Conception de serveurs d'applications ouverts

2/3

Du C/S au middleware Explicite

- CS :
  - Le client et le serveur sont développés en collaboration
- Objet distant :
  - Client et serveur sont liés par une interface
  - La couche réseau est masqué au client et au serveur
- ==> Notion de code applicatif/code non applicatif

Code applicatif/code non applicatif

- Le **code applicatif** est le code propre à l'application développée (aussi appelé **code métier** ou **BusinessCode**)
  - Ex : Banque ==> compte, retrait, dépôt
- Le code non-applicatif est le code non spécifique à l'application développée
  - Ex :
    - Accès réseau, Accès à la base de données, Debug, Log…

==> Pourquoi ne pas automatiser systématiquement le code non-applicatif

Principes

- Le code non-applicatif :
  - Ne doit pas apparaître dans le code applicatif
  - Il est accessible par la notion de services
    - Service de persistance (Base de données)
    - Service de présentation (html/http)
    - Service de log, Service d'authentification
    - Service de cycle de vie
  - Un service présente
    - une interface = ensemble de méthodes
  - La programmation objet permet de masquer entièrement le comportement d'un objet
  - L'association code applicatif/non-applicatif se fait de manière déclarative … (résolue au run-time, souple, adaptable)

Exemple : Logger

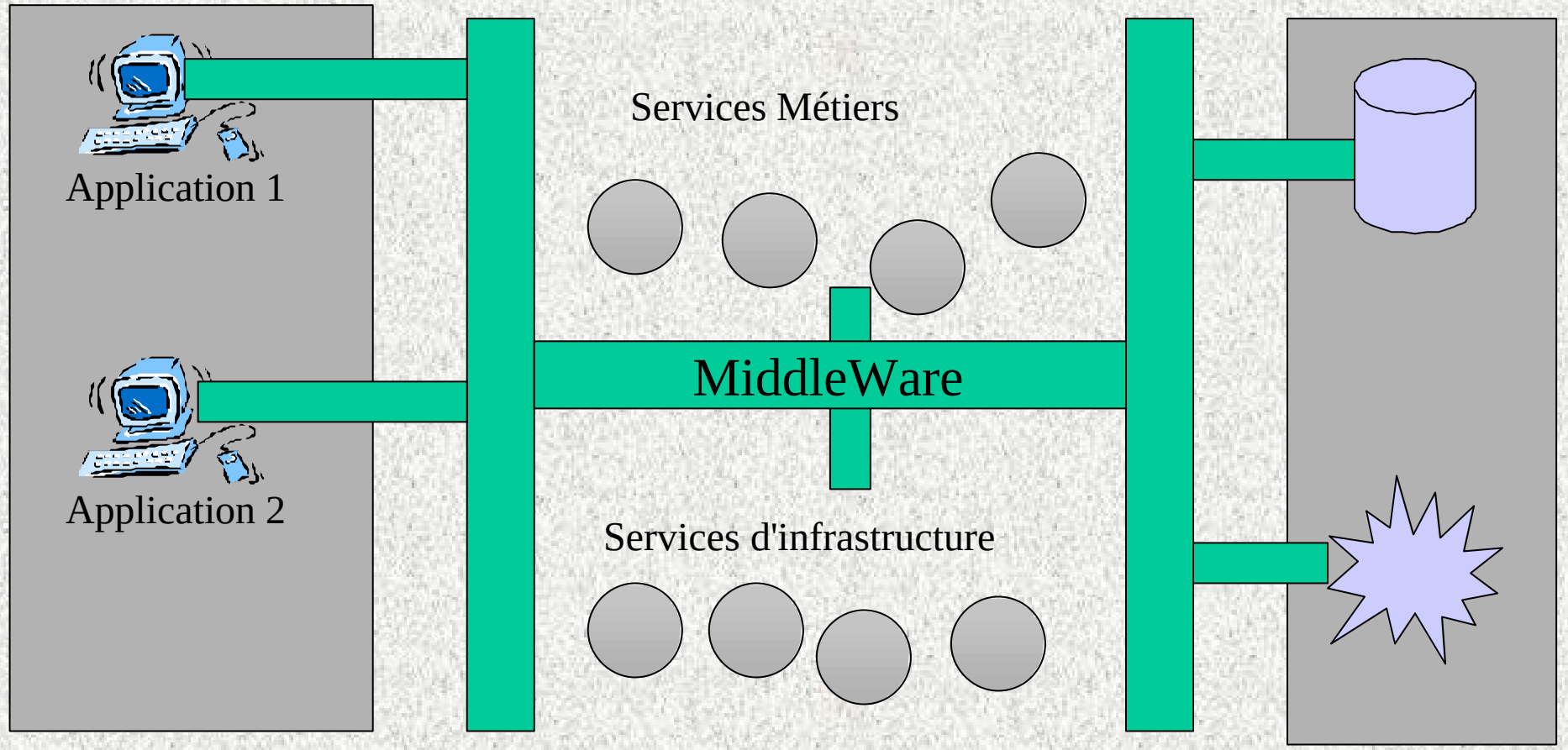
```
public class test{
  public void uneMethode(){
    System.out.println("entrée dans la méthode");
    i++;
    System.out.println("sortie de la méthode");
  }
}
```

==> Quels sont les inconvénients de ce code ?

