ÉCOLE SUPÉRIEURE D'INGÉNIEURS DE LUMINY
UNIVERSITÉ DE LA MÉDITERRANÉE AIX-MARSEILLE II

Département Informatique
Promotion 2008

DOSSIER DE CONCEPTION

Projet CAR

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

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

Date de rédaction : *27/01/2008*

Sommaire

HISTORIQUE DES REVISIONS

Référence	Révision	Date	Auteur(s)	Nature de la révision
Dossier de Conception CAR	00	20/11/2007	Equipe de développement	Version projet initiale
Dossier de Conception CAR	01	11/01/2008	Equipe de développement	Conception Générale AN
Dossier de Conception CAR	02	27/01/2008	Equipe de développement	Conception Détaillée AN et AM

REFERENCE INTERNET

http://jonathan.favier.free.fr/projets/Car/

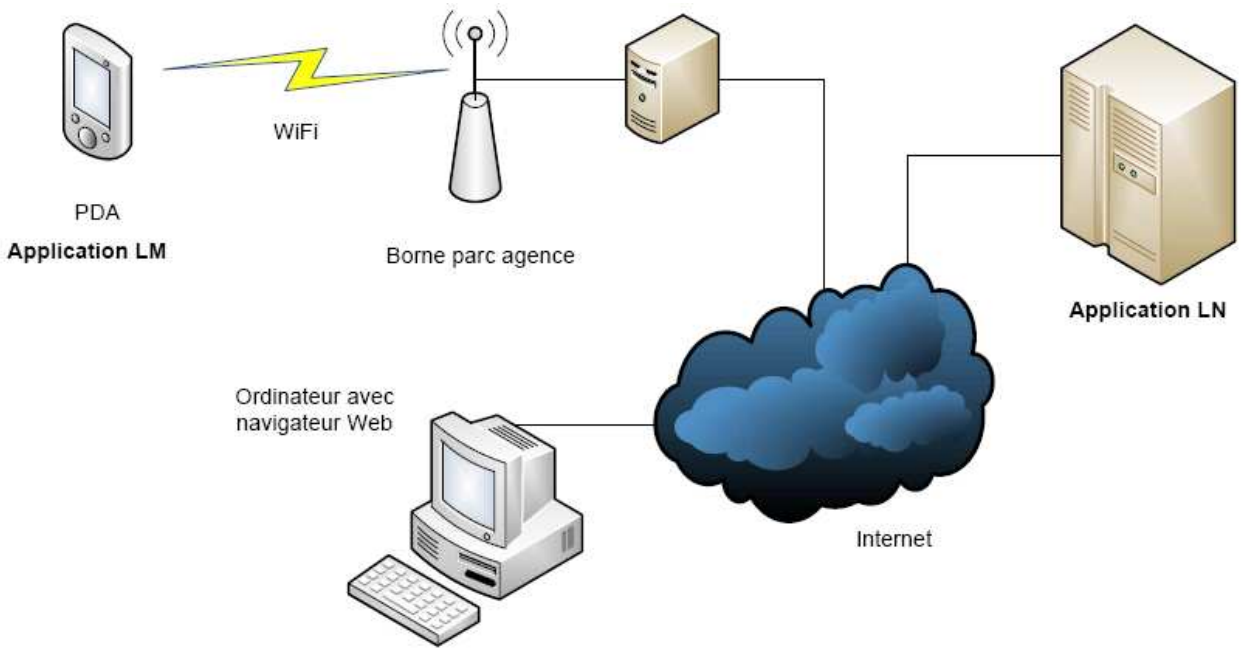
I. DOMAINE D'APPLICATION

1. Objectifs du système

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 contient deux applications, une **application nationale (AN)** installée au siège de la société, et une **application mobile (AM)** 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 doit être 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), destinée aux agents responsables des parcs véhicules, permet à ces derniers de gérer les parcs des véhicules des agences ainsi que les locations.



Architecture générale de l'application CAR

2. Interfaces

A. Interfaces Utilisateur

Le système contient deux applications, une nationale et une mobile. L'**interface de l'application nationale** est une interface WEB (via un navigateur WEB tel que *Internet Explorer* ou *Firefox*) contenant des formulaires et des tableaux de données regroupés par fonctions, le tout agencé par une navigation par onglets. Cette interface WEB permet l'affichage et la saisie de données. L'**interface de l'application mobile** sur PDA est calquée sur les applications par défaut du PDA. Il s'agit également de formulaires et de listes de données, plus un accès rapide à la synchronisation de données. Cette interface prend en compte les contraintes du PDA : technologie disponible, taille de l'écran, résolution. Pour plus d'informations sur ces interfaces, se référer directement à la charte graphique du projet **CAR**.

Ces deux applications comportent à leur point d'entrée une **authentification** par mot de passe de l'employé (ou du client pour l'AN) qui va utiliser les services proposés. En fonction de la nature de l'utilisateur, des droits ou des restrictions seront configurés sur les applications.

B. Interfaces Logicielles

Les interfaces logicielles mentionnées ici sont des interfaces internes au système.

- **Interface avec la base de données centrale :** les deux applications du système CAR doivent interagir avec la base de données centrale (Oracle, installée sur le serveur de base de données) au moyen de pilotes dédiés, de manière synchronisée. Cette base de données rassemble toutes les données manipulées par l'application nationale.

- **Interface avec chaque base de données mobile :** l'application mobile installée sur le PDA doit interagir avec la base de données embarquée qui l'accompagne (Oracle Lite), grâce à un pilote dédié.

- **Interface de synchronisation entre les PDA et le serveur central *:*** tous les PDA contiennent un composant de synchronisation de données avec le serveur de base de données (et inversement).

C. Interfaces de communication

Les interfaces de communication mentionnées ici sont des interfaces internes au système.

- **Interface entre le PDA et le serveur central :** le PDA et le serveur central, lors des synchronisations, communiquent grâce à une liaison WIFI (ou une liaison USB filaire pendant la phase de développement uniquement).

- **Interface entre le client WEB et l'application nationale :** pour accéder aux données et aux services présentés par l'application nationale, une connexion TCP/IP est nécessaire.

- **Interface entre l'application nationale et les points d'accès distants :** la synchronisation entre les bases embarquées des PDA et la base centrale a lieu au niveau de points d'accès distants, reliés eux-mêmes à la base de données centrale. La communication entre ces composants est réalisée grâce à une connexion Internet de type TCP/IP.

3. Bases de données externes

Aucune base de données externe n'est prévue pour interagir avec notre système. Les seules bases de données, mentionnées précédemment, font partie du système à concevoir.

4. Contraintes générales de conception

Plusieurs contraintes provenant de différentes sources sont à prendre en compte dans la phase de conception du système. Ci-dessous, un récapitulatif des contraintes imposées par le maître d'ouvrage dans le cahier des charges :

- L'architecture **J2EE** doit être utilisée pour l'application nationale, avec l'utilisation des composants **Hibernate** et **Spring**.
- La base de données doit être de type **Oracle** pour l'application nationale, et la base de données de la partie mobile **Oracle Lite.** Ces bases de données doivent être **synchronisées à intervalle régulier.**
- Le **modèle du PDA** est imposé/fourni par le maître d'ouvrage.
- La phase de conception n'est démarrée qu'après validation du **modèle métier** (et du modèle conceptuel des données) par le maître d'ouvrage.
- Les **règles de document applicables ACAI** définies comme références dans le cahier des charges doivent être respectées tout au long de la conception.
- Les deux applications devront être en **français**.
- Les connexions aux applications du système CAR doivent être **sécurisées par mot de passe**. Chaque catégorie d'utilisateurs disposera de **droits spécifiques**.

II. DOCUMENTS DE REFERENCE

Les documents suivants sont à utiliser en référence avec la lecture de ce document :

- **Le Cahier Des Charges** dans sa version finale : il contient les spécifications initiales des exigences du maître d'ouvrage. *Référence : CAR-CDC-AVIS-V01*
- **Le Modèle Métier :** il correspond au modèle conceptuel des données, et est accompagné des règles de gestions, des contraintes d'historisation et de cohérence des données.
  Référence : CAR-MM-AVIS-V01
- **La Charte Graphique** : elle a été établie spécialement pour le projet CAR.
  Référence : CAR-CG-AVIS-V01
- **Les Maquettes des écrans et leurs Cinématiques.**
  Référence : Partie clients : http://jonathan.favier.free.fr/projets/Car/AN/
  Administration : http://jonathan.favier.free.fr/projets/Car/AN/connexion.html
- **Le Dossier Développeur :** il contient les instructions techniques de mise en place de l'environnement de développement et de l'architecture.
  Référence : CAR-DD-AVIS-V01
- **Les Règles ACAI** : elles sont été imposées par le maître d'ouvrage dans le CDC.
  - **ACAI**-GuideErgonomie-150
  - **ACAI**-GuidePersistance-150
  - **ACAI**-GuideRéalisation-150
  - **ACAI**-ModèleErgonomie-150

  Référence : http://www.informatique.dgpa.equipement.gouv.fr/

Les documents suivants ont été utilisés pour améliorer les concepts introduits dans ce dossier :

- **Cours de Qualité** *(William Boher-Coy)*
- **Cours de Génie Logiciel Appliqué et de Génie Logiciel** *(William Boher-Coy)*
- **Cours de JAVA/J2EE** *(Yannick Tisson)*
- **Projets MPM, Pharmatica et CAPT** *(projets scolaires menés à l'ESIL en 2006-2007)*

III. NORMES, STANDARDS ET OUTILS

1. Méthodes de conception

Les méthodes de conception sont utilisées afin d'améliorer la qualité de la conception finale.

La méthode de conception **MERISE** a été utilisée pour mettre en place le **Modèle Métier** du système, le **Modèle Conceptuel de Données** (MCD), le **Modèle Logique de Données** (MLD) pour aboutir enfin au script de génération de la base de données. Ces différents modèles ont été créés grâce à l'environnement de conception *Win'Design* de la société Cecima.

Les **recommandations ACAI** en terme de modélisation et de conception-réalisation d'applications (**documents applicables)** ont constitué des références pour les phases d'analyse et de conception du projet . Elles peuvent être vues comme un ensemble de bonnes pratiques permettant d'orienter l'architecture et les choix techniques du système.

Référence : ACAI-GuideModélisation-150

La **norme IEEE 1471 (2000)** représente également une source de recommandations importante en ce qui concerne l'architecture en 5 couches de l'application nationale. Elle préconise ainsi l'utilisation intensive de vues et notamment le **Modèle-Vue-Contrôleur** (MVC – cf. Dossier Développeur) utilisé par la couche « Client».

Les **designs patterns** sont utilisés pour améliorer l'architecture du logiciel. Ce sont des modèles de conception réutilisables qui répondent à des problématiques courantes de conception indépendamment de tout langage. Ces modèles de conception fournissent un support fort pour la mise en oeuvre de principes chers à l'approche par objets : la flexibilité, la réutilisabilité, la modularité, la maintenabilité…

Référence : Catalogue Sun des design patterns appliqués à J2EE (accès public)
http://java.sun.com/j2ee/blueprints/design_patterns/catalog.html

Concernant l'élaboration du système, un **style de conception ascendante (ou « bottom-up »)** a été choisi. Cette approche permet de s'appuyer sur un modèle métier validé par le maître d'ouvrage, puis de le transférer rapidement en modèle objet. Elle permet également d'intégrer les frameworks et l'architecture en couches, dans l'optique de fournir des composants réutilisables et autonomes.

2. Environnement et outils de développement

Le matériel de développement utilisé est une machine préparée pour chaque développeur, équipée de **Windows XP Professionnel** et d'une quantité suffisante de mémoire vive (le minimum a été fixé à **1Go** pour avoir une qualité de développement acceptable, en partie en raison des nombreux services à exécuter).

