N° d'ordre : 03/TCO/STI Année Universitaire : 2007 / 2008

UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEURE POLYTECHNIQUE

DEPARTEMENT TELECOMMUNICATIONS

MEMOIRE DE FIN D'ETUDES

en vue de l'obtention

du DIPLOME d'INGENIEUR

Spécialité : Télécommunication
Option : Système de Traitement de l'Information

par : **RAKOTONDRAINA Tahina Ezéchiel**

J2EE : MISE EN ŒUVRE D'UN SITE WEB, PAR PAIEMENT EN LIGNE, DE TELECHARGEMENT DE SERVICE D'APPLICATION POUR TELEPHONE MOBILE

Soutenu le 06 Janvier 2009 devant la Commission d'Examen composée de :

Président :

M. RATSIMBAZAFY Andriamanga

Examinateurs :

M. RATSIHOARANA Constant

M. RAZAKARIVONY Jules

M. BOTO ANDRIANANDRASANA Jean Espérant

Directeur de mémoire : M. RANDRIARIJAONA Lucien Elino

REMERCIEMENTS

La rédaction de cet ouvrage représente l'accomplissement de cinq années de labeurs et de travail à Vontovorona. Je tiens à remercier ceux qui ont bien voulu me tendre la main.

Je suis reconnaissant envers Monsieur RAMANANTSIZEHENA Pascal, Professeur Titulaire et Directeur de l'Ecole Supérieure Polytechnique d'Antananarivo, de m'avoir accueilli au sein de l'établissement.

Je remercie chaleureusement Monsieur RANDRIAMITANTSOA Paul Auguste, Professeur, Chef de Département Télécommunications pour tous les savoirs qu'il nous a transmis et d'avoir permis l'achèvement de mes études dans les meilleures conditions possibles.

Monsieur RANDRIARIJAONA Lucien Elino, Enseignant chercheur à l'ESPA, Directeur de ce mémoire de fin d'études, pour ses conseils et aides, sans lesquels le mémoire n'aurait pas abouti.

Monsieur RATSIMBAZAFY Andriamanga, Maître de conférences et Enseignant chercheur au sein du Département Télécommunication, qui me fait l'honneur de présider le jury de ce mémoire,

Mes remerciements vont également aux membres du Jury, à savoir :

Monsieur RATSIHOARANA Constant, Maître de conférences et Enseignant chercheur à l'ESPA

Monsieur RAZAKARIVONY Jules, Maître de conférences et Enseignant chercheur à l'ESPA

Monsieur BOTO ANDRIANANDRASANA Jean Espérant, Assistant d'Enseignement et de Recherche au sein du Département Télécommunication

Qui ont eu l'amabilité d'examiner ce mémoire malgré leurs nombreuses occupations.

Mes vifs remerciements s'adressent également à tous les enseignants et les personnels administratifs de l'ESPA.

J'adresse enfin mes profonds remerciements à toute ma famille et mes amis pour leur amour, leurs encouragements et leur soutien durant mes études et à la réalisation de ce mémoire.

TABLE DES MATIERES

TABLE DES MATIERES i

LISTE DES ABREVIATIONS vi

INTRODUCTION GENERALE 1

CHAPITRE 1 GENERALITES SUR LES ARCHITECTURES SYSTEMES 2

1.1. Le contexte 2

1.1.1. Applications multitiers distribuées 2

1.2. Architecture système 3

1.2.1. Introduction sur le client/serveur 3

1.2.2. Modèle générale d'architecture système 5

1.2.2.1. Espace serveur 5

1.2.2.2. Espace client 6

1.2.3. Briques de l'architecture système 6

1.2.3.1. Interface utilisateur 6

1.2.3.2. Applications 6

1.2.3.3. Données 7

1.2.3.4. Middleware 8

1.2.4. Typologie technique 8

1.2.4.1. Type de données véhiculées (TDV) 8

1.2.4.2. Type de dialogue client/serveur (TD) 9

1.2.4.3. Structure et localisation des applicatifs 10

1.2.5. Architectures client/serveur 10

1.2.5.1. Client/serveur à client passif (1/3) 10

1.2.5.2. Client/serveur de données (2/3) 11

1.2.5.3. Client/serveur distribués (3/3) 13

1.2.5.4. Client/serveur à objets distribués (système réparti) 14

1.2.6. Evolution du client/serveur avec Internet 15

1.2.6.1. Standardisation des protocoles 15

1.2.6.2. Influence de la sécurité 16

1.2.6.3. Problème de bande passante 16

1.2.7. Architectures clients/serveurs basées sur Internet 17

1.2.7.1. Architectures C/S HTML/WEB 17

1.2.7.2. Architectures à code mobile 17

1.3. Conclusion 19

CHAPITRE 2 PLATE-FORME J2EE 20

2.1. Introduction 20

2.2. Applications multitiers distribuées de J2EE 20

2.2.1. Composants J2EE 20

2.2.2. Clients J2EE 21

2.2.2.1. Client web *21*
2.2.2.2. Applet *21*
2.2.2.3. Applications clientes *22*
2.2.3. Composants web **22**
2.2.4. Composants métiers **23**
2.2.5. Niveau EIS **24**
2.3. Conteneurs J2EE *24*
2.3.1. Services conteneurs **25**
2.3.2. Types de conteneurs **26**
2.3.2.1. Serveur J2EE *26*
2.3.2.2. Conteneur EJB *26*
2.3.2.3. Conteneur web *26*
2.3.2.4. Conteneur d'application cliente *26*
2.3.2.5. Conteneur d'applet *26*
2.4. API de J2EE *27*
2.4.1. Technologie Enterprise JavaBeans **27**
2.4.2. Technologie Java Servlet **27**
2.4.3. Technologie JavaServer Page **27**
2.4.4. API Java Message Service **28**
2.4.5. API JDBC **28**
2.4.6. API Java Transaction **28**
2.4.7. JNDI **28**
2.4.8. API JavaMail **28**
2.5. Serveurs d'applications *29*
2.5.1. Rôles d'un serveur d'application **29**
2.5.2. Intégration dans l'architecture d'entreprise **29**
2.5.3. Architecture externe d'un serveur d'application **30**
2.5.4. Architecture interne d'un serveur d'application **31**
2.6. Communication entre le client et le serveur : le protocole HTTP *31*
2.7. Développement client *32*
2.8. Développement serveur en Java *33*
2.9. Conclusion *33*

CHAPITRE 3 BASES DE DONNEES **34**
3.1. Présentation générale *34*
3.1.1. Introduction aux bases de données **34**
3.1.2. Objectifs et avantages **34**
3.1.2.1. Indépendance physique. *35*
3.1.2.2. Indépendance logique. *35*
3.1.2.3. Manipulable par des non-informaticiens. *35*
3.1.2.4. Accès aux données efficace. *35*

3.1.2.5. Administration centralisée des données. *36*
3.1.2.6. Non redondance des données. *36*
3.1.2.7. Cohérence des données. *36*
3.1.2.8. Partageabilité des données. *36*
3.1.2.9. Sécurité des données. *37*
3.1.3. Différents types de bases de données **38**
3.1.4. Quelques systèmes existants **39**
3.1.5. Modéliser les données **39**
3.2. Algèbre relationnelle *41*
3.2.1. Opérations de base **41**
3.2.2. Expressions de l'algèbre relationnelle **42**
3.2.3. Quelques remarques sur l'algèbre relationnelle **42**
3.3. Langage SQL *43*
3.3.1. L'obtention des données **44**
3.3.1.1. Expression des projections *45*
3.3.1.2. Expression des restrictions *45*
3.3.1.3. Tri et présentation des résultats *45*
3.3.1.4. Expression des jointures *45*
3.3.1.5. Fonctions statistiques *46*
3.3.2. Mise à jour d'informations **47**
3.3.2.1. Insertion. *47*
3.3.2.2. Mise à jour *48*
3.3.2.3. Suppression. *48*
3.3.3. Définition des données : DDL **48**
3.3.3.1. Types de données *49*
3.3.3.2. Création de tables *49*
3.3.3.3. Expression des contraintes d'intégrité *49*
3.4. Connexion aux bases de données *51*
3.4.1. API JDBC **51**
3.4.1.1. Qu'est ce que JDBC? *51*
3.4.1.2. Structure d'une application JDBC *52*
3.4.2. Différents pilotes **56**
3.4.3. Requêtes précompilées **57**
3.4.4. Commit et Rollback **58**
3.4.5. Méta-base **58**
3.5. Conclusion *59*

