DOSSIER DEVELOPPEUR

Projet CAR

Maître d'ouvrage (enseignant responsable) : *William BOHER-COY*

Titulaire (équipe de conception) : *Jonathan FAVIER*
Samuel ROLLET
Robin HAIDER

Date de rédaction : *26/01/2008*

Sommaire

HISTORIQUE DES REVISIONS

Référence	Révision	Date	Auteur(s)	Nature de la révision
CAR-DD-AVIS-V01.doc	00	15/11/2007	Equipe de développement	Version projet initiale.
CAR-DD-AVIS-V01.doc	01	22/11/2007	Equipe de développement	Modifications sur la partie Oracle Lite et Subclipse Plugin.
CAR-DD-AVIS-V01.doc	02	24/11/2007	Equipe de développement	Ajout de la partie VisualSVN Server et de la Prise en Main du Projet
CAR-DD-AVIS-V01.doc	03	23/01/2008	Equipe de développement	Ajout de la partie Création d'un Module

REFERENCES

- http://www.hibernate.org/
- http://www.jboss.org/
- http://www.labo-sun.com/
- http://www.oracle.com/
- http://www.springframework.org/
- http://fr.wikipedia.org/

- **Fichiers de configuration Eclipse**
  - http://tahe.developpez.com/java/eclipse/
- **JSP, Servlets**
  - http://tahe.developpez.com/java/web/
- **Spring**
  - http://zekey.developpez.com/articles/spring/
  - http://tahe.developpez.com/java/springioc/
- **Architecture 5 couches**
  - http://www.application-servers.com/articles/multicouchesjava/index.html

I. INTRODUCTION

1. Objet

Pour faciliter la maintenance ou l'éventuelle reprise du **Projet CAR** nous avons décidé de rédiger ce Dossier Développeur en accord avec la maîtrise d'ouvrage. Ce dossier comprend le détail de l'installation et de la configuration de tous les outils nécessaires au développement de ce projet (architecture J2EE et environnement de modélisation), ainsi qu'une explication de l'architecture J2EE. Ce dossier présente également les difficultés rencontrées durant l'installation et les réponses apportées en considérant la configuration matérielle et logicielle.

2. Contexte

Ce document a été rédigé dans le cadre d'un projet de fin d'études proposé aux étudiants de 3ème année du département informatique de l'ESIL. L'équipe en charge du projet (le titulaire), composée de **Jonathan FAVIER** (Chef de Projet), **Robin HAIDER** et **Samuel ROLLET** (Ingénieurs d'études et développement), a été supervisée par l'enseignant de l'ESIL M. **William BOHER-COY**.

L'objectif de ce projet est de fournir un système destiné à gérer toutes les agences de location de véhicules de la société **BYRON**. Ce système contiendra deux applications, une **application nationale (LN)** installée au siège de la société et une **application mobile (LM)** installée sur des PDA.

L'application nationale permet aux clients de réserver un véhicule et aux agences de gérer les locations et les réservations, le parc et la maintenance des véhicules, les contrats et les clients, et enfin l'administration du système. Cette application est utilisable sur l'intranet de la société ou à l'extérieur à l'aide d'un navigateur Web.

L'application mobile installée sur des PDA (en mode connecté dans l'agence par liaison WiFi) et destinée aux agents responsables des parcs de véhicules, permet à ces derniers de gérer les parcs des véhicules des agences ainsi que les locations.

3. Configuration initiale

- Microsoft Windows XP SP2
- 1Go de Mémoire RAM
- 1 Processeur Dual-Core
- 1 connexion Internet

4. Composants de l'installation

Eclipse IDE : environnement de développement intégré extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

J2EE : Java 2 Platform, Enterprise Edition est une spécification pour le langage de programmation Java plus particulièrement destinée aux applications d'entreprise. Dans ce but, toute implémentation de cette spécification contient un ensemble d'extensions au framework Java standard (J2SE, Java 2 Standard Edition) facilitant la création d'applications réparties.

JBoss : serveur d'applications J2EE libre entièrement écrit en Java et publié sous licence GPL. Parce que le logiciel est écrit en Java, JBoss Application Server peut être utilisé sur tout système d'exploitation fournissant une machine virtuelle Java (JVM).

JDBC : API Java permettant d'accéder à une base de données, il sert à faire l'interface entre un programme Java et une base de données.

JDK : le **Java Development Kit** est l'environnement dans lequel le code Java est compilé pour être transformé en bytecode afin que la JVM (machine virtuelle de Java) puisse l'interpréter.

Hibernate : framework open source gérant la persistance des objets en base de données relationnelle. Publié sous licence LGPL, il permet de « mapper » une base de données relationnelle en objets. Il permet donc d'abstraire totalement l'accès à la base de données et propose un accès totalement orienté objet aux données.

Spring : framework open source J2EE pour les applications 3-tiers, dont il facilite le développement et les tests. Publié sous licence Apache.

Définitions extraites de Wikipédia

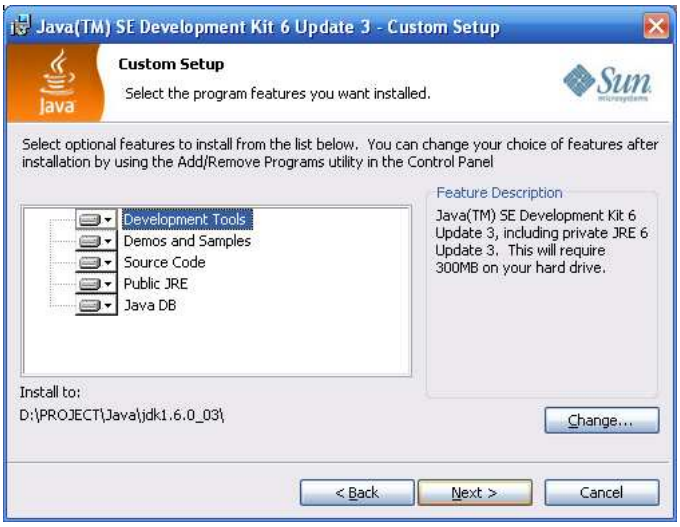
II. INSTALLATION DES OUTILS

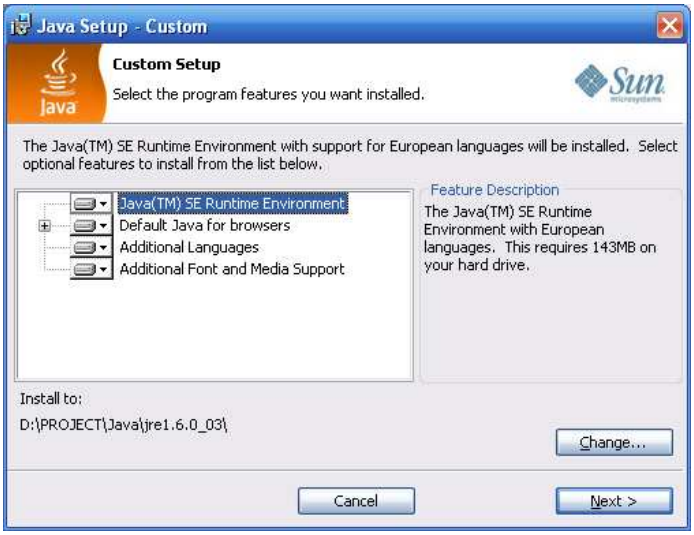
1. JDK

Installation du JDK 1.6 :

- **Programme :** Java(TM) SE Development Kit 6 Update 3
- **Fichier :** jdk-6u3-windows-i586-p.exe
- **URL :** http://java.sun.com/javase/downloads/index.jsp

Cliquer ensuite sur *Download* de la rubrique « JDK 6 Update 3 ». Cocher la case « Accept License Agreement » puis télécharger le fichier précisé précédemment, rubrique « Windows Platform ». Lancer ensuite l'installation, en installant la JDK et la JRE dans le dossier Java de votre dossier projet comme présenté dans les deux fenêtres suivantes :





Changement des variables d'environnement de Windows :

Les variables d'environnement renseignent le système sur l'existence et la localisation (chemin) de diverses applications. Pour les configurer :

→Aller dans *Démarrer / Panneau de configuration / Système / Avancée / Variables d'environnement*. Ajouter ensuite les valeurs aux variables suivantes (utilisateurs ou système, sans importance) :

→ **Variable JAVA_HOME :** valeur jdk1.6.0_03 dans le dossier de votre projet.
Exemple : D:\PROJECT\Java\jdk1.6.0_03

→ **Variable CLASSPATH :** valeur jdk1.6.0_03\lib dans le dossier de votre projet.
Exemple : D:\PROJECT\Java\jdk1.6.0_03

→ **Variable PATH :** valeur jdk1.6.0_03\bin dans le dossier de votre projet.
Exemple : D:\PROJECT\Java\jdk1.6.0_03\bin

2. JBoss

Installation de JBoss:

- **Programme :** JBoss 4.2.1
- **Fichier :** jboss-4.2.1.GA.zip
- **URL :**http://sourceforge.net/project/showfiles.php?group_id=22866&package_id=16942&release_id=523619

- **Instructions :** dézipper l'archive dans votre dossier dédié au projet. Dézipper ensuite l'archive JBossIDE dans le même dossier, puis lancer Eclipse (situé dans ce dossier).

