Introduction au développement Web avec J2EE

Développement Web et J2EE

Un aperçu

P. André

Master MIAGE Nantes

MIAGE

Université de Nantes

Introduction

La motivation : développer des systèmes d'information avec les technologies actuelles.

- bases de données SGBD relationnels
- applicatif portable multi-systèmes d'exploitation
- accès distant Internet, Client-serveur
- interfaces accessibles Navigateurs Web

Application : ARGOSI

- développement avec UML : processus, documentation, outils
- reprise d'un existant : itératif et incrémental
- architecture 3-4 tier : SGBD, serveur Web, client
- travail en groupe : gestion de projet

domaine complexe et évolutif

Introduction

Avertissement

L'objectifs de cette présentation est de faire un tour d'horizon de la problématique mais ne constitue en aucun cas un exposé approfondi du domaine.

Le lecteur consultera les détails techniques dans les documents et ouvrages référencés.

Sommaire de l'exposé

Principales références

- Développement Web [Con00, Tah06, Sar05, Lan03]
- J2EE [Mol05, Sar05, CFS03, Lan03]
- Eclipse et Java [Hol04, Dau04, Dja05, SM03, Dev06, Dou07]
- JSP [Mol05, Sar05, Cha04]
- EJB [CFS03]
- MVC, Struts, Hibernate [Mol05, Sar05, Cha04, Goo02] [Tah06, Dev06]
- ... très riche

Sommaire de la partie

Développement Web

Références

- Développent Web [Con00]
- Développent Web J2EE [Mol05, Sar05, CFS03]
- Autres [Tah06, Lan03, Dou07]

Caractérisation : développer une application logicielle qui utilise des serveurs Web et un réseau (Internet, Intranet, Extranet)

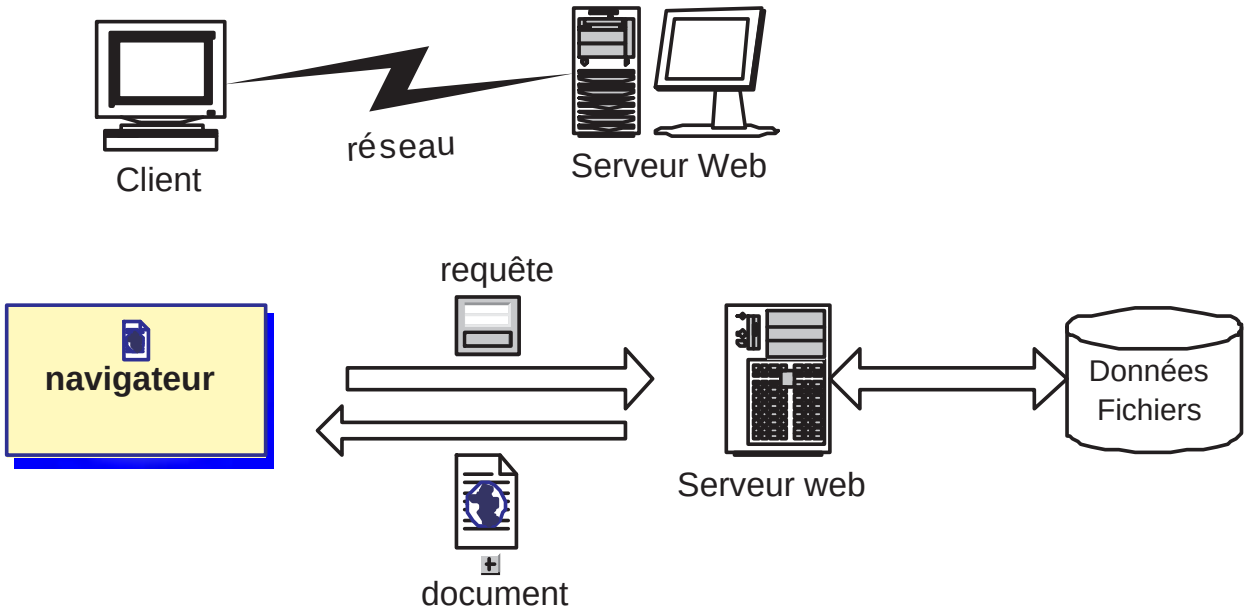
Elément clé : architecture (matérielle, logicielle), déploiement

- approche client/serveur (serveur Web)
- approches distribuées (middlewares, p2p, corba...) pas vu ici

$\Rightarrow$ Architectures Web

- langages
- outils associés

Architecture Web simple (2-tier)



- Client léger / lourd
- Serveur Serveur Web
- Langages/outils HTML, scripts (CGI, Javascript...), applications et plugins multimedia (audio, video) ou bureautique (word, openoffice, pdf...)

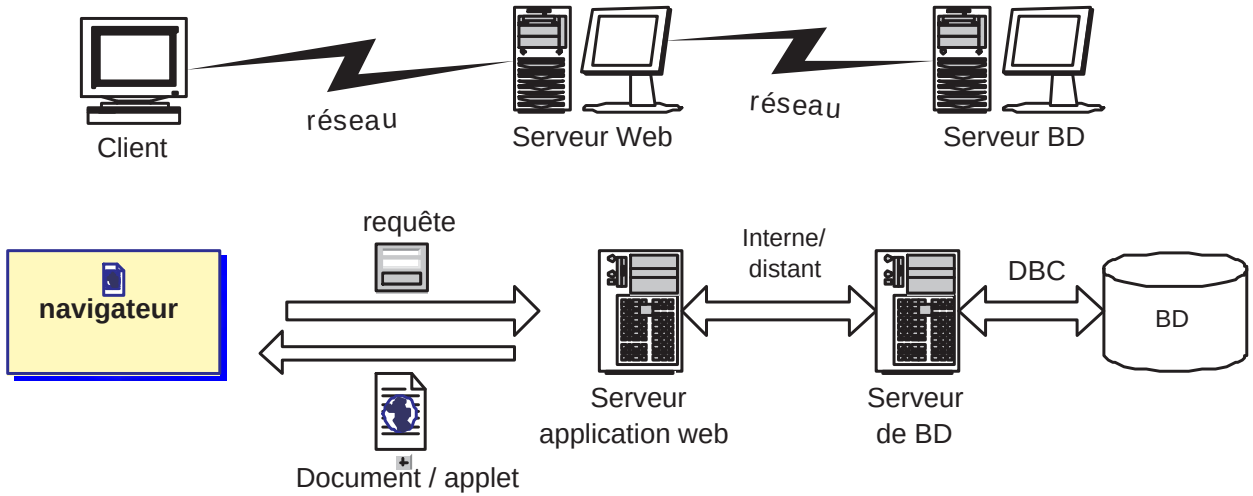
Architecture Web simple (2-tier) suite

Site Web = client + serveur web [Con00]

- Client
  - léger ⇒ IHM seule
    Navigateur Netscape, FireFox, IE, ...
  - lourd ⇒ IHM+Métier
    Navigateur + Plugins, Applications ; JavaWebStart...
    ActiveX, Swing, SWT...
- Serveur Serveur Web Apache, Internet Information Server (MS), IPlanet (netscape avant)...
- Langages/outils HTML, scripts (CGI, Javascript...), applications et plugins multimedia (audio, video) ou bureautique (word, openoffice, pdf...)

ne pas confondre avec architecture 2-tier (IHM+Métier, SGDB) e.g. Java + Oracle

Architecture Web simple (3-tier)



- Client plutôt léger Navigateur Netscape, FireFox, IE, ...
- Serveur Serveur d'application Web
  - serveur web ⇒ requêtes Web
  - serveur applicatif ⇒ Métier
- Serveur Serveur BD (Oracle, MySQL, Postgres...)
- Langages deux du 2-tier + langages spécifiques

Architecture Web simple (3-tier) suite

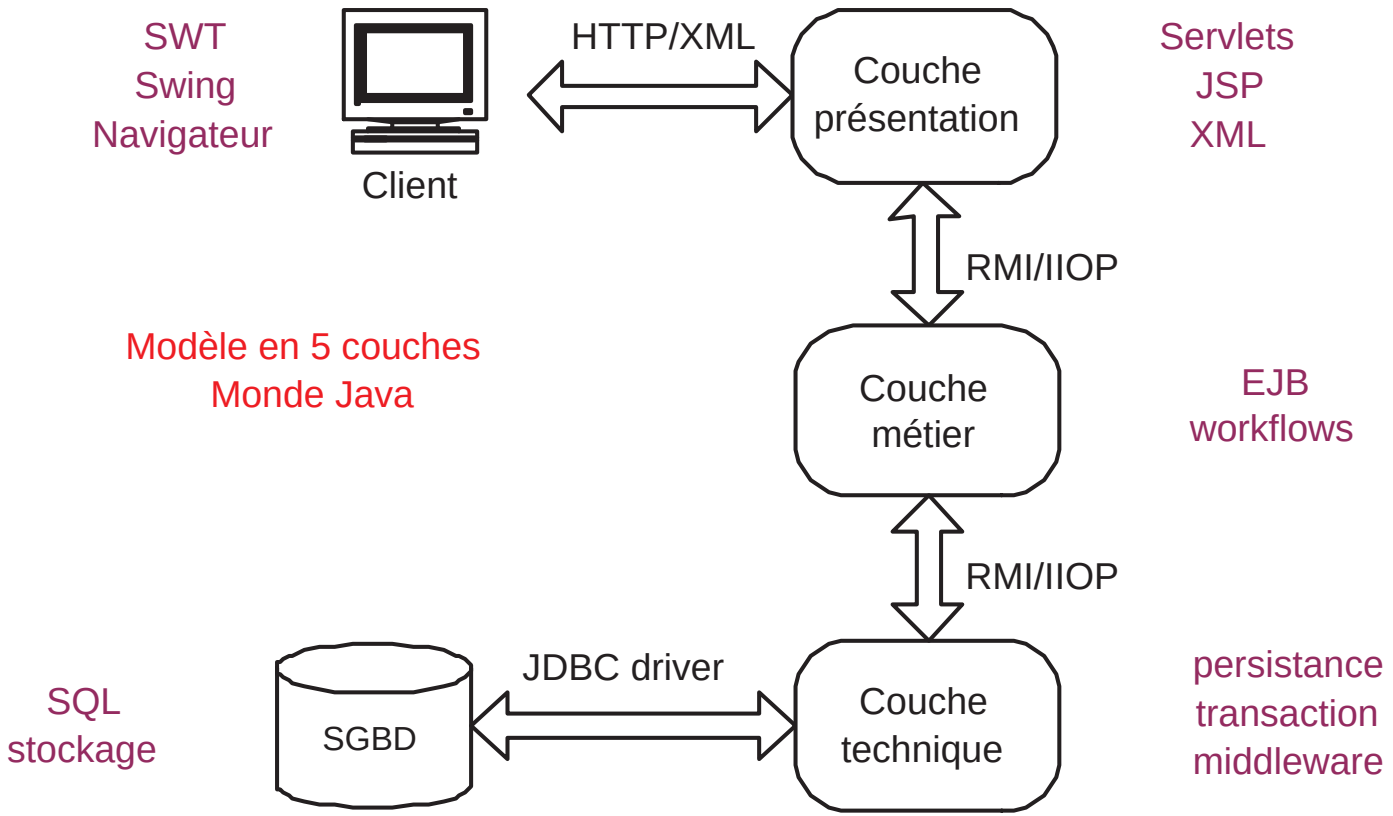
Application Web = client + serveur web + serveur d'application (traitement d'une logique applicative) [Con00]

- Client
- Serveur Serveur d'application Web
  - serveur web (Apache, IIS...)
  - serveur applicatif

  Souvent intégré (Tomcat, Websphere, JBoss, JRun...)
  2 approches
  - compilé : CGI, Internet Server API (MS), Netscape Server API, Servlet (J2EE)
  - interprété : ASP (Active Server Pages MS), JSP (Java Server Pages), Cold Fusion (Allaire)...
- Serveur Serveur BD (Oracle, MySQL, Postgres...)
- Langages deux du 2-tier + PHP, ASP, JSP, servlets, API serveurs web, ...

Architecture Web simple (5-tier)

[Mol05]



Architecture Web simple (5-tier) suite

- Client Navigateur, Applets, Java Web Start, ...
- Présentation de données partie serveur d'application Web rendant possible l'accès de clients riches ou légers
- Services métiers composants fonctionnels
- Services techniques composants gérant la persistance, la concurrence, les transactions...
- Stockage SGBD, XML, fichiers

Répond aux objectifs de la DSI

- déploiement simple, diminution des coûts
- Factorisation de la logique entreprise et séparation claire des préoccupations.
- Délégation de la partie technique à des composants et des équipes spécialisées.