CHAPITRE 4 SERVLETS ET JSP **60**
4.1. Application web *60*
4.1.1. Cycle de vie des applications web **62**
4.1.2. Modules web **63**
4.2. Servlets *64*

4.2.1. Définition **64**
4.2.2. Cycle de vie d'une servlet **65**
4.2.3. API servlet **65**
4.2.4. Structure de fonctionnement interne d'une servlet **65**
4.2.5. Implémentation d'une servlet **66**
4.2.6. Initialisation et destruction d'une servlet **68**
4.2.6.1. Initialisation ***68***
4.2.6.2. En service ***69***
4.2.6.3. Destruction ***69***
4.2.7. Invocation d'une servlet à partir d'un client léger **69**
4.2.7.1. Invocation de la méthode doGet() ***69***
4.2.7.2. Invocation de la méthode doPost() ***71***
4.3. JSP ***72***
4.3.1. Généralités **72**
4.3.2. Comparaison entre JSP et servlet **73**
4.3.3. Architecture **74**
4.3.4. Tags JSP **74**
4.3.4.1. Déclaration tag (<%! %>) ***75***
4.3.4.2. Expression tag (<%= %>) ***75***
4.3.4.3. Directive tag (<%@ directive ... %>) ***75***
4.3.4.3.1. Page directive 75
4.3.4.3.2. Include directive 76
4.3.4.3.3. Tag Lib directive 77
4.3.4.4. Scriptlet tag (<% … %>) ***77***
4.3.4.5. Action tag ***77***
4.3.4.6. Javabeans ***77***
4.3.4.7. Commentaire ***77***
4.3.5. Avantages et inconvénients des JSP **78**
4.4. Conclusion ***80***

CHAPITRE 5 MISE EN ŒUVRE DE L'APPLICATION **81**
5.1. Etat de l'art ***81***
5.2. Choix du Langage Java ***81***
5.3. Choix de la base de données ***81***
5.4. Outils de développement ***81***
5.4.1. MERISE **81**
5.4.2. ECLIPSE et DREAMWEAVER **83**
5.4.2.1. Configuration d'ECLIPSE ***83***
5.4.2.2. Configuration de DREAMWEAVER ***84***
5.4.3. Win' Design et EASYPHP **85**
5.4.4. TOMCAT **86**
5.4.4.1. Configuration de Tomcat ***87***

5.4.4.2. Modification de contexte *87*
5.5. Mise en œuvre de l'application *89*
5.5.1. Règles de gestion **89**
5.5.2. Organigramme **91**
5.5.3. Exemples de pages de l'application **91**
5.5.3.1. Inscription *91*
5.5.3.2. Authentification ou login *93*
5.5.3.3. Page représentant la liste des différents services *93*
5.6. Conclusion *94*
CONCLUSION GENERALE **95**
ANNEXES……**96**
ANNEXE 1 : ARCHITECTURE DES RESEAUX **97**
ANNEXE 2 : RAPPELS HTML **98**
ANNEXE 3 : CODE D'AUTHENTIFICATION **102**
ANNEXE 4: CODE SOURCE JEUX.JSP **103**
ANNEXE 5 : CODE SOURCE SHOWCADDY.JSP **105**
BIBLIOGRAPHIE **107**
FICHE DE RENSEIGNEMENT **109**
RESUME **110**
ABSTRACT **110**

LISTE DES ABREVIATIONS

ACID	Atomicité Cohérence Isolation Durabilité
ADSL	Asynchronous Digital Subscriber Line
ANSI	American National Standards Institute
API	Application Programming Interface
ASP	Active Server Page
AWT	Abstract Window Toolkit
BCFN	Boyce-Codd Form Normal
C/S	Client/Serveur
CGI	Common Gateway Interface
CIN	Carte d'Identité National
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DBMS	DataBase Management System (SGBD)
DCL	Data Control Language
DCOM	Distributed Common Gateway Interface
DDL	Data Definition Language
DLL	Dynamic Linked Library
DML	Data Manipulation Language
DNS	Domain Name Server
ESPA	Ecole Supérieure Polytechnique d'Antananarivo
EIS	Enterprise Information System
EJB	Enterprise Java Bean
FTP	File Transfer Protocol
GUI	Graphic User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IDS	Integrated Data Storage
IIOP	Internet Inter-ORB Protocol
ISO	International Standards Organization
JAR	Java ARchive
J2EE	Java 2 Enterprise Edition

J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JDBC	Java DataBase Connectivity
JDK	Java Developement Kit
JMS	Java Message Service
JNDI	Java Naming Directory Interface
JSP	Java Server Page
JSTL	JavaServer Pages Standard Tag Library
JTA	Java Transaction API
JVM	Java Virtual Machine
L4G	Langage de 4è Génération
LDAP	Lightweight Directory Access Protocol
MCD	Modèle Conceptuel des données
MOM	Message Oriented Middleware
MVC	Model View Controller
ODBC	Open DataBase Connectivity
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Systems Interconnections
PC	Personal Computer
PHP	Personal Home Page
PKI	Public Key Interface
POP	Post Office Protocol
RC	Règle de Calcul
RG	Règle de Gestion
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SGBD	Système de Gestion de Base de Données
SGBDO	Système de Gestion de Base de Données Object
SGBDR	Système de Gestion de Base de Données Relationnelle
SIM	Subscriber Identity Module
SMTP	Simple Mail Transfer Protocol

SPI	Serial Peripheral Interface
SQL	Structured Query Language
TCL	Tool Commande Language
TCP	Transmission Control Protocol
TD	Type de Dialogue
TDV	Type de données véhiculées
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VB	Visual Basic
WAR	Web ARchive
WML	Wireless Markup Language
XML	eXtended Markup Language

INTRODUCTION GENERALE

J2EE (Java 2 Enterprise Edition) est la plate-forme java qui définit le cadre d'un serveur d'application en entreprise. En clair, c'est une plate-forme permettant de créer et de gérer une infrastructure d'objets distribués et de Services Web en vue de leur exploitation dans le domaine de l'utilisation et de la pratique du système d'information en entreprise. Dans le contexte actuel de mondialisation, de globalisation, d'échange ouvert d'information, ... l'entreprise se trouve confrontée à de nouveaux challenges à surmonter et dont il faut faire face pour prétendre survivre et avoir une place au soleil. Il est primordial de se doter d'un système d'information qui permet d'avoir une décision appropriée au moment opportun. Néanmoins, la mise en place d'un tel système d'information exige la maîtrise des différents aspects technologiques mis en jeu. Aussi, il faut donc en premier lieu savoir sur quelle plate-forme doit s'asseoir ledit système et en faire un bon usage pour répondre au mieux aux attentes des acteurs cibles. Dès à présent la plate-forme J2EE qui a été choisie est l'une des meilleures plates-formes sur le marché actuellement.

Ce mémoire va donc explorer en profondeur cette plate-forme J2EE, qui utilise le langage Java, afin de tirer profit de ces différents avantages. Pour ce faire, le plan de ce mémoire se présente comme suit :

En première partie, nous développerons les différentes architectures systèmes les plus courantes. Ce développement sera suivi de la description détaillée de la plate-forme J2EE. De nombreux concepts généraux seront mis en exergues.

En deuxième partie, nous aborderons les bases de données qui font partie intégrante des systèmes d'information d'entreprises. Ce chapitre sera suivi d'une description détaillée de l'un des aspects particuliers de la plate-forme J2EE, en l'occurrence les composants pour le développement d'application Web : les servlets et les Java Server Page (JSP).

Ce sont ces alternatives intéressantes pour la conception d'applications intranet basées sur le web qui feront l'objet d'une réalisation dans le cadre de ce mémoire.

CHAPITRE 1 GENERALITES SUR LES ARCHITECTURES SYSTEMES

1.1. Le contexte [1]

L'ouverture dans le monde grâce à internet et l'évolution de la technologie poussent les développeurs et les concepteurs de site web à chercher et à adopter une approche à la fois simple et fiable pour mettre en œuvre des systèmes

C'est en ce sens qu'une étude sur la plate-forme J2EE ainsi que sur les bases de données et l'élaboration d'un système d'information nous est présentée dans ce mémoire.