→ JBoss utilise par défaut le port 8080. Ce port est aussi utilisé par Oracle pour son interface web. Il convient donc de changer le port par défaut dans le fichiers de configuration de JBoss.

→ Editer les fichiers :

[JBOSS_HOME] \server\all\deploy\jboss-web.deployer\server.xml
[JBOSS_HOME]\server\default\deploy\jboss-web.deployer\server.xml

→ Modifier dans chacun des fichiers l'entrée *Connector port="8080"* par *Connector port="8081"*

3. Eclipse - JBossIDE

L'installation de l'environnement de développement Eclipse se fait en même temps que l'installation du plugin JBossIDE pour Eclipse.

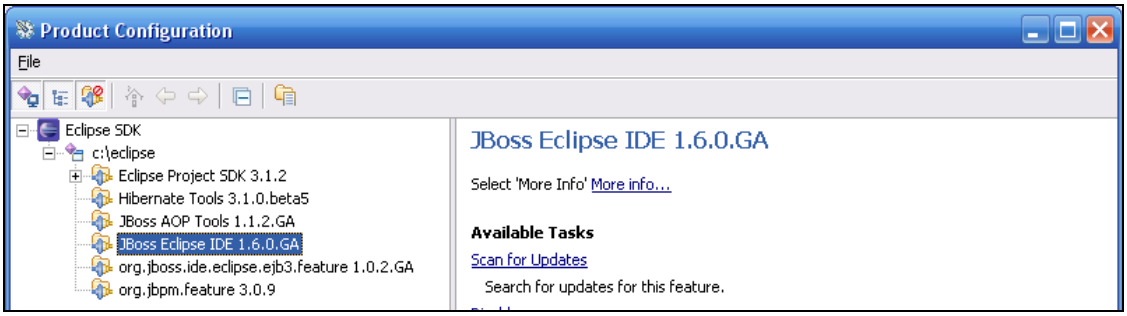
Téléchargement de JBossIDE 2.0.0 :

- **Programme :** Plugin JBossIDE 2.0.0.Beta2
- **Fichier :** JBossIDE-1.6.0.GA-Bundle-win32.zip
- **URL :** http://labs.jboss.com/jbosside/download/index.html

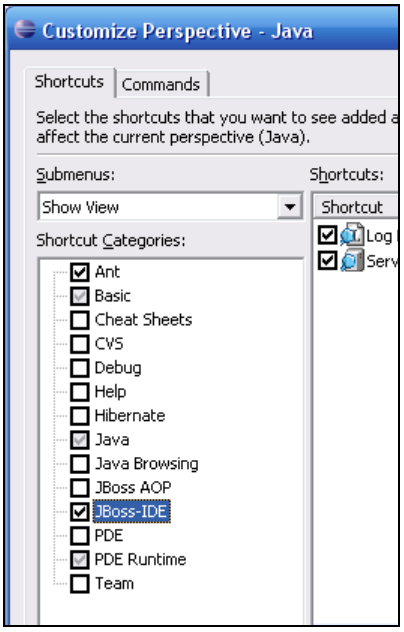
Cliquer ensuite sur le fichier précisé précédemment dans le package « JBossIDE Bundle ».

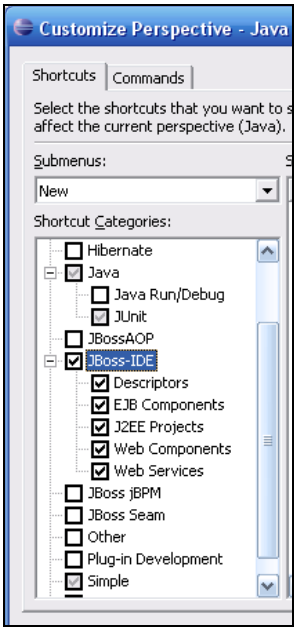
- **Instructions :** Dézipper l'archive dans votre dossier dédié au projet. Puis lancer Eclipse (situé dans ce dossier).

→ Aller dans *Help / Software Update / Manage Configuration*, puis cliquer sur le bouton *Show Disable Features*. Dérouler l'arborescence, puis vérifier que toutes les composantes sont bien activées.

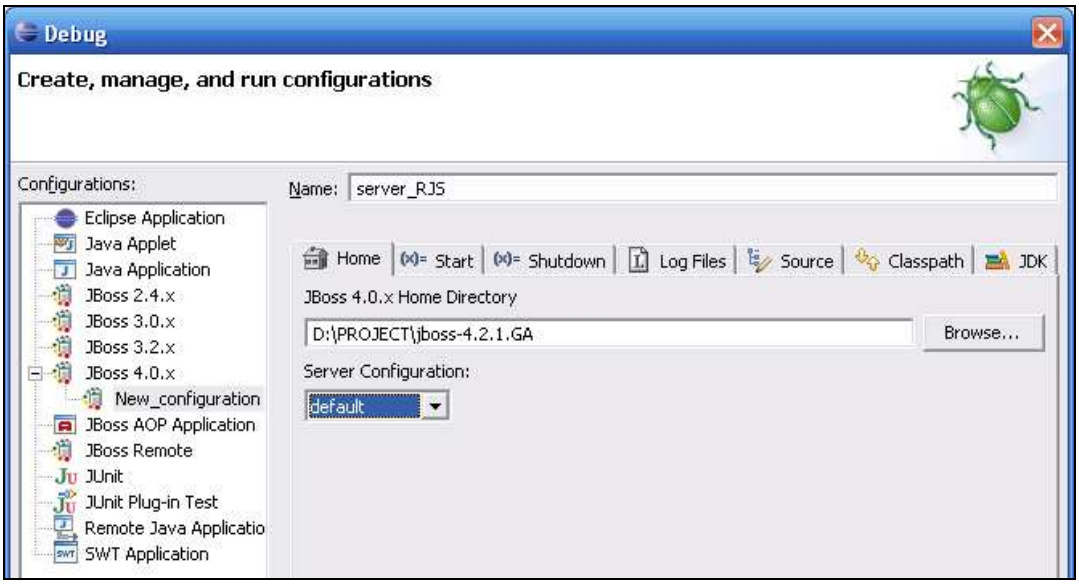


→ Aller dans *Window / Customize Perspective*. Dans le sous-menu *New*, cocher *JBoss-IDE*, puis dans le sous-menu *ShowView*, cocher également *JBoss-IDE*.

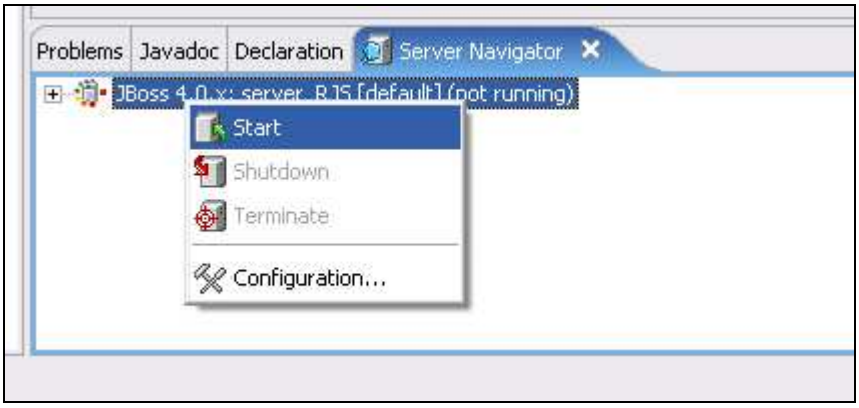




→ Fermer ensuite la fenêtre *Welcome*, puis cliquer sur le menu *Window / ShowView / Server Navigator*. Faire un clic-droit sur la vue *Server Navigator* apparue en bas de l'écran, option *Configuration*. Choisir la version JBoss 4.0.x et cliquer sur le bouton *New*. Nommer ensuite la configuration *server_RJS*, et préciser dans l'onglet *Home* le répertoire où vous avez installé JBoss. Dans l'onglet *Start* ajouter dans le champ *VM Arguments* la valeur *-Xmx512m*. Cela spécifie que la JVM qui exécute le serveur peur utiliser jusqu'à 512Mo de tas contre 64Mo par défaut.



→ Faire un clic-droit sur la vue *Server Navigator*, option *Start* : le serveur JBoss démarre.


Problems
Javadoc
Declaration
Server Navigator
JBoss 4.0.x: server_RJS [default] (not running)
Start
Shutdown
Terminate
Configuration...