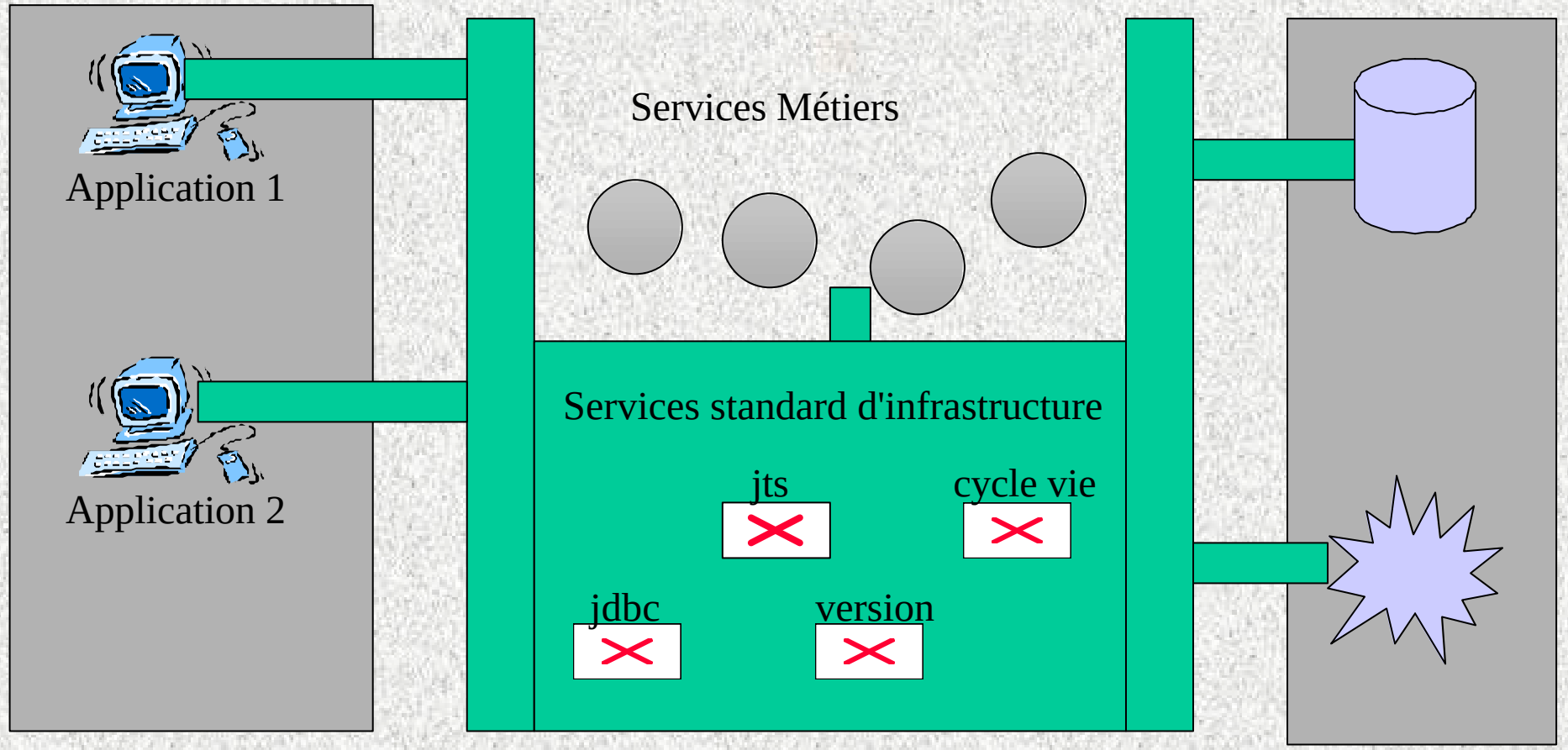
==> Quels sont les avantages de passer par un service ?