cf cycle en Y

Architecture Web Argosi

Architecture Web autour de Java J2EE

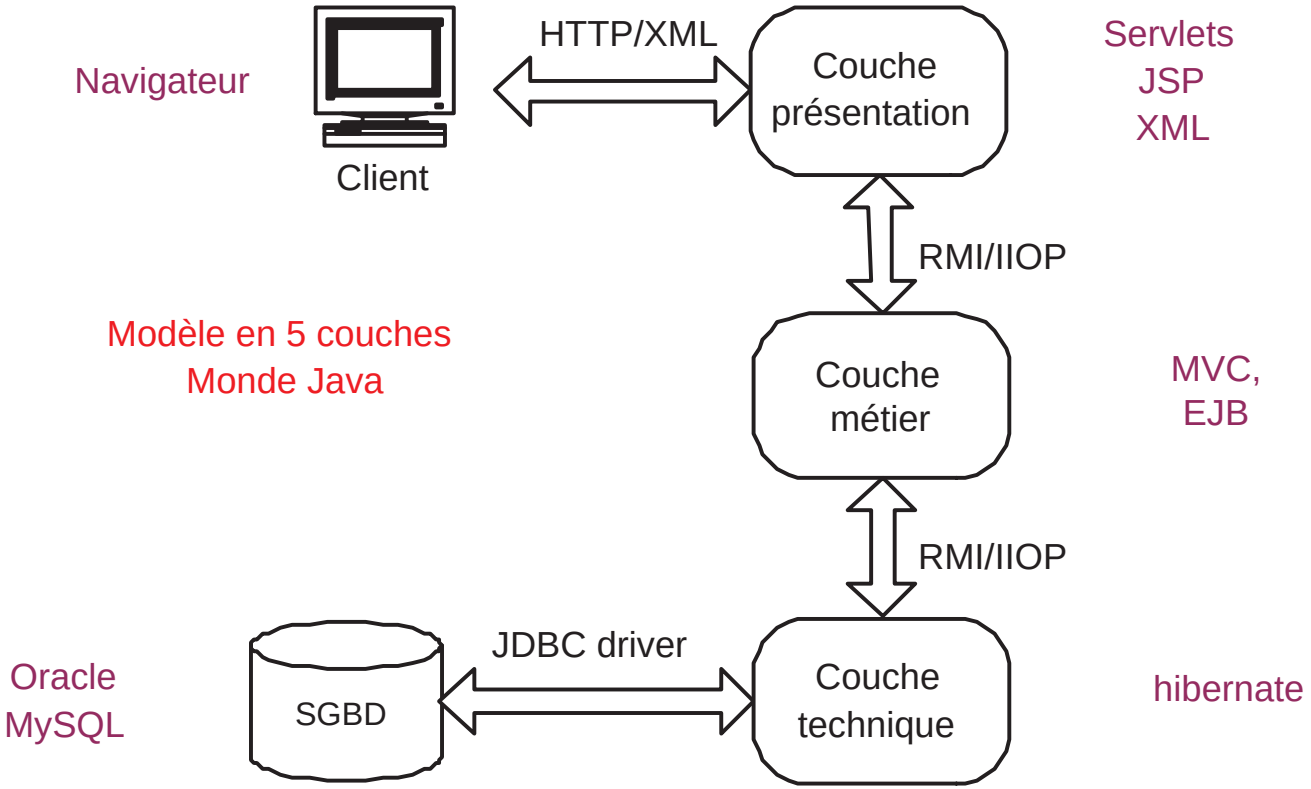


Fig.: Architecture Web simple

Architecture Web (solutions J2EE) suite

Architecture Web autour de Java J2EE

- Client léger Navigateur
- Serveur d'application J2EE Tomcat, Websphere, Jrun, Jboss
  comprenant le serveur Web (servlet, JSP)
  Eclipse - Lomboz (plugin J2EE)
- Couche métier EJB, MVC (Struts)
- Couche technique persistance (Hibernate), ...
- Couche BD Oracle, MySQL, SQL Server, PostGresSQL : JDBC

Architecture Web (patterns)

pour aller plus loin [Con00]

- Pattern Client web léger (*Thin Web Client*) Navigateur seul
- Pattern Client web lourd (*Thixk Web Client*) Navigateur + HTML dynamique, applets Jav, contrôles ActiveX
- Pattern Livraison web (*Web Delivery*) utilisation de IIOP ou DCOM en plus de HTTP pour un système d'objets distribués

Sommaire de la partie

J2EE

Java 2 Enterprise Edition

L'environnement J2EE fournit un ensemble d'API permettant de développer des sites Web dynamiques avec une technologie Java. [Sar05]. (applications réparties)

- définie par Sun `http://java.sun.com/`
- basée sur Java : Standard Edition, Enterprise Edition, Micro Edition
- applications types : systèmes d'information entreprise, commerce électronique...
- ensemble de technologies pour contruire des applications réparties
- implantation de référence : J2EE 5 SDK

sources : [Mol05, Sar05]
`http://www2.lifl.fr/~seinturi/middleware/index.html`
`http://fr.wikipedia.org/wiki/Java_EE`
`http://www.loribel.com`

J2EE technologies et services

Java 2 Enterprise Edition

Un ensemble de technologies pour contruire des apploications réparties

- Serveur d'application à base de
  - *Web component* JSP/servlet (pages web dynamiques)
  - *Business component* EJB (logique applicative)
- Services d'infrastructures (cf Corba)
  - JDBC (Java DataBase Connectivity) : API d'accès aux SGBDs
  - JNDI (Java Naming and Directory Interface) : service de noms (annuaire) de référencement des objets
  - JTA/JTS (Java Transaction API/Service) : service de gestion des transactions distribuées
  - JCA (J2EE Connector Architecture) est une API de connexion au système d'information de l'entreprise, notamment aux systèmes dits ■Legacy■ tels que les ERP.
  - JMX (Java Management Extension) fournit des extensions permettant de développer des applications web de supervision d'applications.
- Services de communication ...

J2EE services

- ... Services de communication
  - JAAS (Java Authentication and Authorization Service) est une API de gestion de l'authentification et des droits d'accès.
  - JavaMail est une API permettant l'envoi de courrier électronique.
  - JMS (Java Message Service) service de gestion des messages asynchrones (appelées MOM pour Middleware Object Message) entre applications.
  - RMI-IIOP est une API permettant la communication synchrone entre objets.

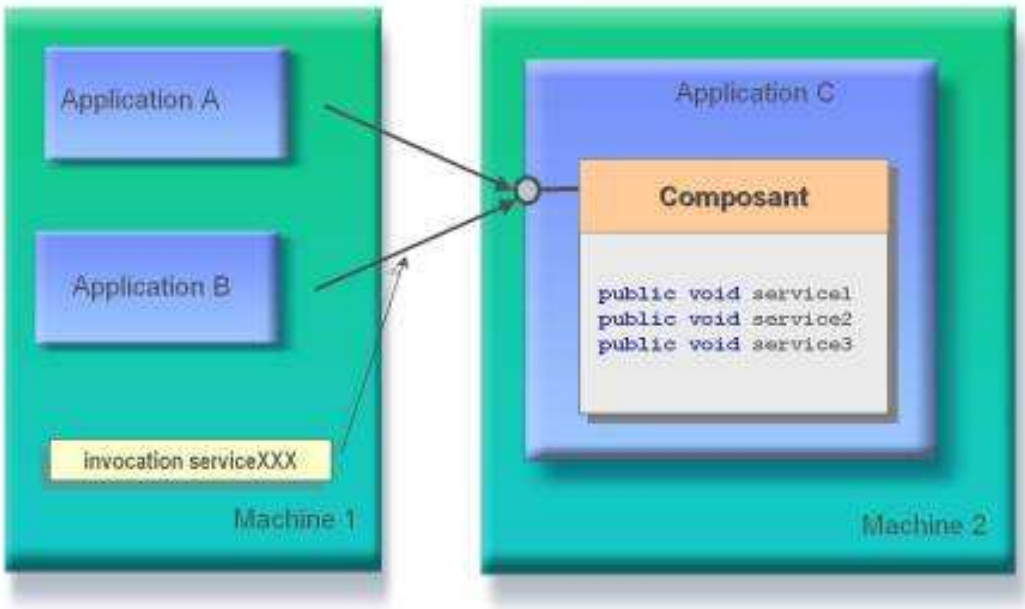
J2EE Plateformes

Plateformes compatibles (processus de certification mis en place par Sun)
Les plateformes sont en majorité J2SE 1.4

- commerciales
  - Websphere (IBM)
  - WebLogic (BEA)
  - Oracle Application Server
  - Sun Java System Application Server
  - ...
- *open source*
  - JBoss
  - JRun
  - JOnAS
  - Geronimo
  - OpenEJB
  - JFox
  - Tomcat
  - ...

J2EE Communications

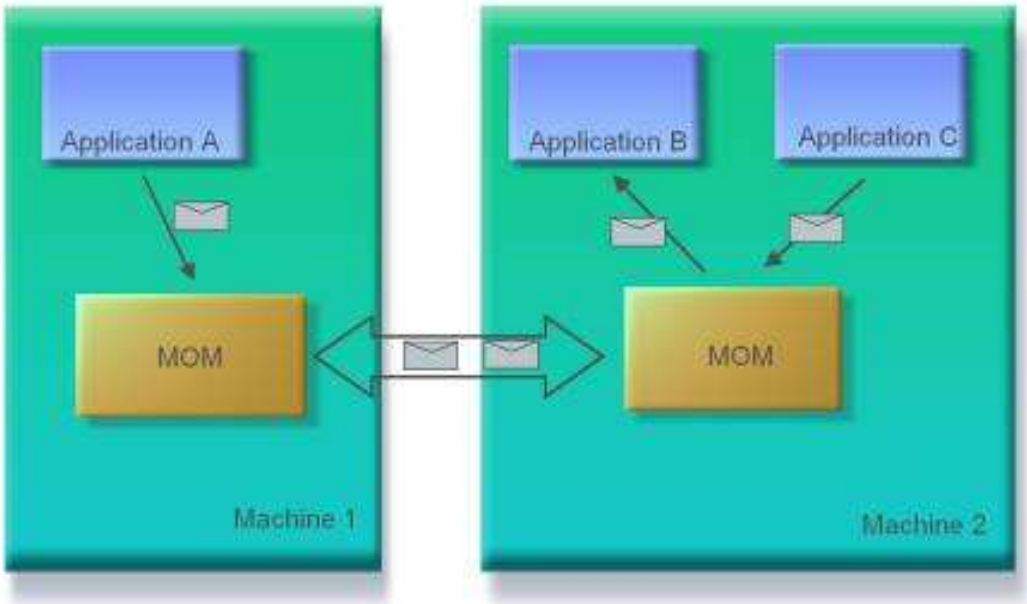
Les technologies d'appel de procédure à distance regroupant les standards tels que CORBA (OMG), RMI (J2EE), DCOM (Microsoft DNA) ou encore .NET Remoting (Microsoft .NET). Leur principe réside dans l'invocation d'un service (i.e. d'une procédure ou d'une méthode d'un objet) situé sur une machine distante indépendamment de sa localisation ou de son implémentation.



Principe des appels de procédure à distance

J2EE Communications

Les technologies d'échanges de messages. Les applications voulant communiquer entre elles s'échangent des messages véhiculés par l'infrastructure MOM selon différents modes de communication que nous détailleront par la suite, ces messages ont une nature complètement générique, ils peuvent représenter tous types de contenu aussi bien du binaire (image, objets sérialisés) que du texte (document XML).



Principe des « Middleware Oriented Message »

Conteneurs J2EE

- Application J2EE = 0..* composants Web + 0..* composants EJB
- Plusieurs rôles :
  - développeur de composants Web,
  - développeur de composants EJB,
  - assembleur d'applications,
  - déployeur et gestionnaire d'applications
- 4 services fournis par le serveur au conteneur EJB
  - cycle de vie,
  - transaction JTS,
  - nommage JNDI,
  - sécurité

  différent de Corba car ces services sont intégrés dès le départ à la plate-forme.

Conteneurs J2EE (suite)

