# EJB : les fondamentaux

*Michel Buffa (buffa@unice.fr), UNSA 2002,*
*modifié par Richard Grin (version 1.1, 21/11/11),*
*avec emprunts aux supports de Maxime Lefrançois*

## Plan du cours

- Introduction générale
- EJB : fondamentaux
- Session Beans
- Entités
- Message-Driven Beans
- Concepts avancés sur la persistance
- Relations avec les entités
- Gestion des transactions

## Enterprise Java Bean

- Les EJB
  *facilitent la création d'applications distribuées pour les entreprises.*
  - S'occupent du traitement métier de l'application
  - Permettent aux développeurs de se concentrer sur les traitements orientés métiers
  - Sont réutilisables
  - Sont assemblables

## Enterprise Java Bean

- Composant serveur qui encapsule une logique métier, qui peut être déployé dans un serveur d'application
- Composé de un ou plusieurs objets
- Les appels aux méthodes par les clients de l'EJB sont interceptés par le conteneur d'EJB

## Rôle du conteneur

- Le conteneur d'EJB s'occupe de certains traitements
  - Cycle de vie du bean
  - Injection de dépendance
  - Accès au bean, communication à distance
  - Sécurité d'accès
  - Accès concurrents
  - Transactions, …

## 2 Types d'EJB

- Session Bean
  - Modélise un traitement
  - Représenté par une classe Java et une interface qui expose certaines méthodes
- Message Driven Bean (MDB)
  - Consomme des messages asynchrones envoyés par des clients
  - Permettent l'interconnexion avec des systèmes différents (non Java EE)

## Session Bean

- Modélise un traitement (business process)
- Correspond à un *verbe*, à une *action*
- Ex : gestion de compte bancaire, affichage de catalogue de produit, vérifieur de données bancaires, gestionnaire de prix…
- Les actions impliquent des calculs, des accès à une base de données, consulter un service externe (appel téléphonique, etc.)
- Souvent client d'autres Beans

## 3 types de Session Bean

- Bean sans état (*stateless*)
  - Pour traiter les requêtes de plusieurs clients,
  - sans garder un état entre les différentes requêtes
  - Exemple : obtenir la liste de tous les produits
- Bean avec état (*stateful*)
  - Pour tenir une conversation avec un seul client,
  - en gardant un état entre les requêtes
  - Exemple : remplir le caddy d'un client avant de lancer la commande (le caddy est rempli en cliquant sur les différentes pages des produits)

## 3 types de Session Bean

- Bean singleton
  - Garantie de n'avoir qu'une seule instance du bean dans tout le serveur d'application
  - Supporte les accès concurrents (configurable)
  - Exemple : bean qui « cache » une liste de pays, utilisé par les classes de l'application pour éviter d'interroger la BD

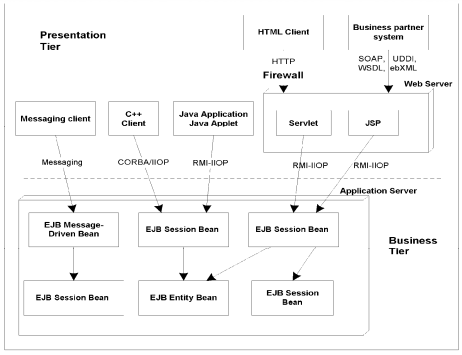
## Message-Driven Bean

- Introduits à partir de la norme EJB 2.0 (aujourd'hui en 3.0)
- Similaire aux Session beans : représentent des verbes ou des actions,
- On les invoque en leur envoyant des messages, souvent d'une autre application
- Ex : message pour déclencher des transactions boursières, des autorisations d'achat par CB
- Souvent clients d'autres beans…

## Entité

- Les applications Java EE utilisent aussi des entités
- Une entité est une classe qui représente des données enregistrées dans une base de données
- Correspond à un nom
- Ex : personne, produit, compte bancaire

## Clients interagissant avec un serveur à base d'EJBs

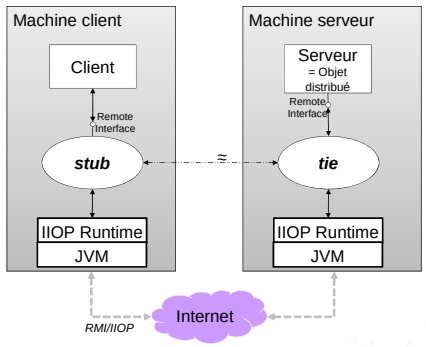


Objets distribués

## Application distribuée

- Une application Java EE peut être distribuée sur plusieurs machines du réseau
- Les containers gèrent les appels distants pour le développeur (utilisent RMI-IIOP)