Les Conteneurs

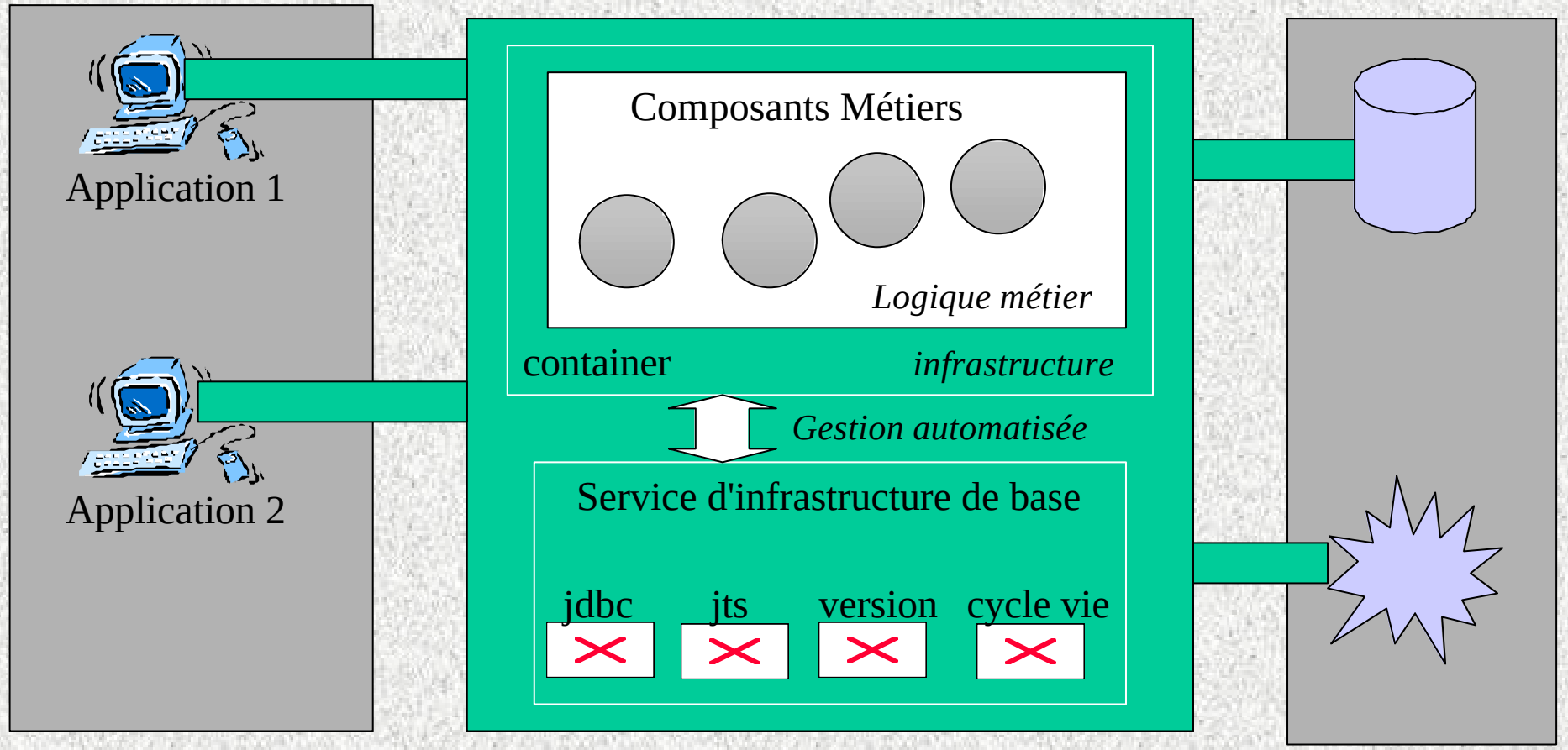
Architectures à Objet Distribués



Serveur de composants de base



Serveur de composants intégré



Exemple de services offerts par un conteneur

- Services internes
  - Gestion de la charge du serveur
    - (cycle de vie, accès client, passivation...)
  - Service de nommage
  - Gestion des accès aux objets métiers
- Services externes
  - Gestion du mapping sur BD relationnelle
  - Gestion des transactions
  - Gestion des échanges de messages

Un conteneur c'est :

- Une boîte qui automatise
  - La communication avec des services non-fonctionnels
  - La gestion des applications
  - Le cycle de vie d'une application pour son client

==> Qui réalise une interception entre le « client » et le « service » afin de réaliser des tâches

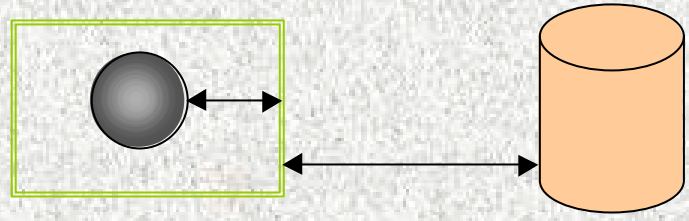
- Economie de code,
- Economie de moyen,
- Simplification pour le programmeur,
- et l'hébergeur

Interception de code 1/3

- Explicite
  - Le développeur inclut son propre code d'accès au service

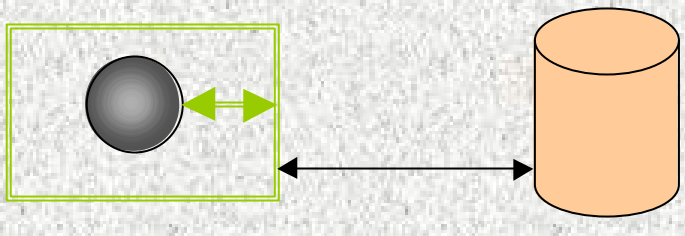
Interception de code 2/3

- Implicite : Le conteneur joue un rôle de proxy
  - Le conteneur fournit une API similaire
  - Le client croit voir une base de données alors qu'il voit le conteneur
  - ==> Avantages



Interception 3/3

- Automatique : Le conteneur automatise la vision du service
  - Le conteneur réalise les opérations standards du client
  - Le client ne voit rien, il est automatiquement peuplé de données
  - ==> Exemple : Base de données, Transaction...