J2EE fournit une approche basée composant sur la conception, le développement, l'assemblage, et le déploiement des applications d'entreprises afin de réduire le coût et augmenter la rapidité de la conception et du développement

La mise en place d'un tel système d'information comprend deux phases à savoir : la phase de conception qui nécessite des méthodes permettant de mettre en place un modèle de base ou référence et la modélisation qui consiste à créer une représentation virtuelle d'une réalité de façon à faire ressortir les points intéressants. Plusieurs méthodes d'analyse existent, mais pour cette étude, la méthode « Merise » sera utilisée et dont les détails seront largement expliqués ultérieurement.

Puisque les solutions existantes pour le développement sont très variées, les technologies servlets et JSP seront adoptées. Ces technologies semblent les plus appropriées pour l'application. Une des raisons de ce choix est que les servlets et les JSP sont écrites en java, et héritent donc des caractéristiques de ce langage.

1.1.1. *Applications multitiers distribuées*

La plate-forme J2EE se base sur le modèle d'application multitiers distribué en général, et sert d'atelier logiciel pour la production d'applications d'entreprises et d'application Web.

Une application multitiers est composée du traitement et de la présentation ; le traitement contient les données et la logique applicative (ou processus métier ou logique métier) et la présentation se charge de présenter les applications ainsi traitées

La logique applicative est divisée en deux composants selon ses fonctions (orientées web ou entreprise). Les différents composants d'applications que comporte une application J2EE sont installés sur différentes machines selon leur niveau dans l'environnement multitiers J2EE.

Pour illustrer ceux-ci, la figure 1.01 montre deux exemples types d'application multitiers basée sur la plate-forme J2EE. La représentation est ici hiérarchisée en niveau et comprend les composants

suivants:

- Composant « niveau client » : s'exécute sur la machine cliente
- Composant « niveau web » : s'exécute sur un serveur web J2EE
- Composant « niveau métier » : s'exécute sur un serveur métier J2EE
- « Niveau système d'information de l'entreprise » (niveau EIS)

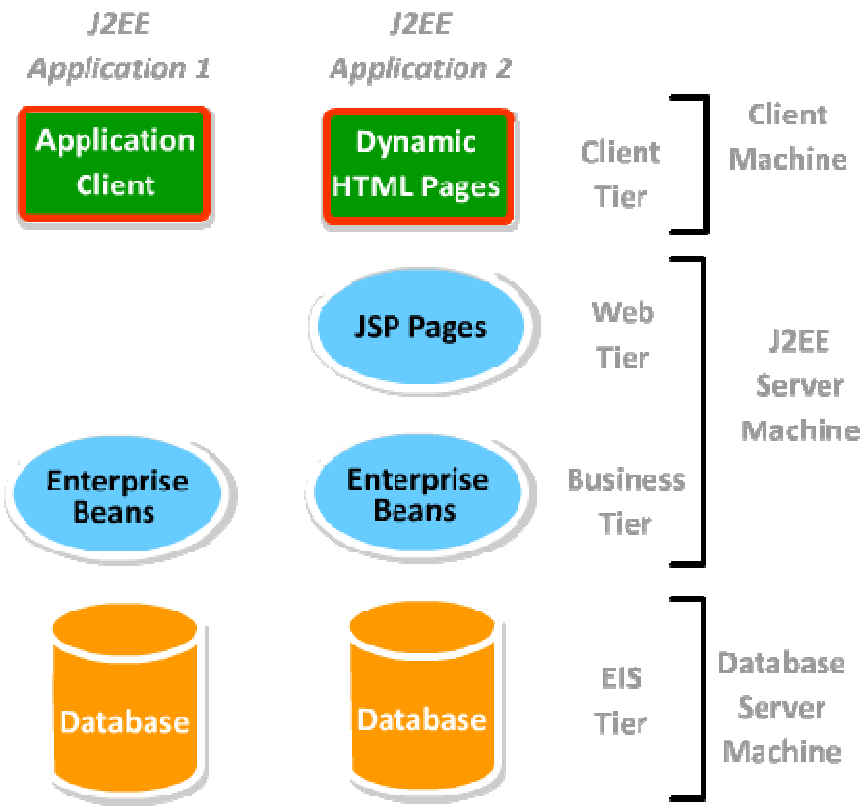


Figure 1.01: Application multitiers

1.2. Architecture système [2]

1.2.1. *Introduction sur le client/serveur*

Auparavant les Mainframes étaient les architectures dominantes dans les entreprises. Les applications et les données étaient centralisées sur un gros ordinateur, dont des terminaux aux faibles capacités permettent juste l'affichage des résultats demandés. Le client/serveur est une évolution du Mainframe et permet par l'utilisation de nouvelles méthodes et techniques de passer outre les limites de l'environnement Mainframe et les systèmes propriétaires. Cette innovation permet ainsi d'améliorer l'interopérabilité, la flexibilité des systèmes.

Dix ans après son apparition, le client/serveur est devenu l'architecture la plus prisée. En fait, il applique un gigantesque découpage aux applications monolithiques des sites centraux pour répartir les charges de traitement entre clients et serveurs. La manière dont étaient conçues et bâties les applications jusqu'ici en a été révolutionnée, et la facilité d'utilisation est désormais offerte aux utilisateurs finaux.

Le client/serveur a entraîné, dans son sillage, la création d'une énorme industrie logicielle dominée par des géants comme Baan, Informix, Lotus, Microsoft, Novell, Oracle, PeopleSoft, SAP, Sun et Sybase. Superstars de la première époque du client/serveur, ses entreprises forment aujourd'hui le nouvel établissement de l'informatique.

Même si cela peut surprendre, une modification silencieuse est en marche au sein même de la révolution client/serveur. La figure 1.02 montre en effet que la technologie client/serveur est en train de passer d'une structure originelle à 2 niveaux à une architecture à 3 niveaux.

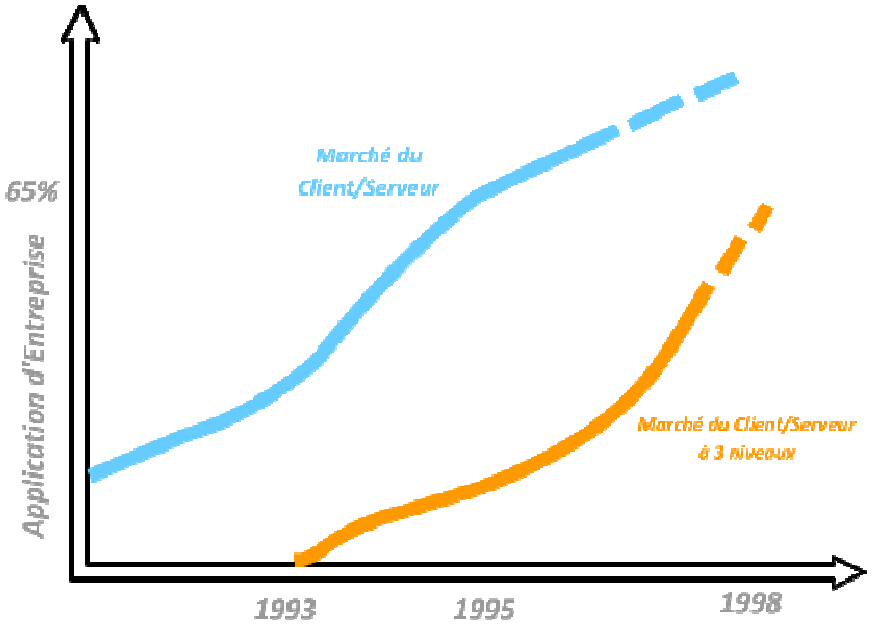


Figure 1.02: Croissance comparée des client/serveur à 2 et à 3 niveaux

L'impact du changement du client/serveur à deux niveaux vers le client/serveur à trois niveaux sera plus important que l'évolution initiale des applications monolithiques (Mainframe) vers le client/serveur. Ce courant vers le modèle à 3 niveaux a été initié par la nécessité d'utiliser la technologie client/serveur pour les grandes applications des entreprises.