Les trois membres de l'équipe de développement exécutent les applications du projet CAR sur leurs propres machines. La base de données Oracle est située sur une quatrième machine personnelle ayant le rôle de serveur central. **Quatre machines** sont donc nécessaires pour le développement du projet.

Concernant l'application nationale articulée sur la plate-forme J2EE, l'**outil de développement Eclipse** est mis à disposition de l'équipe de développement. Les **environnements de développement** qui interviennent dans la conception et le développement du système sont **Hibernate** et **Spring.** Concernant l'application mobile sur PDA, la technologie **J2SE** est utilisée dans l'**environnement de développement Eclipse**.

La **base de données Oracle** est manipulée et testée grâce à l'outil **SQL Developer** (lors de la phase de développement, un ensemble de données tests est utilisé afin d'avoir un support convenable pour les différents services à créer). **L'environnement de conception Win'Design** est quant à lui utilisé pour la réalisation des modèles et pour la génération du script de création de la base de donnée et des tables associées.

La gestion de configuration est effectuée sur un serveur FTP pour la documentation écrite et les composants sources (classes, scripts, fichiers de configurations, pages WEB).
Référence : http://jonathan.favier.free.fr/projets/Car/

Pour plus d'informations sur l'ensemble des outils et technologies utilisées dans ce projet, se référez directement au *Dossier Développeur* ou au *Dossier de Gestion de Configuration*.

3. Notations utilisées

La terminologie utilisée dans le projet CAR est disponible dans le Cahier Des Charges. Les acronymes qui sont les plus couramment utilisés sont **AN** (application nationale CAR) et **AM** (application mobile pour PDA). Les conventions suivantes seront utilisées pour les schémas de modélisation : normes UML 1.5 et conventions MERISE.

4. Conventions de nommage

Ces conventions concernent les répertoires et dossiers, des entités spécifiques dans les applications du système CAR (classes, variables, packages, entités de fichiers de configuration) et dans la base de données (tables, champs). Ces conventions respectent les conventions de codage précisées en annexe dans le document ACAI-GuidePersistance-150.

A. Noms des applications et des packages

Le système complet s'appelle « CAR ». Chaque application possède un nom spécifique qui permet de l'identifier dans le système (AN pour Application Nationale, AM pour Application Mobile). Les packages utilisés dans les sources sont définis comme suit :

- La racine de tout le système est : « CAR »
- L'application centrale est située dans le package : « CAR.AN »
- L'application mobile pour les agents responsables des parcs de véhicules est située dans le package : « CAR.AM »

B. Base de données

Les **tables de la base de données** sont issues de deux catégories de données dans le MCD : les entités (employés, clients, etc.) et les associations (rattachement d'un véhicule à une agence, mise à disposition d'un véhicule par un employé, etc.). Chaque **entité** possède un libellé lisible qui permet de la distinguer clairement (table EMPLOYE au lieu de EMP par exemple). Ce libellé peut contenir des chiffres, mais ne peut pas contenir d'articles dans le nom (ex : LE_CLIENT), ni de verbes (VEHICULE_RATTACHER_A).

Référence : PA52 du Document ACAI-GuidePersistance-150

Les **associations** reliant toujours deux entités au minimum, le libellé des tables correspondantes est une concaténation des deux entités, ce qui donne par exemple « VEHICULE_AGENCE pour la table correspondant à l'association rattachant un véhicule à une agence. Etant donné que plusieurs associations peuvent exister entre 2 tables, on ajoutera alors au libellé de la table le nom caractérisant cette association (RATTACHEMENT, CORRESPONDANCE, etc.). Les noms des tables ne devront jamais dépassés 30 caractères.

Concernant **les champs de la base de données**, les noms des colonnes ont les mêmes règles que les libellés des tables. Le nom de chaque colonne d'une table commence par les 3 premières lettres du nom de la table (hors préfixe). Ceci permet d'identifier très clairement les colonnes de clé étrangère : elles ne commencent pas par les 3 mêmes lettres.

Un **index sur une clé primaire** est créé automatiquement et portera le nom de la clé primaire : PK_nom de la table. Les **index sur les clés étrangères** seront préfixés de FK. On aura FK_nom de la référence.

Concernant les **requêtes SQL** présentes dans les fichiers sources, elles doivent être écrites de la façon suivante : lettres majuscules pour les mots SQL (ex. : SELECT, UPDATE, TO_CHAR) et lettres minuscules pour les noms des objets sur lequel porte la requête (noms des tables, champs, variables, etc.). Avant chaque **mot SQL**, il est souhaitable d'avoir un retour à la ligne et d'aligner les lignes de la requête (avec des espaces et non des tabulations qui ne sont pas portables pour la mise en page). De même, les conditions des sous-requêtes doivent être décalées par rapport à la requête principale.

Référence : PA54 du Document ACAI-GuidePersistance-150

C. Application

Application nationale : nom des couches

Au sein de l'application nationale, il convient de nommer chaque couche de façon prédéfinie. En effet, chaque couche est stockée dans un package différent :

- la **couche physique** n'est pas représentée car il s'agit de la base de données
- la **couche mapping** est stockée dans les packages : *car.an.hibernate* et *car.an.dao*
- la **couche métier** est stockée dans le package : *car.an.business*
- la **couche application** est stockée dans le package : *car.an.controllers*
- la **couche présentation** est stockée dans un dossier séparé *WebRoot* contenant des jsp et les ressources associées.

Application nationale : couche mapping

Au sein de la couche mapping, plusieurs conventions de nommage sont à déterminer. On distingue deux types de fichiers : les fichiers de mapping (.hbm.xml) et les classes d'entités métiers (.java). Ces deux types de fichiers sont générés automatiquement à partir de la base de données.

Les classes de mapping développées dans ce projet utilisent le mapping généré par Hibernate. Ces classes d'accès aux données (Data Access Object) sont nommées suivant le schéma *<NomDonneesAccedees>DAO*. Des interfaces définissent pour chaque DAO les méthodes offertes. Chaque classe DAO implémente l'interface qui lui correspond. Ces interfaces sont nommées suivant le schéma *I<NomDonneesAccedees>DAO*.

Application nationale : couche métier

Au sein de la couche métier, plusieurs conventions de nommage sont également à mettre en place. Tout d'abord, les fichiers sont organisés par entités, les entités étant elles-mêmes organisées par groupes d'entités selon l'axe de fonctionnalité (activité commerciale, employés, clients etc.). Chaque entité est alors représentée par 2 fichiers principaux (d'autres fichiers complémentaires peuvent s'ajouter) :

- les managers, nommés *<NomManager>Business*
- les interfaces associées, implémentées par les managers : *I<NomManager>Business*

Application nationale : couche application

La couche application est constituée des différents contrôleurs permettant d'implémenter chaque service demandé. Ces contrôleurs sont organisés par axes de fonctionnalités (à l'image de la couche métier). Tous les contrôleurs sont nommés suivant le model *<Nom>Controller*

Spring

Spring définit des objets et leurs dépendances dans un fichier de configuration *xml*. Les objets qui font référence aux classes crées pour représenter les différentes couches sont nommées suivant le model suivant *nomClasse* pour une classe *NomClasse*.

5. Standards de programmation

L'équipe de développement suit un ensemble de conventions de codage qui permettent une homogénéisation des sources :

- Les **commentaires** doivent être rédigés pour chaque classe et chaque méthode en respectant la norme JavaDoc, afin de permettre à toute personne entrant dans le projet de reconnaître l'utilité, les entrées et les sorties de ces entités.
- Les **conventions de codage Java** utilisées sont celles recommandées par les ACAI (cf. Guide de conception-réalisation).
- Les **conventions de codage SQL** sont également celles recommandées par les ACAI (cf. Guide de la persistance).
- Les **conventions approuvées par le W3C** sont également appliquées dans le cadre du développement WEB (HTML et CSS).

IV. CONCEPTION GENERALE

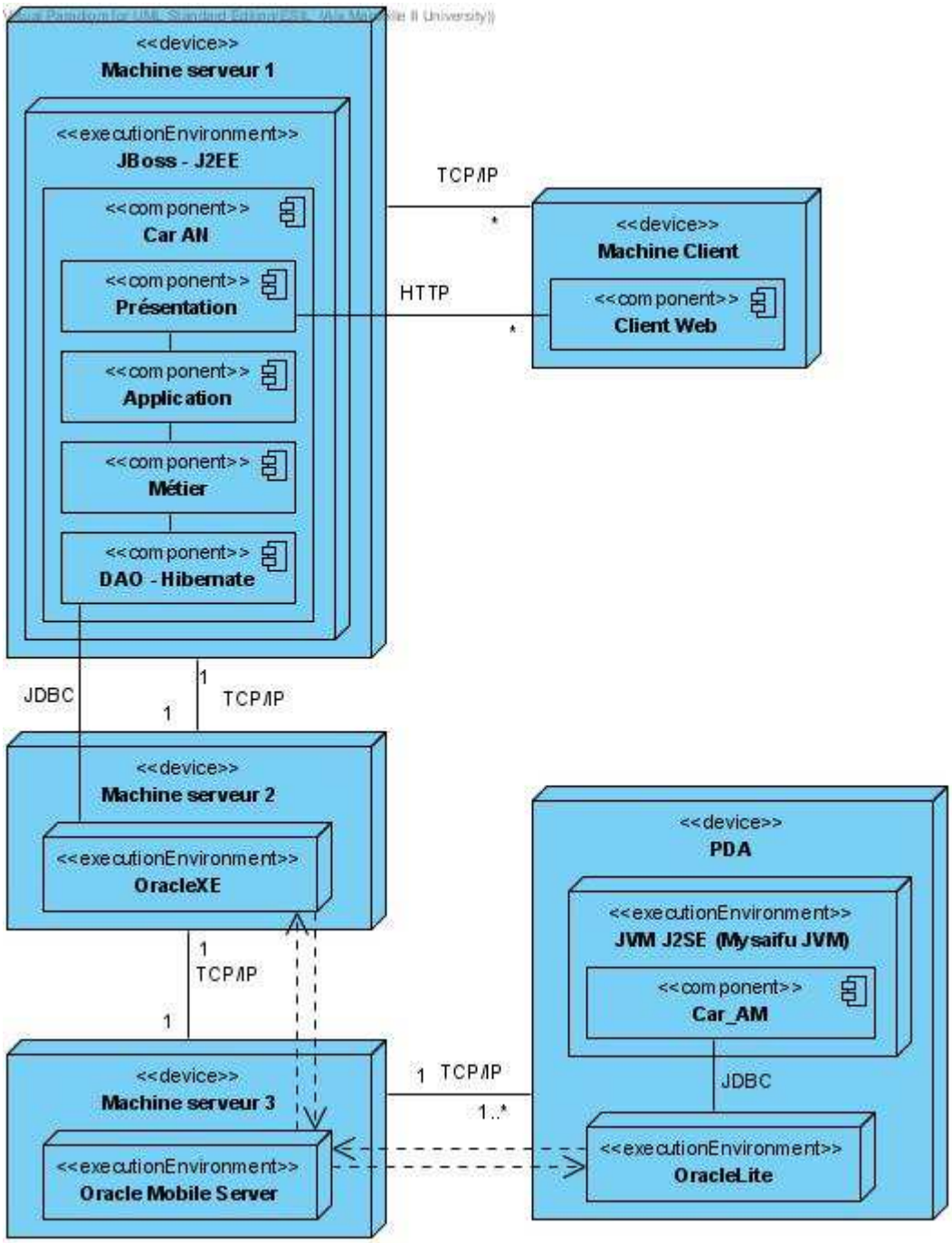
1. Langages utilisés

Voici la liste des différents langages utilisés dans le projet ***CAR*** :

- **SQL** pour les scripts de création de la base de données (création de tables et insertions de tuples dans la BD) ainsi que pour les requêtes spécifiques de l'AM.
- **PL/SQL** pour les déclencheurs.
- **HQL** (langage à requêtes de Hibernate, similaire à SQL) pour les requêtes spécifiques dans l'AN.
- **Java 1.5** pour le développement de l'application nationale, **Java 1.4** pour l'application mobile.
- **JSP** pour le développement des vues dans l'AN.
- **XHTML**, **CSS** et **Javascript** à pour le contenu des vues JSP dans l'AN.
- **XML** pour les fichiers de configuration *Spring*, *Hibernate* et *Tomcat.*

2. Diagramme de déploiement

Voici un diagramme de déploiement résumant les différents composants qui font partie du système Car.

