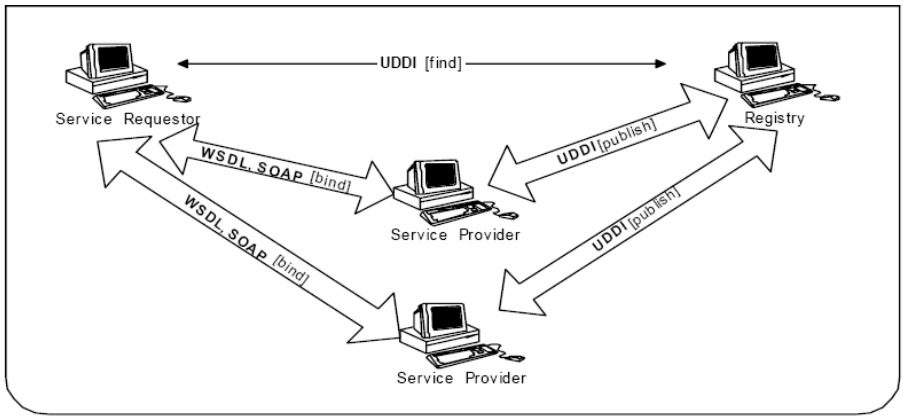
XML et Web services

- JAX-WS = *API for XML-based Web Services*
- Implémentation logicielle des spécifications WS reposant sur un ensemble de protocoles et de standards de base :
  - SOAP (*Simple Object Access Protocol*)
    - format d'échange de messages
    - requête/réponse HTTP encapsulent un message SOAP = enveloppe XML respectant un schéma défini

XML et Web services

- WSDL (*Web Service Description Language*)
  - description des messages et des types de données
  - des outils permettent de générer des WSDL (XMLSpy, etc.)
- UDDI (*Universal Description Discovery and Integration*)
  - Annuaire (*registry ≠ repository*)
  - Fournit la manière de publier et de découvrir des informations sur les Web services
  - Utilisé lorsque le nombre de WS est important (> 50 souvent)

XML et Web services



XML et Web services

- API XML complémentaires
  - JAXP = *Java Api for Xml Processing*
    - API standard permettant de *parser* et de transformer des documents XML
    - Supporte DOM, SAX, XSLT
  - JAXB = *Java Architecture for Xml Binding*
    - API standard permettant de représenter un schéma XML en objet java
  - JAXR = *Java Api for Xml Registries*
    - Accès générique à des registres sur Internet (UDDI et ebXML)

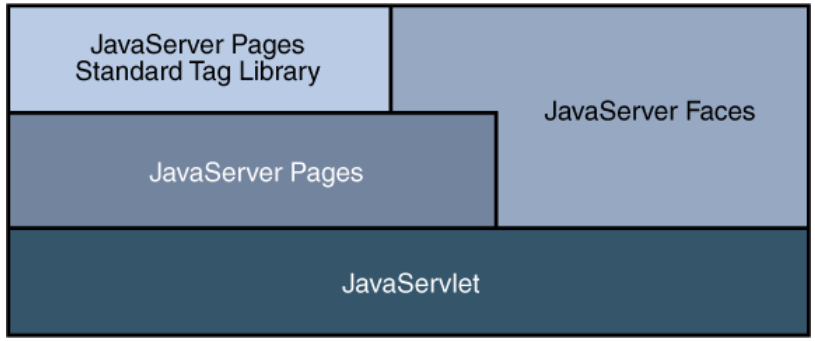
Composants Web

- Servlets/JSP : constituent la brique de base des applications Web dynamique en Java
- Cette technologie étend les fonctionnalités d'un serveur web en traitant de manière objet les requêtes/réponses HTTP
- Intégration à serveur web via un conteneur Web
  - Servlet
    - Composant exécuté sur le serveur
    - Implémente le paradigme requête-réponse (HTTP principalement)

Composants Web

- JSP
  - Extension des servlets
  - Document texte contenant des parties statiques (HTML, XML) et dynamiques (déterminant comment se construit la page pour le client)
  - Extensible via les *custom tags*
- JSTL
  - librairie encapsulant des fonctions récurrentes aux applications JSP
  - Utilisation du XML
- JSF : *framework* de création d'interfaces utilisateurs pour les applications web, basé sur les technologies JSP et Servlet

Composants Web



Composants EJB

- EJB = Entreprise JavaBean
- Composants implémentant des fonctionnalités métiers
  - Brique utilisée de manière autonome ou avec d'autres composants J2EE pour exécuter une logique métier
- Il fonctionnent à travers un conteneur d'EJB
  - Séparation de la logique métier et du code système géré par le conteneur
  - Le conteneur a en charge la création, la destruction, la passivation ou l'activation de ses composants en fonction des besoins

Composants EJB

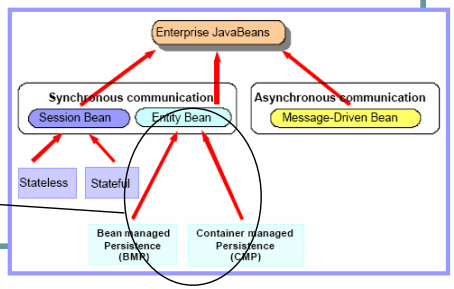
- Le client via un appel RMI (généralement) va rechercher un EJB par son nom logique JNDI et appeler une/des méthodes de cet objet
  - EJB Moins performants que RMI mais fournit plus de services (transactions, pool d'objets, etc.)
- Avant la version 3.0, un EJB est accompagné d'un ou plusieurs descripteur de déploiement
- Depuis la version 3.0, le modèle EJB utilise le principe d'annotations java (meta-données) pour spécifier toute la configuration et les propriétés transactionnelles de l'objet