L'Internet, les intranets, les objets distribués et les composants propulsent aujourd'hui la structure à 3 niveaux au cœur du client/serveur.

Le client/serveur est donc devenu la forme dominante de l'informatique et le modèle à 3 niveaux la forme dominante du client/serveur.

1.2.2. *Modèle générale d'architecture système*

Toutes les architectures informatiques client/serveur présentent des caractéristiques communes :

- Elles intègrent une interface utilisateur souvent graphique (GUI)
- Elles fonctionnent grâce à des applications
- Les applications qui les animent manipulent des données.

C'est la répartition de ces trois composants entre le client et/ou les serveurs qui caractérisent les différentes architectures.

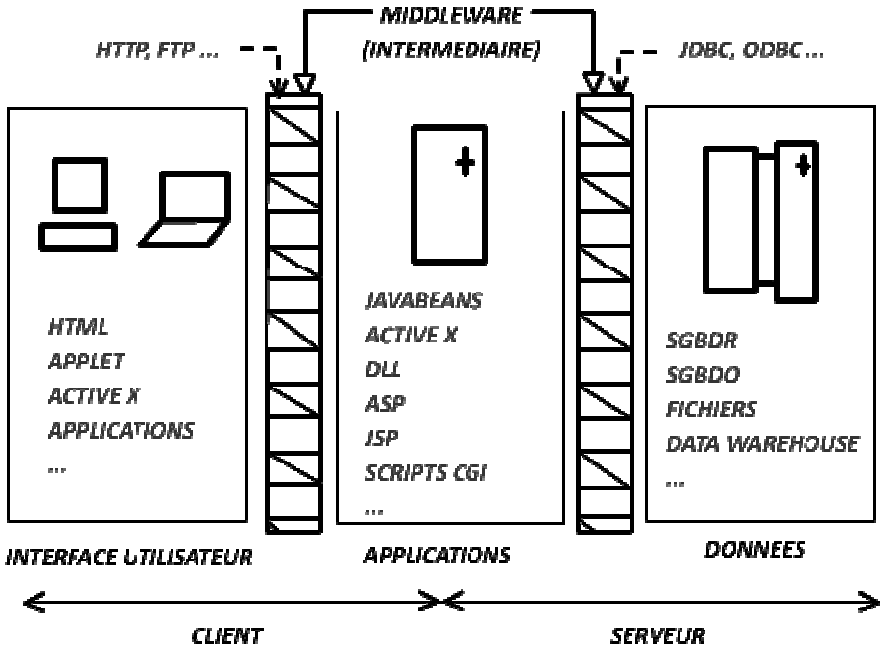


Figure 1.03: Architecture client-serveur

1.2.2.1. Espace serveur

Voici quelques exemples de serveurs :

- Serveur objet

  Le serveur objet exécute la logique applicative auquel le client se connecte par l'intermédiaire d'un protocole d'objets distribués. Les objets qu'héberge ce serveur peuvent éventuellement être conditionnés sous la forme de composant.

- Moniteur transactionnel

Son but est de coordonner l'exécution de transactions distribuées sur plusieurs machines. Par exemple : si l'utilisateur veut à la fois réserver un avion et un hôtel, à l'aide d'une même application, alors l'application devra s'assurer que la transaction a lieu intégralement ou pas du tout. Le moniteur transactionnel sera alors très utile pour coordonner ces deux réservations simultanées et s'assurer de l'atomicité (ACID) de l'opération composée.

- Serveur web

Le serveur web selon le cas a un rôle de serveur de fichiers ou celui de middleware

- Serveur de sécurité

Le serveur de sécurité qui gère l'authentification des utilisateurs et le chiffrement des flux de communication. La norme LDAP (Lightweight Directory Access Protocol) mise en œuvre par Netscape à la fin des années 90, s'impose aujourd'hui comme le standard de serveur d'annuaire distribué. LDAP est ainsi au centre de tous les systèmes de sécurité à base de cryptage à clé public (PKI : Public Key Interface).

1.2.2.2. Espace client

Le logiciel client n'est pas lié à la machine qui le fait tourner, il doit être portable et mobile. De ce fait, l'espace client est souvent unique pour un utilisateur donné et sa localisation physique est en principe indéterminée.

1.2.3. *Briques de l'architecture système*

1.2.3.1. Interface utilisateur

Aujourd'hui, les interfaces utilisateurs sont interactives, multimédia, et de préférence portables, c'est-à-dire des interfaces graphiques pouvant fonctionner sur tous les systèmes d'exploitation et conservent toujours le même aspect.

1.2.3.2. Applications

Le choix en matière de langage de développement est considérable, et souvent de haut niveau, c'est à dire qu'il reste éloigné des préoccupations au niveau des matérielles, comme la gestion de la mémoire ou la programmation réseau ; Ces détails sont réglés automatiquement. Le développement peut donc se concentrer sur la logique de l'application, sur les langages de scripts (Perl. PHP, TCL, JavaScript, VBScript) et sur les langages objets (Java, C++, VB.Net, C#). Ces

langages de scripts sont très simples et permettent de développer rapidement des petites applications. Pour des projets de plus grandes envergures, il est nécessaire d'avoir recours au langage objet de troisième génération. Dans la pratique, les applications d'entreprises mettent souvent en jeu des combinaisons de ces deux types de langage.
Cependant, le problème du développement informatique ne se limite pas au choix d'un langage. La qualité d'une application informatique s'exprime selon quatre axes :

- Facilité de développement (FAIRE)
- Facilité de déploiement (INSTALLER)
- Facilité à être maintenu ou maintenabilité ('METTRE A JOUR)
- Facilité de réutilisation (RECYCLER)

Dans les architectures client/serveur, l'optimum de qualité est atteint par le recours à la programmation objet ou à base de composant (ActiveX, DLL….) Ces briques logicielles qui servent à construire l'édifice applicatif sont généralement développées dans des langages compilés de troisième génération comme C++ ou Java. La gestion des interfaces graphiques de type HTML est par contre simplifiée par l'utilisation de langage script côté serveur. Actuellement la multiplication des solutions offrent des combinaisons de scripts et de composants : le langage de scripts joue le rôle d'intermédiaire entre le serveur d'application et les composants qu'il héberge (ASP/ActiveX, JSP/EJB, JavaScript/EJB).

1.2.3.3. Données

La grande majorité des données de gestion est encore à l'heure actuelle stockée dans des bases de données non relationnelles, dans des « Mainframes ». Jusqu'à une époque très récente, la plupart des éditeurs de sites intranet avaient purement et simplement oublié ce fait. Les accès se limitaient aux données relationnelles (Sybase, Oracle à travers SQL) souvent exclusivement au travers de l'interface ODBC (Open DataBase Connectivité) de Microsoft. Les Systèmes de Gestion de Base de Données Objet ne percent toujours pas. Leurs qualités sont pourtant indéniables lorsque l'on mène un projet objet :

- La persistance des données objets est gérée presque automatiquement
- Il n'est pas nécessaire de constituer une correspondance entre la représentation des données et leur représentation sous forme relationnelle.
- Les performances sont aussi bonnes voire meilleures qu'avec le SGBDR
- L'administration de base de données est souvent plus aisée qu'avec le SGBDR

1.2.3.4. Middleware

Pour lier efficacement les différentes composantes d'une application client/serveur, il est parfois nécessaire d'avoir recours à des logiciels spécifiques : « les middlewares » Comme leur nom l'indique, ce sont des intermédiaires. Ils servent à simplifier le travail du développeur en se chargeant d'un point particulier. Ils sont de plusieurs types, les principaux étant :

- Les middlewares d'accès aux données :

Ils mettent en communication les applications avec les différentes sources de données de l'entreprise.

- Les ORB (Object Request Broker) :

Leur but premier est de rendre plus transparente la communication avec les applications situées sur des machines distantes grâce à ce qu'on appelle objets distribués (interopérabilités). Citer comme exemple : CORBA, DCOM, RMI.

- Les moniteurs transactionnels :

Leur fonction est de coordonner les transactions distribuées sur plusieurs machines

- Les Messages Oriented Middleware (MOM) :

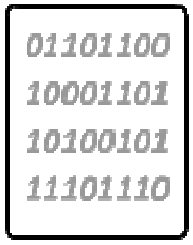
Il s'agit de serveurs intermédiaires dans la communication entre application.