4. Hibernate et JDBC

Travailler dans les deux univers que sont l'orienté objet et la base de données relationnelle peut être lourd et chronophage. **Hibernate** se propose de joindre ces deux univers, à travers le mapping objet/relationnel. Le terme **Mapping Objet/Relationnel** (ORM) décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle, basée sur un schéma SQL.

Hibernate s'occupe du **transfert des classes Java dans les tables de la base de données** (et des types de données Java dans les types de données SQL). Il permet également de **requêter** les données et propose des moyens de les récupérer. On peut voir Hibernate comme une fine surcouche de JDBC qui lui ajouterait une dimension objet.

Le plugin **Hibernate Tools** est destiné à gérer le mapping et son association avec la base de données depuis Eclipse. Il permet ainsi de se connecter à la base de données pour générer les fichiers de mapping et les classes java associées pour les tables présentes dans la base.

Installation de Hibernate :

Hibernate IDE est installé en même temps que JBoss IDE.

- **URL de référence :** http://www.hibernate.org/30.html

JDBC permet d'accéder à la base de données, il sert à faire l'interface entre un programme Java et une base de données. Le driver **JDBC** est propre à la base de données que l'on souhaite utiliser. Dans le cadre de ce projet, la version de JDBC à utiliser est contenue dans un fichier Jar appelé "ojdbc14_g.jar" qui est disponible au téléchargement sur le site d'Oracle ou après l'installation d'oracle dans *[ORACLE_HOME]\app\oracle\product\10.2.0\server\jdbc\lib*. Cette archive devra être incluse dans votre projet Eclipse.

5. Spring Framework

Spring est un framework qui permet de « remplacer » la lourdeur des serveurs d'application lourds. En effet, on parle de « conteneur léger ». Il prend en charge la création et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration décrivant l'ensemble de ces relations. L'un des avantages principal est qu'il n'impose pas l'héritage ou l'implémentation d'une quelconque interface ou classe, contrairement aux EJB.

Installation de Spring Framework :

- **Programme :** Spring IDE 1.3.6
- **Fichier :** pring-framework-2.0.7-with-dependencies.zip
- **URL :** http://www.springframework.org/download

- **Instructions :**

→ Décompresser l'archive. Les archives .jar se trouvent dans *[SPRING_HOME]\dist* . Ces archives devront être incluses dans votre projet Eclipse pour utiliser spring.

6. Spring IDE

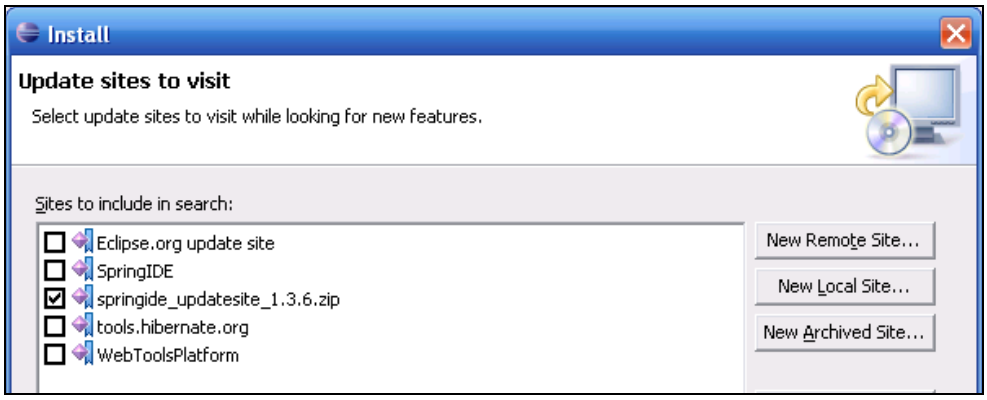
Spring IDE est une interface graphique simplifiant la configuration des fichiers utilisés par le Framework Spring. Il se présente sous la forme de plugins pour la plateforme Eclipse.

Installation de Spring IDE :

- **Programme :** Spring IDE 1.3.6
- **Fichier :** springide_updatesite_1.3.6
- **URL :** http://springide.org/updatesite_1.x/springide_updatesite_1.3.6.zip

- **Instructions :**

→ Dans eclipse, aller dans *Help/Software Update/Find and Install*. Cocher l'option *Search for new Features to install*, puis cliquer sur *Suivant*. Cliquer sur *New Archived Site…*, puis sélectionner le fichier archive précédemment téléchargé. Cocher ensuite l'option dans la liste, puis cliquer sur *Finish*.



→ Une fois la fenêtre ci-dessous ouverte, cocher *springide_updatesite_1.3.6.zip* puis cliquer toujours sur *Suivant* jusqu'à la fin de l'installation. Redémarrer ensuite Eclipse.

→ Une fois Eclipse redémarré, aller dans *Window / Customize perspective*, puis cocher l'option *Spring*.

- **Utilisation :**

→Pour utiliser Spring Framework dans un projet existant : clic-droit sur le projet et choisir *Add Spring Project Nature*.
→Pour afficher la vue de Spring Framework : Sélectionner dans la menu : *Window/Show View/Spring Bean*
→Dans la vue, clique-droit sur le projet et sélectionner *Properties*. Dans l'onglet *Config Files*, cliquer sur *Add…* pour spécifier un fichier de configuration xml de spring.

7. VisualSVN Server et Subclipse Plugin

VisualSVN Server est une adaptation de la partie serveur de Subversion. Cette adaptation a pour avantage d'offrir une interface graphique agréable et fonctionnant sous Windows. L'application est en outre accompagnée d'un serveur Apache hébergeant une interface Web de consultation des fichiers sauvegardés.

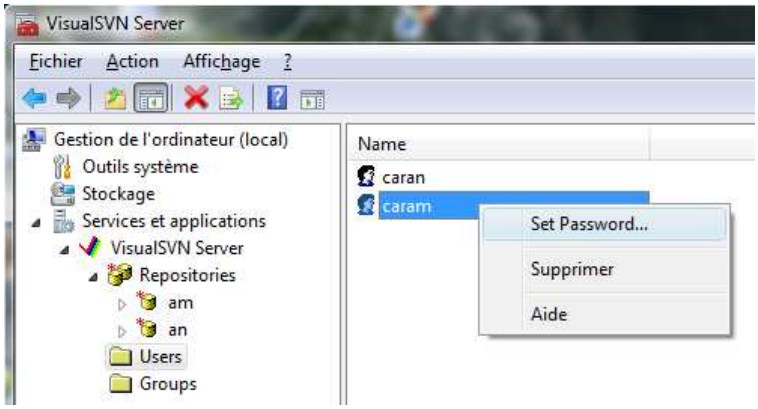
L'installation de VisualSVN Server ne pose pas de problème. Il est seulement demandé de choisir un port pour le serveur Apache. Nous avons choisi le port 8443 proposé par défaut. Une fois le logiciel installé, lancer l'interface de configuration « VisualSVN Server Manager ». Nous y avons alors créé, dans l'ordre :

- Le groupe `admins` auquel appartiendront les utilisateurs.

- Les utilisateurs `caran` et `caram` qui auront respectivement accès à l'application nationale et à l'application mobile.

- Les *repositories* `an` et `am` pour stocker les données des deux applications. Chaque *repository* correspond à un emplacement dans le disque dur.

Le manager permet d'administrer simplement le serveur SubVersion.



L'interface Web de VisualSVN Server permet d'accéder aux différentes versions des applications AM et AN. Elle est accessible à l'adresse `maison.xalti.com:8443/svn`.

Subclipse est un plugin pour le support de Subversion dans Eclipse IDE.

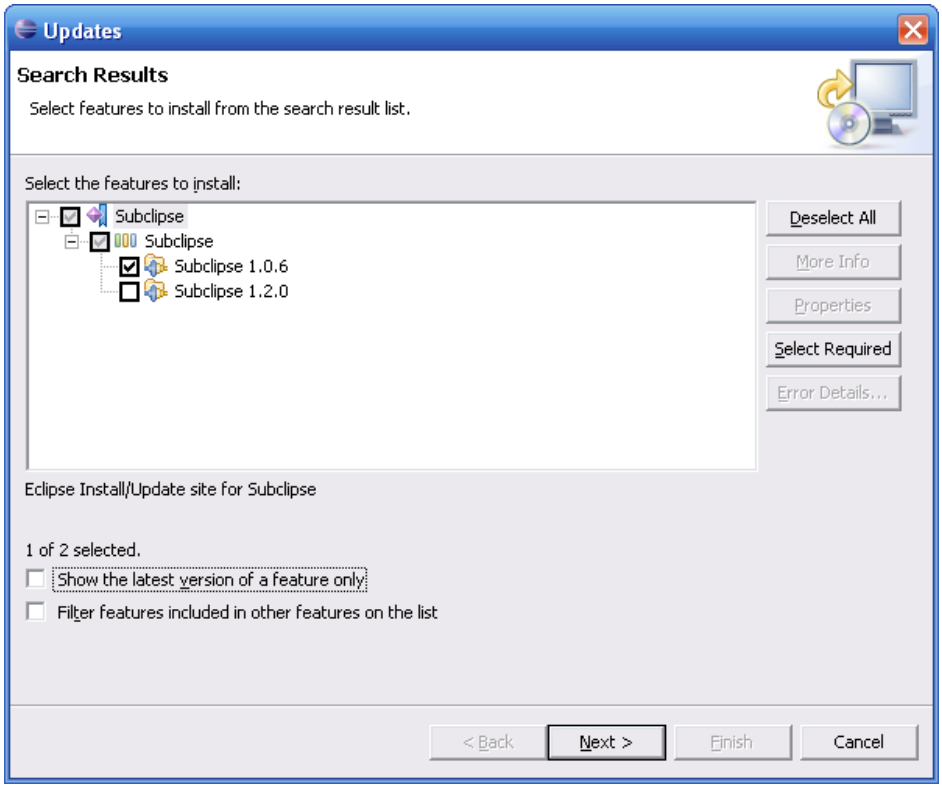
<u>Installation de Subclipse :</u>

- **Programme :** Subclipse 1.0.6
- **URL :** http://subclipse.tigris.org/update_1.0.x