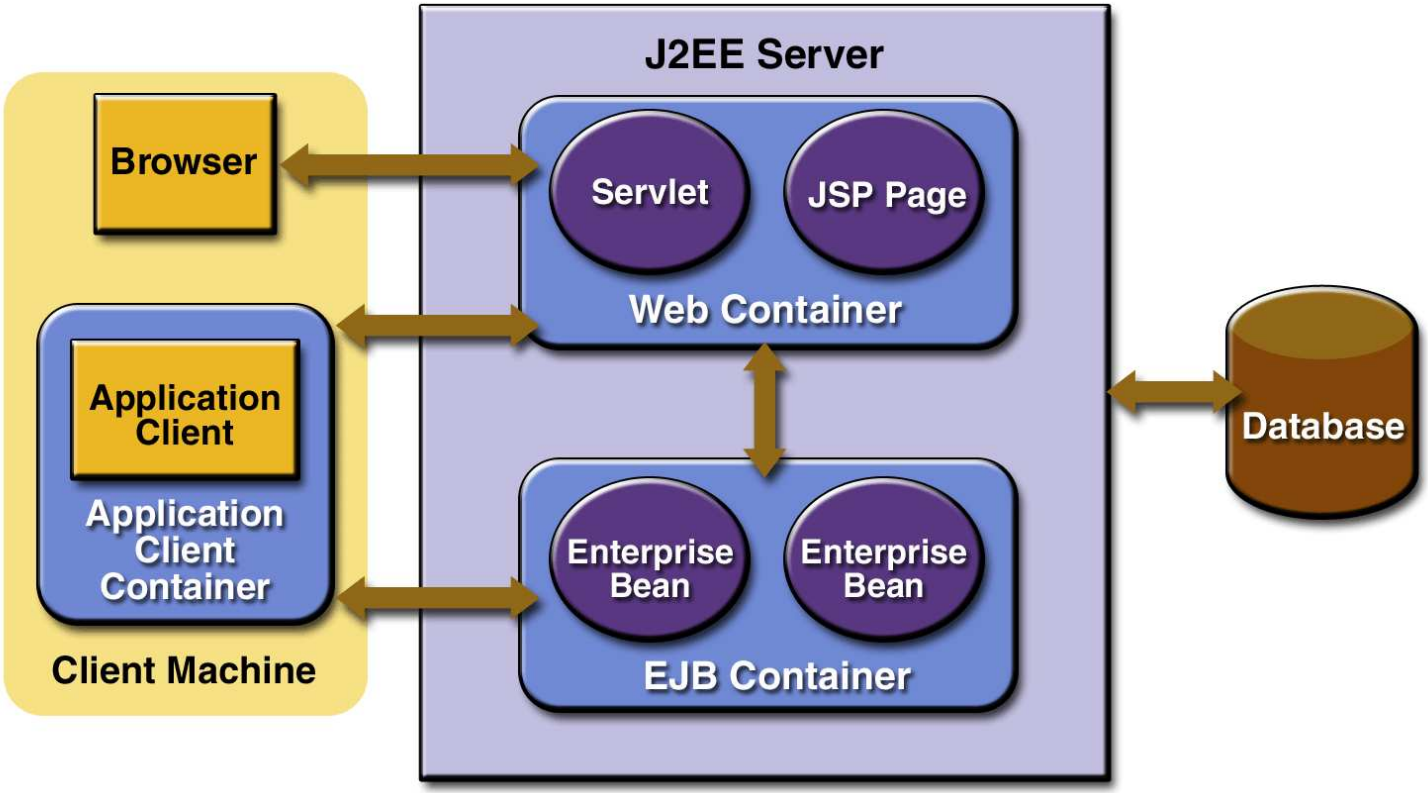


Fig.: Conteneurs J2EE

`http://java.sun.com/j2ee/1.3/docs/tutorial/doc/Overview4.html`

Architecture J2EE

Architectures Web autour de Java J2EE

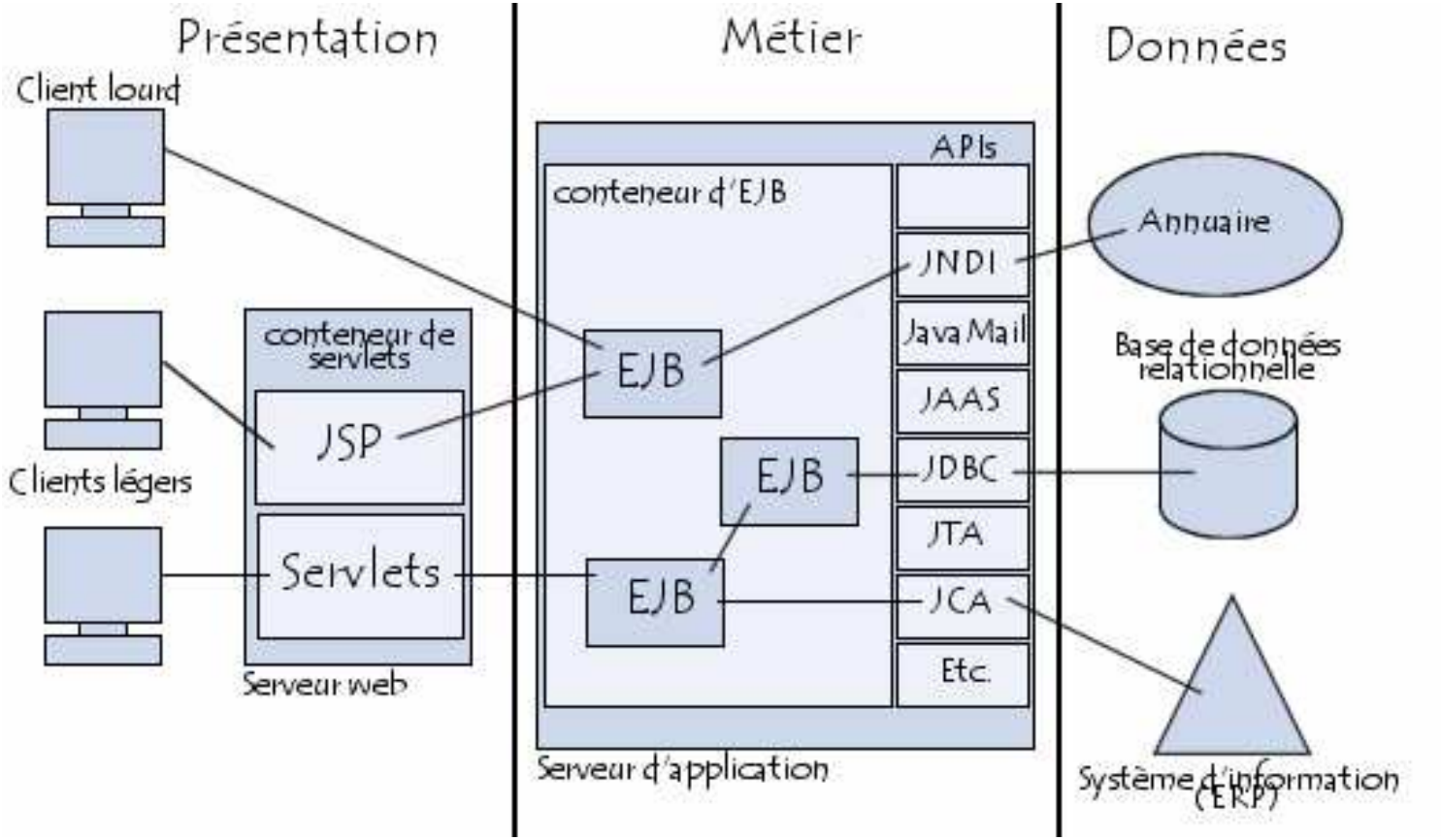


Fig.: Vue globale

http://www.commentcamarche.net/j2ee/j2ee-intro.php3

Architecture J2EE

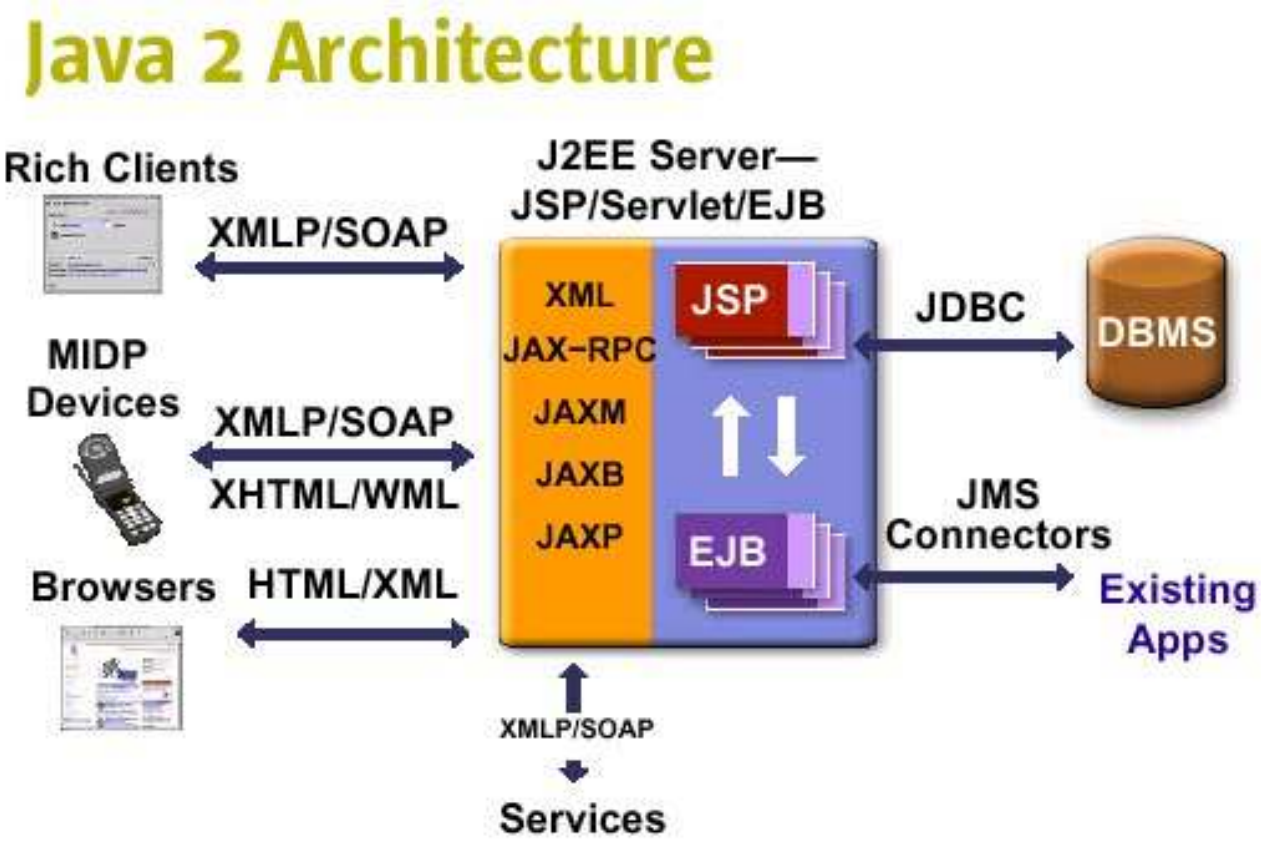


Fig.: Vision service Web

http://www.loribel.com

Architecture J2EE 5-tier

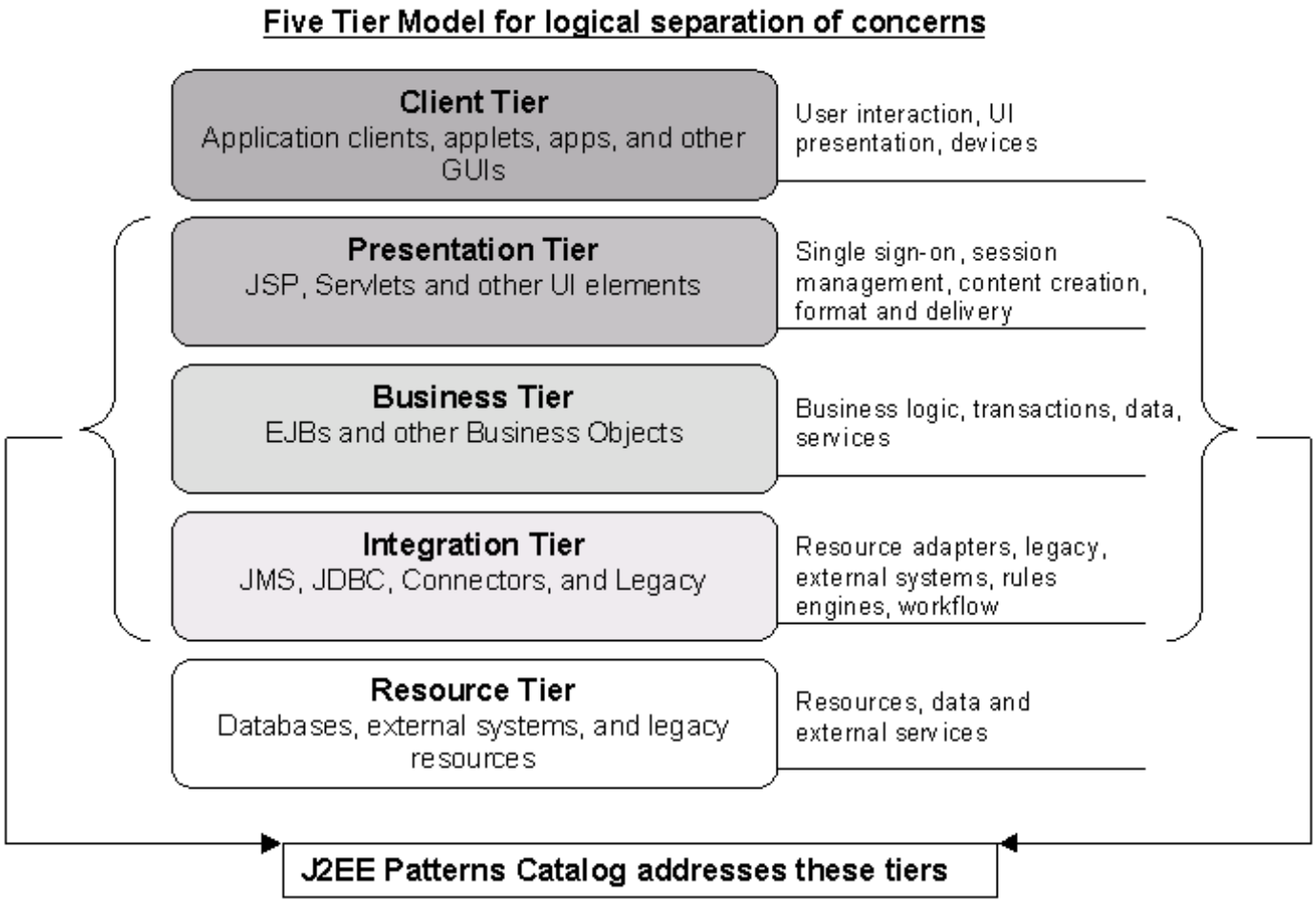


Fig.: Architecture J2EE 5-tier

Architecture J2EE n-tier

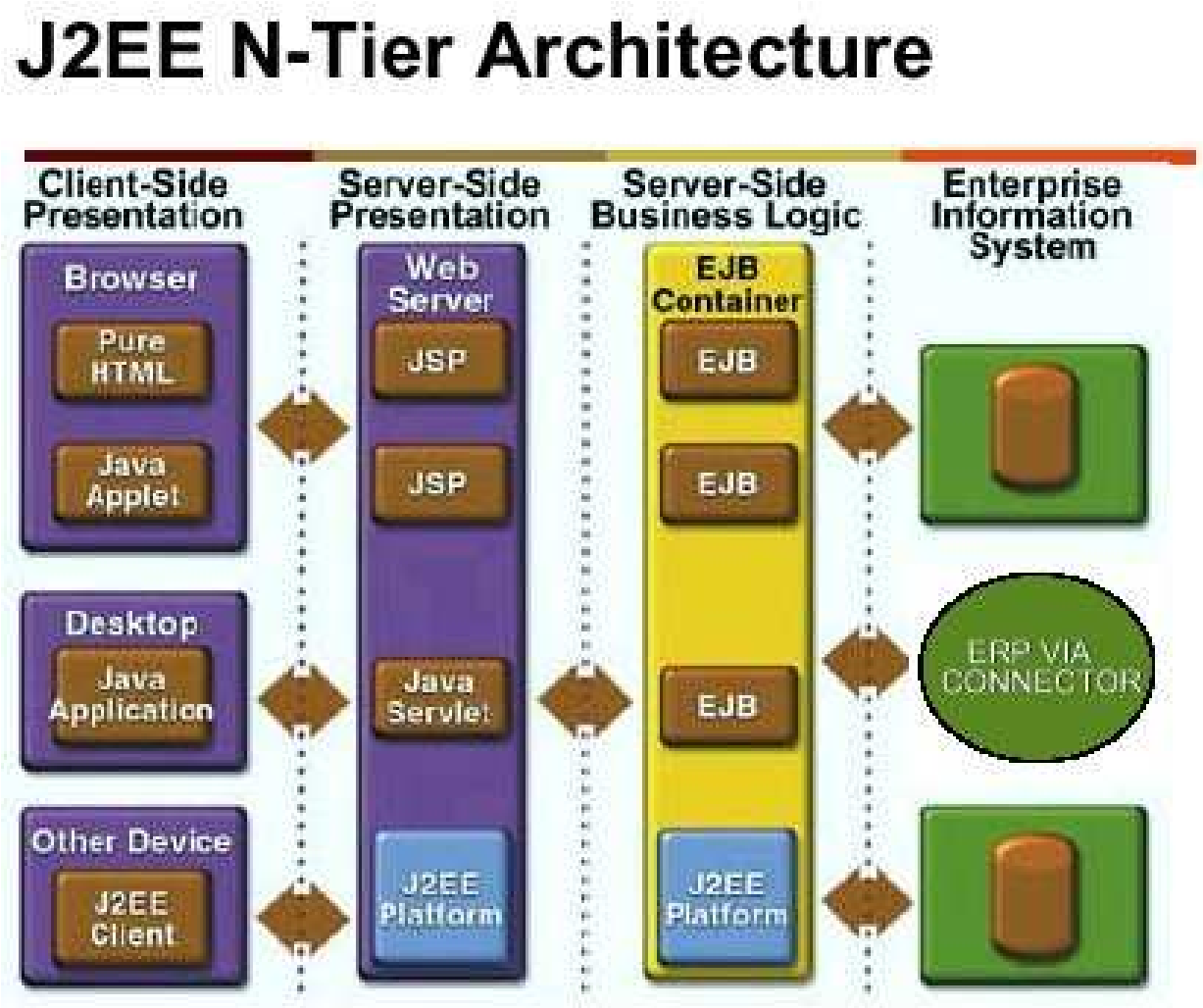


Fig.: Architecture J2EE n-tier

Sommaire de la partie

JSP et servlets

Programme Java s'exécutant côté serveur Web

- Servlet : programme "autonome" stocké dans un fichier `.class` sur le serveur
- JSP : programme **source** Java embarqué dans une page `.html`

	côté client	côté serveur
.class autonome	applet	servlet
embarqué dans .html	JavaScript	JSP

exécutable avec tous les serveurs Web (Apache, IIS,...) ayant un moteur de servlet/JSP (e.g. Tomcat)