1.2.4. *Typologie technique*

Pour comparer les architectures entre elles, le paragraphe qui suit s'attache à déterminer quelles sont les caractéristiques significatives d'une architecture client/serveur.

1.2.4.1. Type de données véhiculées (TDV)

La figure 1.04 représente le TDV



HTML, SQL, IIOP,
Objets sérialisés, XML ...

Figure 1.04: Données véhiculées

De quelle nature est l'information qui circule entre le client et le serveur ?

- S'agit-il d'une information de présentation (interface HTML) auquel cas le client s'apparente à un terminal passif (il doit juste afficher ce qu'on lui envoie) ?
- Ou bien s'agit-il uniquement de données extraites d'une base SQL, auquel cas le client a pour responsabilités de déterminer comment afficher ces données ?
- Ces données représentent-elles des appels de fonction à distance (avec le protocole CORBA-IIOP par exemple) ?
- Ou enfin, ces données sont-elles des objets sérialisées, c'est à dire transformées en une série d'octets propre à être transmise au travers d'un réseau ?

1.2.4.2. Type de dialogue client/serveur

La figure 1.05 représente le type de dialogue client/serveur

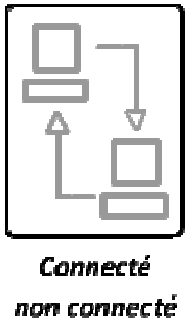


Figure 1.05: Dialogue client-serveur

Le HTTP est un protocole déconnecté : on demande une page HTML, le serveur web la renvoie (avec les éventuels fichiers qui s'y rattachent comme les images, ou des applets Java), et la communication est coupée. Comment obtenir un contexte transactionnel avec un tel protocole ?

Par exemple : comment faire en sorte qu'un utilisateur puisse acheter des produits en ligne au travers de page HTML, et que chaque nouvelle page qu'il visite se souvienne de ce qu'il avait acheté lors des pages précédentes. A l'inverse, les protocoles tel que CORBA - IIOP - RMI - DCOM, ou plus simplement le protocole TCP sont dits connectés. A la manière d'une communication téléphonique, la ligne reliant les interlocuteurs est dédiée : le dialogue est interrompu explicitement soit par le client, soit par le serveur.

1.2.4.3. Structure et localisation des applicatifs

La figure 1.06 représente la structure et localisation des applicatifs

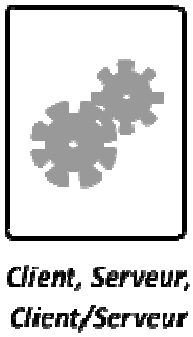


Figure 1.06: *Localisation des applicatifs*

La centralisation des applicatifs est d'un attrait certain, car elle facilite la maintenance, le déploiement, la gestion des accès concurrent aux données, des transactions, etc. L'intégrisme centralisateur doit céder le pas à une réflexion plus rationnelle sur l'organisation du système d'information. Par exemple : certains métiers imposent la mobilité et l'autonomie : on doit travailler sans être connecté au serveur. Comment y parvenir s'il n'y a pas de code sur le poste client ?

1.2.5. *Architectures client/serveur*

1.2.5.1. Client/serveur à client passif (1/3)

Le terme client/serveur appliqué à ce type d'architecture présente des clients passifs qui n'exécutent rien. Le client ici se présente sous forme de terminal ou l'émulation de terminal. C'est le serveur qui héberge les applications.

Dans ce type d'architecture, la logique applicative (métier) et les données sont concentrées en un seul point : « le site central ». Le client (c'est à dire le terminal) ne sert qu'à la visualisation et à l'insertion des données.

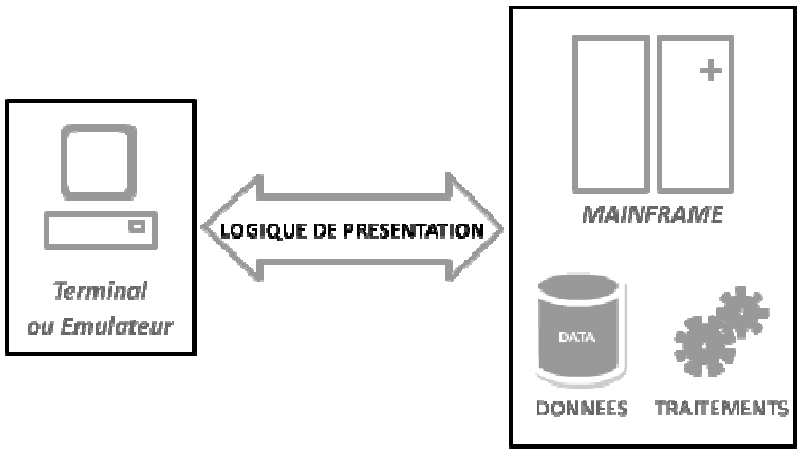


Figure 1.07: Architecture 1/3

Ce type de système d'information a fait ces preuves. Sûr, sécurisé, robuste, il satisfait parfaitement à la première loi de l'informatique d'entreprise : « il faut que ça fonctionne ». La programmation est simple puisque l'accès aux données est généralement très bien intégré. En effet, l'avantage est de pouvoir travailler dans la mémoire d'un ordinateur puissant, particulièrement stable et qui est spécialement conçu pour le traitement de gestion : « le mainframe ». Le flux réseau entre le client et le serveur ne véhicule que des informations de présentation : de tableau de caractères ou maps 3270 par exemple.

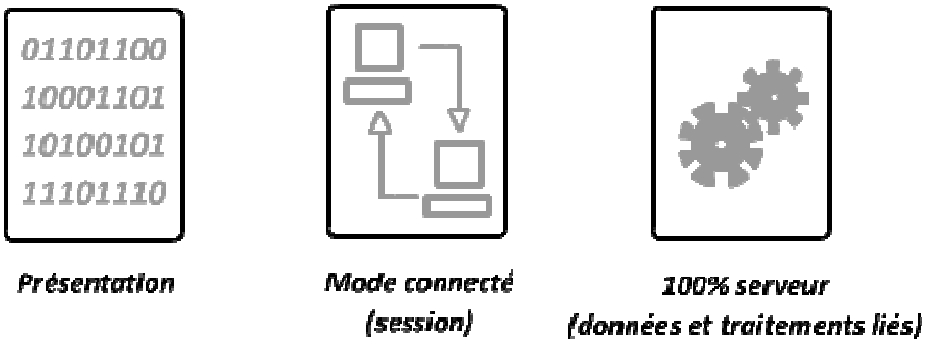


Figure 1.08: Eléments constituants de l'architecture 1/3

1.2.5.2. Client/serveur de données (2/3)

Ce type d'architecture est apparu avec la vague de PC du milieu des années 80. La loi de Grosh stipule qu'à puissances égales, plusieurs petites machines coûtent moins cher qu'une grosse. Pragmatique, les entreprises ont donc adhéré au PC. Cependant le mainframe n'est pas mort pour autant car il contenait déjà toute la logique applicative et toutes les données de l'entreprise. Opérer une migration intégrale vers les PC n'était pas envisageable. Ainsi un nouveau type d'architecture s'est développé autour du PC : « le client/serveur de données ».

Le PC étant capable d'exécuter des traitements, pourquoi ne pas en profiter ? Par conséquent, le serveur perd ces prérogatives, et n'est désormais utilisé qu'à la gestion des données.
C'est l'avènement des SGBDR et des outils L4G (SQL Windows. PowerBuilder) qui permettent rapidement de développer des applications pour un PC avec accès aux données du serveur.
Le client concentre ainsi l'ensemble de la logique applicative et communique au serveur à l'aide d'un langage de requête standardisé : « SQL ». Grâce au PC, les interfaces peuvent alors devenir beaucoup plus graphiques accroissant la qualité d'information disponible dans un même écran et améliorant les possibilités de manipulation de données. Les GUI peuvent augmenter notablement la productivité des utilisateurs.