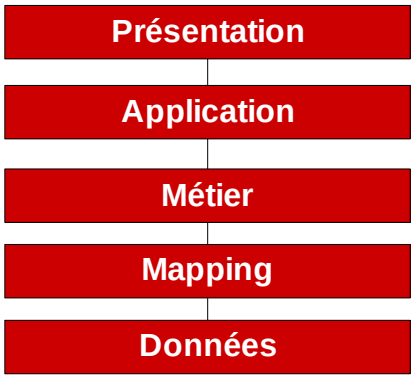


3. Architecture des Composants

Chaque application bénéficie d'une architecture séparée, où le seul point commun est la base de données centrale. Dans ce paragraphe, nous allons nous attarder sur les problèmes d'architecture fondamentaux du système. Tous les autres choix de conception sont expliqués dans le chapitre « Conception détaillée ».

A. Architecture 5 couches pour l'application nationale

L'application nationale est divisée en cinq couches de fonctionnalités, totalement autonomes les unes des autres, et communiquant par un système de file : chaque couche ne dialogue qu'avec les couches voisines supérieure et inférieure. Toutes les couches doivent agir de façon transparente les unes des autres. La séparation des couches est la suivante :



Couche Données : cette couche contient les données physiques stockées dans la base de données Oracle. Elle ne requiert pas d'implémentation Java particulière et fonctionne simplement comme un espace de consultation massif.

***Couche Mapping* :** cette couche contient l'implémentation des accès à la base de données afin de la masquer à la couche métier. Cette couche est entièrement gérée par Hibernate.

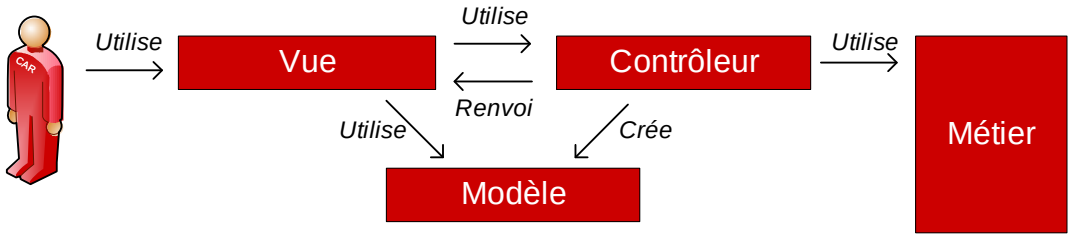
Couche Métier : cette couche contient les objets métiers de l'application. Il existe un objet par fonctionnalité de l'application. Ces objets implémentent les fonctionnalités spécifiques relatives à la location de voitures, et font le lien entre la couche contrôleur et la couche mapping.

Couche Application : cette couche contient la partie fonctionnelle de l'application. Elle s'appuie sur les objets métier pour réaliser les actions sollicitées par l'utilisateur par l'intermédiaire de la couche présentation. Elle est en charge de vérifier la validité des requêtes de la couche présentation. Il existe un contrôleur par fonctionnalité de l'application. Le schéma utilisé est le MVC (Modèle Vue Contrôleur), mis en œuvre en utilisant Spring MVC. Plus d'informations sont disponibles dans le paragraphe suivant.

***Couche Présentation* :** cette couche représente les interfaces qui permettent de présenter les contenus générés par la couche présentation, grâce à des vues dynamiques (JSP). Elle se charge d'afficher des contenus à l'utilisateur et de lui offrir des interfaces avec la couche contrôleur pour interagir avec l'application.

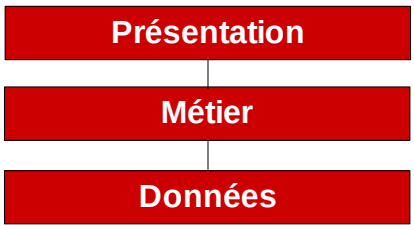
L'architecture MVC

Les couches présentation et application s'appuient sur l'architecture « Modèle-Vue-Contrôleur » qui permet de séparer le fonctionnel de l'interface. Cette architecture est réalisée par une conjonction d'un contrôleur et d'un nombre quelconque de pages JSP (les vues) qui offrent le rendu à l'utilisateur. Les données calculées par le contrôleur et fournies aux vues pour être affichées sont les modèles. Le contrôleur s'appuie sur les couches inférieures pour obtenir ces données.



Plus précisément, l'utilisateur sollicite une action par l'intermédiaire d'une Vue. Cette action est transmise au contrôleur, qui en vérifie la validité et s'appuie sur la couche métier pour l'effectuer. Enfin le contrôleur renvoi sur la vue correspondant à la demande de l'utilisateur.

B. Architecture de l'application mobile



Sur les applications PDA, la stratégie adoptée est d'utiliser un modèle en couches similaire. Cependant, la taille de l'application n'impose pas une séparation aussi fine que l'application nationale. Nous appliquons donc un modèle à 3 couches, organisé comme suit :

Couche Présentation : cette couche effectue la présentation des données à l'utilisateur par l'intermédiaire de formulaires et de listes de données interactives. L'utilisateur accède à l'application par cet unique moyen.

Couche Métier *:* de même que pour l'AN, cette couche contient d'une part les objets métier, correspondant aux données métier à manipuler dans l'application, d'autre part, les services métiers reliés à ces objets (création, recherche, suppression, modification et autres services spécifiques). A l'image de l'AN, cette couche s'occupe de garantir que le métier de l'application est respecté, que les règles liées aux données sont toujours suivies. Dans cette couche, on retrouve également les méthodes d'accès à la base de données (présentes dans la couche Données dans l'application nationale).

Couche Données *:* cette couche contient les données physiques stockées dans la base de données mobile Oracle Lite.

Synchronisation des données

La problématique de la synchronisation prend une part importante dans la conception du système et notamment de la base de données et des parties applicatives basses. Pour rappel ces synchronisations interviennent lorsqu'un utilisateur de PDA (commercial ou responsable de parc d'agence) veut faire coïncider les données qu'il possède sur son unité mobile avec celles stockées en base de données centrale. Le cas d'utilisation type se présente ainsi :

- Au démarrage du PDA, le responsable de parc de véhicules se synchronise avec la base Oracle pour avoir des données à jour ;
- Lors de ses déplacements dans le parc de véhicules, il peut consulter, modifier, insérer ou supprimer des données chargées dans la base Oracle Lite sans connexion réseau ;
- En fin de journée, avant d'éteindre le PDA, il se synchronise avec le point d'accès Wifi de l'agence pour soumettre ses mises à jour et récupérer celles des autres ; Ces

synchronisations peuvent être plus fréquentes dans le cas de procédures de mise à disposition ou de retour de véhicules.

Problématique : quels conflits peuvent être engendrés par ces synchronisations ?

Du fait que les responsables de parc de véhicules utilisent le PDA en mode déconnecté, il n'existe pas de conflit de modification lors de l'utilisation du PDA. Ces conflits peuvent par contre apparaître lors de la synchronisation avec la base de données centrale. En cas de conflits pendant la synchronisation entre les données présentes sur le PDA et celles présentes dans la base centrale, la priorité est donnée aux données contenues dans la base de données centrale. Cette politique de priorités est configurée à l'aide de *Mobile Database Workbench*, le composant de la suite *Oracle Lite* permettant de construire les éléments de synchronisation.

Ce problème de conflit d'identifiants et d'écrasement des données est critique et peut apparaître fréquemment dans un contexte d'utilisation quotidienne du système CAR. Nous avons donc décidé qu'une plage d'identifiants serait attribuée au PDA, de manière à éviter tout conflit. La solution optimale (nettement plus complexe à mettre en œuvre) serait d'associer un second identifiant, propre à chaque PDA, au premier identifiant utilisé dans l'ensemble de l'application. Une seconde solution serait sinon de concaténer à l'identifiant incrémenté un identifiant unique de machine. Etant donné que chaque machine possède sa propre et unique base de données embarquée, le système serait à l'abri de tout conflit. La gestion des identifiants des PDA devrait dans ce cas être centralisée au niveau de l'application *Mobile Server*.

4. Structure des données globales, des fichiers et des bases de données

Les données sont centralisées dans une unique base de données Oracle 10g installée sur le serveur de base de données Oracle XE (les scripts de création des tables et des tuples de la base de données sont livrés avec les applications AN et AM). Ces données sont utilisées par l'application AN, mais également par l'application AM qui les récupère ou les modifie lors de la synchronisation avec le serveur (l'application AM récupère donc une copie de certaines tables de la BD).

5. Stratégie de traitement des erreurs et des exceptions

Dans ce paragraphe sont exposées les différentes stratégies mises en place pour gérer les erreurs, i.e. empêcher l'application de s'effondrer sur n'importe quelle erreur et rendre les erreurs compréhensibles pour l'utilisateur.

Stratégie d'exceptions par couches

Afin de permettre une gestion localisée des exceptions, il a été décidé que chaque couche prenne en compte. Ainsi, la couche mapping est en charge des exceptions lors des accès à la base. La couche contrôleur est en charge des exceptions dues au mauvais formatage des données et de transmettre des messages d'erreur textuels à la couche présentation pour l'utilisateur.
De cette façon, chaque couche agit de façon autonome et la réutilisation des composants est possible.

Vérification des données saisies

Une des principales sources d'erreurs dans les applications est due à des saisies erronées de la part de l'utilisateur. Ces erreurs peuvent être de plusieurs niveaux : erreurs de formatage, non suivi des règles métier, contraintes de base de données. Elles doivent être interceptées le plus tôt possible dans le cheminement de l'architecture afin d'une part que la logique soit respectée (chaque couche a un rôle particulier, il doit être rempli sans supposer que d'autres couches le feront à sa place), et d'autre part que le nombre de traitements ne soit pas excessivement grand. Les différents niveaux de vérifications de données sont les suivants :

- *couche contrôleur* : les formulaires sont validés. On peut alors vérifier que les données ne sont pas absentes, ou mal formatées. Ce sont des vérifications sans logique métier, simplement applicable à tout type de formulaire.

- *couche métier* : une fois les valeurs saisies validées, la prochaine étape est de vérifier l'adéquation métier de ces données. Par exemple, une voiture apparaissant dans des mouvements

ne peu pas être supprimée mais simplement retirée de la circulation. Ce genre de règles métiers doivent être vérifiées par cette couche.

- *couche mapping* : cette couche se charge de vérifier toutes les règles de gestion implémentées dans la base de données. Cela permet de passer au crible toutes les erreurs de saisies qui seraient soit mal gérées par les couches supérieurs, soit appartenant à des cas très spécifiques d'erreurs. Notamment, la gestion des transactions est primordiale. Hibernate se charge d'une grande partie de ces problématiques, il convient toutefois de définir les contraintes sur la base de données avec prudence et précision.

Sécurisation des accès aux données

Les données sont accessibles au travers de différents écrans, offrant les différentes fonctionnalités de l'application. Les utilisateurs de la partie administration de l'AN doivent s'identifier pour y accéder. De plus un rôle est définit pour chacun d'eux. Pour chaque rôle, les fonctionnalités accessibles sont définies dans la base de données. L'application est développée pour prendre en compte les fonctionnalités accessibles à l'utilisateur courant. Pour cela elle n'affiche que les écrans autorisés à l'utilisateur. De plus certaines portions de formulaires ou de menus peuvent être masquées.