- **Instructions :**

→ Dans Eclipse, aller dans *Help/Software Update/Find and Install*. Cocher l'option *Search for new Features to install*, puis cliquer sur *Suivant*. Cliquer sur *New Remote Site…*, puis entrez l'URL *http://subclipse.tigris.org/update_1.0.x* et nommez la Subclipse. Cocher ensuite l'option dans la liste, puis cliquer sur *Finish*.

→ Une fois la fenêtre ci-dessous ouverte, décocher l'option : *Show the latest version of a feature only*. Cocher Subclipse 1.0.6 puis cliquer toujours sur *Suivant* jusqu'à la fin de l'installation. Redémarrer ensuite Eclipse.



- **Configuration pour partager un projet Eclipse existant sur un serveur SVN :**

- Clic droit sur le projet dans l'arborescence : *Team/Share Project*
- Choisir *SVN*, faire *Next*
- Choisir *Create a new repository location*, faire *Next*
- Saisissez l'URL de votre projet SVN où vous voulez publier votre projet local.
- Cliquer sur *Finish* pour publier, ou *Next* pour saisir des options (nom du projet, commentaire sur la publication).

- **Configuration pour récupérer un projet existant sur un serveur SVN :**

- Dans le menu : *File/New*
- Choisir : *SVN/Check out projects from SVN*, fair *Next*
- Choisir *Create a new repository location*, fair *Next*
- Saisissez l'URL de votre projet SVN.
- Dans l'arborescence, sélectionner le dossier du projet que vous voulez récupérer en local et cliquer sur *Finish*

8. Oracle

Le serveur situé au siège de la société CAR héberge la base de données centrale du système. La version d'Oracle 10g qui a été installée est la XE (eXpress Edition) car elle répond aux besoins tout en étant plus simple et moins consommatrice de ressources que les versions plus complètes. Lors de la procédure d'installation, il convient de laisser les paramètres par défaut. Le nom et le mot de passe de l'utilisateur (qui est l'administrateur de la base de données) sont au choix de l'utilisateur.

Une fois la base de données créée il convient d'utiliser l'interface Web d'administration de Oracle pour créer un utilisateur (et son schéma associé). Pour cela, il faut se connecter avec l'utilisateur *SYSTEM* et le mot de passe choisi lors de l'installation. Pour la création de ce prototype, nous avons choisi l'utilisateur CAR avec comme mot de passe « car ». Il est conseillé de se servir de cet utilisateur pour accéder à la base.

Installation de Oracle XE 10g :

- **Programme :** Oracle Express Edition
- **Fichier :** OracleXE.exe
- **URL :** http://www.oracle.com/technology/products/database/xe/index.html

Pour télécharger le fichier d'installation, inscrivez-vous sur le site Oracle.

- **Instructions :**

→Double-cliquer sur le fichier d'installation puis suivre les différentes étapes de l'installation. Accepter la licence puis choisir le dossier de votre projet pour installer Oracle.
→Choisir le mot de passe de votre base de données pour les comptes *root*, *SYS* et *SYSTEM* (mot de passe « *car* » dans notre cas). Suivre ensuite normalement les étapes, puis cliquer sur *Finish* pour terminer l'installation.

Note : *Pour se connecter à la base de données, il faut utiliser le port 1521 et le SID « xe ». Les autres informations sont dépendantes de votre installation et de votre configuration (hostname, nom d'utilisateur…).*

9. Oracle Client

Oracle Client est une application permettant de visualiser la base de données Oracle.

Installation de Oracle Client XE :

- **Programme :** Oracle Express Edition Client
- **Fichier :** OracleXEClient.exe
- **URL :** http://www.oracle.com/technology/products/database/xe/index.html

Pour télécharger le fichier d'installation, inscrivez-vous sur oracle.

- **Instructions :**

→Double-cliquer sur le fichier d'installation, puis suivre chaque étape de l'installation. Choisir ensuite le dossier de votre projet pour installer Oracle.
→Choisir le mot de passe de votre base de données pour les comptes *root*, *SYS* et *SYSTEM* (mot de passe « *car* » dans notre cas). Suivre ensuite normalement les étapes puis cliquer sur *Finish* pour terminer l'installation. Cocher *« How to Start… »* pour afficher le guide de démarrage.

10. Oracle Lite

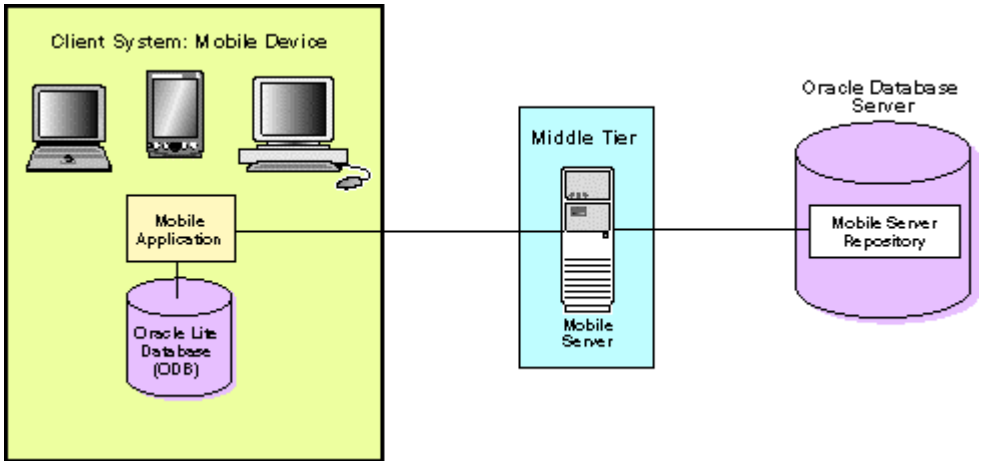
A. Oracle Lite et Mobile Server

Dans notre architecture J2EE, trois composantes interviennent dans le fonctionnement de la base de données :

- **Oracle XE** sur le serveur :
  - URL : `jdbc:oracle:thin:@maison.xalti.com:1521:xe`
  - User : car
  - Password : carmanager
  - SID : xe

- **Oracle Lite** sur le PDA :
  - Base : car
  - User : system
  - Password : carmanager

- **Mobile Server** sur le serveur ou sur une machine intermédiaire :
  - URL : `maison.xalti.com:8765`
  - Interface d'administration : `maison.xalti.com:8765/webtogo`
  - User : administrator
  - Password : carmanager



L'architecture mobile d'Oracle se compose de trois éléments principaux.

Dans l'assistant d'installation d'Oracle Lite, choisir d'abord d'installer le Mobile Server puis, dans un deuxième temps, le MBK (Mobile Development Kit). Ces deux composants peuvent se trouver sur deux machines différentes. Le MBK contient en particulier le *Mobile Database Workbench* que nous allons utiliser pour déployer notre base mobile.

Au lancement du Mobile Server plusieurs erreurs sont susceptibles d'apparaître ("javac non trouvé" et/ou "version de la JVM non compatible (requise : 48, disponible : 50)"). Il est alors nécessaire de préciser l'emplacement de javac dans :

```
C:\Oracle\OracleLite\mobile_oc4j\j2ee\mobileserver\config\server.xml
```

La ligne par défaut …

```
<java-compiler name="javac" in-process="false" options="-J-Xmx1024m -encoding UTF8" extdirs="C:\Oracle\OracleLite\jre\1.4.2\lib\ext" />
```

… est à remplacer par la ligne modifiée ainsi :

```
<java-compiler name="javac" in-process="false" options="-J-Xmx1024m -encoding UTF8" bindir="C:\Java\jdk1.6.0_03\bin" extdirs="C:\Java\jdk1.6.0_03\jre\lib\ext" />
```

Il peut alors utile de modifier la variable d'environnement PATH après l'installation afin de remplacer les pointeurs vers la JRE livrée avec Oracle Lite par des pointeurs équivalent pointant vers la JRE de son système.