Références

- [Mol05, Sar05, Cha04]
- `http://www2.lifl.fr/~seinturi/middleware/index.html`
- `http://fr.wikipedia.org/wiki/JavaServer_Pages`
- `http://www.serli.com/`

Servlet (définition)

Classe java héritant de `HttpServlet` (ou compilation d'une JSP)

```
All Implemented Interfaces:
    java.io.Serializable, Servlet, ServletConfig

public abstract class HttpServlet
   extends GenericServlet
   implements java.io.Serializable
```

La méthode `service` définit le code à exécuter.

```
protected  void  service(HttpServletRequest req,
                         HttpServletResponse resp)
 void  service(ServletRequest req, ServletResponse res)
```

`req` représente la requête renseignée par le moteur
`resp` représente la réponse HTML construite dans la servlet ("à coup de `out.println`")

Servlet (utilisation)

Déploiement et exécution de Servlet

- *bytecode* `.class` stocké sur le serveur
- désigné par une URL
- le chargement de l'URL provoque l'exécution de la servlet
  - étendent le comportment du serveur Web
  - sont exécutés par un moteur
- Chaque servlet n'est instanciée qu'une fois : ses données persistent entre deux invocations (ex : compteur évolue)
  - `init()` démarrage (chargement)
  - `destroy()` destruction (arrêt moteur, déchargement, arrêt volontaire)

Servlet (utilisation)

Contenu généré

- HTML, GIF, PDF, DOC...
- type MIME (text/html, image/gif...)

Fonctionnalités supplémentaires

- sessions
- cookies
- upload fichier
- chaînage des servlets
- concurrence single/multi-thread
- données globales partagées entre servlets

JSP

JSP = Java Server Pages

Une JSP est un fichier contenant du code HTML et des fragments de code Java exécutés sur le moteur de Servlets

- Comparable aux langages côtés serveur de type PHP, ASP, ...
- Les pages JSP sont converties en Servlet par le moteur de Servlets lors du premier appel à la JSP
- ensemble de technologies pour contruire des applications réparties
- implantation de référence : J2EE 5 SDK

JSP

- Le JavaServer Pages ou JSP est une technologie basée sur Java qui permet aux développeurs de générer dynamiquement du code HTML, XML ou tout autre type de page web. La technologie permet au code Java et à certaines actions prédéfinies d'être ajoutés dans un contenu statique.
- La syntaxe du JSP ajoute des balises XML, appelées actions JSP, qui peuvent être utilisées pour appeler des fonctions. De plus, la technologie permet la création de bibliothèques de balises JSP (taglib) qui agissent comme des extensions au HTML ou au XML. Les bibliothèques de balises offrent une méthode indépendante de la plate-forme pour étendre les fonctionnalités d'un serveur HTTP.
- Les JSP sont compilées par un compilateur JSP pour devenir des servlets Java. Un compilateur JSP peut générer un servlet Java en code source Java qui peut à son tour être compilé par le compilateur Java, ou peut générer le pseudo-code Java interprétable directement.

inspiré de `http://fr.wikipedia.org/wiki/JavaServer_Pages`

JSP (principes)

- Une page utilisant les Java Server Pages est exécutée au moment de la requête par un moteur de JSP, fonctionnant généralement avec un serveur Web ou un serveur applicatif. Le modèle des JSP étant dérivé de celui des servlets (en effet les JSP sont un moyen d'écrire des servlets).
- Lorsqu'un utilisateur appelle une page JSP, le serveur Web appelle le moteur de JSP qui crée un code source Java à partir du script JSP, compile la classe afin de fournir une servlet à partir du script JSP...

  Le moteur de JSP ne transforme et compile la classe que dans le cas où le script JSP a été mis à jour. La compilation (bytecode) et la recompilation sélective font de cette technologies une des plus rapides pour créer des pages dynamiques. En effet, la plupart des technologies de pages actives (ASP, PHP, ...) reposent sur un code interprété, ce qui requiert beaucoup de ressources pour fournir la réponse HTTP. Actuellement seuls les scripts `FastCGI` (code compilé écrit en langage C) sont plus rapides car ils ne nécessitent pas une machine virtuelle pour exécuter l'application.

inspiré de `http://www.commentcamarche.net/jsp/jspintro.php3`

JSP (fonctionnement suite)

De plus, les JSP possèdent toutes les caractéristiques faisant la force de Java :

- les JSP sont multithreadées,
- les JSP sont portables,
- les JSP sont orientées objet,
- les JSP sont sûres, ...

Mécanismes

- plusieurs zones <% ... %> peuvent cohabiter dans une même JSP
- lors du premier chargement (ou après modification) le moteur rassemble les fragments (zones) JSP dans une classe, la compile, l'instancie. La page JSP est un objet Java dans le moteur.
- à chaque chargement le moteur exécute l'objet dans un *thread*.
- en cas d'erreur Java le message est récupéré dans le navigateur.

JSP (fonctionnement Eclipse, Tomcat)

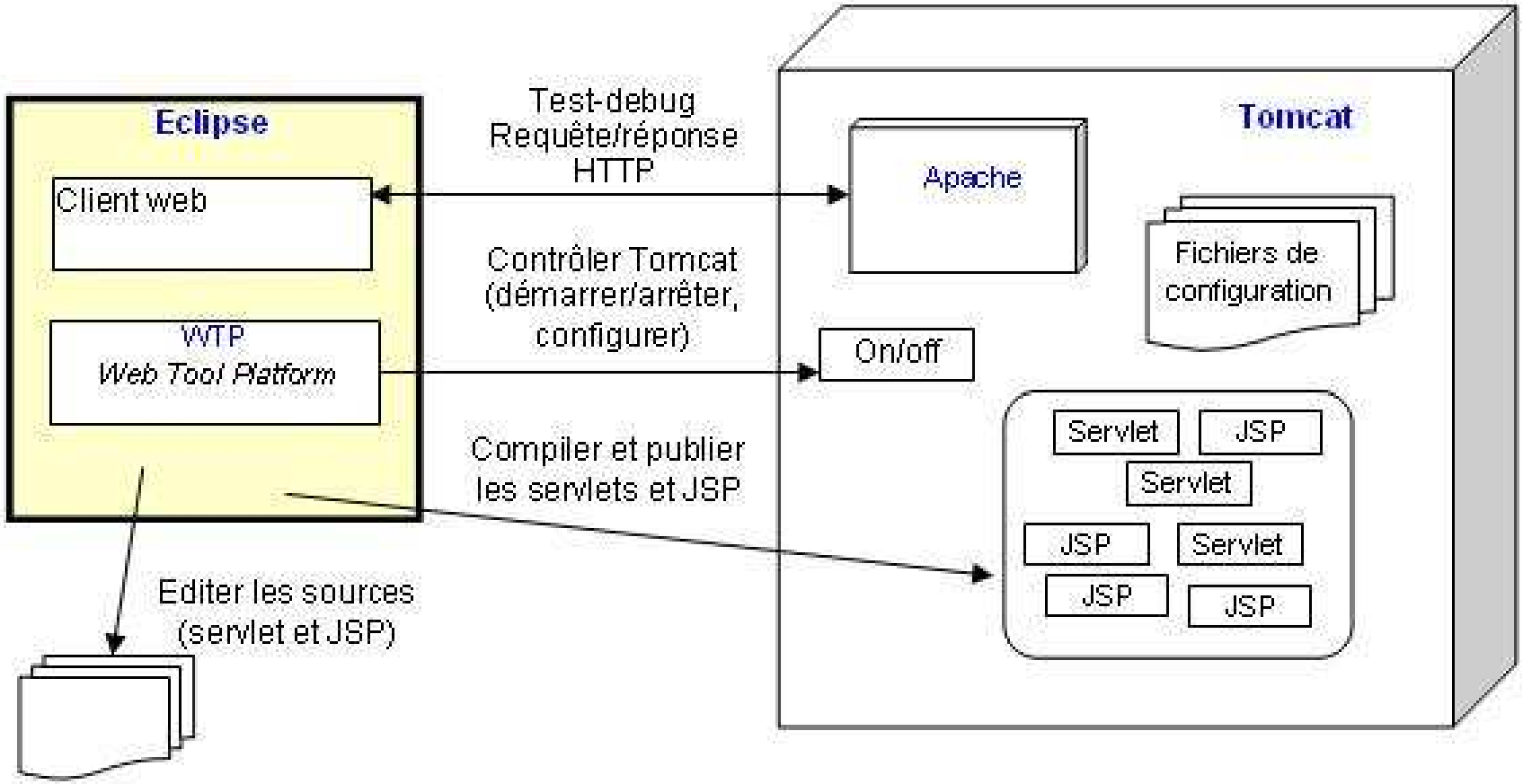


Fig.: Archi JSP/Tomcat

`http://www.while-true-do.com/wtdo-website-fr/img-contents/`

JSP (balises)

Une JSP contient :

- les données statiques comme le HTML
- directives <% directive attribut1=valeur1 ... %> qui agissent sur la compilation (page, include, taglib)
- les scripts, méthodes et variables (implicites, déclarées) :
  déclarations <%! int variableDeClasse = 0; %>,
  scriptlet <% %> méthode _jspService() du Servlet,
  expression <%= variable %>,
  commentaire <%- - Voici un commentaire JSP - -%>
- les actions : inclusion <jsp :include ... >, délégation <jsp :forward ... >, plugin (ancien), propriétés <jsp :getProperty ... >, beans <jsp :jsp :useBean ... >...
- les balises personnalisées et bibliothèques.