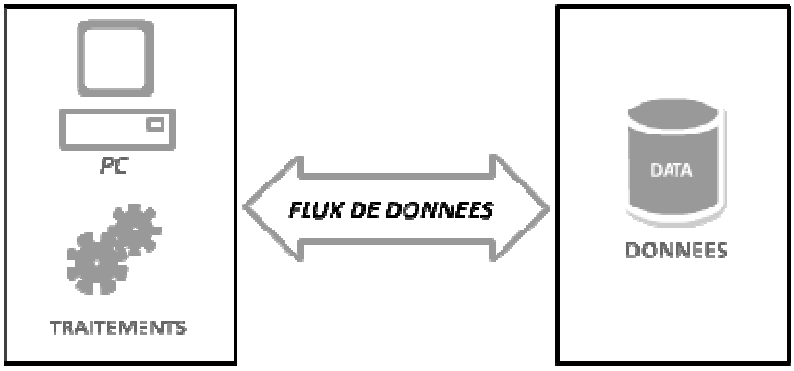


Figure 1. 09: Architecture 2/3

A peine dix ans plus tard le bilan est sans appel : le coût de maintenance de ce type d'architecture, démontré par la figure ci-dessus, est prohibitif. Par ailleurs, les déploiements à grande échelle sont complexes (en cas de mise à jour, il faut réinstaller l'application sur les postes clients). Ainsi le mainframe est resté indispensable et il a survécu à la vague des PC.

Bref, la nature des éléments constituant cette architecture est résumée par la figure 1.10 suivant:

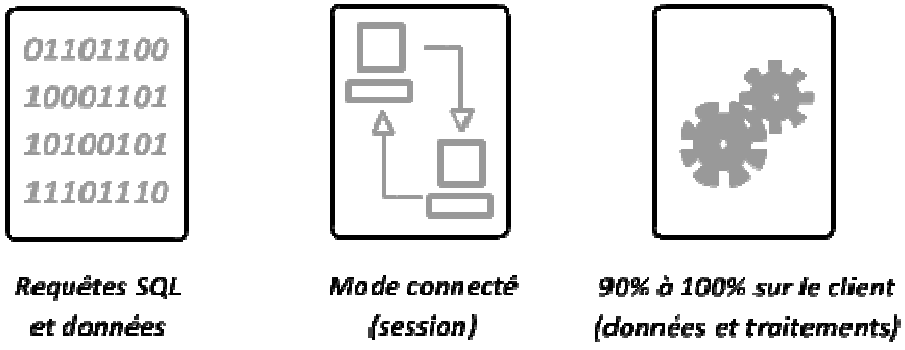


Figure 1.10: Eléments constituants de l'architecture 2/3

1.2.5.3. Client/serveur distribués (3/3)

Les débuts industriels de ce type d'architecture marquent le début des années 90. On commence à comprendre que la décentralisation intégrale des traitements vers les PC n'est pas forcément une bonne chose. Quelques portions de programme sous la forme de « procédures stockées » seront transférées progressivement sur le serveur.

Mais le problème de déploiement et de la maintenance des applications reste presque inchangé. Certains éditeurs d'outils L4G prennent la mesure du problème et proposent la solution basée sur des serveurs d'applications : N AT STAR, PowerBuilder, Forte, ... Ces serveurs hébergent des traitements appelés par des protocoles RPC propriétaires.

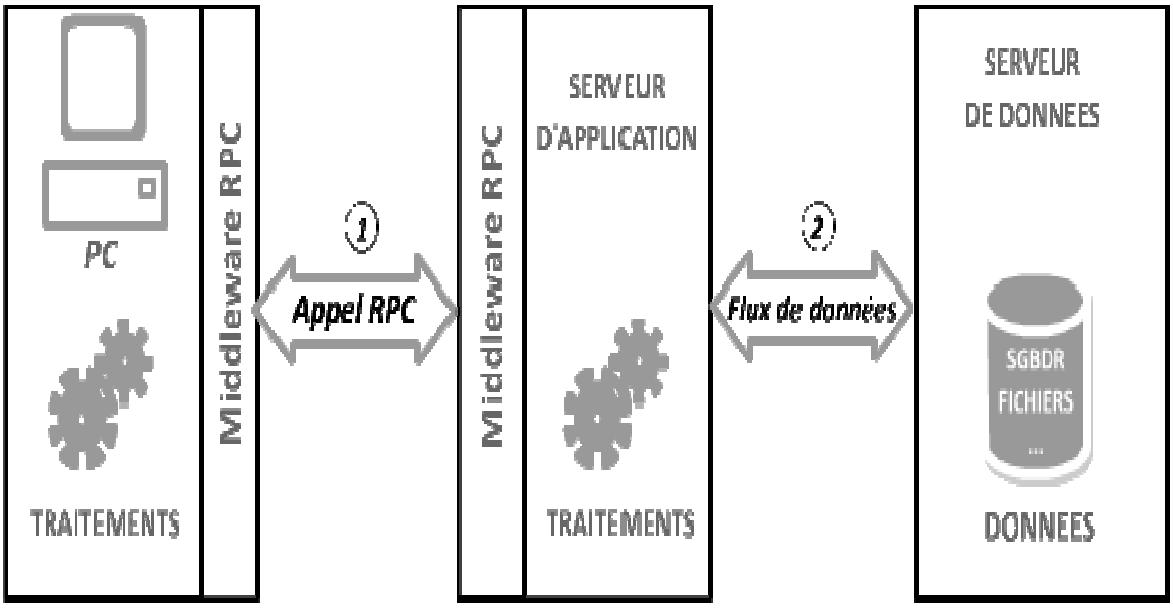


Figure 1.11: Architecture 3/3

Le principe de cette architecture est simple, il ne s'agit plus de disposer le code à exécuter sur le client, mais plutôt sur un serveur tiers (appelé « serveur d'application », car il offre la possibilité d'exécuter des portions d'application). Ainsi, il est au faveur du client de demander aux serveurs d'exécuter telle ou telle procédure. Le serveur effectue l'opération demandée puis retourne le résultat au client. Ainsi le client se trouve-t-il déchargé d'une bonne part des traitements. Par ailleurs, les données ne circulent presque plus qu'entre le serveur d'application et la base de données ce qui économise la bande passante réseau.

En résume, la nature des éléments constituants cette architecture est :

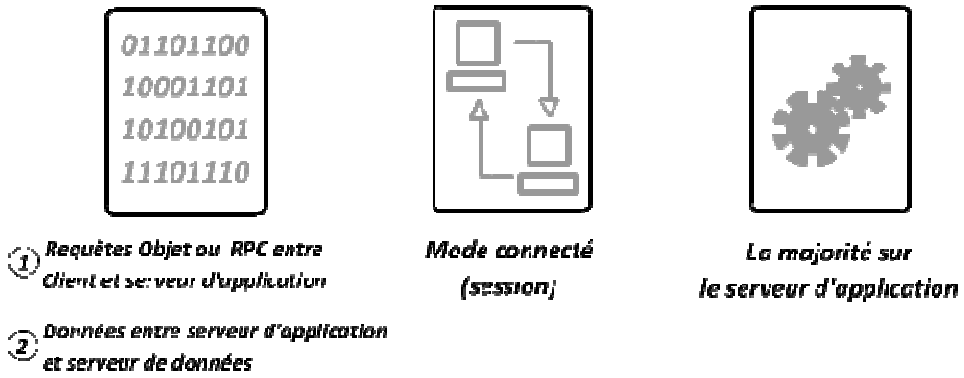


Figure 1.12: Eléments constituants de l'architecture 3/3

1.2.5.4. Client/serveur à objets distribués (système réparti)

Cette architecture se compose des différentes couches à savoir:

- **Couche Interface Utilisateur** : Couche chargée de gérer les interactions entre l'utilisateur et l'application (Browser Internet...).
- **Couche Logique de Présentation** : Elle permet de définir ce que doit afficher l'interface utilisateur et la manière dont les requêtes doivent être traitées.
- **Couche Logique Applicative** : Modélise les règles métiers de l'entreprise.
- **Couche Service d'Infrastructure :** Fonctionnalités fournies aux composants (connexions, transactions...).
- **Couche Données** : Données de l'entreprise.