Une fois l'installation du Mobile Server et du MBK sur le PC, charger sur le PDA l'installateur d'Oracle Lite proprement dit :

```
C:\Oracle\OracleLite\Mobile\Sdk\wince\ppc2003\cabfiles\olite.us.ppc2003.armv4_sdk.CAB
```

Trois logiciels constituent la partie PDA d'Oracle Lite :

- **Oracle Lite Device Manager** : configuration de la connexion au serveur.
  - ServerURL : http://maison.xalti.com:8765/webtogo/devmgr
  - Server : maison.xalti.com:8765
  - User : administrator
  - Passwd : carmanager

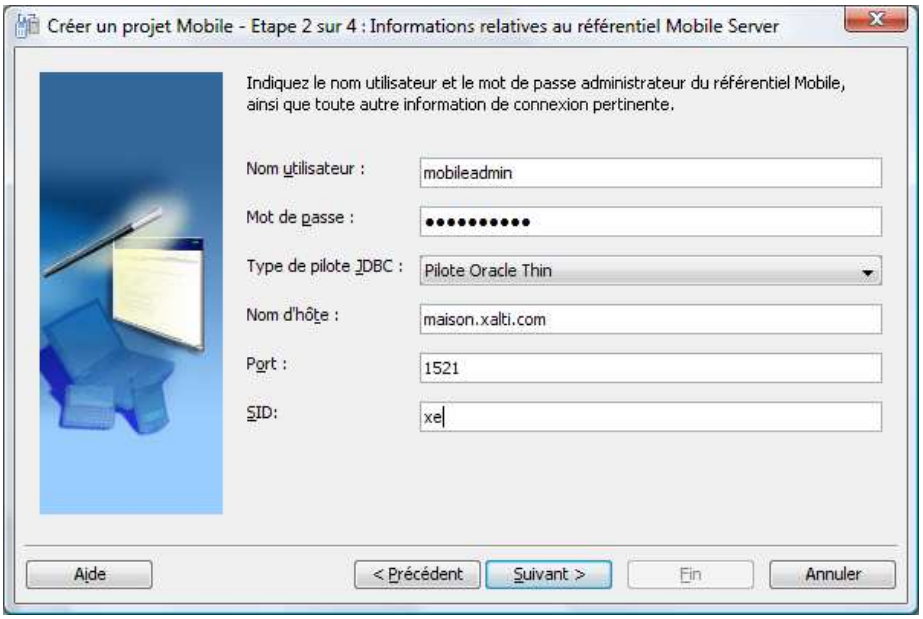
- **Oracle mSync** : synchronisation de la base embarquée et du Mobile Server.
  - User : car
  - Passwd : carmanager
  - Server : maison.xalti.com :8765

- **Oracle mSQL** : accés à la base embarquée.

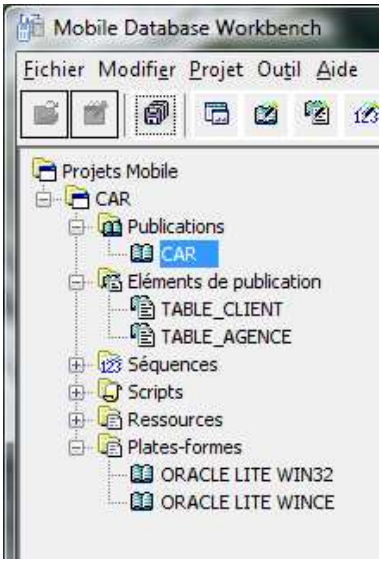
La synchronisation s'effectue manuellement à partir du PDA à l'aide de l'outil mSync. Celle-ci ne prend que quelques secondes.

B. Mise en place du processus de synchronisation

- **Création d'une publication avec Mobile Database Workbench** : On choisit de créer un nouveau projet puis autant d'*éléments de publication* qu'il y a de tables à synchroniser. Pour chaque *élément de publication* il est possible de choisir quelles colonnes seront disponibles sur le PDA et synchronisées. Il faut ensuite créer une *publication* et l'associer avec les différents *éléments de publication*. Cette publication donnera son nom à la base embarquée sur le PDA.



Saisie des données de connexion à la base Oracle lors de la création du projet dans le Mobile Database Workbench.



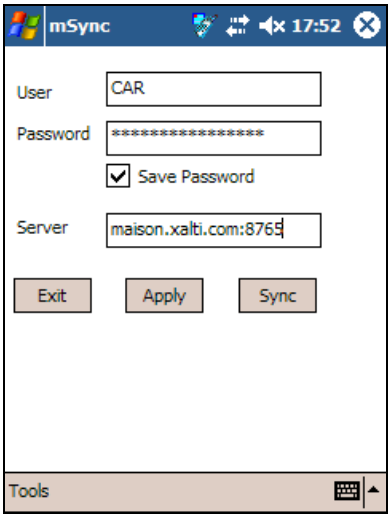
Une fois les *éléments de publication* et la *publication* créés, enregistrer le projet et sélectionner la commande « déployer » dans le menu « Fichier » afin de valider l'opération.

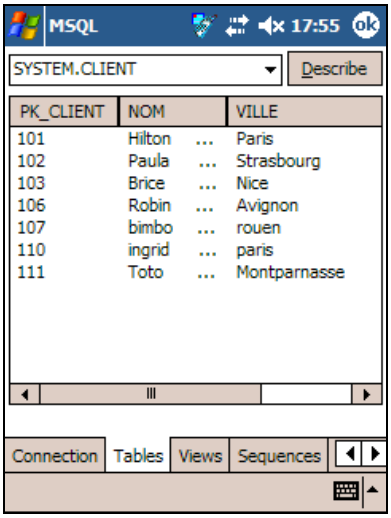
- **Ajout de la publication dans le Mobile Server à l'aide de Packaging Wizard** : Choisir « Créer une définition d'application » puis entrer les informations relatives au Mobile Server. Lorsque cela vous sera demandé, un explorateur de fichiers vous permettra de charger le fichier du projet *Mobile Database Workbench*. L'URL de la base de référence sera de la forme `jdbc:oracle:thin:@maison.xalti.com:1521:xe`.

- **Configuration du Mobile Server** : L'interface webtogo du Mobile Server est accessible à l'adresse `maison.xalti.com:8765/webtogo`. On trouvera dans la rubrique *Serveurs Mobile Server / Applications* la publication générée à l'étape précédente. La solution la plus propre consiste alors à créer un utilisateur (rubrique *Serveurs Mobile Server / Utilisateurs*) et à l'associer avec la publication (dans l'onglet *Accès*).

L'application CAR, publiée à l'aide du Packaging Wizard, doit être associée à l'utilisateur CAR nouvellement créé. Il ne reste alors plus qu'à lancer la synchronisation.

- **Synchronisation depuis le PDA** : Connecter le PDA à Internet puis saisir les informations de connexion au Mobile Server dans l'interface de configuration de mSync.





Une fois la synchronisation effectuée avec mSync, la copie embarquée de la base est consultable et modifiable avec mSQL. Cette base porte le nom donné à la publication dans le Database Workbench.

11. Autres

D'autres logiciels peuvent être installés pour une visualisation plus complète du projet et de son développement. Ces logiciels ne sont pas indispensables pour faire fonctionner le système.