Composants EJB

- Les EJB proposent :
  - des services avec ou sans conservation d'état entre les appels (EJB *session bean*)
  - d'accomplir des tâches de manière asynchrone (EJB *message-driven bean*)

Dans J2EE 5.0 les beans entity ont été remplacés par l'API *peristence entities*

Une entité représente des données persistentes stockées dans une ligne d'une table d'une SGBD



Composants EJB : *Session Bean*

- Un *session bean* représente une conversation transitoire (*transient*) avec un client
- Objet métier non persistant
- Typologie :
  - *stateless* : ne conservent pas leur état en mémoire entre deux appels du client
  - *stateful* : conservent leur état entre deux appels

Composants EJB : *Message-driven bean*

- Un *Message Driven Bean* est un composant métier recevant des messages de manière asynchrone
  - Le client n'a pas besoin de figer son exécution durant le traitement du MDB
- Les clients n'appellent pas directement des méthodes mais utilisent JMS pour produire un message et le publier dans une file d'attente
  - À l'autre bout, le MDB est à l'écoute de cette file d'attente et se « réveille » à l'arrivée d'un message. Il extrait ce dernier de la file d'attente, en récupère le contenu puis exécute un traitement

Composants EJB

```
...
@Stateless
public class MyBean {

    // business code
    public void myMethod() {
        ...
    }
}
...
```

```
...
@Stateful
public class MonBean {

    private String myAttribute;

    // business code
    public void myMethod() {
        ...
    }
}
...
```

```
...
@MessageDriven
public class MyMDB implements MessageListener {

    public void onMessage (Message msg) {
    // traitement métier
    }

...
}
```

Composants EJB : Entité

- Bean Entité (avant EJB 3.0)
  - Objet métier persistant
  - Deux types :
    - Container-Managed Persistence (CMP) : bean dont la persistance est directement assurée par le conteneur d'EJB
    - Bean-Managed Persistence (BMP) : bean dont la persistance a dû être programmée par le développeur

JPA

- JPA = Java Persistence API
  - Nouveau service depuis J2EE 1.5
  - API standard pour gérer la persistance
    - Approche objet/relationnel (modèle/SGBDR)
  - Remplace les EJB *entities*
  - Couche d'abstraction s'appuyant sur JDBC

Plan du chapitre

- Présentation J2EE
- Architecture
- Composants et services
- **Développement d'une application J2EE et Frameworks**

Développement d'application J2EE

- Processus général :
  - Création de composants applicatifs J2EE qui seront regroupés en modules, puis déployés individuellement ou comme application J2EE
- Création de composants applicatifs
  - Modélisation des règles métiers sous la forme de composants applicatifs
  - Regroupement des composants selon leur type (Web, EJB, etc.) avec, pour chaque module J2EE, un descripteur de déploiement. Ces modules composeront l'application J2EE

Développement d'application J2EE

- Assemblage de l'application (*packaging*)
  - Les modules J2EE développés sont assemblés de différentes manières : tâches *ant*, Maven, ...
  - Ils peuvent être déployés :
    - comme applications à part entière ou
    - être assemblés avec un descripteur de déploiement J2EE et être déployés en tant qu'application J2EE. L'application J2EE est alors packagée dans un fichier possédant l'extension .ear.
- Déploiement de l'application
  - Consiste à installer et à configurer des modules ou l'application sur la plateforme J2EE

Développement d'application J2EE

- Application J2EE (aggrégation de différents tiers)
  - Fichier « .ear » + descripteur « application.xml »
- Niveau Web
  - Web : fichier « .war » + descripteur « web.xml »
- Niveau EJB
  - Fichier « .jar » + descripteur « ejbjar.xml »
- Niveau accès aux données (connecteurs JCA)
  - Fichier « .rar » + descripteur « rar.xml »

Frameworks

- Il est parfois lourd d'utiliser un composant standard de J2EE :
  - trop ouvert, trop complexe, *design patterns* à connaître et à implémenter
- Solution : les frameworks
  - couches d'abstraction s'appuyant sur d'autres briques plus génériques visant à simplifier l'utilisation de certaines technologies
  - briques spécialisées intégrant un ensemble de bonnes pratiques et un cadre de codage

Frameworks : Struts

- Ne nécessite qu'un simple moteur de servlet contrairement aux EJB
- Implémentation du modèle MVC2
- Basé sur les technologies Servlet/JSP
- Autres frameworks : JSF, Spring-MVC, ...

Frameworks : Hibernate

- Permet l'accès à une SGBD
- Mapping objet/relationnel : abstraction des accès à la SGBD par l'utilisation d'objets mappés sur les données
- Puissant et performant :
  - *lazy-loading*
  - filtres
  - etc.

Frameworks : Spring

- Ne nécessite qu'un simple moteur de servlet contrairement aux EJB
- Conteneur léger : fabrique d'objets
  - Définition d'interfaces
- Prise en charge de la création et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration
- POA : programmation orienté aspect