Pour empêcher l'accès aux écrans non autorisés, en plus de les masquer, un contrôleur spécifique, appelé intercepteur vérifie qu'un utilisateur peut bien y accéder pour chaque requête.
Les droits d'accès sont éditables par l'administrateur qui dispose de tous les droits sur tous les écrans de l'application.

6. Justification des choix d'architecture, des composants et du langage

Les choix de l'architecture sont en partie imposés par le maître d'ouvrage. Celui-ci préconise l'utilisation d'Oracle comme SGBDR, ainsi que le langage JAVA autour des frameworks J2EE pour l'application nationale. D'autre part, le modèle des PDA est aussi imposé par le maître d'ouvrage. Il s'agit de HP iPAQ embarquant l'OS Windows Mobile 5.0.

L'application nationale a été développée selon une architecture 5 couches. Ce choix permet d'accroître l'indépendance et la réutilisation des composants. Ainsi pour chacune des couches, il est possible de changer le choix technique ou l'implémentation de façon transparente pour les autres couches. D'autre part le découpage en couche permet une plus grande réutilisation synonyme d'économie à terme.

En contrepartie, il faut être conscient qu'une telle architecture réduit les performances générales du système du fait du cheminement indirect par les différentes couches. De plus, le primo investissement est plus lourd en terme de développement. Les économies se réalisent sur les extensions et la maintenance.

L'architecture des applications sur PDA est une architecture 3 tiers. La synchronisation entre les PDA et la base de données centrale s'appuie sur les composants fournis par Oracle.

V. CONCEPTION DETAILLEE DES COMPOSANTS

1. Système d'Information

A. Règles de gestion

Les règles de gestion décrivent la nature des relations entre les entités d'un système d'information. L'ensemble de ces règles permet de définir un système correspondant à une problématique métier précisément adaptée aux besoins du client. Ces règles de gestion sont utilisées directement dans le modèle métier : chaque règle correspond à une relation entre 2 (ou plusieurs) entités. Les entités métiers sont indiquées en gras.

- R1 : une **catégorie de véhicules** rassemble un ou plusieurs **modèles de véhicules**. Un **modèle** appartient à une seule **catégorie**.
- R2 : un **modèle** correspond à un ou plusieurs **tarifs de location**, en fonction de différents paramètres. Un **tarif de location** peut correspondre à plusieurs **véhicules**.
- R3 : un **équipement spécial** peut être monté sur un véhicule.
- R4 : un **véhicule** appartient à un **modèle**. Un **modèle** rassemble un ou plusieurs **véhicules**.
- R5 : un **client** peut avoir un responsable, **employé** de l'agence.
- R6 : un **client** peut bénéficier d'un **contrat partenaire**.
- R7 : un **contrat d'abonnement** est signé avec un **client** de type *Particulier*. Il peut inclure une remise.
- R8 : un **contrat d'abonnement** nécessite une **carte d'abonnement**. Cette carte peut correspondre à plusieurs contrats.
- R9 : un **client professionnel** travaille pour une entreprise.
- R10 : un **client** peut effectuer une ou plusieurs **réservations.** Une **réservation** correspond à un seul **client**. Une réservation effective est une location.
- R11 : une **réservation** est faite pour une **catégorie de véhicule** donnée.
- R12 : une **réservation** peut contenir des **équipements spéciaux**.

- R13 : une **réservation** est payée avec un **mode de règlement** particulier (chèque, CB, etc.).

- R14 : un **employé** de type *agent commercial* met à disposition un **véhicule** à une date donnée pour une **réservation** donnée. Un **employé** de type *agent commercial* effectue le retour d'un **véhicule** à une date donnée pour une **réservation** donnée.

- R15 : un **employé** de type *responsable d'agence* est responsable d'une **agence**.

- R16 : un **employé** est soit rattaché à une **agence**, soit directement au siège.

- R17 : un **véhicule** est présent dans une agence à un moment donné. Une **agence** peut contenir plusieurs **véhicules** dans son parc de véhicules.

- R18 : des **inspections** sont effectuées sur tous les **véhicules**. Chaque **véhicule** est inspecté à une date donnée. Une **inspection** est toujours effectuée lors de la mise à disposition d'un **véhicule**, et lors de son retour.

- R19 : une **inspection** peut ne donner lieu à aucun **constat**, ou donner lieu à plusieurs.

- R20 : un **constat** peut avoir pour objet un ou plusieurs **types de problèmes**.

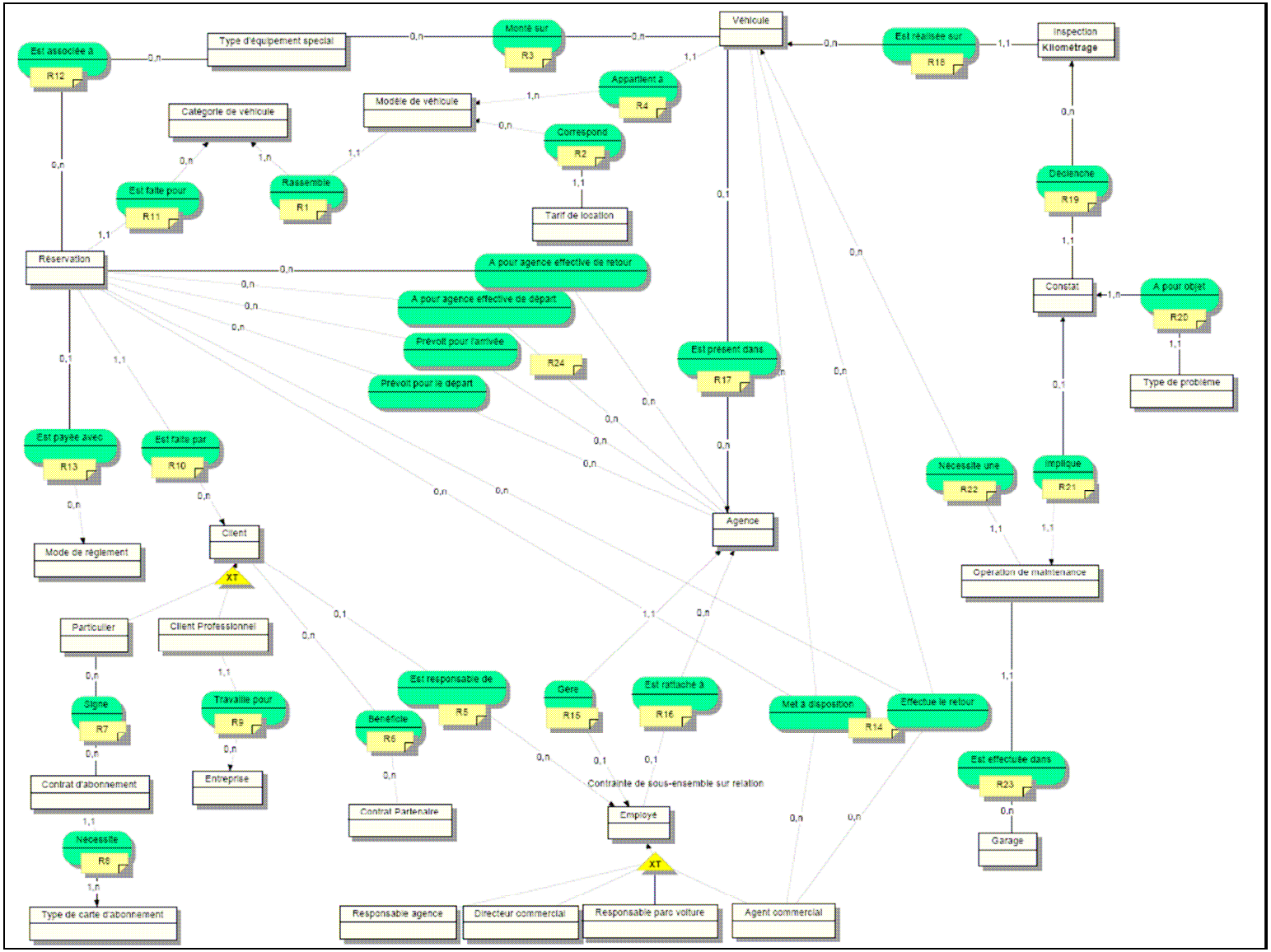
- R21 : un **constat** nécessite une **opération de maintenance** de type corrective.

- R22 : une **opération de maintenance** est effectuée régulièrement de manière préventive sur un **véhicule**, sans que cela implique un constat.

- R23 : une **opération de maintenance** est effectuée dans un **garage**.

- R24 : une **réservation** a une **agence** de départ et une **agence** de retour prévue. Lorsque la **réservation** devient une location**,** la location a une **agence** de départ effective, et lorsque le retour est effectué, une **agence** de retour effective.

B. Modèle Métier

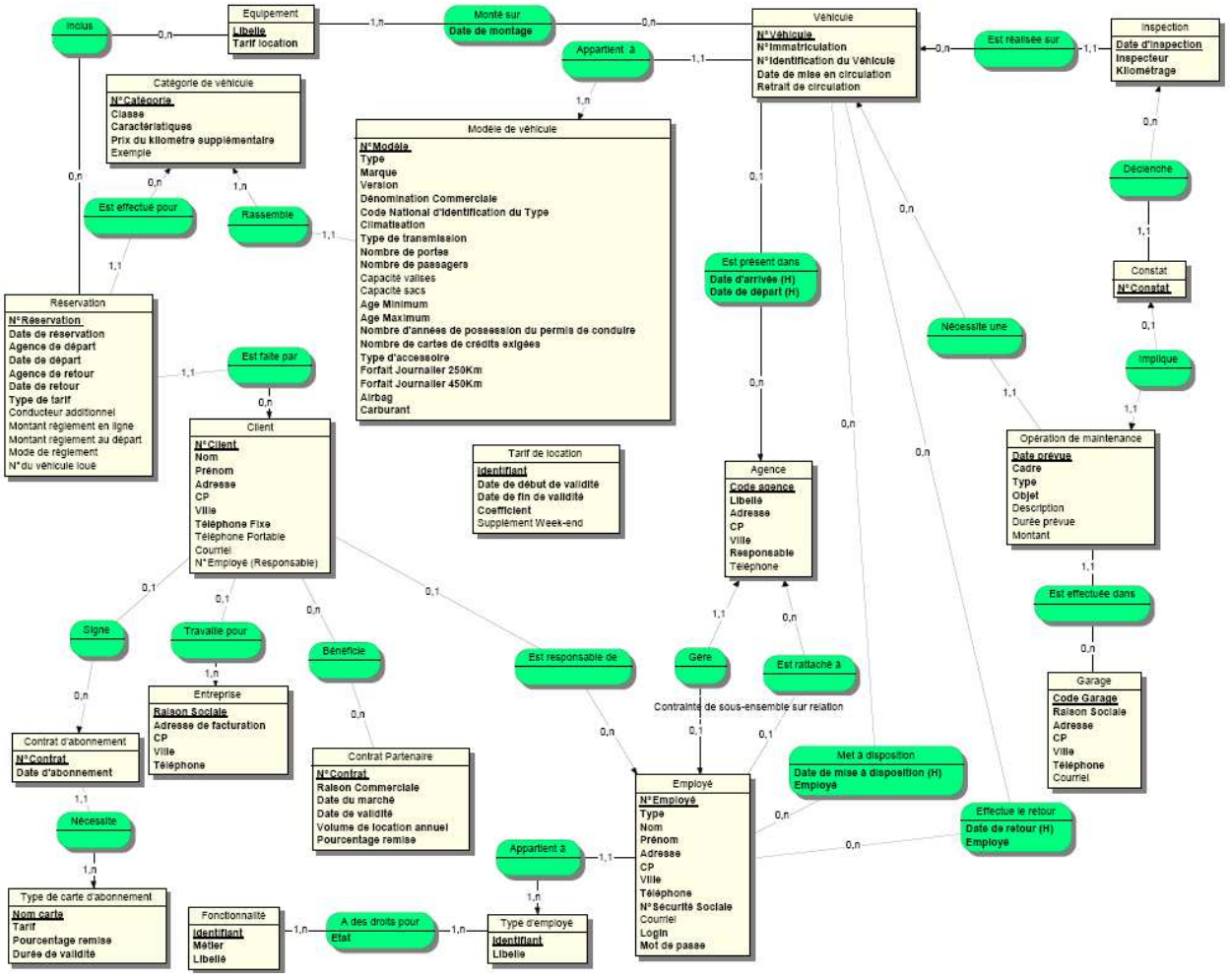

Véhicule
Est associée à
R12
Type d'équipement spécial
Monté sur
R3
Appartient à
R4
Modèle de véhicule
Catégorie de véhicule
Correspond
R2
Tarif de location
Est faite pour
R11
Rassemble
R1
Est réalisée sur
R18
Inspection
Kilométrage
Déclenche
R19
Constat
A pour objet
R20
Type de problème
Réservation
A pour agence effective de retour
A pour agence effective de départ
Prévoit pour l'arrivée
Prévoit pour le départ
R24
Est présent dans
R17
Agence
Est payée avec
R13
Est faite par
R10
Mode de règlement
Client
XT
Particulier
Client Professionnel
Signe
R7
Contrat d'abonnement
Nécessite
R8
Type de carte d'abonnement
Travaille pour
R9
Entreprise
Est responsable de
R5
Bénéficie
R6
Contrat Partenaire
Gère
R15
Est rattaché à
R16
Contrainte de sous-ensemble sur relation
Employé
Responsable agence
Directeur commercial
Responsable parc voiture
Agent commercial
Met à disposition
R14
Effectue le retour
Nécessite une
R22
Implique
R21
Opération de maintenance
Est effectuée dans
R23
Garage