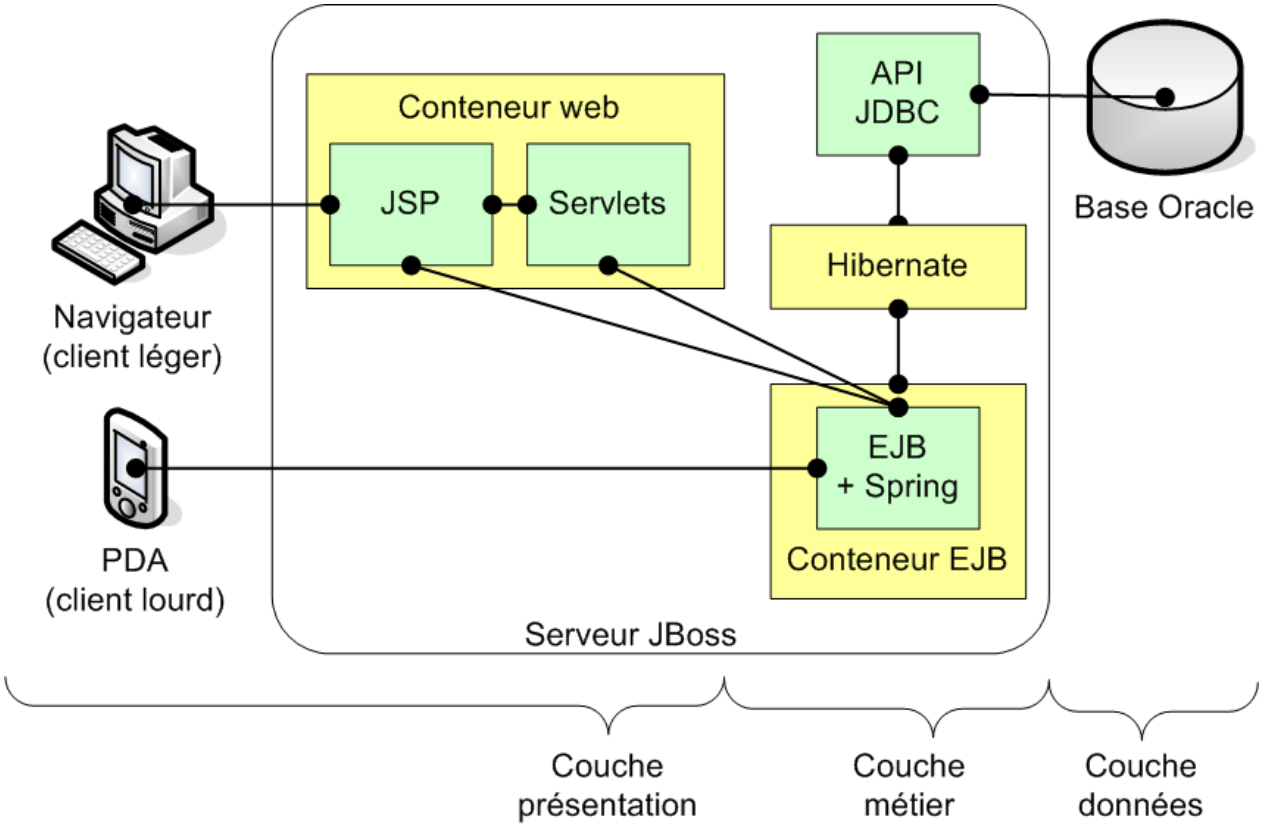
Le logiciel **Win'Design V5.2** a été installé pour réaliser le modèle métier, le Modèle Conceptuel de Données (MCD), le Modèle Logique de Données (MLD) et le script de génération de la base de données. Il permet de réaliser différentes étapes de la conception générale aux spécifications détaillées pour la production du cahier des charges. Il est disponible en version *Démonstration* sur à l'adresse http://www.win-design.com.

Le logiciel **Oracle SQL Developer** permet de manipuler la base de données Oracle créée pour le projet (insertions, modifications, suppressions des tables et des tuples, utilisation de requêtes SQL, etc.). Il est disponible en version gratuite sur le site d'Oracle à l'adresse http://www.oracle.com .

III. J2EE

1. Architecture générale

J2EE (Java 2 Enterprise Edition) est un ensemble d'APIs visant le développement d'applications orientées entreprise. Parmi ces APIs, certaines existent déjà dans la version "standard" de Java (Java 2 Standard Edition), tandis que d'autres ne sont présentes que dans la version orientée entreprise. Ainsi on a les APIs EJBs (Enterprise Java Beans) ou JMS (Java Message Service) qui sont spécifiques au JDK J2EE, à l'inverse des APIs JDBC ou RMI qui existent aussi dans le JDK J2SE. La plate-forme J2EE présente une solution optimale pour développer des applications robustes, sécurisées et évolutives. En effet, choisir cette technologie, c'est suivre un certain nombre de règles. Le but est en effet de séparer au maximum l'application en couches.



2. Architecture 5 couches

- **La couche Physique** correspond à la structure physique, au gestionnaire des données (concerne le SGBD *Oracle* dans le projet).

- **La couche Mapping** réalise les accès vers la couche Physique, et permet de mettre en relation l'application orientée objet et la base de données (concerne le framework *Hibernate* dans le projet).

- **La couche Entreprise** correspond aux objets structurants de l'entreprise, aux « objets métiers » relatifs à l'activité de l'entreprise, transversaux à toutes les applications. Ces objets n'intègrent aucune notion fonctionnelle. Cette couche propose des services d'accès à ces objets au travers de méthodes de création, recherche, modification, suppression. Ces méthodes contiennent les règles de gestion associées aux différentes opérations. La *couche Entreprise* utilise les objets générés par la *couche Mapping* pour fournir des services à la *couche Application*. Par exemple, en ce qui concerne la gestion des employés des agences dans le projet, cette couche contient des objets hérités des objets générés dans la couche *Mapping* et qui représentent les personnes.

- **La couche Application** regroupe la logique fonctionnelle d'une application, telle qu'elle est définie dans les spécifications fonctionnelles détaillées. La *couche Application* utilise les services de la *couche Entreprise* pour réaliser le fonctionnel spécifié sous forme de services. Ces services retournent des objets de *niveau application*, qui sont autant de vues sur les objets entreprise. Pour notre exemple de gestion des employés, cette couche contient les fonctionnalités qui permettent de gérer les personnes en appelant des fonctions des objets de la *couche Entreprise*.

- **La couche Client** (sous-couche de la couche Présentation) représente l'interface utilisateur. Elle est amenée à changer fréquemment dans le cas d'une application WEB. Elle concerne le *navigateur WEB* et l'application WEB basée sur des *pages JSP* dans notre projet. Les pages JSP sont transformées en servlets par le serveur d'application, des objets Java répondant aux requêtes des utilisateurs qui sont hébergées sur le conteneur de servlets coté serveur – *JBoss* dans notre cas.

3. Le modèle MVC (Model View Controller)

Le pattern Modèle-Vue-Contrôleur est l'un des modèles les plus connus. Il a été tout d'abord élaboré par Xerox lors de leur premier système fenêtré et plus particulièrement pour gérer les interactions avec l'utilisateur.

Le problème que résout le modèle MVC est celui de la simplification de trois grandes fonctions communes à de nombreuses applications :

- La maintenance des données dans un stockage de *back-end* ou sur un système distant.
- La construction de la couche présentation destinée à l'utilisateur final.
- La gestion de la logique conditionnelle qui décide des écrans qui sont présentés à l'utilisateur, de ce qui se passe lorsqu'une erreur survient et de la façon et du moment exacts où les systèmes distants sont mis à jour.

Il est donc possible de combiner l'ensemble de ces modules et d'avoir au final un système qui fonctionne. Cependant les problèmes se posent lorsque vous souhaitez maintenir le code. En effet dans le cas des JSP, les concepteurs HTML chargés du *look and feel* et ceux qui maintiennent le code de traitement sont différents. MVC traite ce problème en séparant le code en trois parties distinctes :

- Les composants Modèle, qui maintiennent les données dans le stockage.
- Les composants Vue, qui constituent la couche présentation destinée à l'utilisateur final.
- Les composants Contrôleur, qui gèrent la logique conditionnelle. C'est elle qui décide des écrans qui sont présentés à l'utilisateur, qui gère les erreurs et la mise à jour des systèmes distants.

MVC simplifie donc la maintenance de l'application et empêchent les trois types de logiques de se mélanger. Il permet également de masquer les détails d'implémentation de chaque domaine aux deux autres et réduit ainsi les dépendances de codage entre eux. De ce fait, MVC permet de définir une frontière naturelle entre les concepteurs web qui maintiennent le code html et la couche présentation.

Un des avantages considérable dans le développement d'application web (jsp, servlet) est que le modèle MVC facilite grandement la gestion des exceptions (car elles sont toutes gérées via le contrôleur).

D'autres avantages s'appliquent à toutes les formes de traitement conditionnel. Voici quelques exemples :

- Si différentes Vues sont nécessaires en fonction des données qui sont extraites d'une base de données ou d'un système distant, le Contrôleur peut décider de la page à présenter.
- Si votre application change en fonction de l'heure et/ou la date de la journée, le Contrôleur peut gérer cela facilement.
- Si un processus de saisie de données nécessite plusieurs pages, dont certaines sont facultatives.

IV. PRISE EN MAIN DU PROJET

1. Tutoriels

Pour mieux appréhender les concepts et technologies mises en œuvre dans ce projet, de nombreux tutoriels sont disponibles.

- Pour les bases du développement Java Web :
  Introduction à la programmation WEB en Java – Serge Tahé
  http://tahe.developpez.com/java/web/
  Développement web en Java avec Eclipse et Tomcat – Serge Tahé
  http://tahe.developpez.com/java/eclipse/

- Pour les architectures 5 couches :
  Modèle multicouche Architecture à 5 couches – Stève SFARTZ
  http://www.application-servers.com/articles/multicouches/index.html
  Architecture à 5 couches : du modèle à l'implémentation JAVA – Stève SFARTZ
  http://www.application-servers.com/articles/multicouchesjava/index.html

- Pour la prise en main de Spring :
  Les bases du développement web MVC en Java – Serge Tahé
  http://tahe.developpez.com/java/baseswebmvc
  Spring : théorie & pratique - Steve Hostettler
  http://zekey.developpez.com/articles/spring/

- Pour la prise en main de JBossIDE :
  JBoss Eclipse IDE Tutorial An introduction and walkthrough of JBoss Eclipse IDE
  http://docs.jboss.com/jbosside/tutorial/build/en/html/index.html

- Pour la prise en main d'Hibernate:
  Hibernate Tools Reference Guide - Chapter 3. Eclipse Plugins
  http://www.hibernate.org/hib_docs/tools/reference/en/html/plugins.html
  Débuter avec Hibernate sous Eclipse - Julien DEFAUT
  http://defaut.developpez.com/tutoriel/java/eclipse/hibernate/fichiers/hibernate.pdf

2. Accès au Serveurs – Tunnel SSH

Le projet éclipse est configuré pour utiliser un serveur SVN ainsi qu'une base de données Oracle. Cette configuration est faite pour des serveurs installés en local (adresse : *localhost*) ou des serveurs distants (pour cela il faut mettre un place un Tunnel SSH). Cette configuration se réalise sans avoir à éditer les URL des serveurs configurés dans le projet.