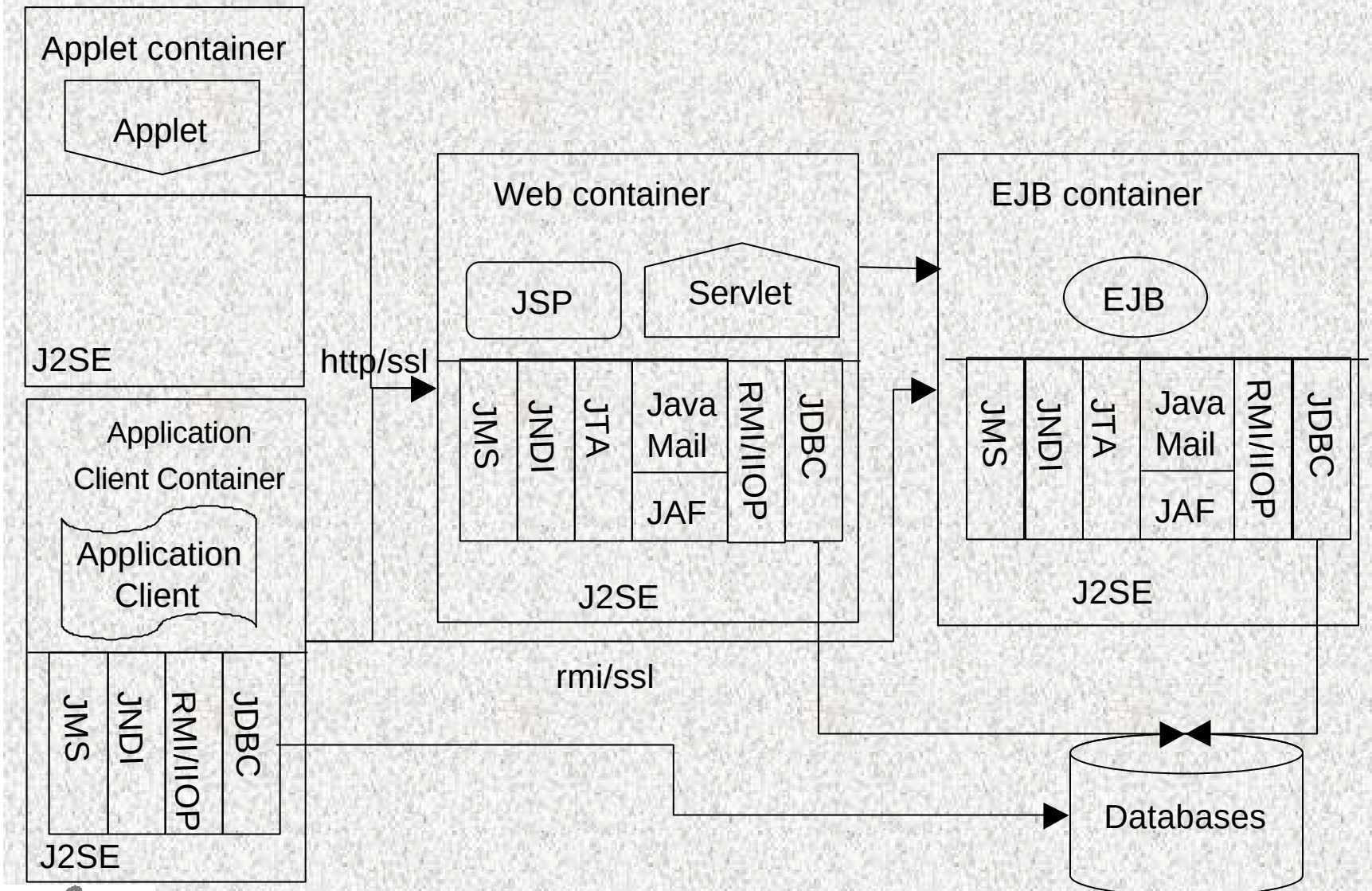


Les Serveurs d'applications

Un serveur d'application

- Application qui cherche à simplifier la programmation, et l'administration de grands systèmes
- Hébergeant :
  - Des containers
    - Pages Web
    - Composants Métier (EJB)
  - Des services
    - Nommage
    - Base de données/Mapping sur Base
    - Moniteurs transactionnels
    - Déploiement ...
  - Des API sur les services
    - JDBC/JTS/JMS…
- Deux grandes familles de serveurs d'applications
  - Les interfaces utilisateurs
  - Les applications distribuées

The J2EE Architecture



Les offres de serveur d'applications

- Serveur d'application J2EE
  - Weblogic BEA,WebSphere IBM, Iplanet Sun, Oracle
  - WebObject (Jonhatan/Jonas) INRIA
  - Jboss (Free)
- Autres serveurs d'applications
  - Microsoft .net
  - Zope (Python)
  - OpenACS
  - Serveur CORBA (OrbixWeb)