http://www.dil.univ-mrs.fr/~massat/ens/java/jsp1.html

Déploiement

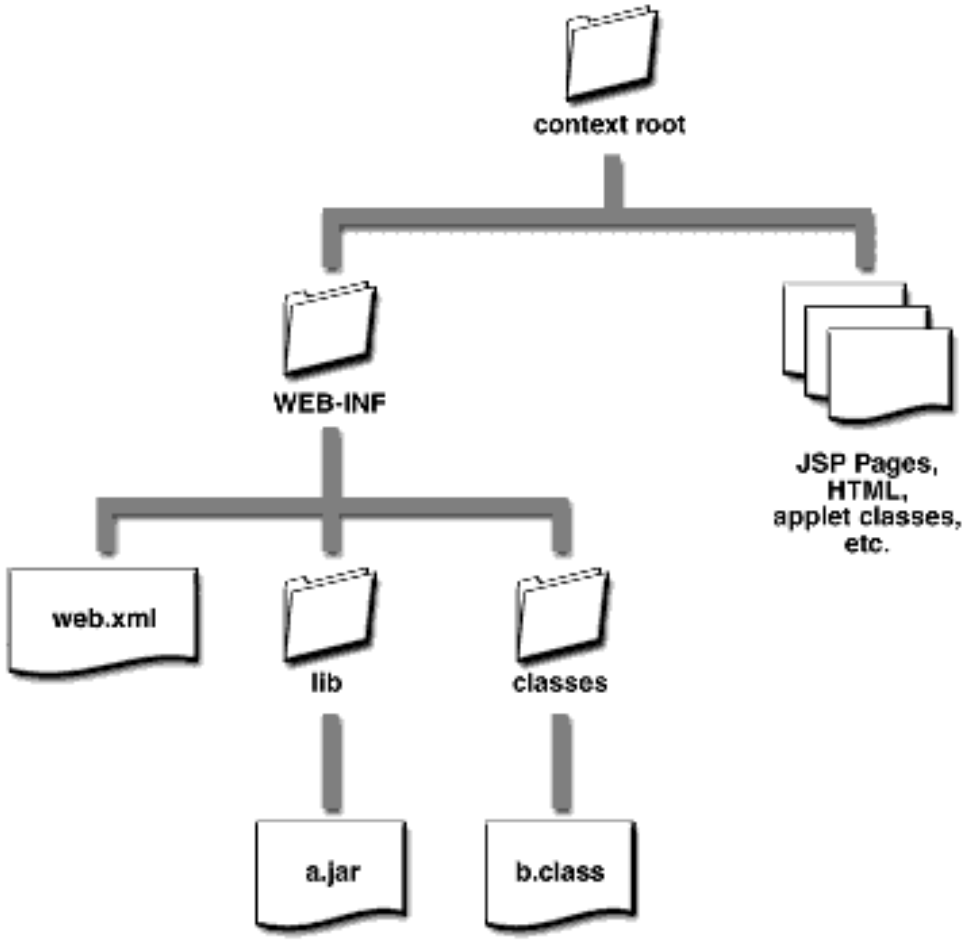


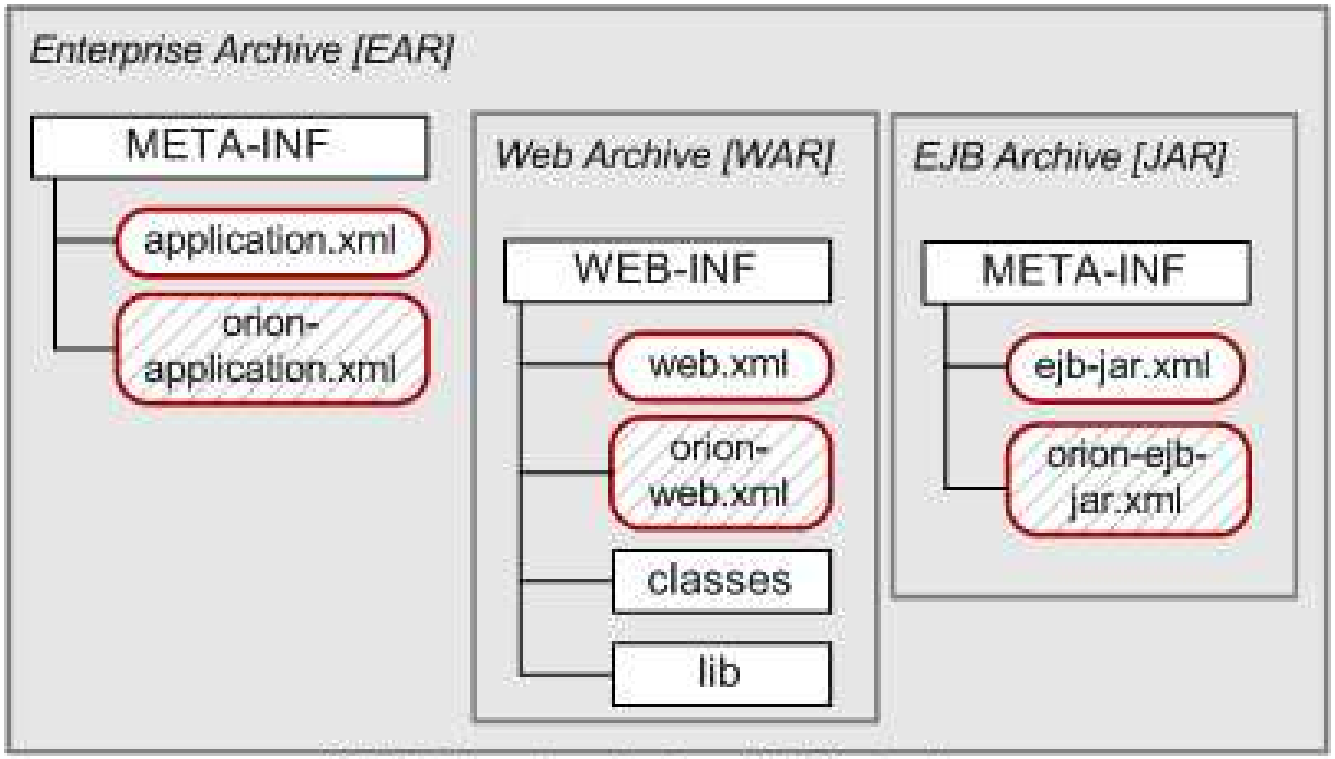
Fig.: Déploiement J2EE (varie selon le serveur)

`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/deployment4.html`

Déploiement

- WEB-INF : pages Web
  - descripteur `web.xml`
  - lien pages web et code
- META-INF : applications, EJB...
  - manifeste `manifest.mf`
  - descripteur `application.xml`
  - archives (EAR, JAR)...
- bibliothèques
- autres
-

Déploiement



J2EE Enterprise Archive [EAR] structure.
Grey boxes represent ZIP-style archives, white boxes directories and red, rounded boxes deployment descriptors.

Fig.: Déploiement J2EE (avec archives et beans)

`http://www.jguru.se/jguru/control/Developers/J2EE/Deployment`

En savoir plus

1. livre JSP [Cha04]
2. slides `jsp.pdf` source `http://www.serli.com/`
3. `http://fr.wikipedia.org/wiki/JavaServer_Pages`
4. intro
   `http://www.dil.univ-mrs.fr/~massat/ens/java/jsp1.html`

   et aussi
5. [Mol05, Sar05]
6. `http://www2.lifl.fr/~seinturi/middleware/index.html`
7. Baron `http://mbaron.ftp-developpez.com/javaee/jsp.pdf`
8. Doudou `http://www.jmdoudoux.fr/java/dej/index.htm`
9. Tahé `http://tahe.developpez.com/java/eclipse/`
10. `http://tecfa.unige.ch/guides/tie/html/java-jsp/java-jsp.ht`

Sommaire de la partie

Enterprise Java Beans

"Au départ un Java bean est un composant qui peut être manipulé visuellement dans un environnement de développement graphique. L'idée était de pouvoir incorporer ces composants à des applets par exemple sans avoir à recompiler leur codes."
"Les Enterprise Java Beans se différencient des JB dans le fait qu'ils ne sont pas conçus pour être travaillés visuellement mais pour fonctionner uniquement du côté serveur. Ils ont besoin d'un conteneur pour fonctionner [...] permettre au développeur de se concentrer sur les objets métiers et non sur les traitements annexes."

[Cha04]

Composants EJB

"Les EJB sont faits pour faciliter la vie du développeur qui n'a pas à connaître les API de bas niveau comme la gestion des connexions aux BD (Connection Pooling) ou comme la gestion du nombre d'instances parallèles (Multi-threading)... Architecture distribuée multi-serveurs et multi-applications." [Cha04]

Modèle de composants pour le développement d'applications d'entreprises

- logique applicative côté serveur
- services systèmes fournis par le conteneur (persistance, sécurité, transactions).
- la présentation est à la charge du client
- les *beans* sont portables d'un serveur à l'autre

Composants EJB (détails)

Concrètement un EJB est est un groupe de deux interfaces accompagné d'au moins une classe dans un module contenant un descripteur de déploiement. Quatre étapes sont nécessaires pour construire un EJB :

- création de l'interface Home qui contrôle le cycle de vie de l'EJB
- création de l'interface Component qui contient la logique applicative
- création du bean qui contient les méthodes
- création du deployment descriptor qui décrit l'EJB et les services dont il a besoin.

La gestion des transactions, la persistance des données et la sécurité sont directement prises en charge par les EJB. Les EJB communiquent entre eux en utilisant le JNDI (Java Naming Directory Interface).
Rappel : Les EJB sont invocables dans les JSP.

Composants EJB (interfaces d'accès)

Chaque EJB fournit deux interfaces d'accès distant (et éventuellement deux locales Local)

- Remote les services métier (méthodes) fournis par le *beans*
- RemoteHome interface de gestion du composant (création, recherche, destruction d'instances de *beans*)

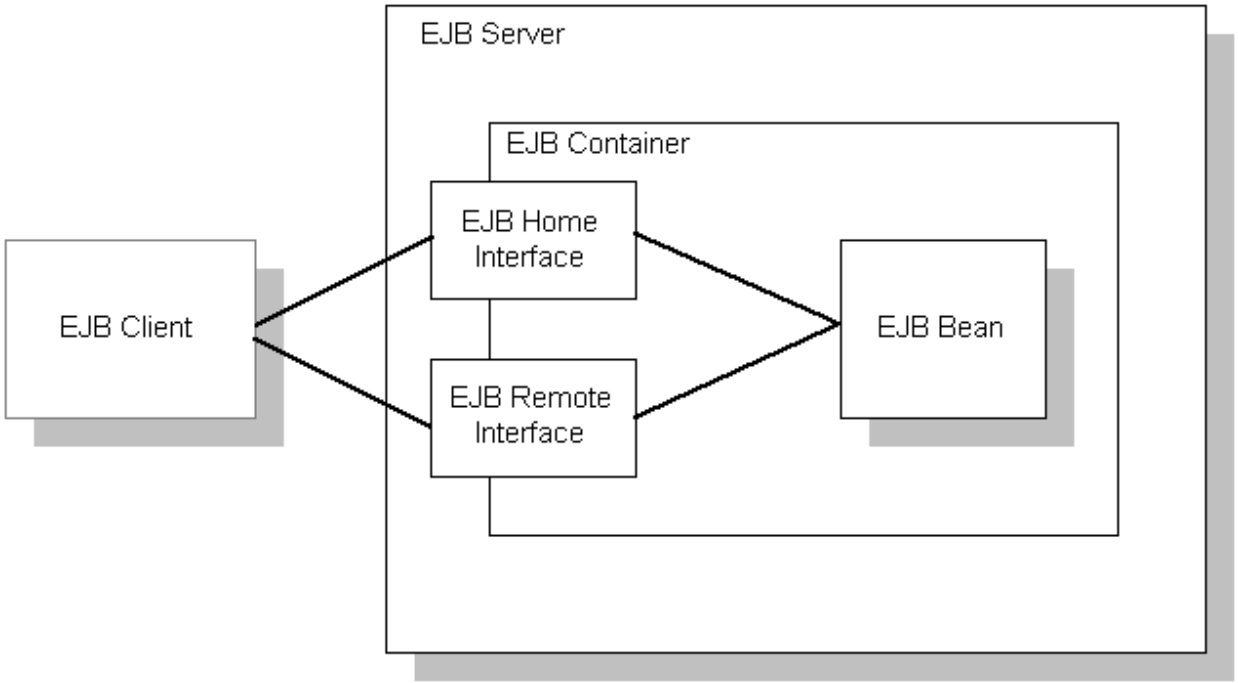


Fig : EJB

Composants EJB

Trois types de composants

- session tâche client
  - Stateless Session Bean -
    La particularité principale d'un Stateless Session Bean est de ne pas conserver d'état entre les différents appels.
  - Stateful Session Bean -
    La particularité principale d'un Statefull Session Bean est de conserver son état entre différents appels de méthodes.
- entity objet métier persistant
  - Container Managed Persistence CMP -
    persistance gérée par le conteneur
  - Bean Managed Persistence BMP -
    persistance gérée par le bean
- message-driven MDB -
  gestion de messages asynchrones, MOM (*Message Oriented Middleware*)

`source:L.Seinturier`

Composants EJB (persistance)

entity bean objet métier persistant ($\Leftrightarrow$ n-uplet d'une relation)

- n'est pas lié à la durée de vie des sessions clients
- est partagé par plusieurs clients
- ses données sont génères de manière persistante ; trois catégories de variables d'instance : persistante, relationnelle (clé), temporaire.
- est identifié par une clié primaire
- est relié à d'autres *entity bean* (cf relations 1-1, 1-n, n-1, n-n mono/bi-directionnel)
- CMP - persistance gérée par le conteneur, configuration via un descripteur de déploiement, le développeur n'écrit pas de classes concrètes, requêtes EJB QL
- BMP - persistance gérée par le bean : optimiser la gestion des données, utiliser d'autres supports que JDBC (fichiers, JDO, JDBC, Hibernate...)

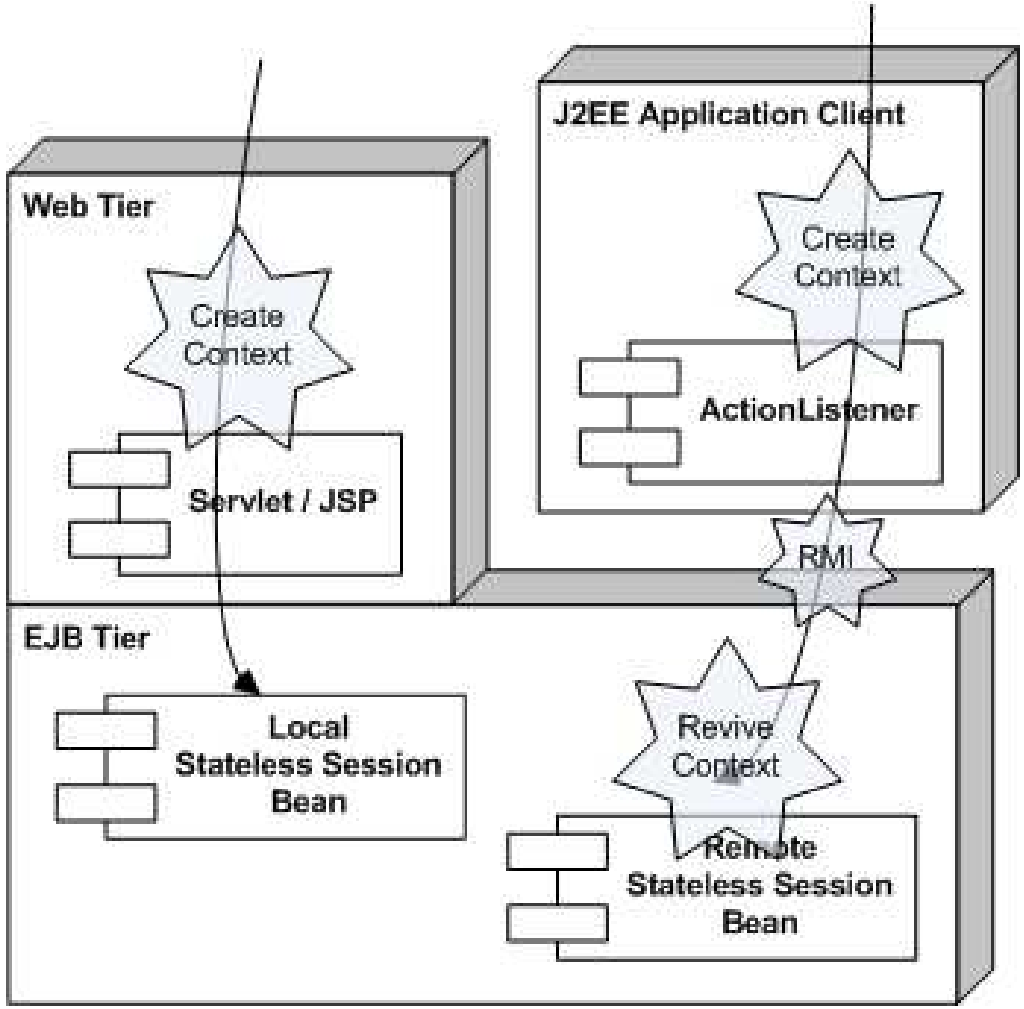
Composants EJB (persistance : comparaison)

Pas de différences entre les deux approches pour les interfaces et clés primaires

Caractéristique	CMP	CMP
classe	`abstract`	normale
accès SGBD	automatique	*à coder*
état persistant	pas de varibales (`abstract`)	variables normales
accesseurs	obligatoires	facultatifs
`findByPrimaryKey`	généré automatiquement	*à coder*
autres recherches	requêtes EJB QL	*à coder*
valeur de retour de `ejbCreate`	`null`	la valeur de la clé primaire

Composants EJB (communications)

- JSP balises spécifiques
- Objet (applet, autre bean...) protocoles RMI-IIOP + service JNDI



EJB 3.0

La spécification 3 des EJB a tout d'abord été élaborée en vue de simplifier la conception d'EJB du côté développeur.