Il existe deux raisons pour utiliser cette configuration :

- Permettre l'utilisation d'une base de données locale pour effectuer des tests, ou en cas d'indisponibilité de la base de données centrale. Dans ce cas on n'utilise pas le Tunnel SSH.
- Permettre une connexion aux serveurs distants depuis un réseau où les ports sont filtrés (exemple : réseau universitaire). Dans ce cas on utilise le Tunnel SSH.

<u>Pour mettre en place le Tunnel SSH :</u>

- **Programme :** Putty
- **Fichier :** putty.exe
- **URL :** http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

- **Instructions :**

→Ce programme ne nécessite pas d'installation. Pour le lancer : double-cliquer sur le fichier.
→Dans la section *Connection/SSH/Tunnels* de l'arborescence, configurer le tunnel SSH :

- ✓ **Pour la base de donnée** : saisir *Source Port* : 1521 (Port part défaut de la base de données Oracle), *Destination* : url_du_serveur:1521 (dans notre cas url_du_serveur = maison.xalti.com). Cliquer ensuite sur *Add*.
- ✓ **Pour le serveur SVN :** saisir *Source Port* : 8443 (Port part défaut du serveur SVN), *Destination* : url_du_serveur:8443 (dans notre cas url_du_serveur = maison.xalti.com). Cliquer ensuite sur *Add*.

→Dans la section *Session* de l'arborescence, configurer la connexion à la passerelle :
Host Name : user_name@url_passerelle (par exemple *r506685@sas.esil.univmed.fr*)
Protocol : SSH
Donner ensuite un nom à cette connexion dans *Saved Session* puis cliquer sur *Save*.
→Pour lancer la connexion : double-cliquer sur son nom dans le cadre *Load, save or delete a stored session* ; ou sélectionner celle-ci dans le même cadre et cliquer sur *Open*.
→Dans l'écran suivant saisir votre mot de passe d'utilisateur.

3. Arborescence du Projet

Le projet pour l'application nationale ***Car*** est nommé *Car_AN* et s'organise comme suit :

- Le dossier *src* contient les sources java des classes de l'application : Beans, Servlets, POJO… Ce dossier est structuré en packages : *beans*, *servlets*, *business*, *dao*.

- Le dossier *dist* contient l'archive *Car_AN.war* (Web Application aRchive). Il s'agit d'une archive utilisée pour déployer l'application nationale ***Car*** sur un serveur d'application. Elle contient l'ensemble des pages, des classes compilées et des ressources de l'application.

- Le dossier *lib* contient les archives *.jar* utilisées par l'application web.

- Le dossier *properties* contient les données textuelles et les références sur les ressources affichées dans l'application.

- Le dossier *WebRoot* représente la racine de l'application web. Il contient l'arborescence de l'application pour les parties *jsp*, *css*, *images* et *javascript*. Il contient aussi un dossier *WEB-INF*. Celui-ci est indispensable pour une application web. Il contient un dossier *classes* pour les classes compilées et autres fichiers qui doivent se trouver dans le *CLASSPATH* de l'application ; un dossier *lib* où doivent se trouver les archives *.jar* utilisées par l'application. Ces deux dossiers sont vides dans le projet Eclipse. Ils seront automatiquement remplis lors du packaging de l'application dans le fichier *Car_AN.war*.

4. Propriétés du Projet

Plusieurs éléments ont étés configurés dans ce projet. Ils sont accessibles en effectuant un clic-droit sur le projet puis en sélectionnant l'option *Properties*.

A. XDoclet

XDoclet est une librairie de génération de code par l'insertion de tags Javadoc spécifiques dans les codes sources. Il est utilisé ici pour la compilation des classes Servlet. Les classes compilées sont placées dans un dossier */bin* qui n'apparait pas dans l'arborescence du projet. Il produit aussi le ficher *web.xml* qui décrit les éléments de l'architecture web : les Servlets, leur classe, leurs paramètres et leur URL associée.

B. Packaging Configuration

La Packaging Configuration se charge de créer l'archive *.war* (Web Application aRchive). Il s'agit d'une archive utilisée pour déployer l'application nationale **Car** sur un serveur d'application. C'est ici que sont spécifiés les dossiers et fichiers du projet à inclure dans

l'application web ainsi que l'endroit où ils seront placés dans l'archive. Voici le contenu des dossiers inclus dans cette archive:

- */WebRoot* inséré à la racine de l'archive
- */bin* placé dans *WEB-INF/classes*
- */properties* dans *WEB-INF/classes*
- */lib* dans *WEB-INF/lib*

Le serveur d'application inclut dans le CLASSPATH de l'application web qu'il déploie les dossiers *WEB-INF/classes* et *WEB-INF/lib*, rendant ainsi accessible à toutes les classes et pages JSP l'ensemble des ressources utilisées et développées dans le projet. Les fichiers inclus sont :

- *web.xml* dans *WEB-INF*
- *sprinx.xml* placé dans *WEB-INF* (description des classes prises en charge par Spring).

5. Compilation et Déploiement

Voici les opérations à effectuer pour préparer et effectuer le déploiement du projet :

→Lancer XDoclet :

- Clic-droit sur le projet puis *Run XDoclet* ; ou *Ctrl+Maj+F1*

→Lancer la Packaging Configuration :

- Clic-droit sur le projet puis *Run Packaging* ; ou *Ctrl+Maj+F2*

→Déployer le projet :

- S'il n'est pas démarré, démarrer le serveur d'application (expliqué dans la partie II.3).
- Clic-droit sur l'archive */dist/Car_AN.war* puis *Deployment/Deploy To*, choisir le serveur *server_RJS* et cliquer sur *OK* ; ou *Ctrl+Maj+F3.*

6. Spring IoC

Spring Inversion of Control met en œuvre la programmation par contrat. Cela permet de supprimer la dépendance entre les couches de l'application, et de faciliter l'intégration des composants entre eux.

On définit une interface pour chaque classe de chaque couche. Ces interfaces sont implémentées par des classes qui respectent les règles de programmation des Beans. Une classe qui utilise les méthodes offertes par une autre fait seulement référence à l'interface de celle-ci. Elle possède un attribut du type de l'interface dont elle veut utiliser les fonctionnalités. Elle possède les accesseurs et mutateurs sur cet attribut respectant la convention de nommage des Beans. Elle ne crée pas d'instance d'une implémentation de l'interface (laissé à Spring).

Le fichier *spring.xml* définit les dépendances entre les objets. Il définit l'implémentation d'interface qu'il est nécessaire à instancier pour les attributs d'un objet. Ce fichier doit être inclus dans le *CLASSPATH* de l'application web déployée. L'instanciation d'un Bean peut se faire explicitement par un appel à *XMLBeanFactory*.

7. Création d'un Module

Ce chapitre explique rapidement comment créer les différents composants pour les 4 couches développées au dessus de la base de données. Il suppose que les concepts et techniques de développement sont connus du développeur, et ne visent qu'à lui présenter le détail de l'architecture mise en place dans l'application Car.

A. Fichier de Configuration Spring

Le fichier XML de configuration de Spring se trouve dans le dossier */spring* et est nommé : *anServlet-servlet.xml* . C'est dans celui-ci que sont définis les objets à instancier pour chaque couche et leurs dépendances. Ce fichier est organisé par ordre alphabétique pour regrouper les composants de chaque couche et pour simplifier son édition. Les paragraphes suivants feront régulièrement référence à ce fichier pour déployer un nouveau composant. Il définit :

- le Data Source pour l'accès à la base,
- le SessionFactory d'Hibernate,
- les DAO pour l'accès aux données,
- les objets métiers,
- les contrôleurs de l'application web,
- le dispatcheur vers ces contrôleurs,
- l'intercepteur chargé de contrôler l'accès aux ressources,

ainsi que leurs dépendances entre eux.

B. Couche Mapping

La mapping en lui-même est réalisé automatiquement par Hibernate. Les classes produites par Hibernate se trouvent dans le dossier *src*, dans *car.appli.hibernate.* Si une nouvelle table est ajoutée dans la base de données il convient de régénérer les classes de mapping en utilisant la configuration *Car_AN_configuration* du "Hibernate code génération" dans Eclipse. Les nouveaux fichiers **.hbm.xml* produits dans le dossier *src* et situés dans *car.appli.hibernate* doivent être rajoutés comme ressources pour le mapping du SessionFactory d'Hibernate dans le fichier de configuration Spring.