Le développement d'applications sur les SA

- Développement du code applicatif :
  - Phase de développement « classique »
  - L'appel à des services externes se fait soit :
    - de manière explicite dans le code
    - de manière implicite
- Packaging du code
  - Le code est regroupé dans une archive (jar, tar, rpm)
    - Du code d'exécution
    - Des indications de dépendances
    - Des indications d'interaction avec les services
- Déploiement du code code
  - Le code est déployé sur une machine d'exploitation
    - Si il y a des bugs, l'ensemble du code est réinstallé

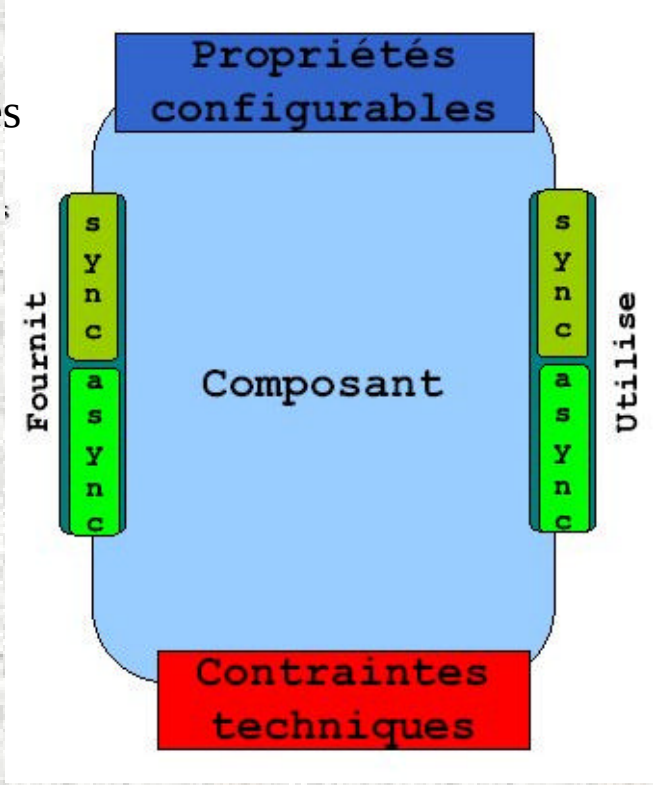
Les composants logiciels

Les composants

- Quoi ?
  - Définition usuelle
    - module logiciel autonome pouvant être installé sur plusieurs plates-formes
    - qui exporte différents attributs, propriétés ou méthodes
    - qui peut être configuré
    - qui peut être transporté / déployé
  - But
    - Brique de base pour concevoir des applications par composition

Modèle à composant : Composant

- Coopération
  - Interface fournie
    - Composantes, interfaces, opérations, propriétés
  - Interface requise
    - Composition et références
  - Mode de communication
    - sync, async, flots
- Propriété configurable du composant
- Contraintes techniques (Qos)
  - middleware : placement, sécurité, transaction
  - interne : cycle de vie, persistance
  - implantation : os, bibliothèque, version