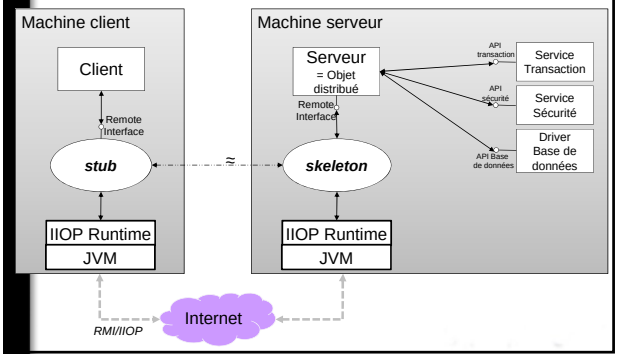
## Les objets distribués
## … *RMI-IIOP*



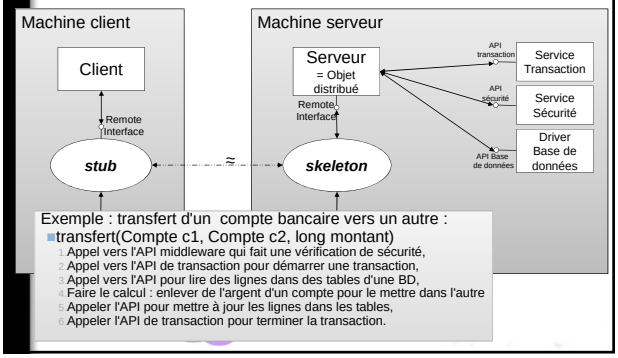

## Les objets distribués et le middleware

- Lorsqu'une application devient importante, des besoins récurrents apparaissent : sécurité, transactions,etc…
- C'est là qu'intervient le *middleware*!
- Deux approches
  1. Middleware explicite,
  2. Middleware implicite

## Les objets distribués
### … Middleware explicite

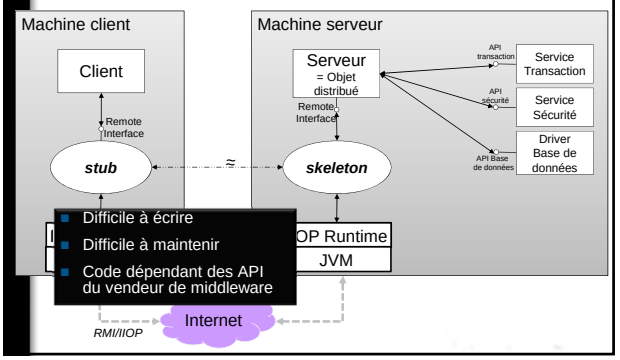



## Les objets distribués
### … Middleware explicite

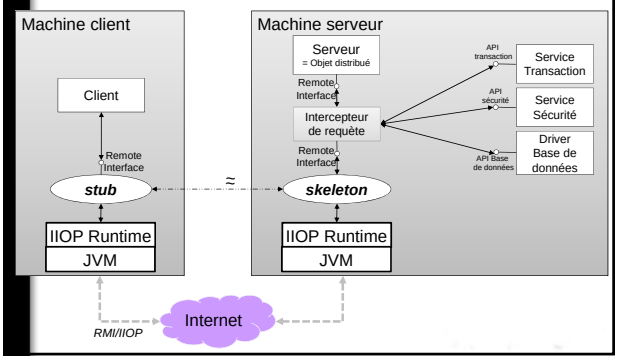



Exemple : transfert d'un compte bancaire vers un autre :
- transfert(Compte c1, Compte c2, long montant)
  1. Appel vers l'API middleware qui fait une vérification de sécurité,
  2. Appel vers l'API de transaction pour démarrer une transaction,
  3. Appel vers l'API pour lire des lignes dans des tables d'une BD,
  4. Faire le calcul : enlever de l'argent d'un compte pour le mettre dans l'autre
  5. Appeler l'API pour mettre à jour les lignes dans les tables,
  6. Appeler l'API de transaction pour terminer la transaction.

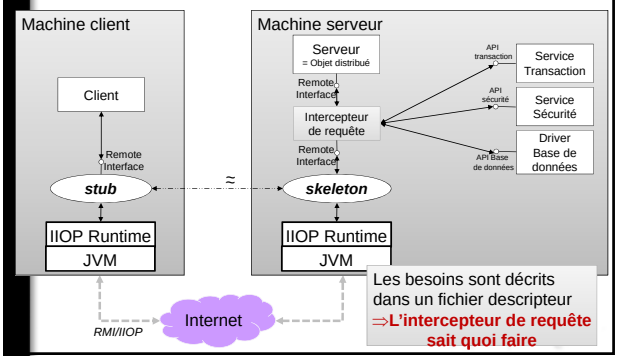
## Les objets distribués
### … Middleware explicite