L'apparition dans l'industrie des technologies objets offre une solution plus complète que les RPC classiques. En 89 une association à but non lucratif, l'OMG (Object Management Group) est créé pour promouvoir un modèle d'architecture objet distribué : CORBA Common Object Request Broker Architecture

CORBA est basé sur l'utilisation d'un nouveau middleware : l'ORB est chargée de transmettre des requêtes entre les objets situés dans la mémoire d'ordinateurs reliés entre eux par un réseau. Associé à la programmation objet, CORBA autorise une répartition des traitements et des données presque transparente. En outre CORBA est une architecture normalisée, supportée et

mise en œuvre par de nombreux acteurs du marché de l'informatique. ORACLE, Sun MicroSystem, FTP ne sont que quelque nom parmi les 850 membres de l'OMG. Cette architecture ressemble beaucoup à l'architecture 3/3. Néanmoins, les principales différences sont les suivantes :

- CORBA est standardisée et objet, alors que l'architecture 3/3 ne l'est pas
- Les requêtes RPC propriétaires sont remplacées par des requêtes IIOP
- Le middleware RPC est remplacé par un ORB quelconque, ORBÏX, Visiobroker. Un ORB peut être donné du côté du client et un autre du côté du serveur, ce qui n'était pas possible avec les protocoles propriétaires
- Il y a souvent plus qu'un serveur à objets distribués alors qu'on se limitait à un serveur d'application dans l'architecture 3/3.

1.2.6. *Evolution du client/serveur avec Internet*

1.2.6.1. Standardisation des protocoles

L'absence d'ouverture des architectures client/serveur classiques est l'une des raisons de leur faible capacité d'évolution. Evolution signifie ici augmentation des fonctionnalités offertes par le système et non renouvellement des productions ou des technologies.

La croissance exponentielle d'Internet s'explique notamment par la standardisation des protocoles. Cette standardisation adoptée au niveau d'une instance de normalisation a permis d'avoir un système qui n'est plus des agglomérats de composants propriétaires, mais plutôt d'un système basé sur des standards de plus en plus ouverts.

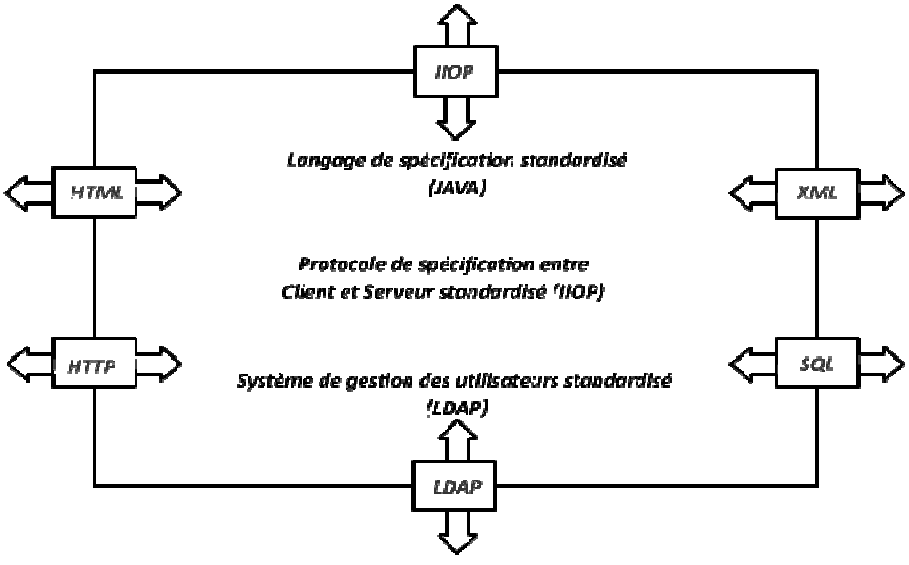


Figure 1.13: Standardisation des protocoles

Certes, l'interopérabilité (capacité à interchanger des produits issus de différents éditeurs) et la portabilité (capacité à changer de système d'exploitation) ne sont pas toujours au rendez-vous. Mais chaque composant de l'architecture offre aujourd'hui une interface standardisée qui constitue un point d'accès adapté à une évolution sans heurts dans le temps:

- JAVA pour les langages de programmation
- HTML pour les interfaces graphiques
- HTTP pour la diffusion d'information et de programme
- XML pour les échanges de données
- IIOP pour la communication inter-objet
- LDAP pour l'authentification

1.2.6.2. Influence de la sécurité

Avec l'ouverture des réseaux sur le monde extérieur, la sécurité est devenue une caractéristique obligatoire des architectures client/serveur. Elle a en réalité des composantes distinctes

- La protection des réseaux contre les intrusions, c'est à dire l'interdiction de l'accès à des services autres que ceux qui sont officiellement permis

- L'authentification des utilisateurs.

Les contraintes très fortes de sécurité sur Internet imposent des limitations aux architectures clients/serveurs dans l'entreprise. En particulier la présence de firewall peut empêcher ou limiter l'utilisation de certains protocoles de haut niveau.

1.2.6.3. Problème de bande passante

Sur Internet, il n'est pas rare pour un utilisateur connecté par modem de rencontrer des débits réseaux inférieur à 10 ko/s. Difficile dans ces conditions de proposer sur Internet les mêmes services, les même applications que celles qui sont exploitées à l'intérieur de l'entreprise. En réseau local, la bande passante par utilisateur est généralement supérieure jusqu'à atteindre un débit de 100Mbit/s. Les contraintes de bande passante sont dans certains cas insurmontables. Certes, des solutions comme Internet par câble ou l'ADSL permettent d'envisager des débits par utilisateurs proches de ceux des réseaux locaux

1.2.7. *Architectures clients/serveurs basées sur Internet*

Il existe en fait que deux types fondamentaux d'architecture basée sur Internet

- Architectures clients/serveurs HTML/WEB
- Architectures à code mobile.

1.2.7.1. Architectures C/S HTML/WEB

La figure 1.14 ci-dessous illustre l'architecture C/S HTML/WEB

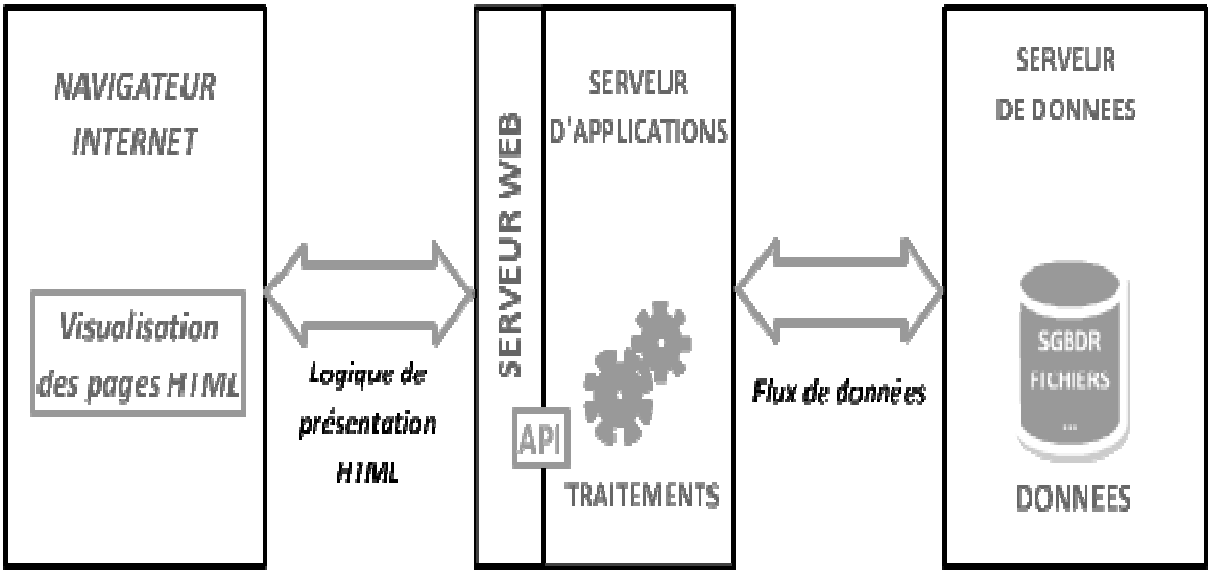


Figure 1.14: Les architectures C/S HTML/WEB

L'architecture HTML/WEB est historiquement la première à avoir été mise en œuvre. Elle repose sur l'utilisation du serveur web en tant que middleware. En d'autres termes, toutes les requêtes de l'utilisateur passent par le serveur web, et celui-ci les dirige vers les codes applicatifs adéquats.