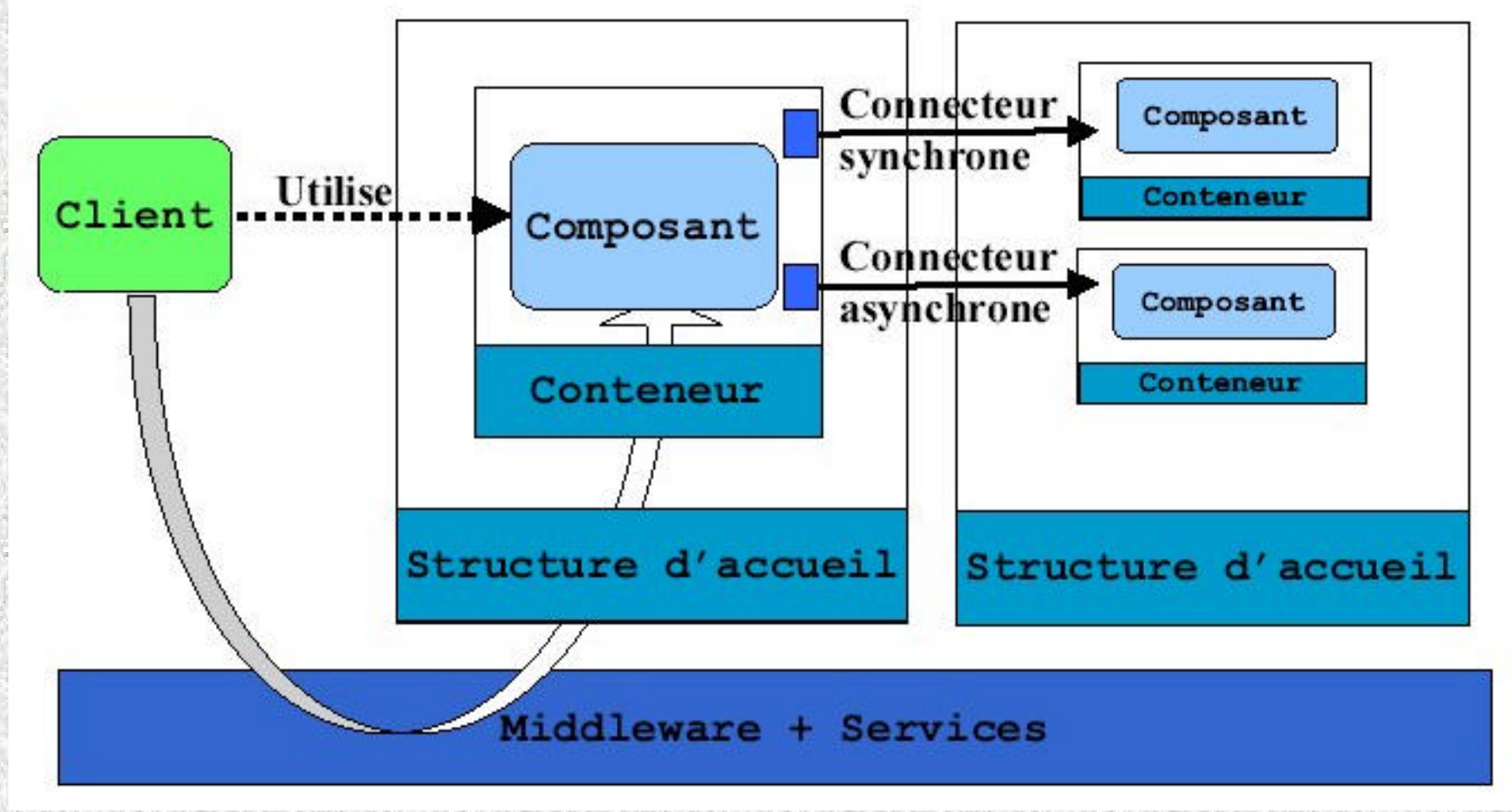


@ src : didier Donsez

Modèle à composants : Conteneur et Structure d'accueil

- Conteneur
  - Encapsulation d'un composant (et ses composantes)
  - prise en charge (masque) les services systèmes
  - nommage, sécurité, transaction, persistance ...
  - prise en charge partielle des connecteurs
  - invocations et événements
  - techniquement par interposition (ou délégation)
- Structures d'accueil
  - espace d'exécution des conteneurs et des composants
  - médiateur entre les conteneurs et les services systèmes
  - des + comme le téléchargement de code (navigateur)

Modèle à composants : conteneurs & structures d'accueil



Composants : de l'installation à l'introspection

- Installer les composants
  - technologie de packaging
  - production des conteneurs
- Créer les composants
  - par des fabriques (maisons / « home »)
  - configuration des valeurs initiales
- Retrouver les composants
  - services de désignation (Nommage ou Vendeur) ou maisons
- Utiliser
  - invocation synchrone et événements
- Introspection
  - découvrir leurs APIs (fonctionnelle)
  - découvrir les connecteurs (structurelle)

Construction par assemblage de composants

- Construction par assemblage plutôt que ingénierie de développement
  - réduire les besoins en compétence technique
  - focaliser l'expertise sur les problèmes du domaine
- Langage de description d'Architecture (ADL)
  - capturer les composants
  - fonctionnalités et besoins
  - capturer les connecteurs
  - composition et modes de communication
  - impédance entre composants => adaptateurs
  - C'est le point faible des solutions industrielles !

Exemple : les EJB

Enterprise JavaBeans

- Enterprise Java Beans :
  - Composants logiciels serveur
- Objectif
  - Standardiser le développement et le déploiement de composants serveurs écrits en Java
  - Le développement ne se fait que sur l'interface métier de l'objet (code fonctionnel)
  - Le conteneur d'EJB prend en charge tout ce qui n'est pas du code fonctionnel