2. Base de données

A. Modèle Conceptuel de Données

La représentation du système d'information a été réalisé à l'aide du modèle métier et de la méthode MERISE. Cette méthode nous a permis de réaliser le Modèle Conceptuel de Données ou MCD, qui fait le lien entre le modèle métier et le modèle logique. Le MCD est un modèle dit « entités-relations » qui modélise l'ensemble des règles de gestion et des contraintes du système. Il correspond globalement au modèle métier, en détaillant le contenu des entités métiers (attributs).

La construction du MCD a été réalisé à partir des spécifications du cahier des charges, du modèle métier et des réunions avec le maître d'ouvrage. Celui-ci a été généré grâce au logiciel ***WinDesign***, qui permet de transformer le MCD en MLR et de générer le script de la base de données correspondant.



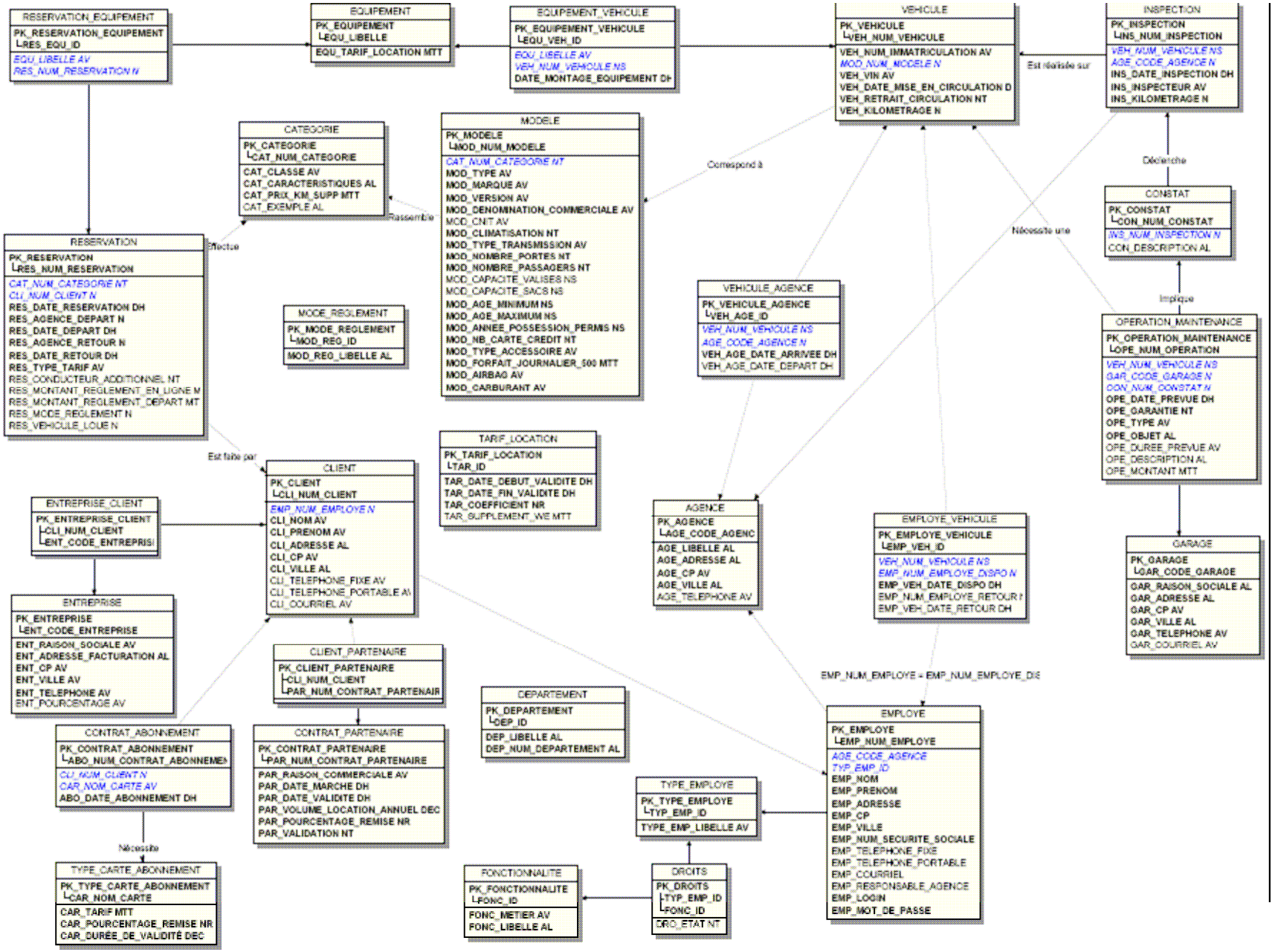
B. Modèle Logique Relationnel

Le MLR est le modèle logique correspondant au MCD. Il représente les tables et les colonnes de la base de données, le schéma de la base de données relationnelles. Chaque entité du MCD correspond à une table, de même que les associations 0..n ←→ 0..n et les associations contenant des attributs.

Dans le schéma du MLR ci-dessous (généré par WinDesign), les attributs des tables contiennent un préfixe de 3 à 4 lettres correspondant à la table dont ils sont issus (voir guide ACAI). Par exemple, les attributs de la table EMPLOYE ont pour préfixe « EMP_ ». Cette technique permet de repérer rapidement les clés étrangères dans une table (indiqué dans le schéma ci-dessous en bleu clair en italique). Les attributs obligatoires sont indiqués en gras. Les clés primaires sont regroupés dans des libellés contenant PK en préfixe (FK pour les clés étrangères, non affichées

Le paramétrage consiste aussi à définir le type de données pour chaque champ. Les types de chaque champs sont définis à partir des informations contenues dans le cahier des charges. On utilisera ainsi des NUMBER pour les entiers (la taille du NUMBER variant en fonction de la précision attendue, ou de la quantité d'enregistrements attendue dans la table) , des CHAR(32) ou des VARCHAR2(255) pour les textes, des DATE pour les dates, etc.

L'historisation consiste à garder une traçabilité de l'état d'une entité ou d'une relation à travers les modifications et suppressions intervenues dans le temps. Elle est représentée par le formalisme MERISE dans le MCD (voir précédemment pour les relations entre les agences et les véhicules par exemple). L'historisation des dates d'arrivée et de départ des véhicules dans les agences, ainsi que des dates de mise à disposition et de retour des véhicules a été traduite par 2 tables, EMPLOYE_VEHICULE (pour la mise à disposition et le retour) et VEHICULE_AGENCE (pour les dates de départ et de retour).