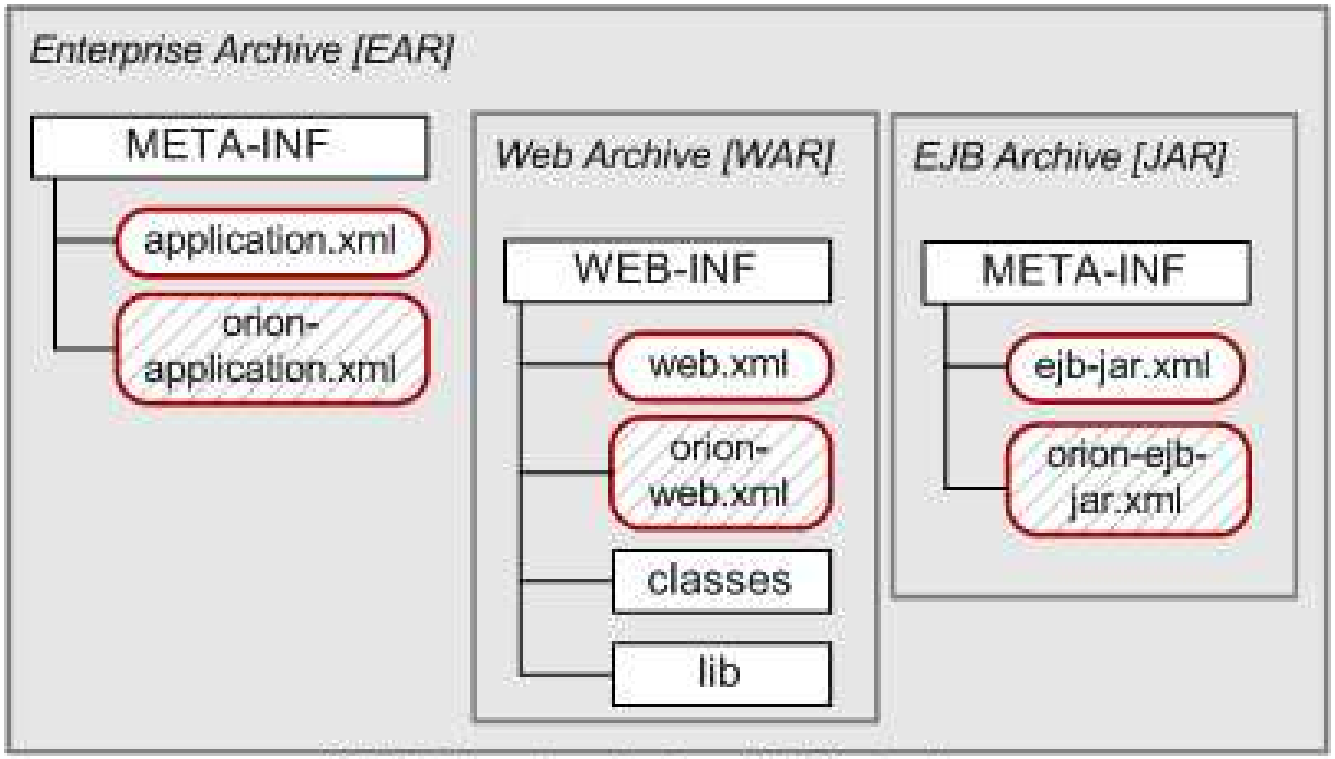
- Simplification de la définition des interfaces, suppression d'un bon nombre de points requis dans la version 2.1 (plus besoin d'hériter d'une super interface ou classe)
- Simplification pour la création de la classe du Bean.
- Simplification des API pour l'accès à l'environnement du Bean : définition par simple injection dépendante
- Introduction de l'utilisation des annotations en Java qui sont utilisées à la place du descripteur de déploiement
- Simplification concernant la persistance d'objet par la définition par l'utilisation facilitée de mapping objet/relationnel basée sur l'utilisation direct de classes Java et non de composants persistants.

`http://www.labo-sun.com/resource-fr-essentiels-836-3-java-j2ee-ejb-3-les-entreprise-java-bean-version-3-javabeans-.htm`

Déploiement

- Application EJB = archive EAR
  - descripteur `application.xml` de l'appplication
  - archive `.war` par *web bean*
  - archive `.jar` par *bean*
- Archive WAR
  - descripteur `.xml` du *web bean*
  - code (JSP, servlet.class)
- Archive JAR
  - descripteur `ejb-jar.xml` du *bean*
  - code `.class` du *bean*

Déploiement



J2EE Enterprise Archive [EAR] structure.
Grey boxes represent ZIP-style archives, white boxes directories and red, rounded boxes deployment descriptors.

Fig.: Déploiement J2EE (avec archives et beans)

`http://www.jguru.se/jguru/control/Developers/J2EE/Deployment`

En savoir plus

1. livre EJB 3.0 [dtS06]
2. `http://fr.wikipedia.org/wiki/Enterprise_JavaBeans`
3. `http://www.labo-sun.com/resource-fr-essentiels-836-0-java-j2ee-ejb-3-les-entreprise-java-bean-version-3-javabeans-.htm`
4. Design Pattern J2EE

et aussi

5. [Mol05, Cla03]
6. `http://www2.lifl.fr/~seinturi/middleware/index.html`
7. Doudou `http://www.jmdoudoux.fr/java/dej/chap047.htm`

Sommaire de la partie

MVC

Pattern Model/View/Controller de Smalltalk =
séparer les préoccupations du code

- Model le cœur de l'application (données et accès, calculs)
- View présentation des données
- Controller interaction de l'utilisateur

Construction du *framework* d'IHM décentralisée en Smalltalk (ça a changé progressivement pour passer progressivement en centralisé - fin du *polling* depuis la version 5i4 de Visualworks)

Le MVC a ensuite été popularisé par les *design patterns*.

MVC

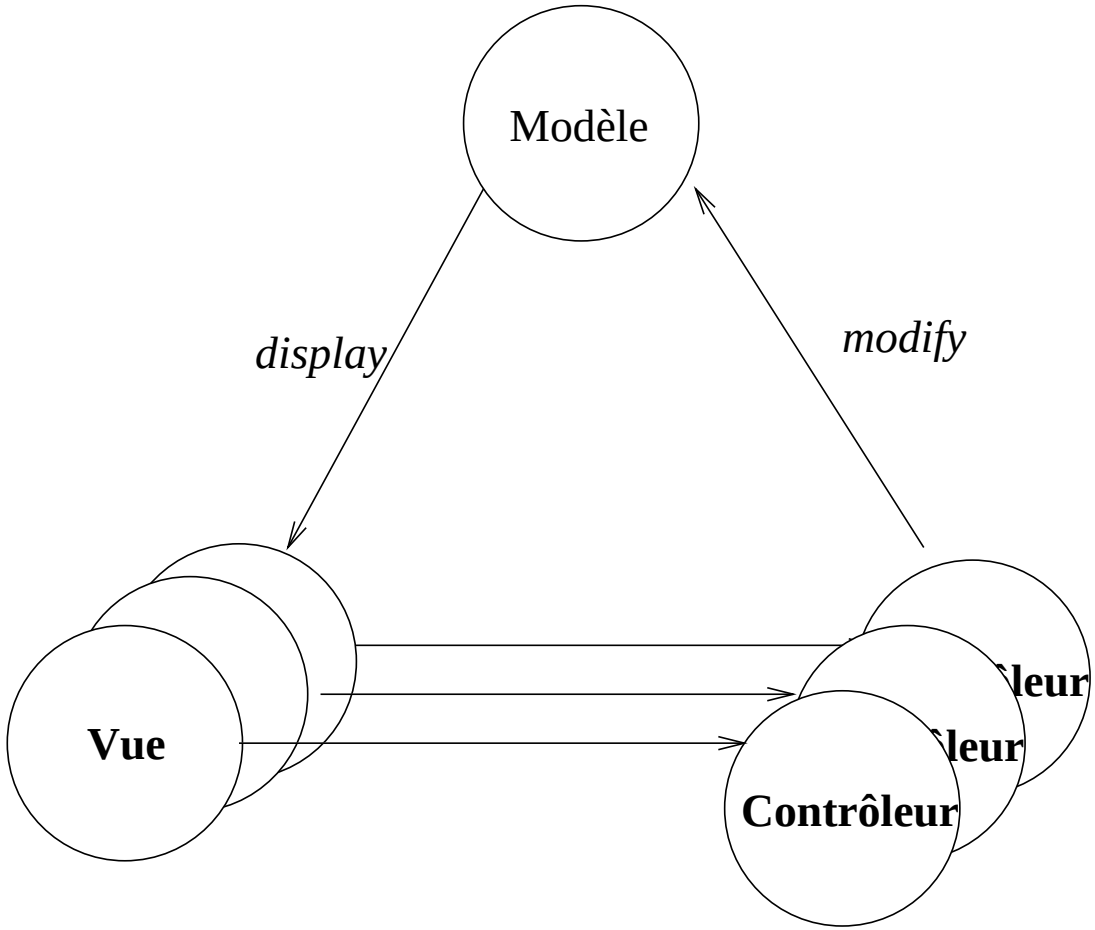


Fig.: MVC simplifié Smalltalk

MVC

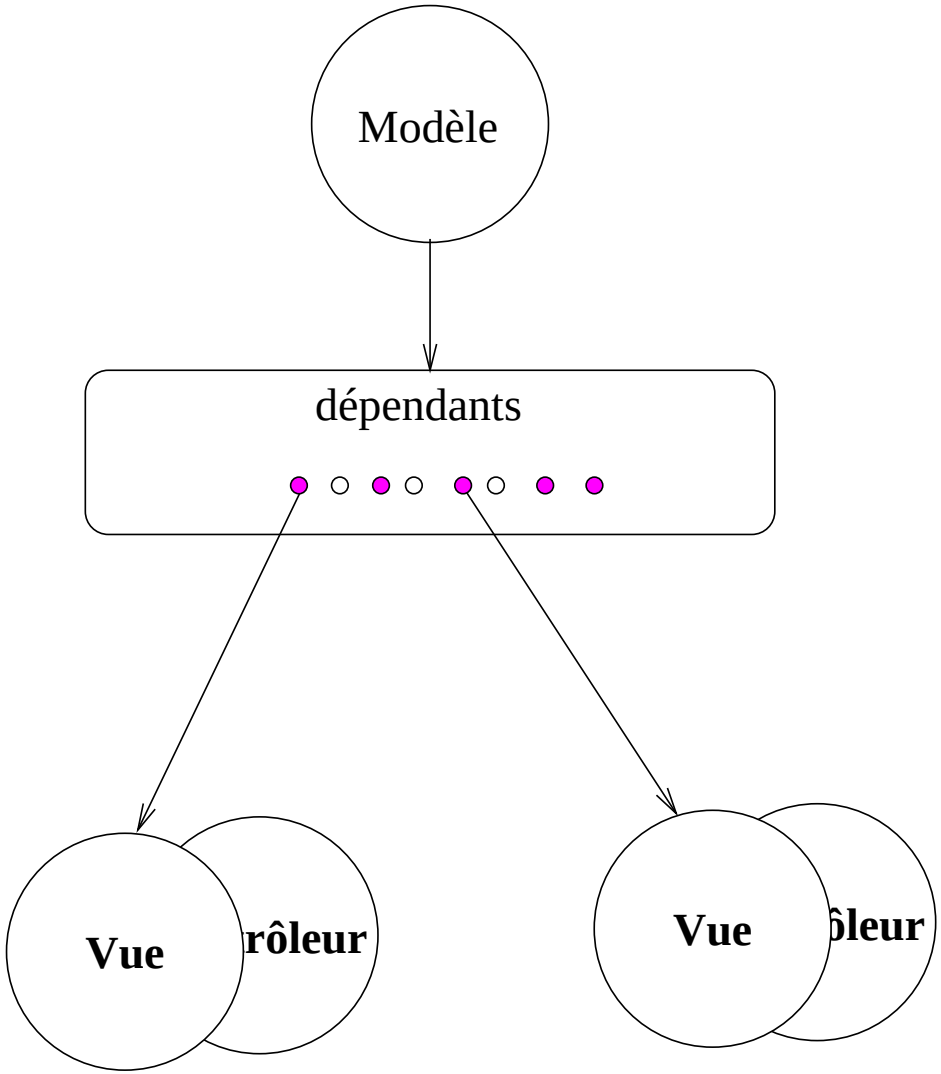


Fig.: MVC 4.0 Smalltalk

MVC (J2EE)

Pattern Model/View/Controller en J2EE

- Model The model is commonly represented by entity beans, although the model can be created by a servlet using a business object framework such as Spring.
- View The view in a Java EE application may be represented by a Java Server Page, which may be currently implemented using JavaServer Faces Technology (JSF). Alternatively, the code to generate the view may be part of a servlet.
- Controller The controller in a Java EE application may be represented by a servlet, which may be currently implemented using JavaServer Faces (JSF).

Dans le MVC2 (à l'instar de Smalltalk) on centralise le contrôle (par exemple une seule servlet).

`http://wpetrus.developpez.com/java/struts/`
`http://en.wikipedia.org/wiki/Model-view-controller`

MVC, (Struts)

Frameworks : Barracuda, Hammock, Tapestry, Webwork, Struts

Struts

Framework pour développer des applications web J2EE selon le MVC.
Struts fait partie du projet Jakarta (Fondation Apache) qui propose aussi Tomcat, Cactus, JMeter...

- Model à la charge du développeur.
- View JSP/servlet, Struts propose des taglibs pour l'intégration JSP.
- Controller Struts implémente un contrôleur principal (représenté par la classe `ActionServlet`) et des sous contrôleurs (correspondant aux classes `Action`).