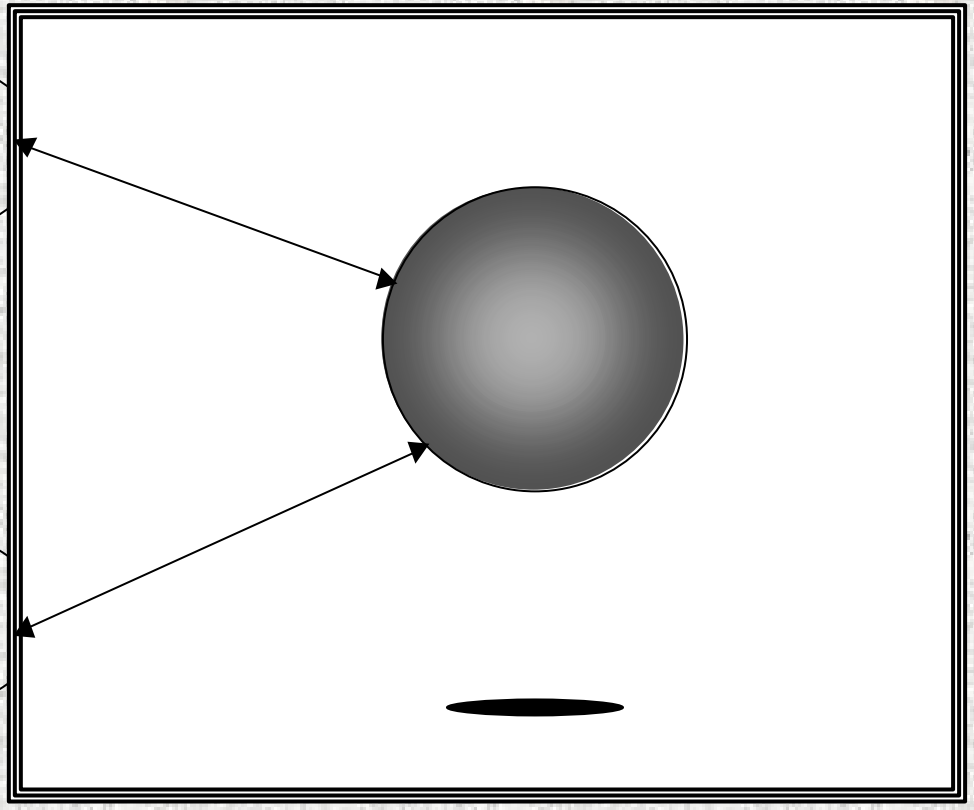
2 principes

- Les interfaces :
  - Représentent la vision "client" d'un objet
    - L'interface Home, représente la vision "cycle de vie" de l'objet
    - L'interface Remote, représente la vision "fonctionnelle » de l'objet
  - Elles sont concrétisées par une classe d'implantation
- Le conteneur
  - Exécute la classe d'implantation
  - Gère les aspects non-fonctionnels