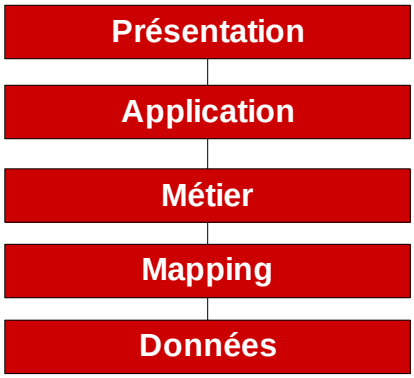

RESERVATION_EQUIPEMENT
PK_RESERVATION_EQUIPEMENT
RES_EQU_ID
EQU_LIBELLE AV
RES_NUM_RESERVATION N
EQUIPEMENT
PK_EQUIPEMENT
EQU_LIBELLE
EQU_TARIF_LOCATION MTT
EQUIPEMENT_VEHICULE
PK_EQUIPEMENT_VEHICULE
EQU_VEH_ID
EQU_LIBELLE AV
VEH_NUM_VEHICULE NS
DATE_MONTAGE_EQUIPEMENT DH
VEHICULE
PK_VEHICULE
VEH_NUM_VEHICULE
VEH_NUM_IMMATRICULATION AV
MOD_NUM_MODELE N
VEH_VIN AV
VEH_DATE_MISE_EN_CIRCULATION D
VEH_RETRAIT_CIRCULATION NT
VEH_KILOMETRAGE N
Est réalisée sur
INSPECTION
PK_INSPECTION
INS_NUM_INSPECTION
VEH_NUM_VEHICULE NS
AGE_CODE_AGENCE N
INS_DATE_INSPECTION DH
INS_INSPECTEUR AV
INS_KILOMETRAGE N
Déclenche
CONSTAT
PK_CONSTAT
CON_NUM_CONSTAT
INS_NUM_INSPECTION N
CON_DESCRIPTION AL
Implique
OPERATION_MAINTENANCE
PK_OPERATION_MAINTENANCE
OPE_NUM_OPERATION
VEH_NUM_VEHICULE NS
GAR_CODE_GARAGE N
CON_NUM_CONSTAT N
OPE_DATE_PREVUE DH
OPE_GARANTIE NT
OPE_TYPE AV
OPE_OBJET AL
OPE_DUREE_PREVUE AV
OPE_DESCRIPTION AL
OPE_MONTANT MTT
GARAGE
PK_GARAGE
GAR_CODE_GARAGE
GAR_RAISON_SOCIALE AL
GAR_ADRESSE AL
GAR_CP AV
GAR_VILLE AL
GAR_TELEPHONE AV
GAR_COURRIEL AV
Nécessite une
CATEGORIE
PK_CATEGORIE
CAT_NUM_CATEGORIE
CAT_CLASSE AV
CAT_CARACTERISTIQUES AL
CAT_PRIX_KM_SUPP MTT
CAT_EXEMPLE AL
Classifie
MODELE
PK_MODELE
MOD_NUM_MODELE
CAT_NUM_CATEGORIE NT
MOD_TYPE AV
MOD_MARQUE AV
MOD_VERSION AV
MOD_DENOMINATION_COMMERCIALE AV
MOD_CNIT AV
MOD_CLIMATISATION NT
MOD_TYPE_TRANSMISSION AV
MOD_NOMBRE_PORTES NT
MOD_NOMBRE_PASSAGERS NT
MOD_CAPACITE_VALISES NS
MOD_CAPACITE_SACS NS
MOD_AGE_MINIMUM NS
MOD_AGE_MAXIMUM NS
MOD_ANNEE_POSSESSION_PERMIS NS
MOD_NB_CARTE_CREDIT NT
MOD_TYPE_ACCESSOIRE AV
MOD_FORFAIT_JOURNALIER_500 MTT
MOD_AIRBAG AV
MOD_CARBURANT AV
Correspond à
RESERVATION
PK_RESERVATION
RES_NUM_RESERVATION
CAT_NUM_CATEGORIE NT
CLI_NUM_CLIENT N
RES_DATE_RESERVATION DH
RES_AGENCE_DEPART N
RES_DATE_DEPART DH
RES_AGENCE_RETOUR N
RES_DATE_RETOUR DH
RES_TYPE_TARIF AV
RES_CONDUCTEUR_ADDITIONNEL NT
RES_MONTANT_REGLEMENT_EN_LIGNE M
RES_MONTANT_REGLEMENT_DEPART MT
RES_MODE_REGLEMENT N
RES_VEHICULE_LOUE N
Effectue
Est faite par
MODE_REGLEMENT
PK_MODE_REGLEMENT
MOD_REG_ID
MOD_REG_LIBELLE AL
VEHICULE_AGENCE
PK_VEHICULE_AGENCE
VEH_AGE_ID
VEH_NUM_VEHICULE NS
AGE_CODE_AGENCE N
VEH_AGE_DATE_ARRIVEE DH
VEH_AGE_DATE_DEPART DH
TARIF_LOCATION
PK_TARIF_LOCATION
TAR_ID
TAR_DATE_DEBUT_VALIDITE DH
TAR_DATE_FIN_VALIDITE DH
TAR_COEFFICIENT NR
TAR_SUPPLEMENT_WE MTT
CLIENT
PK_CLIENT
CLI_NUM_CLIENT
EMP_NUM_EMPLOYE N
CLI_NOM AV
CLI_PRENOM AV
CLI_ADRESSE AL
CLI_CP AV
CLI_VILLE AL
CLI_TELEPHONE_FIXE AV
CLI_TELEPHONE_PORTABLE AV
CLI_COURRIEL AV
AGENCE
PK_AGENCE
AGE_CODE_AGENC
AGE_LIBELLE AL
AGE_ADRESSE AL
AGE_CP AV
AGE_VILLE AL
AGE_TELEPHONE AV
EMPLOYE_VEHICULE
PK_EMPLOYE_VEHICULE
EMP_VEH_ID
VEH_NUM_VEHICULE NS
EMP_NUM_EMPLOYE_DISPO N
EMP_VEH_DATE_DISPO DH
EMP_NUM_EMPLOYE_RETOUR
EMP_VEH_DATE_RETOUR DH
ENTREPRISE_CLIENT
PK_ENTREPRISE_CLIENT
CLI_NUM_CLIENT
ENT_CODE_ENTREPRIS
ENTREPRISE
PK_ENTREPRISE
ENT_CODE_ENTREPRISE
ENT_RAISON_SOCIALE AV
ENT_ADRESSE_FACTURATION AL
ENT_CP AV
ENT_VILLE AV
ENT_TELEPHONE AV
ENT_POURCENTAGE AV
CLIENT_PARTENAIRE
PK_CLIENT_PARTENAIRE
CLI_NUM_CLIENT
PAR_NUM_CONTRAT_PARTENAIR
DEPARTEMENT
PK_DEPARTEMENT
DEP_ID
DEP_LIBELLE AL
DEP_NUM_DEPARTEMENT AL
EMP_NUM_EMPLOYE = EMP_NUM_EMPLOYE_DIS
EMPLOYE
PK_EMPLOYE
EMP_NUM_EMPLOYE
AGE_CODE_AGENCE
TYP_EMP_ID
EMP_NOM
EMP_PRENOM
EMP_ADRESSE
EMP_CP
EMP_VILLE
EMP_NUM_SECURITE_SOCIALE
EMP_TELEPHONE_FIXE
EMP_TELEPHONE_PORTABLE
EMP_COURRIEL
EMP_RESPONSABLE_AGENCE
EMP_LOGIN
EMP_MOT_DE_PASSE
CONTRAT_ABONNEMENT
PK_CONTRAT_ABONNEMENT
ABO_NUM_CONTRAT_ABONNEMEN
CLI_NUM_CLIENT N
CAR_NOM_CARTE AV
ABO_DATE_ABONNEMENT DH
Nécessite
CONTRAT_PARTENAIRE
PK_CONTRAT_PARTENAIRE
PAR_NUM_CONTRAT_PARTENAIRE
PAR_RAISON_COMMERCIALE AV
PAR_DATE_MARCHE DH
PAR_DATE_VALIDITE DH
PAR_VOLUME_LOCATION_ANNUEL DEC
PAR_POURCENTAGE_REMISE NR
PAR_VALIDATION NT
TYPE_EMPLOYE
PK_TYPE_EMPLOYE
TYP_EMP_ID
TYPE_EMP_LIBELLE AV
TYPE_CARTE_ABONNEMENT
PK_TYPE_CARTE_ABONNEMENT
CAR_NOM_CARTE
CAR_TARIF MTT
CAR_POURCENTAGE_REMISE NR
CAR_DUREE_DE_VALIDITE DEC
FONCTIONNALITE
PK_FONCTIONNALITE
FONC_ID
FONC_METIER AV
FONC_LIBELLE AL
DROITS
PK_DROITS
TYP_EMP_ID
FONC_ID
DRO_ETAT NT

3. Application nationale (AN)

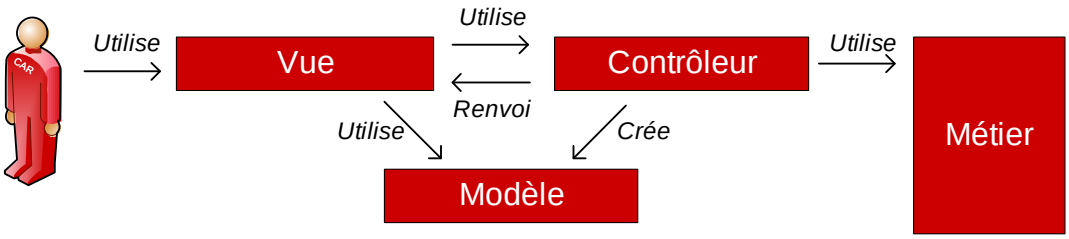
Architecture détaillée du composant

L'application nationale se divise en deux parties : une partie destinée aux clients, permettant principalement d'effectuer la réservation d'un véhicule en ligne ; et une partie administration, destinée aux employés et offrant toutes les fonctionnalités de gestion définies dans le cahier des charges.

Le choix de l'architecture à 5 couches a été effectué. Nous allons dans les prochains paragraphes détailler le fonctionnement de chaque couche. Voici un schéma récapitulatif :



Les couches Présentation et Contrôleur ont été développées en utilisant le paradigme MVC :



Pour mettre en œuvre cette architecture nous avons utilisé :

- Oracle 10g XE pour la couche Données ;
- Apache Commons DBCP pour le pool de connections ;
- Hibernate pour le mapping de la base de données ;
- Spring IOC pour les liens entre les différentes couches ;
- Spring MVC pour la mise en place du MVC ;
- JSTL (JavaServer Pages Standard Tag Library) pour l'affichage des données dans les JSP.

Spring permet de réduire la dépendance entre les différentes couches et diminue la quantité de code nécessaire par l'intermédiaire de fichiers de configuration.

Spring IOC

Spring IOC permet de réduire la dépendance entre les couches de l'application. Il s'agit en fait de séparer l'appel à la couche inférieure de son implémentation. L'inversion de contrôle permet de mettre en place cette séparation. Les éléments des couches « Mapping » et « Métiers » sont définis par des interfaces puis implémentés dans des classes distinctes. La couche supérieure ne connaît que l'existence des interfaces dont elle utilise les méthodes prédéfinies. Ainsi le changement d'implémentation d'une couche n'affecte pas la couche supérieure.

L'instanciation des objets de chaque couche est prise en charge par Spring. Un fichier de configuration XML définit les dépendances effectives entre les couches. Grâce à ce fichier, Spring peut savoir quels objets instancier et comment les mettre en relation.

Hibernate

Les DAO vont utiliser des primitives fournies par Hibernate pour faire appel à la base de données. Notamment, le langage HQL doit être utilisé pour créer des requêtes. Le framework Spring fournit une encapsulation permettant de faire appel aux méthodes Hibernate plus simplement. En effet il permet de mettre en œuvre l'inversion de contrôle entre les DAO, Hibernate et le driver JDBC. Pour cela il faut utiliser un Pool de connections pour simplifier l'accès à la base de données (ici, Apache Commons DBCP).

Spring MVC

Spring MVC permet de mettre en œuvre facilement le paradigme MVC en limitant les dépendances entres Modèles, Vues et Contrôleurs. Spring offre des types de base pour implémenter les contrôleurs et les modèles. De plus, il permet de mettre en œuvre un contrôleur central qui se charge de dispatcher les requêtes de la couche Présentation sur le contrôleur approprié. Enfin les opérations dites « génériques » inhérentes au modèle MVC sont réalisées par le framework.

Couche Physique

La couche physique nécessite Oracle 10 XE pour sa mise en place. Elle a été modélisée par un Modèle Métier puis par un Modèle Logique Relationnelle en utilisant WinDesign. Ces diagrammes sont présentés plus haut. Le script de création de la base est produit par WinDesign. Il doit ensuite être exécuté dans la base de données. Un jeu de données de test est aussi mis en place sous forme d'un script chargé de remplir la base avec des données cohérentes pour tester l'application.

Lors de la phase de développement, la base de données est située sur une machine distante, chaque développeur y accède par Internet. Cela implique des temps de latence importants entre chaque traitement et accès à la base de données, et donc un ralentissement global de l'application. La communication avec la couche physique se fait au moyen d'un driver JDBC fournit par Oracle.

Couche Mapping

Cette couche est composée de trois catégories de fichiers : les fichiers de mapping Hibernate, les objets métiers et les DAO (Data Access Object). Nous allons présenter les trois types de fichiers et expliquer comment nous devons les gérer vis-à-vis des services attendus pour cette couche. Les fichiers de mapping et les objets métiers sont générés automatiquement par le plugin Hibernate que nous avons installé dans notre environnement de développement.

A. Les fichiers de mapping

Ces fichiers contiennent des informations XML permettant à Hibernate de faire la liaison entre un objet métier et son équivalent dans la base de données. Par exemple, le fichier indique à quelle colonne de table correspond telle propriété d'une classe mappée.
Il convient de paramétrer Hibernate pour qu'il prenne en charge l'incrémentation automatique des identifiants des tables.

B. Les objets métiers de mapping

Ces objets sont des classes java, totalement calquées sur les objets présents en base de données. Ils sont couramment appelés les POJOs (Plain Old Java Objects). Ils sont générés automatiquement à partir des fichiers mapping.

C. ***Les DAO***

Ces objets sont chargés de faire le lien avec la couche métier en lui offrant toutes les méthodes nécessaires pour utiliser les données dans la couche physique. Ils masquent la technologie utilisée (ici Hibernate) à la couche métier. Ainsi un changement de technologie pour la couche physique n'entrainerait pas de modification des couches autre que celle du Mapping. Il existe un DAO par type de données de la base de données. Ces DAO permettent :

- L'accès à l'ensemble des données d'un type ;
- L'accès à un seul élément d'un type par un critère définit ;
- La modification d'un élément ;
- L'insertion d'un nouvel élément ;
- La suppression d'un élément.

Pour respecter les contraintes de Spring MVC, chaque DAO est définit par l'intermédiaire d'une interface qui définit l'ensemble de ses méthodes. L'implémentation fait appel à Hibernate pour accéder à la couche physique. Les requêtes sont écrites en HQL (Hibernate Query Language).

Couche Métier

La couche métier a comme rôle primordial de proposer un ensemble de services génériques sur les données, tout en faisant respecter les règles métier et les contraintes inhérentes au domaine d'activité.