L'accès aux données se fait par des DAO qui se trouvent dans le dossier *src*, dans *car.appli.dao*. Un DAO correspond en général à une classe de mapping, donc à une table de la base de données (les fichiers DAO peuvent tout de même exécuter des requêtes impliquant plusieurs tables). Pour créer un nouveau DAO, il faut tout d'abord créer une interface définissant les méthodes à implémenter. Par convention, le nom des interfaces commence par un *I* et de terminent

par *DAO*, de la forme : *IMesDonneesDAO*. Cette interface doit être implémentée par une classe qui étend *HibernateDaoSupport*, nommée de la façon suivante : *MesDonneesDAO* .

L'instanciation de ces objets est faite par Spring. Il faut donc les rajouter dans le fichier *anServlet-servlet.xml*, en spécifiant une référence sur le SessionFactory d'Hibernate (*anSessionFactory*) définit dans le même fichier.

C. Métier

Les objets de la couche métier se trouvent dans le dossier *src*, dans *car.appli.business*. Une classe métier correspond à une "fonctionnalité métier" définie dans le cahier des charges (Employés, Clients, Réservations…).

Pour créer une nouvelle fonctionnalité métier, il faut tout d'abord créer une interface définissant les méthodes à implémenter. Par convention, le nom des interfaces commence par un *I* et de termine par *Business*, de la forme : *IMonMetierBusiness*.

Cette interface doit être implémentée par une classe nommée de la façon suivante : *MonMetierBusiness*. Les méthodes de ces classes peuvent contenir de la **logique purement métier**, comme la mise en œuvre d'une réservation/location, ou simplement être des **CRUD** permettant d'accéder à la couche DAO. La référence aux DAO de la couche inférieure se fait par le biais d'objet dont le type est l'**interface** et non l'implémentation de ces DAO, sur le modèle : *IMesDonneesDAO mesDonneesDao.* De plus, il faut ajouter un mutateur pour cet objet dont le nom suit la forme : *setMesDonneesDao(IMesDonneesDAO mesDonneesDAO).*

L'instanciation de ces objets est faite par Spring. Il faut donc les rajouter dans le fichier *anServlet-servlet.xml* en spécifiant une référence sur les instances des DAO définies plus tôt (dont ils ont besoin).

D. Contrôleurs

Les objets de la coucher contrôleur se trouvent dans le dossier *src*, dans *car.appli.controllers* . Ils sont séparés en deux packages, un pour la partie d'administration réservée aux employés, et l'autre pour la partie réservée aux clients de l'entreprise.

Une classe contrôleur correspond à une "fonctionnalité métier" définie dans le cahier des charges (Employés, Clients, Réservations…). Elle est chargée de : vérifier les données saisies par l'utilisateur, contrôler les pages demandées, renvoyer l'utilisateur sur la page qu'il a demandé et interagir avec la couche métier pour effectuer des traitements métiers, récupérer ou sauvegarder des données. Chaque contrôleur implémente *Controller* pour permettre la mise en œuvre de Spring MVC.

La référence aux objets métiers de la couche inférieur se fait par le biais d'objet dont le type est l'**interface** et non l'implémentation de ces objets métiers, sur le modèle : *IMonMetierBusiness monMetierBusiness.* De plus il faut ajouter un mutateur pour cet objet dont le nom suit la forme : *setMonMetierBusiness(IMonMetierBusiness monMetierBusiness).*

L'instanciation de ces objets est faite par Spring. Il faut donc les rajouter dans le fichier *anServlet-servlet.xml*, en spécifiant une référence sur les instances des objets métiers définies plus tôt, dont ils ont besoin. De plus, pour les rendre accessible par la couche présentation au travers d'une URL, il faut rajouter une propriété de mapping au dispatcher *urlMapping*, qui est définit dans le fichier *anServlet-servlet.xml*. Cela permet de spécifier l'url qui, quand elle est appelée, entrainera l'exécution du contrôleur qui lui est associé.

Enfin, les données qui sont passées à la couche présentation sont placées dans le modèle (de MVC). Pour cela un *Map* est utilisé pour stocker les données, associées à une clef qui les identifie.

E. Présentation

La couche présentation est chargée d'afficher des données à l'utilisateur, et de lui offrir les interfaces nécessaires pour naviguer et interagir avec l'application. La couche présentation se compose de pages JSP, associées à un style CSS et de fonctionnalités JavaScript pour faciliter l'utilisation par l'utilisateur.

Toutes ces ressources sont regroupées dans le dossier /*WebRoot* dans des sous dossiers leur correspondant. Les pages JSP se trouvent dans /*WebRoot/jsp/admin* et /*WebRoot/jsp/client*. Les parties communes à toutes les pages sont regroupées dans un sous dossier */part* .

Chaque page inclut donc les composants *part/top.jsp* et *part/bottom.jsp* pour uniformiser le rendu. L'accès aux données du modèle se fait principalement en utilisant les balises offertes par la *JSTL*, mais aussi et les récupérant par l'intermédiaire de l'objet *pageContext* qui existe dans chaque JSP. L'appel à la couche contrôleur se fait par des liens vers les URL définit plus tôt. Aucuns liens directs entre JSP ne doit exister pour assurer la cohérence de l'application.

F. Internationalisation

Cette application a été développée dans le but de permettre son portage rapide vers une autre langue. Pour ce faire, aucun texte n'est présent directement dans la couche présentation ou dans les autres couches. Le texte de l'application est stocké soit :

- dans la base de données, ce qui les rend facilement remplaçables par une traduction
- dans des fichiers *.properties*, dans /*WebRoot/properties/*

Il existe un fichier **.properties* par fonctionnalité métier de l'application. Ces fichiers contiennent l'ensemble des données affichées de façon statique dans la couche présentation. Pour chaque donnée, il est spécifié un unique identifiant qui est utilisé dans les JSP pour les afficher. Les JSP accèdent à ces données par l'intermédiaire d'un *ResourceBundle*. Ces données peuvent donc être remplacées par leur équivalent dans une autre langue.

G. Contrôle d'accès

Le contrôle d'accès aux pages de la partie d'administration se fait par une authentification des utilisateurs qui crée une session dans laquelle est placée un objet *car.appli.beans.ControleAcces* . Celui-ci contient l'identité de l'utilisateur ainsi que l'ensemble des droits dont il dispose comme ils sont définit dans la base de données. Ces données sont utilisées par un **intercepteur**, *car.appli.controllers.admin.AccesInterceptor* qui intercepte toutes requêtes sur les URL des contrôleurs. Il se charge de vérifier que l'utilisateur a biens les droits nécessaires pour accéder à cette ressource ; si ce n'est pas le cas, il est renvoyé sur la page d'accueil. Pour ajouter des fonctionnalités, il faut tout d'abord ajouter les droits associés dans les tables DROITS et FONCTIONNALITE. Enfin, il faut vérifier dans *AccesInterceptor* que l'utilisateur possède bien les droits pour accéder à cette fonctionnalité.

H. Configuration de l'accès à la base

L'accès à la base de donnée est définit dans le fichier *anServlet-servlet.xml* pour l'objet *anDataSource*, qui est le DataSource de l'application. Il faut donc spécifier ici l'URL, le nom d'utilisateur et son mot de passe, pour que l'application soit capable de communiquer avec la couche Données.

I. Classes Utilitaires

Des classes utilitaires ont étés développées pour simplifier la tâche des développeurs, notamment pour crypter les mots de passe ou pour manipuler les dates. Ces classes sont situées dans le dossier /*src*, dans *car.fwk* .

V. CONCLUSION

Problèmes et réponses apportées

Problème de version

Un problème de version est apparu avec JBoss IDE, qui contient Eclipse 3.1. Cette version n'est pas compatible avec la version de 2.0 SpringIDE. Il a donc été nécessaire d'installer une ancienne version de Spring, la version 1.3, pour faire fonctionner l'architecture.

Problème de port

JBoss et Oracle utilisent par défaut le port 8080. Ce port est aussi utilisé par Oracle pour son interface web et par JBoss pour l'affichage des contenus web. Il convient donc de changer le port par défaut dans le fichier de configuration de JBoss. Comme exposé dans l'installation de JBoss.

Problème de JVM pour JBoss

Par défaut une JVM est crée avec un tas ne pouvant pas dépasser 64Mo. Le serveur JBoss est exécuté par une JVM. Celui-ci peut avoir besoin de plus de mémoire que celle disponible par défaut. Le problème s'est posé quand nous avons commencé à utiliser Hibernate. Pour remédier à ce problème il faut lancer la JVM qui exécute le serveur avec l'option *–Xmx<taille><unité>* comme précisé dans la II. 3. Eclipse – JBossIDE.