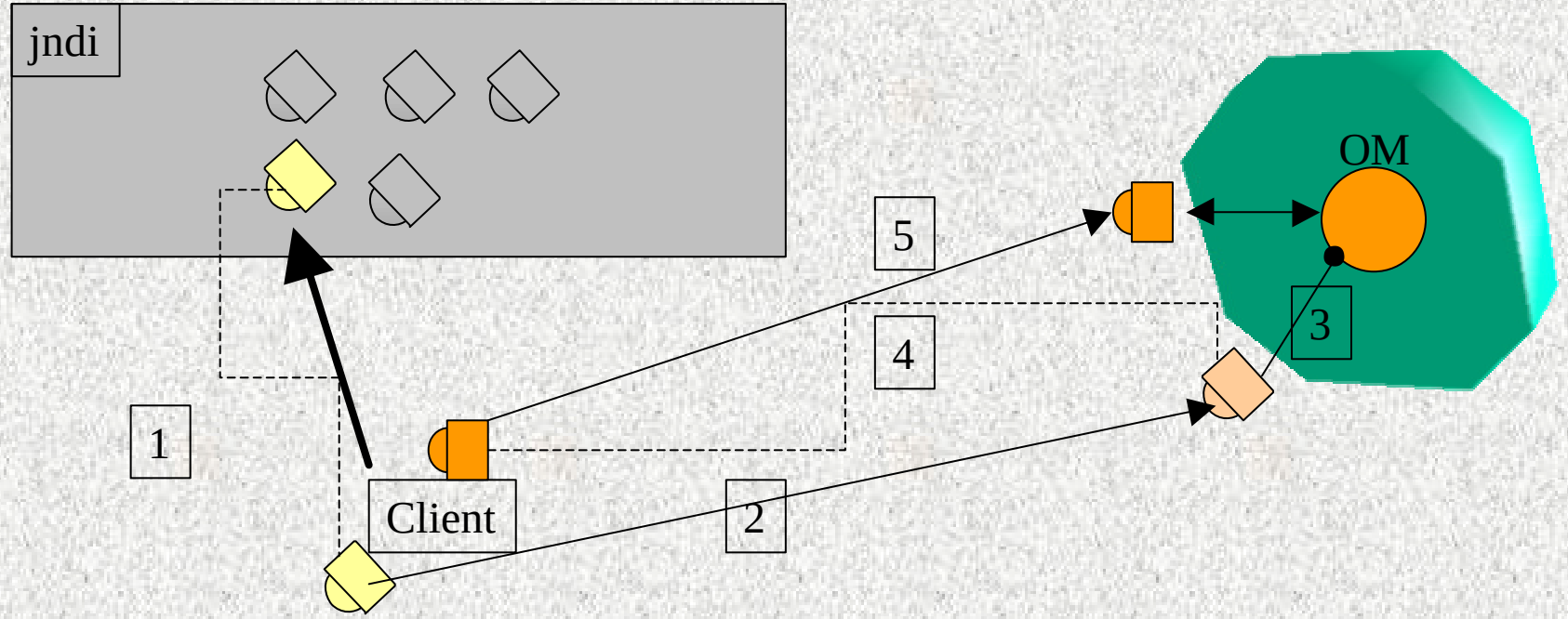
La vision d'un EJB

Home
- cycle de vie
- meta-information

Remote
- code métier



Le fonctionnement global



Les services offerts par le conteneur

- **Contrôle de la charge** : Il permet de contrôler le nombre d'instances actives, par l'intermédiaire de l'usine de fabrication
- **Transparence du réseau** : Les interfaces sont concrétisées par des stubs rmi lorsque client et serveur sont distants
- **Surveillance des instances inactives** : Une instance non utilisée pendant un certain temps est automatiquement sauvée et purgée de la mémoire
- **Gestion des time-out** : Une requête d'un objet qui met plus de 30 secondes est automatiquement annulée
- **Gestion des données** : Les données d'un objet sont automatiquement synchronisées avec une base de données
- **Gestion des événements** : Le conteneur notifie automatiquement un objet si un message réseau arrive
- **Cryptage, Sécurité** ...

Principe de conception des EJB

- Le modèle des EJB est fondé sur quatre concepts pour la conception de systèmes distribués
  - Approche de serveurs sans-états
  - Approche orientée session
  - Approche objet persistant
  - Approche objet orientés messages
- Les spécifications EJB parlent de
  - Session Beans
    - Stateless Session Bean
    - Statefull Session Bean
  - Entity Beans
    - Container-Managed Persistence
    - Bean-Managed Persistence
  - Message-Driven Bean

Cycle de développement d'un EJB session

1) Définir les interfaces métier
   - Interface Remote
2) Définir les méthodes de gestion du cycle de vie
   - Interface Home
3) Définir le code des fonctions
   - Classe Bean
4) Définir les caractéristiques intrinsèques du bean dans un fichier de description
5) Définir les caractéristiques de déploiement dans un fichier de description
6) Déployer le bean
7) Réaliser le(s) clients qui utilisent le service

Interface Métier (Remote)

```
package exemple.fibonacci;

public interface Fibonacci extends javax.ejb.EJBObject {
   public int getFibonacciNumber(int n) throws java.rmi.RemoteException;
}
```

Interface Home

```
package exemple.fibonacci;
import javax.ejb;
import java.rmi;

public interface FibonacciHome extends EJBHome {
 public Fibonacci create() throws CreateException, RemoteException;
}
```

Développement du Bean

```
package exemple.fibonacci;
import java.rmi.*;
import javax.ejb.*;
public class FibonacciBean implements SessionBean {
  public void ejbPassivate(){}
  public void ejbActivate(){}
  public void ejbRemove(){}
  public void setSessionContext(SessionContext ctx){}
  public void ejbCreate() {
    System.out.println("Cet EJB Fibonacci est créé";
  }
  public int getFibonacciNumber(int n){...}
}
```

Descripteur du Bean

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
    JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
 <enterprise-beans>
   <session>
      <ejb-name>fibonacci</ejb-name>
      <home>exemple.fibonacci.FibonacciHome</home>
      <remote>exemple.fibonacci.Fibonacci</remote>
      <ejb-class> exemple.fibonacci.FibonacciBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
    ...
 </enterprise-beans>
</ejb-jar>
```

Descripteur du déploiement

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD WebLogic 6.0.0
    EJB//EN" "http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd" >
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>fibonacci</ejb-name>
    <stateless-session-descriptor>
      <pool><max-beans-in-free-pool>100</max-beans-in-free-pool></pool>
    </stateless-session-descriptor>
    <transaction-descriptor>
       <trans-timeout-seconds>300</trans-timeout-seconds>
    </transaction-descriptor>
    <jndi-name>fibonnaci</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

Déploiement du bean

- Eventuellement (pré-compilation)
- Jar + cp

- Au déploiement
  - Inscription de l'Usine (interface Home) sur le containeur
  - Dépôt du stub de manipulation de l'Usine sur le service de nommage

Développement d'un client

```
public class Client{
  public static void main(String [] argv){
    try{
      FibonacciHome home=(FibonacciHome)context.lookup("fibonaci");
      Fibonacci jacques=home.create();
      jacques.getFibonnaciNumber(10);
    ...
  }
}
```

Développement d'un client JSP

```
<html><head><title> <%= pagetitle %> </title></head>
<h2><font color=#DB1260><%= pagetitle %></font></h2>
<%@ page import="example.fibonacci.*"%>
<%!String pagetitle = "Suite de fibonnaci";%>
<%try {
 ctx = getInitialContext();
 FibonnaciHome homeFib = (FibonnacciHome)ctx.lookup("fibonnaci");
 uneSuite=homeFib.create();
 out.println("fib(7)="+uneSuite.getFibonacciNumber(7));
}catch(Exception e){e.printStackTrace();
%>
</body></html>
```

Points forts

- Notion de container
  - Robustesse, standardisation, évolution
- Interface de développement standardisées
  - Pas / Peu de phase de prise en main
- Automatisation de nombreuses tâches
  - Gestion de la persistance, transactions…
- Intégration à l'API java
- Marché explosant