Il existe un objet métier pour chaque fonctionnalité de l'application. Les opérations les plus fréquentes sont les opérations CRUD : Create, Read, Update, Delete (Créer, Lire, Modifier, Supprimer). Ces opérations sont primordiales et doivent être l'ensemble minimal de méthodes fournies par toutes les entités. Elles font simplement le lien entre la couche Mapping et la couche Application.

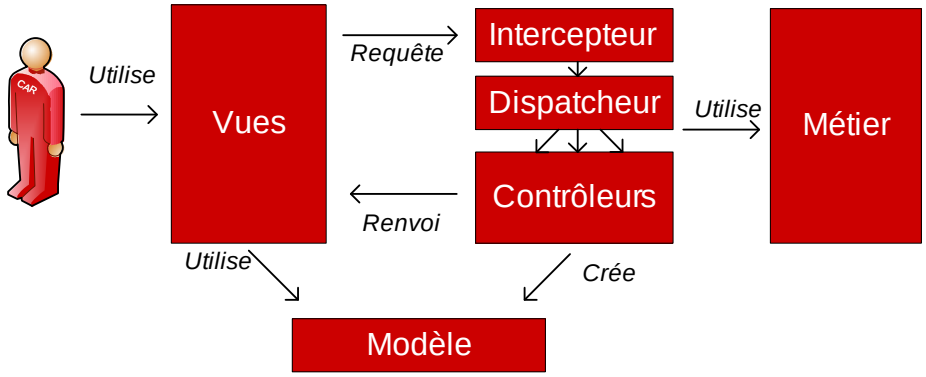
Ces opérations peuvent être bien plus que de simples CRUD, en intégrant par exemple de la logique métier spécifique au domaine de la location de voiture. D'autres opérations spécifiques au métier sont aussi présentes quand cela est nécessaire.

Couche Application

Cette couche a pour rôle d'implémenter les cas d'utilisations qui seront présentés aux futurs bénéficiaires du système. L'architecture de la couche se base sur le modèle MVC expliqué plus tôt. Un contrôleur est mis en place pour chaque fonction de l'application. De plus, un intercepteur se charge de contrôler les accès, et un dispatcheur de renvoyer les requêtes sur le contrôleur approprié. Les contrôleurs sont en charge de :

- vérifier que les données saisies sont correctes
- faire appel à la couche métier pour effectuer les traitements demandés
- placer dans le Model les données nécessaires à la couche présentation
- renvoyer l'utilisateur sur la couche présentation correspondant à sa demande

Ils se basent sur le type *Controller* fournit par Spring MVC.



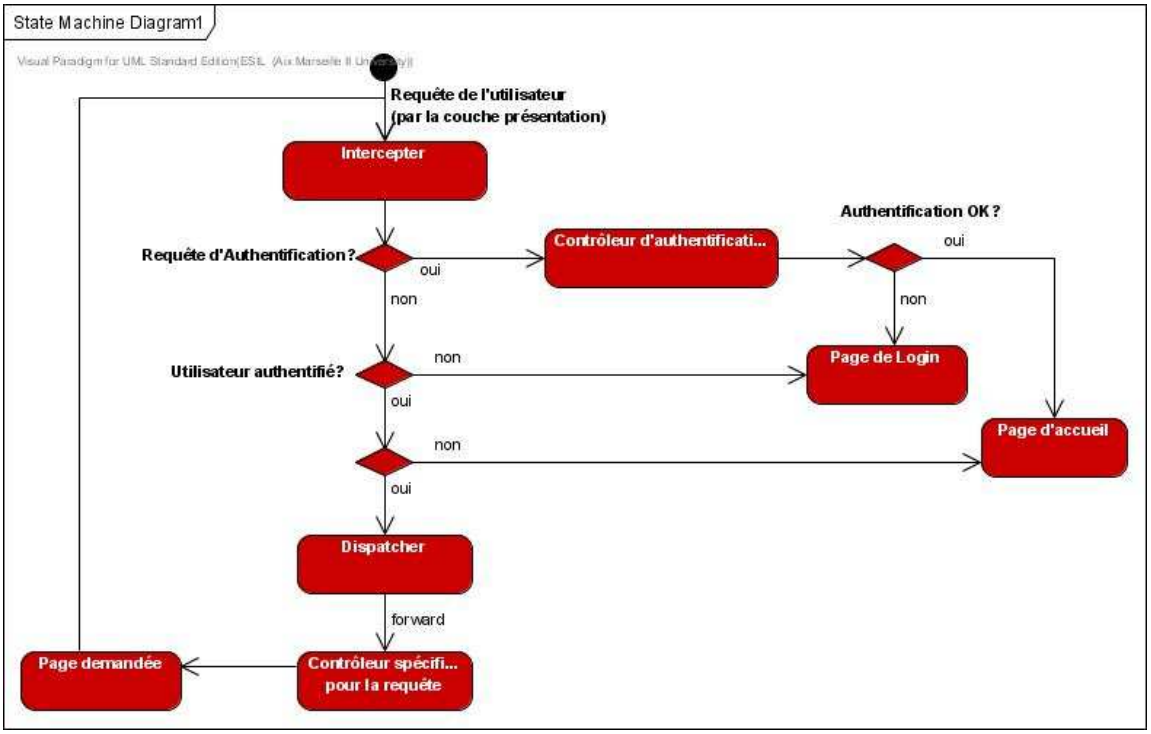
A. Sécurisation par un intercepteur

Comme présenté précédemment, cette couche intègre un Intercepteur chargé de contrôler que l'utilisateur à bien accès aux écrans qu'il demande. Pour ce faire, l'intercepteur empêche l'accès à toutes les pages sauf celle d'identification, tant que l'utilisateur n'est pas authentifié. A l'authentification, le contrôleur chargé de vérifier l'identité de l'utilisateur crée une session. Il place dans cette session l'ensemble des informations nécessaires pour qu'il puisse ensuite naviguer dans les différents écrans : identité, fonction, et droits d'accès associés.

A chaque demande de page par l'utilisateur, l'intercepteur reçoit cette demande et contrôle que l'utilisateur à bien accès à cette page. Si la réponse est positive, il laisse le programme s'exécuter ; sinon, il redirige l'utilisateur sur la page d'accueil en lui indiquant qu'il ne peut accéder à la ressource qu'il a demandé.

B. Dispatcher

Un objet spécifique est chargé de diriger les requêtes sur le contrôleur approprié. Cet objet est un dispatcheur fournit par Spring, et configuré pour diriger chaque requête sur le contrôleur associé.

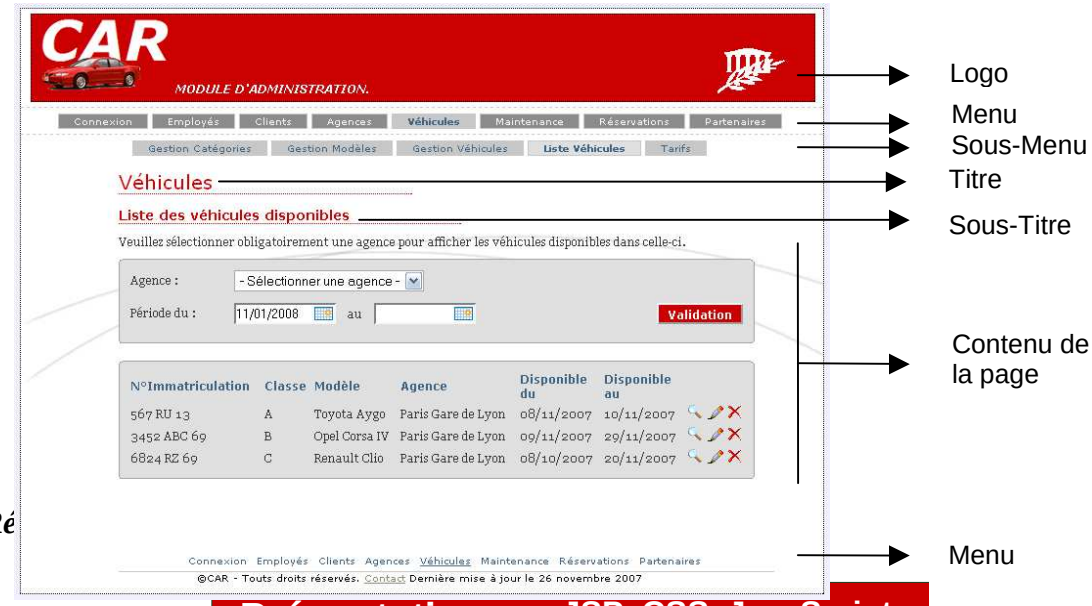


C. Gestion des Erreurs

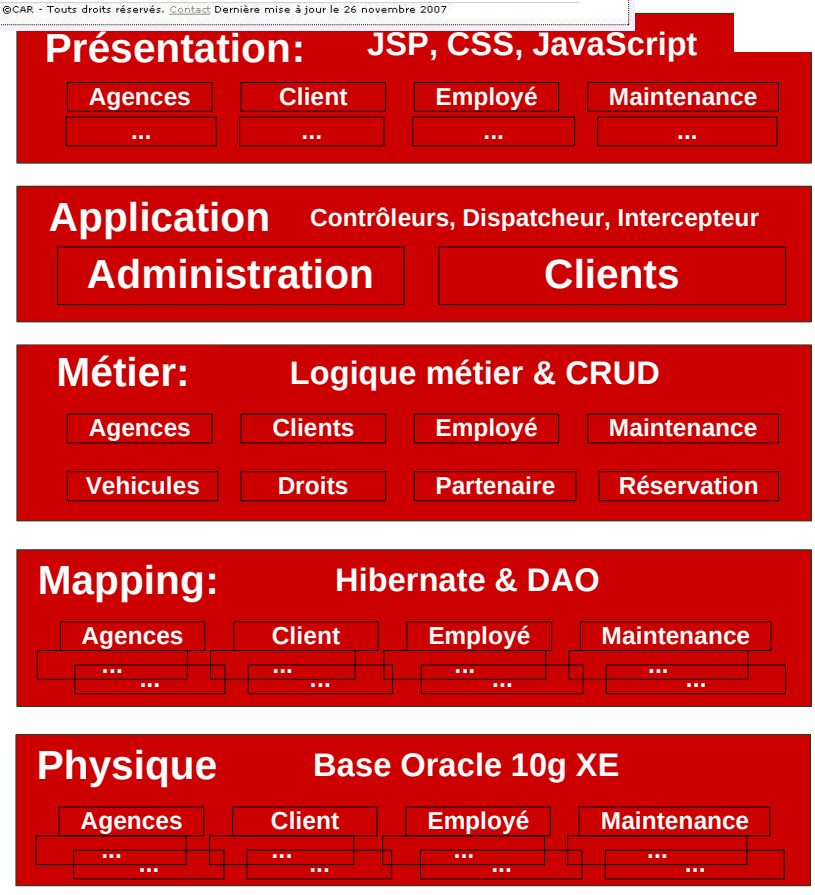
Lorsqu'une erreur est rencontrée par un contrôleur, il place un message d'erreur dans le model. Celui-ci sera ensuite automatiquement affiché à l'utilisateur par la couche présentation.

Couche Présentation

La couche présentation est prise en charge par des pages JSP. Ces pages JSP sont structurées pour standardiser l'apparence rendue : les zones communes sont séparées et incluses par chaque page. L'affichage des données du modèle utilise la JSTL (JavaServer Pages Standard Tag Library) qui offre des balises pour simplifier l'affichage de ces données.



Ré



Répartition HTML/CSS/JavaScript

Il est important de bien faire la distinction, dans toutes les pages web qui sont développées, entre le côté fonctionnel et le côté graphique. L'HTML remplit le côté fonctionnel, i.e. que du contenu, sans se soucier de la forme. L'utilisation des conteneurs de type « div » se prête à cette séparation du contenu, à l'inverse des tableaux. En effet, un tableau utilisé pour la disposition d'éléments n'a aucune signification fonctionnelle, seulement esthétique. Ce n'est pas le rôle de l'HTML. Les tableaux sont utilisés uniquement pour l'affichage de listes de données.