## Les objets distribués
### … Middleware implicite

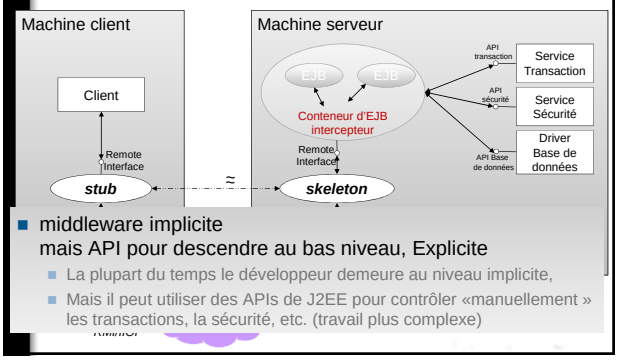



## Les objets distribués
### … Middleware implicite




Les besoins sont décrits dans un fichier descripteur
⇒**L'intercepteur de requête sait quoi faire**

## Les EJB : des objets distribués
### … *RMI-IIOP* au cœur des EJBs




- middleware implicite
  mais API pour descendre au bas niveau, Explicite
  - La plupart du temps le développeur demeure au niveau implicite,
  - Mais il peut utiliser des APIs de J2EE pour contrôler «manuellement » les transactions, la sécurité, etc. (travail plus complexe)

Un peu d'implémentation avec EJB 2.x

## EJB Object

- EJB 3.0 simplifie la tâche du développeur en cachant des détails d'implémentation
- L'étude de EJB 2.x permet de comprendre comment fonctionnent les EJB
- Pour chaque EJB écrit par le développeur, le serveur d'application crée un objet (*EJB Object*) qui contient le code qui va permettre au serveur d'intercepter les appels de méthode de l'EJB

## Rôle de l'EJB Object

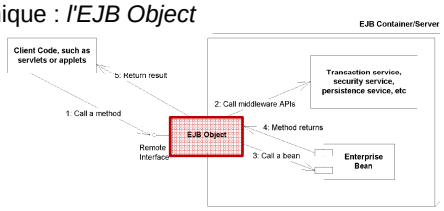
- Les clients n'invoquent jamais directement les méthodes de la classe du Bean
- Les appels de méthodes sont en fait envoyés à l'EJB Object
- Une fois les traitements effectués pour les transactions, sécurité,.. le container appelle les méthodes de la classe du bean

## Constitution d'un EJB : *EJB Object*

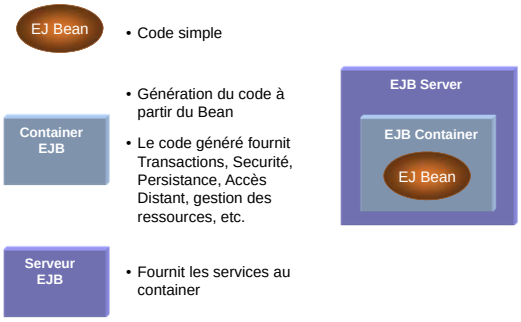
- Que se passe-t-il lors de l'interception ?
  - Prise en compte des transactions,
  - Sécurité : le client est-il autorisé ?
  - Gestion des ressources + cycle de vie des composants : threads, sockets, connexions DB, pooling des instances (mémoire),
  - Persistance,
  - Accès distant aux objets,
  - Threading des clients en attente,
  - Clustering,
  - Monitoring : statistiques, graphiques temps réel du comportement du système…
  - …

## Constitution d'un EJB : *EJB Object*

- Container = couche d'indirection entre le client et le bean
- Cette couche est matérialisée par un objet unique : *l'EJB Object*

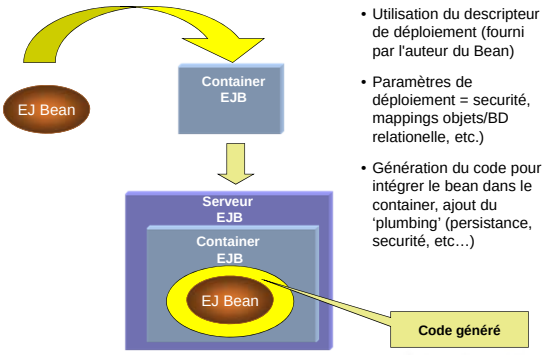



## EJB : classe du Bean et EJB Object




- Code simple
- Génération du code à partir du Bean
- Le code généré fournit Transactions, Securité, Persistance, Accès Distant, gestion des ressources, etc.
- Fournit les services au container

## EJB Object : génération du code




- Utilisation du descripteur de déploiement (fourni par l'auteur du Bean)
- Paramètres de déploiement = securité, mappings objets/BD relationelle, etc.)
- Génération du code pour intégrer le bean dans le container, ajout du 'plumbing' (persistance, securité, etc…)