Sources : [Cha04]
`http://www-igm.univ-mlv.fr/~dr/XPOSE2003/COPIN/`
`http://www.exe6.net/docs/MVC2_Struts.pdf`

`http://wpetrus.developpez.com/java/struts/`

MVC, (Struts)

Rôles

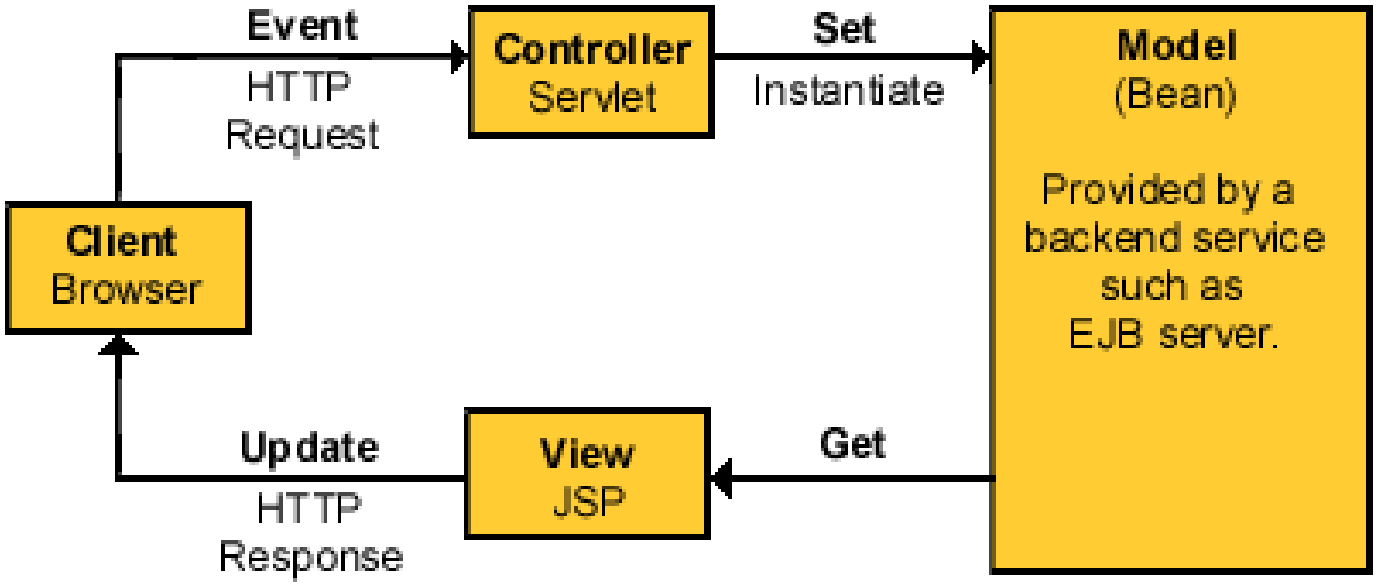


Fig.: Rôles dans Struts

`http://www.ibm.com/developerworks/library/j-struts/`

MVC, (Struts)

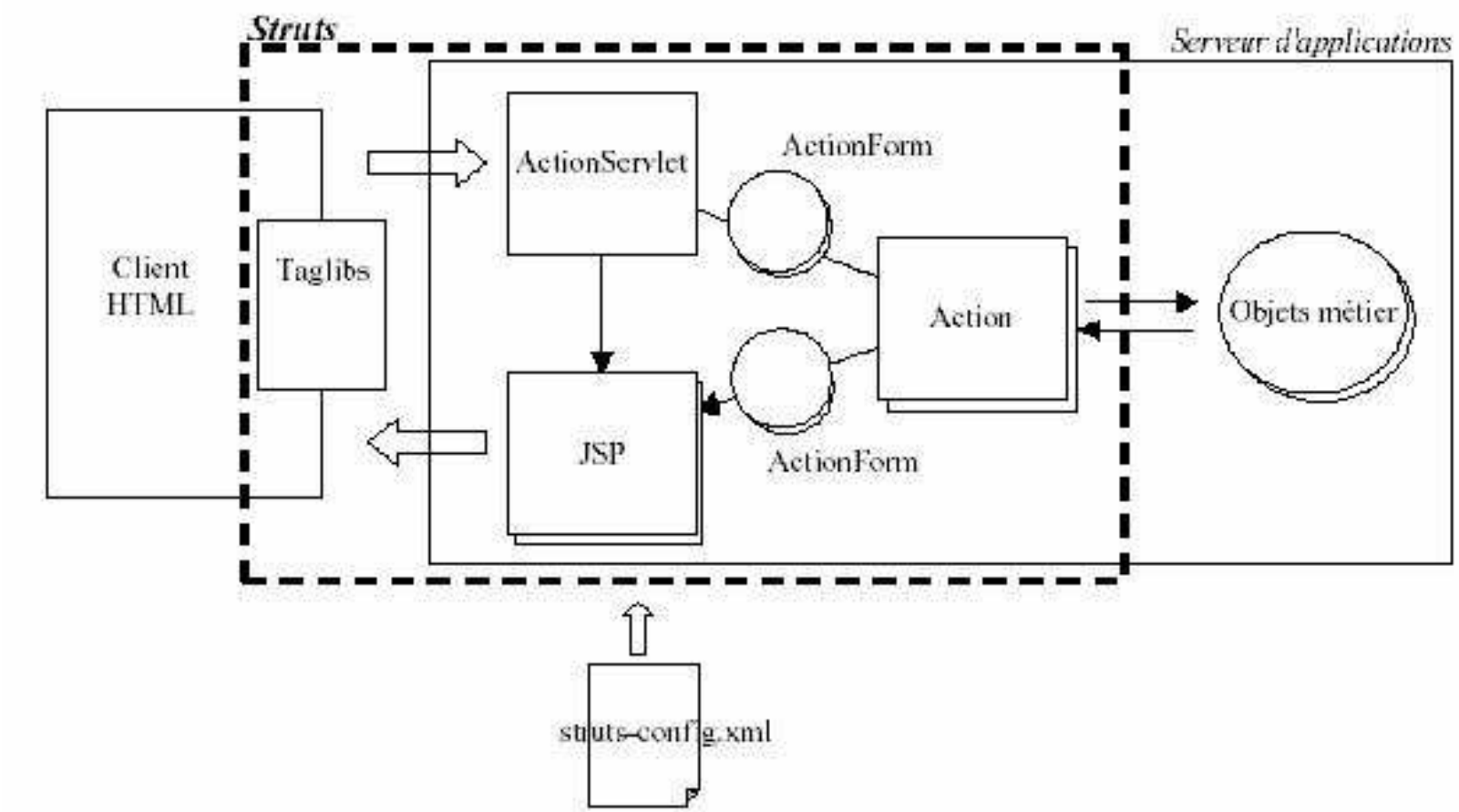


Fig.: Struts - détails

http://www-igm.univ-mlv.fr/~dr/XPOSE2003/COPIN/

MVC, (Struts)

Cycle de vie d'une opération :

- Sur le schéma ci-avant, le client envoie une requête à l'`ActionServlet`.
- Grâce au fichier de configuration `Struts-config.xml`, l'`ActionServlet` aiguille la requête vers l'`Action` appropriée.
- L'action réalise alors le traitement adéquat. Si besoin, cette action utilise les `ActionForm` nécessaires et effectue les opérations utiles sur le modèle.
- L'action renvoie ensuite le résultat du traitement (réussite, échec...).
- A partir de cette valeur, l'`ActionForm` est alors capable de déterminer le résultat à renvoyer au client (redirection vers une autre page JSP...).

`http://www-igm.univ-mlv.fr/~dr/XPOSE2003/COPIN/`

Persistance des données

- La persistance et les transactions sont des éléments fondamentaux dans les applications de systèmes d'information.
- Le passage à la programmation à objets à temporairement remis en cause l'hégémonie des bases de données relationnelles BD/R (apparition des BD objets dans les années 1990).
- Les performances, l'existant et l'adaptation des interfaces ont permis aux SGBD/R de s'imposer (définitivement ?).
- Le stockage XML reste marginal pour les gros volumes et les applications industrielles (performances, nature, lisibilité, interrogation...).
- Le nœud du problème réside dans les interfaces Web/Objet avec les BD/R : disposer d'outils **Object-Relational Mapping (ORM)** qui établissent le lien entre deux représentations *radicalement* différentes.

Cela fait 10 ans que le domaine bouge fortement. Voyons la situation en Java.

Persistance des données (Java) 1/2

Persistance des données avec Java 5 et EJB 3.0, JDO 2.0, JPA,...
(avec ou sans le Web ! !)

- JDBC L'API JDBC (Java DataBase Connectivity) permet aux applications Java d'accéder par le biais d'une interface commune à des sources de données pour lesquelles il existe des pilotes JDBC. Normalement, il s'agit d'une base de données relationnelle.
- JDO Java Data Object est la première spécification de Java permettant la persistance transparente. JDOQL est le langage d'interrogation spécifié par la norme JDO (Java Data Object). Le langage reprend la syntaxe de Java. Il prend en compte l'héritage et permet une navigation très simple. JDO 2.0 est quasi finalisé sous Apache Software Foundation.

Source : Wikipedia

Persistance des données (Java) 2/2

- SDO Service Data Objects est un standard initié par IBM, BEA, Xcalia puis supporté par Oracle, Siebel, SAP pour faciliter l'adoption des architectures orientées SOA. Les objectifs de SDO sont de simplifier l'accès aux données, unifier le modèle de programmation, et encourager l'adoption de patterns J2EE. Contrairement à JDO qui est lié à Java, SDO est indépendant du langage de programmation.
- EJB 3.0 La spécification Enterprise Java Beans 3.0 fait elle-même parti de la plate-forme J2EE 5.0. La persistance des données en EJB3 est possible à l'intérieur d'un conteneur EJB3 aussi bien que dans une application autonome J2SE en dehors d'un conteneur particulier.
- JPA L'API de persistance Java des données Java Persistance API fait partie de la spécification EJB 3. Cette API réalise la fusion des travaux sur Hibernate avec la continuité des spécifications EJB précédentes 2.0 et 2.1.

Source : Wikipedia

Persistance des données (Java)

JDO Java Data Object

La simplicité d'utilisation de JDO basée sur la manipulation de POJOs (*Plain Old Java Objects*) a fait le succès de ce standard. Le développeur peut manipuler les objets sans se soucier de la persistance.
Potentiellement, le standard permet l'accès à des sources de données de nature diverse. Trois grandes catégories d'implémentations existent :

- les implémentations qui ne supportent que les SGBD relationnels
- les implémentations qui ne supportent que les SGBD Objet
- les implémentations qui supportent des sources de données hétérogènes (SGBDR, SGBDO, XML, Mainframe) et également l'accès à des services (WebServices, JMS (Java messaging service), JCA (Java Connector Architecture), Cobol).

Dans les cas des implémentations supportant des sources hétérogènes, l'un des bénéfices apportés est la flexibilité du Système d'Information. Il devient en effet possible de changer de source de donnée par simple paramétrage.
Source : Doudou, Wikipedia

Persistance des données (Java)

JDBC et JDO ont les différences suivantes :

JDBC	JDO
orienté SQL	orienté objets
le code doit être ajouté explicitement	code est ajouté automatiquement
	gestion d'un cache
	mapping réalisé automatiquement ou à l'aide d'un fichier de configuration au format XML
utilisation avec un SGBD uniquement	utilisation de tout type de format de stockage

Source : Doudou

Persistance des données (Java)

DAO Data Access Object est un patron de conception.

- L'utilisation de DAO permet de s'abstraire de la façon dont les données sont stockées au niveau des objets métier. Ainsi, le changement du mode de stockage ne remet pas en cause le reste de l'application. En effet, seules ces classes dites "techniques" seront à modifier (et donc à re-tester). Cette souplesse implique cependant un coût additionnel, dû à une plus grande complexité de mise en œuvre.
- Le DAO définit donc une interface qui va exposer les fonctionnalités utilisables. Ces fonctionnalités doivent être indépendantes de l'implémentation sous jacente. Par exemple, aucune méthode ne doit avoir de requêtes SQL en paramètre. Pour les même raisons, le DAO doit proposer sa propre hiérarchie d'exceptions.

Source : Doudou, Wikipedia

Pattern DAO (Java)