Le CSS remplit le côté graphique : disposition des éléments, couleurs, tailles. La feuille de style est commune à toutes les pages du site, afin de garder une unité et de pouvoir modifier le design du site très rapidement. Le JavaScript n'est utilisé que pour l'aide à la saisie. Il n'intervient pas dans la validation des données saisies : calendrier pour les dates par exemple. En effet l'utilisateur peut choisir de désactiver le JavaScript dans son navigateur. La validation des données est donc entièrement déléguée à la couche contrôleur.

Compatibilité Firefox/Internet Explorer

La problématique de compatibilité entre les deux navigateurs les plus utilisés par les internautes doit être prise en compte très tôt, car ce sont surtout les styles CSS qui doivent être écrits avec ces contraintes. En effet, certains styles ne sont reconnus que par l'un ou l'autre des navigateurs. La feuille de style a été définie dès l'établissement des maquettes des écrans. Elle peut donc être utilisée tout au long du développement pour être testée.

Authentification utilisateur

L'application CAR comporte une interface de gestion destinée à gérer les différents processus de l'entreprise. Cette partie de l'application n'est pas destinée à être accessible par une personne extérieure. L'authentification des employés est donc nécessaire.

Les connexions utilisateurs sont effectuées par identification classique : login et mot de passe. Ces informations sont stockées dans la base de données, leur vérification est donc immédiate lors de l'authentification de l'utilisateur.

Gestion des droits d'utilisateurs

Au sein de la société, il existe plusieurs catégories d'employés qui n'ont pas accès aux mêmes fonctionnalités. Les groupes d'employés considérés sont les suivants :

- Responsable Agence
- Responsable Parc Véhicules
- Agent Commercial
- Directeur Commercial

Il est possible de définir des fonctionnalités au sein de l'application. Celles-ci pourront, pour chaque type d'employé, contenir le type d'autorisation d'accès défini. Par exemple, les agents commerciaux n'auront pas accès aux fonctions liées à la gestion des employés. La solution est donc de stocker une matrice fonction/types d'utilisateurs qui permettra de savoir à un moment

précis quels droits sont accordés à tel type d'utilisateur pour telle fonctionnalité (voir le CDC pour les fonctionnalités et leurs utilisateurs potentiels).

Sessions utilisateurs

L'authentification d'un utilisateur conduit à l'ouverture d'une session qui permettra à celui-ci de naviguer sur tout le site sans devoir s'identifier à chaque page. La technologie employée pour la réalisation du site (JSP) permet de définir une session utilisateur contenant un nombre quelconque de variables de sessions, définies et existantes pendant toute la durée de la session.

Il est important de conserver le login (ou le nom associé) de l'employé connecté, pour des raisons de traçabilité. Les droits alloués à cet utilisateur doivent être conservés. On conserve donc les droits associés au type d'utilisateur auquel appartient l'employé. Lors de la connexion de l'utilisateur, une entité contenant les droits de celui-ci sera donc initialisée pour la session ouverte.

Gestion interne de l'authentification

Il convient de détailler les différents composants qui rentrent en jeu lors d'une authentification utilisateur. Lors de la soumission du formulaire d'identification par l'utilisateur, une première vérification est effectuée, permettant de s'assurer que ni le champ de mot de passe, ni le champ de login ne sont vides. Ensuite, les données sont comparées avec les données stockées en base :

- vérifier que le mot de passe correspond bien à l'utilisateur ;
- dans le cas où le mot de passe est erroné, rediriger vers le formulaire avec une notification d'erreur ;
- dans le cas où le mot de passe est correct, charger les droits de l'utilisateur et rediriger vers la page d'accueil de l'application.

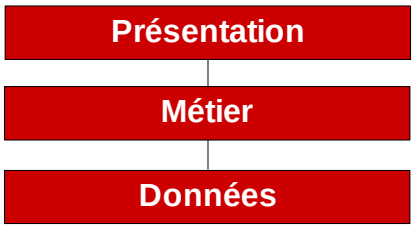
Expiration de session et reconnexion

Toute session de connexion possède un « time out », qui correspond à la durée pendant laquelle une session sans activité reste connectée. Au-delà de ce temps, l'utilisateur doit saisir ses informations de connexion à nouveau.

4. Application mobile (AM)

Architecture détaillée du composant

L'application mobile a été développé selon une architecture 3-couches, séparant la logique métier des données et de leur présentation. Cette architecture est moins évoluée et moins flexible que celle utilisée dans l'application nationale mais est tout à fait adaptée aux plateformes mobiles puisqu'elle concilie évolutivité du code et performances à l'exécution.



Pour mettre en œuvre cette architecture nous avons utilisé :

- *Oracle Lite 10.3* pour le stockage des données sur le PDA.
- *Oracle Database Workbench* et *Mobile Server* pour la synchronisation des données côté serveur.
- *Oracle Device Manager* et *Oracle MSync* pour la synchronisation des données côté unité mobile.
- *MySaifu JVM 0.3* pour l'exécution de l'application sur le PDA.

Synchronisation et stockage des données

La suite Oracle Lite 10 fournit plusieurs outils permettant de configurer et mettre en place une architecture asynchrone permettant de synchroniser les données de plusieurs unités mobiles avec une base centrale. Cette architecture nécessite l'installation et la configuration de plusieurs applications réparties classiquement entre trois machines.

La première machine est le serveur Oracle 10g central sur lequel un compte administrateur spécifique doit être créé. C'est avec ce compte que seront rapatriées les données du PDA.

La seconde machine sert d'intermédiaire entre Oracle XE et Oracle Lite. Elle exécute *Oracle Mobile Server* qui a pour rôle d'enregistrer les différentes applications, unités mobiles et utilisateurs mobiles. Mobile Server peut alors être configuré pour gérer et programmer les synchronisations ainsi que les éventuelles mises à jour applicatives. Les fonctionnalités de *Mobile Server* sont complétées par celles d'*Oracle Database Workbench*. Cette outil permet de définir les différentes tables et colonnes qui seront accessibles depuis l'application mobile ainsi que les

éléments de synchronisation associés. Un projet finalisé dans *Database Workbench* doit être déployé sur le *Mobile Server* afin d'être pris en compte lors des synchronisations suivantes.

La dernière machine impliquée dans le processus de synchronisation est l'unité mobile. En plus de l'application CAR et de la base de données embarquée Oracle Lite, deux applications doivent être installées et configurées. La première, *Oracle Device Manager*, est l'outil qui permet d'associer le PDA à un compte *Mobile Server*. C'est avec cet outil que pourront en outre être déclenchées d'éventuelles mises à jour et synchronisations automatisées. *Oracle MSync* permet quant à lui de déclencher manuellement une synchronisation. C'est avec cet outil que le responsable de parc d'agences pourra récupérer l'état courant du parc de véhicules et faire remonter les actions qu'il aura effectuées face au client. Une fois l'application configurée, deux clics suffisent pour déclencher une synchronisation. Ce fonctionnement est pratique car il ne nécessite pas une connexion permanente à Internet et permet de profiter de la proximité de points d'accès sans fil.

L'Application Mobile

L'application mobile se présente sous la forme d'un client *lourd* en Java, exécuté sur la plateforme d'exécution libre Mysaifu. Le code de cette application se limite à une compatibilité avec Java 1.4, version la plus avancée supportée par la JVM. L'application se décompose en trois couches qui communiquent entre elles à l'aide d'appels de méthodes Java.

La couche Présentation

La couche présentation correspond à l'interface entre l'application (le traitement métier) et l'utilisateur. L'API Swing nécessitant des ressources trop importantes au regard de ce que supporte un PDA, l'IHM n'utilise que les bibliothèques AWT.

L'IHM de l'application est contrôlée par une classe principale qui se charge d'afficher l'écran demandé. Les différents écrans sont implémentés sous la forme de calques dont l'ordre est géré par la classe principale. Chaque calque a la possibilité d'appeler un autre calque, dans la mesure où l'utilisateur authentifié dans le système a le droit de visualiser les données associées. L'appel d'un calque par un autre peut s'accompagner du passage d'un objet métier (cf. paragraphe dédié).

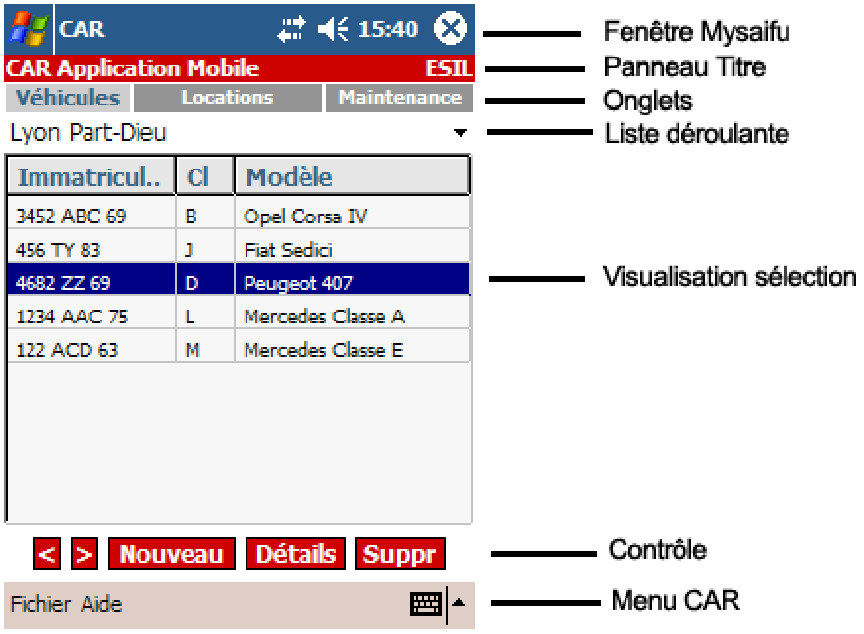


Figure : Exemple d'écran de l'application mobile. Celui-ci est composé d'éléments fixes (tels que les onglets d'accès aux modules) et d'éléments contenus dans le panneau couramment affiché.

L'ensemble des écrans est chargé au démarrage de l'application mobile afin de rendre plus confortable la navigation entre les différents panneaux.

La couche Métier

Cette couche se compose de classes faisant l'intermédiaire entre la base de données et la couche présentation. Ces classes extraient les données demandées par l'utilisateur et les délivrent à l'IHM. Celle-ci est alors susceptible de renvoyer de nouvelles données à la couche métier suite à une saisie ou une autre opération de l'utilisateur. La couche métier sera alors responsable du contrôle de ces données et de leur enregistrement dans la base Oracle Lite embarquée sur le PDA.

Est associé à la couche métier un ensemble de classes décrivant les entités manipulées dans l'application, à l'instar de ce que génère Hibernate dans l'application nationale. Ces classes métier permettent d'ajouter une valeur sémantique aux informations échangées et facilitent l'implémentation de règles de contrôle de ces informations.

Lorsqu'une erreur est rencontrée par une classe de la couche métier, celle-ci est transmise à la couche présentation qui l'affiche sur l'IHM et modifie éventuellement son comportement en conséquence (par exemple le refus de la validation d'un formulaire mal renseigné).

La couche Données

La couche physique nécessite Oracle Lite 10.3 pour sa mise en place. Le modèle métier qu'implémente cette couche physique est commun aux applications AN et AM. Le script de création de base est géré de manière transparente par le Mobile Server qui se charge de créer la base lors de la synchronisation si celle-ci n'existe pas.

La base de données à laquelle accède l'application mobile est située sur le PDA, ce qui permet d'obtenir des temps de latences très courts (l'agrément d'utilisation de l'application est cependant conditionné par la puissance du processeur embarqué). La communication avec la couche physique se fait au moyen d'un driver JDBC fournit par Oracle.