## Les interfaces

## Interfaces

- Pour chaque EJB session, le développeur doit fournir une (ou 2) interface qui indique les méthodes de l'EJB que les clients de l'EJB pourront appeler
- Les autres méthodes de l'EJB servent au bon fonctionnement de l'EJB
- Un EJB session peut avoir une interface locale et une interface distante

## Interface locale

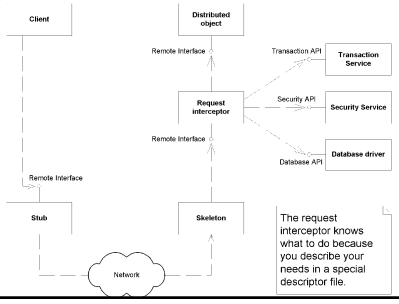
- Si l'EJB n'a qu'une seule interface locale, il ne peut être utilisé que par les classes qui sont dans le même container
- Le développeur peut ne fournir aucune interface ; en ce cas, une interface locale est automatiquement créée, qui contient toutes les méthodes publiques de l'EJB

## Interface distante

- Indispensable si l'EJB peut être utilisé par des classes qui ne sont pas dans le même container (application distribuée)
- Pour manipuler un EJB à travers une interface locale, le serveur d'application utilisera RMI-IIOP, ce qui implique
  - des performances moins bonnes
  - les paramètres et les valeurs de retour sont transmis par recopie des valeurs (références pour un appel local)

## Avec les interfaces distantes

- Problème : la création de bean et l'appel de méthode distante coûtent cher !




## Avec les interfaces distantes

- Commentaires sur la figure précédente
  1. Le client appelle un *stub (souche)*,
  2. Le *stub* encode les paramètres dans un format capable de voyager sur le réseau,
  3. Le *stub* ouvre une connexion sur le *skeleton* (squelette),
  4. Le *skeleton* décode les paramètres,
  5. Le *skeleton* appelle *l'EJB Object*,
  6. *L'EJB Object* effectue les appels *middleware*,
  7. *L'EJB Object* appelle la méthode du bean,
  8. Le Bean fait son travail,
  9. On fait le chemin inverse pour retourner la valeur de retour vers le client !
  10. …Sans compter le chargement dynamique des classes nécessaires !

## Conclusion

- Favoriser les interfaces locales
- Ne jamais utiliser d'interfaces distantes si les EJBs et leurs clients sont dans le même container

Packaging

## Descripteur de déploiement standard

- Pour informer le container des besoins middleware, on utilise *un descripteur de déploiement* (XML)
  - Standardisé,
  - A l'extérieur de l'implémentation du bean.
  - Attention si on les écrit à la main !
  - Outils d'aide au déploiement : IDEs
  - Descripteurs peuvent être modifiés après le déploiement sans devoir recompiler
- application.xml, web.xml, ejb-jar.xml, faces-config.xml

## Descripteur de déploiement spécifique

- Descripteurs spécifiques au serveur d'application
  - Chaque vendeur ajoute des trucs en plus : load-balancing, persistance complexe, clustering, monitoring…
  - Dans des fichiers spécifiques (glassfish-resource.xml, glassfish-web.xml ou glassfish-application.xml avec GlassFish)

## Format de distribution des applications

- Les applications Java EE sont distribuées avec des fichiers jar (format zip)
- Le fichier jar contient les interfaces et classes Java, les fichiers de configuration et les ressources utilisées par l'application (images, sons,…)

## Types de fichiers d'archive

- Jar (Java ARchive) : fichier d'archive habituel qui contient des EJB, des classes Java ordinaires et les ressources associées
- War (Web ARchive) : fichier d'archive pour le Web, qui contient des servlets, des fichiers HTML, des pages JSF, des EJB et les ressources associées
- Ear (Entreprise ARchive) : réunissent des modules jar ou war

## Fichier ear

- Les applications Web peuvent ne contenir qu'un seul fichier war
- Les applications plus complexes, par exemple qui utilisent des MDB, contiennent plusieurs fichiers jar qui sont réunis en un seul fichier ear (format jar avec une structure particulière qui permet de contenir plusieurs fichiers jar)

Service Oriented Architecture

## SOA (Service Oriented Architecture)

- **SOA** = un paradigme de conception pour lequel une application est composée de services faiblement couplés qui peuvent tourner sur des cpu différents et qui sont facilement localisables
- **Service** = composant de l'application qui remplit une fonctionnalité bien définie, avec une interface bien définie
- Un service ne dépend pas d'un contexte externe

## EJB et SOA (Service Oriented Architecture)

- **Web Service** = un exemple de SOA, adapté au Web
- Exemple : Google fournit de nombreux services Web tels que Google Map ou Google Calendar
- Plusieurs spécifications de Java EE permettent de créer facilement des services Web à partir des EJB (en particulier JAX-WS et JAXB)