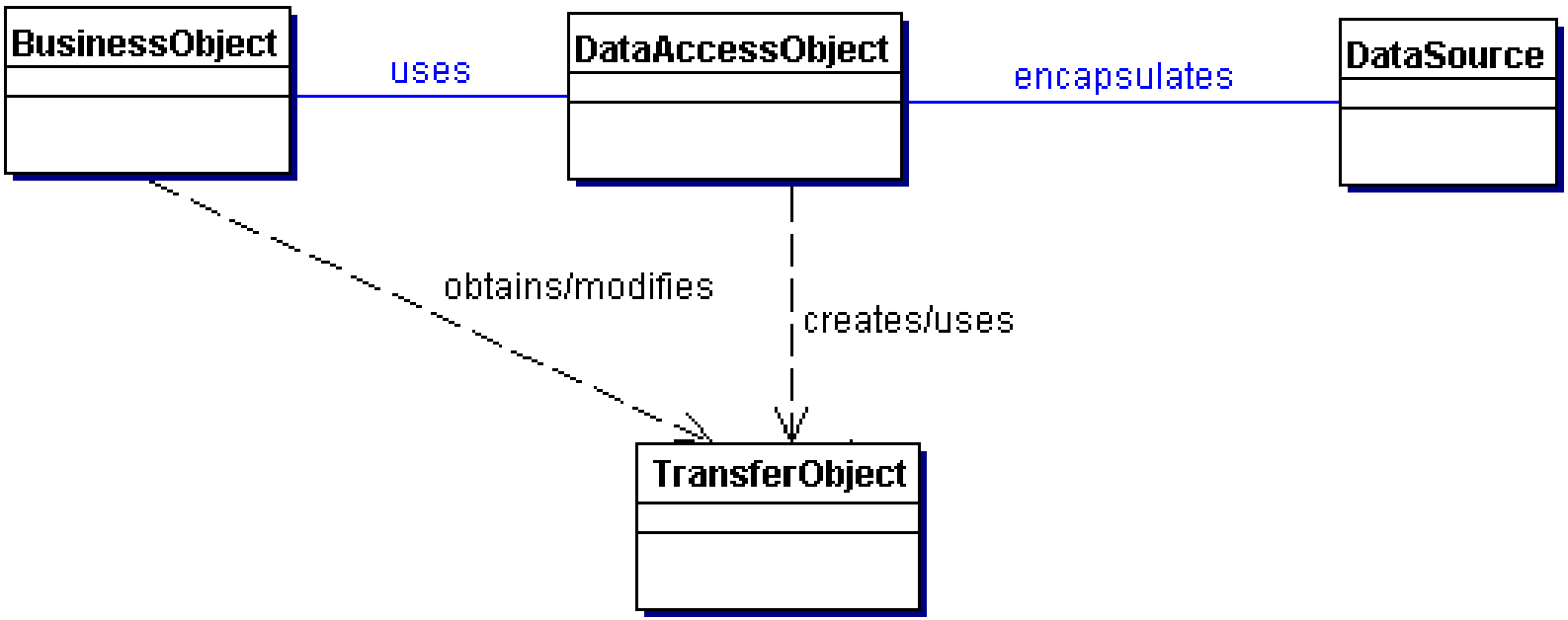
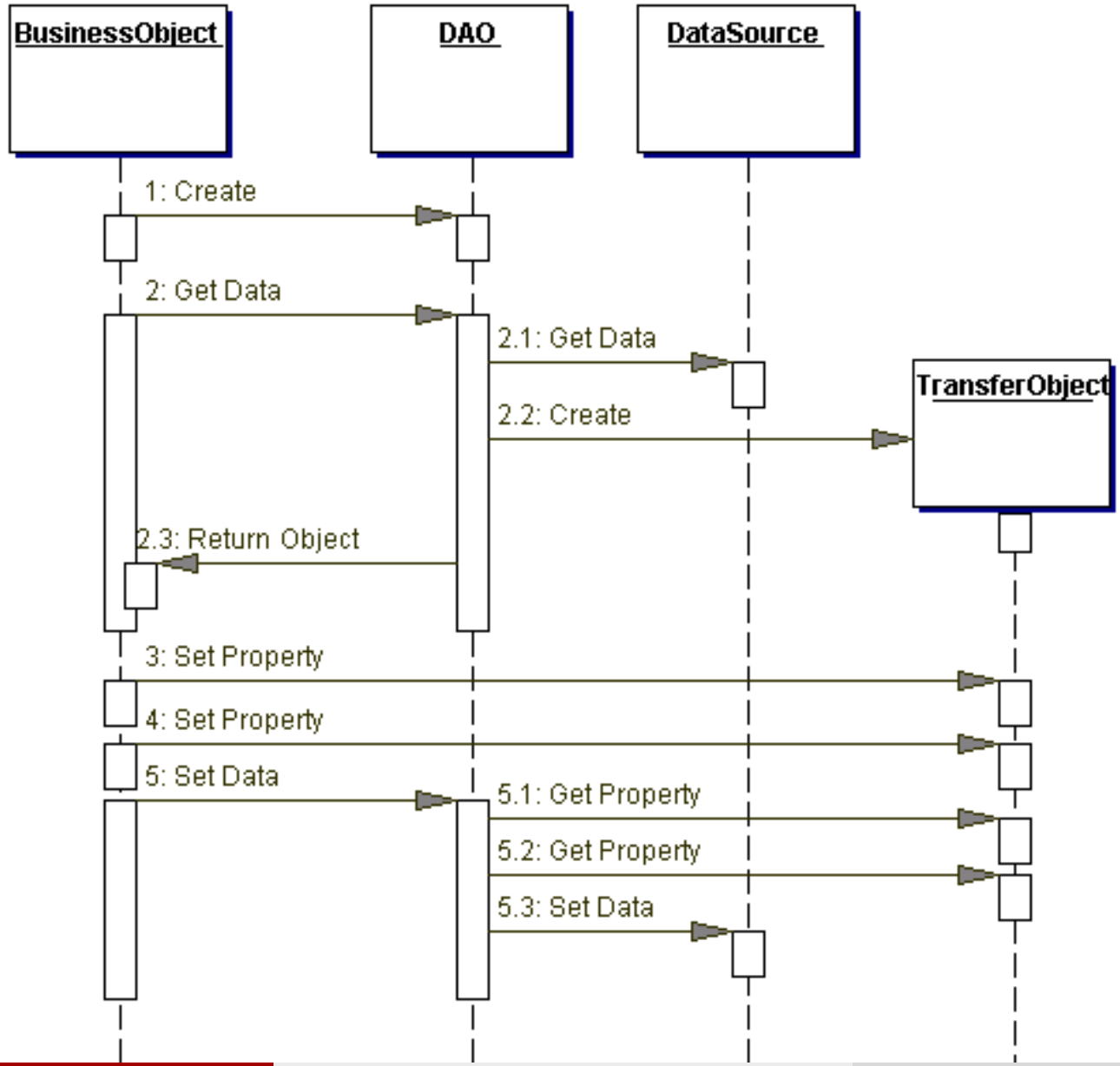


Fig.: Pattern DAO en Java

`http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html`

Pattern DAO (Java)



Persistance des données (Java)

JPA Java Persistance API

- JPA fait partie de la spécification EJB 3. C'est une synthèse standardisée des meilleurs outils du sujet (Hibernate, Toplink, ...). L'API repose sur
  - l'utilisation d'entités persistantes sous la forme de POJOs
  - un gestionnaire de persistance (*EntityManager*) qui assure la gestion des entités persistantes
  - l'utilisation d'annotations
  - la configuration via des fichiers xml
- JPA peut être utilisé avec Java EE dans un serveur d'application mais aussi avec Java SE (avec quelques fonctionnalités proposées par le conteneur en moins).

Source : Doudou

Persistance des données (Java)

JPA Java Persistance API

- JPA est une spécification : il est nécessaire d'utiliser une implémentation pour la mettre en oeuvre. L'implémentation de référence est la partie open source d'Oracle Toplink : Toplink essential. La version 3.2 d'Hibernate implémente aussi JPA.
- JPA ne peut être utilisé qu'avec des bases de données relationnelles.
- La version 3.0 des EJB utilise JPA pour la persistance des données.
- L'implémentation de référence est incluse dans le projet Glassfish. Elle peut être téléchargée unitairement à l'url :
  `https://glassfish.dev.java.net/downloads/persistence/JavaPersistence.html`
  Cette implémentation de référence repose sur l'outil TopLink d'Oracle dans sa version essential.

Source : Doudou

`http://java.sun.com/developer/technicalArticles/J2EE/jpa/`

JPA (tahe)

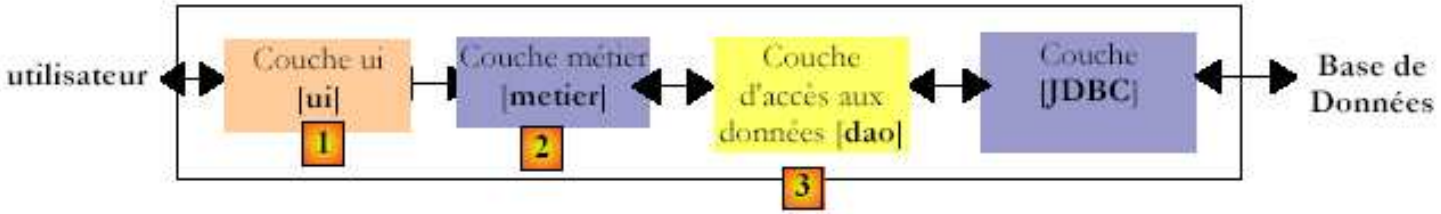


Fig.: Architecture multicouche classique

- la couche [1], appelée ici [ui] (User Interface) est la couche qui dialogue avec l'utilisateur, via une interface graphique Swing, une interface console ou une interface web. Elle a pour rôle de fournir des données provenant de l'utilisateur à la couche [2] ou bien de présenter à l'utilisateur des données fournies par la couche [2].
- la couche [2], appelée ici [metier] est la couche qui applique les règles dites métier, c.a.d. la logique spécifique de l'application, sans se préoccuper de savoir d'où viennent les données qu'on lui donne, ni où vont les résultats qu'elle produit.
- la couche [3], appelée ici [dao] (Data Access Object) est la couche qui fournit à la couche [2] des données pré-enregistrées (fichiers, bases de données, ...) et qui enregistre certains des résultats fournis par la couche [2].
- la couche [JDBC] est la couche standard utilisée en Java pour accéder à des bases de données. C'est ce qu'on appelle habituellement le pilote Jdbc du SGBD.

`http://tahe.developpez.com/java/jpa/`

JPA (tahe)

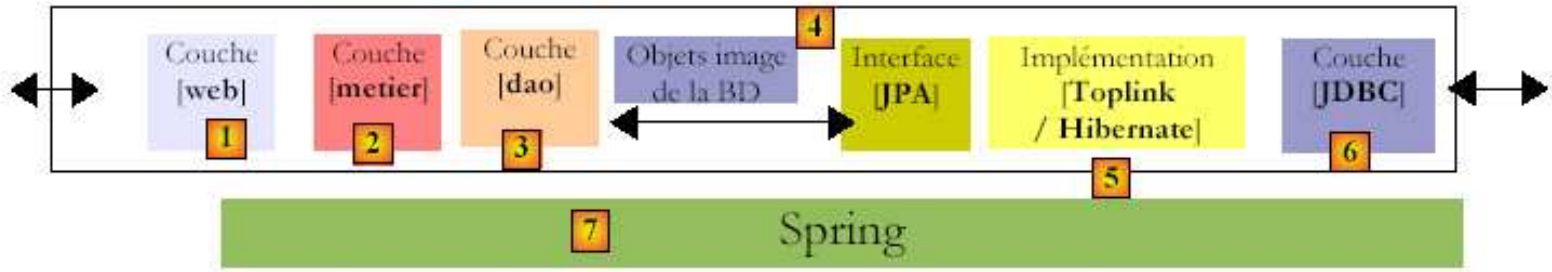


Fig.: Architecture multicouche avec JPA

- La couche [dao] dialogue maintenant avec la spécification JPA, un ensemble d'interfaces. Le développeur y a gagné en standardisation. Avant, s'il changeait sa couche ORM, il devait également changer sa couche [dao] qui avait été écrite pour dialoguer avec un ORM spécifique. Quelque soit le produit qui implémente JPA, l'interface de la couche JPA reste la même.
- La couche [4] des objets, image de la BD est appelée "contexte de persistance". Une couche [dao] s'appuyant sur Hibernate fait des actions de persistance (CRUD, create - read - update - delete) sur les objets du contexte de persistance, actions traduites par Hibernate en ordres SQL.
- La couche [Hibernate] est une couche d'abstraction qui se veut la plus transparente possible. L'idéal visé est que le développeur de la couche [dao] puisse ignorer totalement qu'il travaille avec une base de données.

`http://tahe.developpez.com/java/jpa/`

Hibernate

- Hibernate est un framework open source gérant la persistance des objets en base de données relationnelle.
- Hibernate est adaptable en terme d'architecture, il peut donc être utilisé aussi bien dans un développement client lourd, que dans un environnement web léger de type Apache Tomcat ou dans un environnement J2EE complet : WebSphere, JBoss Application Server et WebLogic de BEA Systems (voir (en) BEA Weblogic).
- Hibernate apporte une solution aux problème d'adaptation entre le paradigme objet et les SGBD en remplaçant les accès à la base de donnée pas des appels à des méthodes objet de haut niveau.
- Hibernate fournit au développeur, un langage HQL (Hibernate Query Language) pour interroger le contexte de persistance
- Spring conteneur léger *open source* supportant Hibernate.

`http://fr.wikipedia.org/wiki/NHibernate`
`http://www.jmdoudoux.fr/java/dej/chap042.htm`

`http://www.hibernate.org/`

En savoir plus

1. livre EJB 3.0 [dtS06]
2. `http://tahe.developpez.com/java/jpa/`
3. `http://fr.wikipedia.org/wiki/Java_Persistance_API` et les autres
4. `http://ippon.developpez.com/articles/java/persistance/solutions/`
5. `http://java.sun.com/developer/technicalArticles/J2EE/jpa/`
6. `http://www.jmdoudoux.fr/java/dej/chap039.htm`

et aussi

7. [Mol05, Cla03]
8. `http://www2.lifl.fr/~seinturi/middleware/index.html`
9. `http://fr.wikipedia.org/wiki/ORM`

Outils

textcolordarkredAide mémoire

- IDE : Eclipse [plugins ex : MyEclipse (J2EE, JSP, EJB, Struts...)], Netbeans, Lomboz (Eclipse spécifique J2EE)...
- Serveurs J2EE (pas tous EJB) : Tomcat, JBoss, Jonas, JRun, ...
- Développement : Scripts Ant, Test JUnit, Cactus, IHM JFace (surcouche SWT), Traces Log4J, balises JSP Taglibs
- Versions CVS, SVN
- Génération code, doc et suivi XDoclet, AndroMDA
- XML-Objets Castor
- Code et qualité Checkstyle, métriques JDepend, performances JMeter
- Moteur de templates Velocity (génération de code, SQL...)
- Spring conteneur léger open source supportant Hibernate.

☞ Liste en constante évolution...

Conclusion

Résumé

- Approche répandue pour les applications SI
- Technologies autour de Java, Logiciel libre
- Documentation riche (Logiciel libre)
- Complexe...

Mise en pratique

- Projet ARGOSI
- Développement Web Java UML
- Base existante


Christophe Calandreau, Alain Fauré, and Nader Soukouti.
EJB 2.0 - Mise en oeuvre.
Dunod, 2003.
ISBN 2-10-004729-9.


Eric Chaber.
JSP - Avec Struts, Eclipse et Tomcat.
Collection InfoPro, 01 Info. Dunod, 1 edition, 2004.
ISBN 2-10-008241-8.


Gilles et al. Clavel.
Java, la synthèse - Concepts, architectures, frameworks.
Dunod, 4 edition, 2003.
ISBN 2-10-007102-5.


Jim Conallen.
Concevoir des applications Web avec UML.
Eyrolles, 2000.
ISBN 2-212-09172-9, 1e édition.


Berthold Daum.

Eclipse - Développement d'applications Java.
Dunod, 1 edition, 2004.
ISBN 2-10-048733-7.


Club Developpez.
Cours sur Java.
Developpez.com, 2006.
travail collaboratif.


Karim Djaafar.
Eclipse et JBoss - Développement d'applications J2EE professionnelles, de la conception au déploiement.
Eyrolles, 1 edition, 2005.
ISBN 2-212-11406-0.


Jean-Michel Doudou.
Développons en Java avec Eclipse.
Developpez.com, 2007.
25/01/2007 - version 0.80 (635 pages).


Laboratoire Supinfo des technologies Sun.

EJB 3 - Des concepts à l'écriture du code - Guide du développeur.
Dunod, 1 edition, 2006.
ISBN 2-10-050623-4.


James Goodwill.
Mastering Jakarta Struts.
John Wiley & Sons, Inc., New York, NY, USA, 2002.


Steve Holzner.
Eclipse - Développement d'applications Java.
O'Reilly, 1 edition, 2004.
ISBN 2-84177-264-0.


Dick Lantim.
.NET.
Eyrolles, 2003.
ISBN 2-212-11200-9.


Jérôme Molière.
J2EE.
Eyrolles, 2 edition, 2005.

ISBN 2-212-11574-1.

Éric Sarrion.
J2EE.
Eyrolles, 1 edition, 2005.
ISBN 2-84177-380-9.

Pierre-Yves Saumon and Antoine Mirecourt.
Le guide du développeur Java 2 - Meilleures pratiques avec Ant, Junit et les design patterns.
Eyrolles, 1 edition, 2003.
ISBN 2-212-11275-0.

Serge et al. Tahé.
Développement web avec Java.
Developpez.com, 2006.
travail collaboratif.