1.2.7.2. Architectures à code mobile

Les architectures à code mobile se déclinent en deux catégories distinctes :

- Architecture à code mobile de type client/serveur de données
- Architecture à code mobile de type client/serveur distribué

Le point commun entre ces deux modèles est le téléchargement de la partie cliente de l'application afin d'établir le dialogue client/serveur.

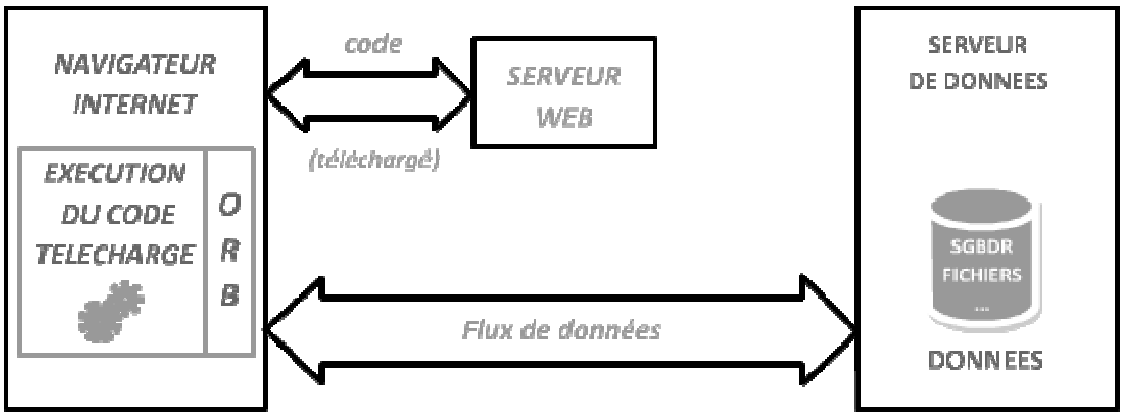


Figure 1.15: Les architectures C/S de données

Le type d'architecture C/S donné de la figure 1.15 n'est en fait qu'une adaptation à internet du module client/serveur de données. Dès lors que le code téléchargé depuis le serveur web commence à s'exécuter, on retombe dans un schéma de centralisation des applicatifs sur le client.

Il y a tout de même un avantage de taille par rapport au client/serveur de donnée classique, le problème de la distribution de code chez le client est résolu. En effet, si le code mobile sur le serveur web et mis à jour et qu'un client téléchargera la page html contenant ce code mobile, il recevra également la nouvelle version du code.

Le second type d'architecture à code mobile correspond à une adaptation du client/serveur à objet distribué ou web (figure 1.16). Dans un premier temps, l'application cliente est téléchargée dans le navigateur, ensuite celle-ci se connecte à un serveur d'application et communique avec lui par l'intermédiaire d'un protocole RPC (type CORBA-IIOP)

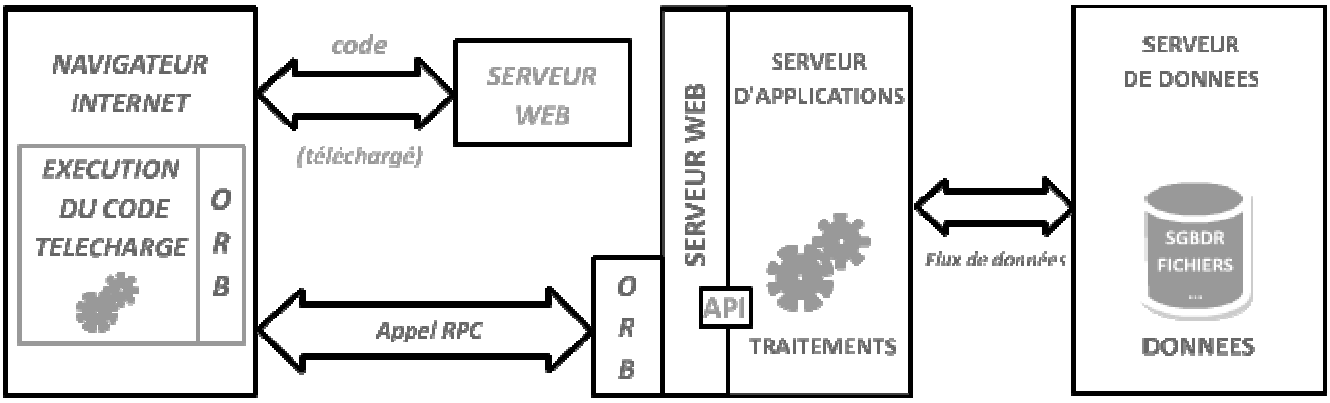


Figure 1.16: Les architectures C/S distribuées

Il s'agit probablement du modèle d'architecture le plus complexe mais aussi le plus évolué et le plus prometteur.

Types	Description
Client/Serveur HTML/WEB	Serveur web : Apache Serveur d'application : IBM WebSphere hébergeant des servlets qui accèdent à des composants EJB Source de données : SGBDR Oracle
Client/Serveur à code mobile : client/serveur de données	Client : Applet Java Protocole de communication client/serveur : protocole propriétaire Oracle mis en œuvre par le driver JDBC de l'éditeur Source de données : SGBDR Oracle
Client/Serveur mobile : client/serveur distribués	Client : Applet Java (utilisant ORB/RMI) Protocole de communication client/serveur : RMI Serveur d'application : BEA WebLogic Application Server Protocole de communication entre serveur d'application et base de données : protocole propriétaire véhiculant un dialogue SQL Source de données : SGBDR SYBASE

Tableau 1.01: Exemple de serveur

1.3. Conclusion

Dans ce chapitre nous avons pu voir les différents types d'architecture système connus ainsi que leurs caractéristiques. Cette architecture permet de mettre en œuvre des applications multi-niveau. Depuis l'utilisation des Mainframes jusqu'à nos jours, elle a rencontré divers changement du fait des évolutions de la technologie, des besoins et des demandes des utilisateurs et des clients. Avec l'ouverture du réseau sur le monde extérieur le système doit être bien architecturé pour avoir un maximum de sécurité et une facilité de déploiement pour les composants. En effet, un système tournant sur une architecture client/serveur est à la fois pratique et fiable. Et c'est sur ce type d'architecture que va s'appuyer le présent mémoire.

CHAPITRE 2 PLATE-FORME J2EE

2.1. Introduction [1] [3] [4]

La plate-forme J2EE est la proposition de Sun pour le développement et la mise en œuvre d'application d'entreprise multiniveaux.

La plate-forme J2EE s'appuie entièrement sur le langage JAVA et de ce fait bénéficie de ces caractéristiques. Sun caractérise le langage Java comme étant simple, orienté objet, interprété, robuste, sécurisé, portable et multitâche.

J2EE fournit :

- Un modèle de développement de composant web (servlet, JSP) et de composant métier (EJB) sous forme d'API Java
- Un ensemble de service (JDBC, JTA, JNDI, JMS, RMI/IIOP, JavaMail, XML) utile au composant sous forme d'API Java
- Un modèle de création de module web (.war), de module EJB (jar) et de module d'entreprise (.ear) associé à des descripteurs au format XML utile pour le déploiement des applications d'entreprise.

J2EE est avant tout une norme permettant à des sociétés tiers de développer leur propre serveur d'application qui implémente totalement (conformément à la norme) ou partiellement les spécifications de Sun. Aujourd'hui, il existe une quarantaine de serveur d'application sur le marché. Les plus répandus étant IBM WebSphere et BEA WebLogic. Les serveurs d'application Java s'appuient entièrement sur les APIs de la plate-forme J2EE. De ce fait quel que soit l'outil de développement Java utilisé pour le développement de servlet, JSP, EJB, ... on reste quasi indépendant du serveur d'application pour la mise en œuvre.

2.2. Applications multitiers distribuées de J2EE [1] [2] [4]

2.2.1. *Composants J2EE*

Les applications J2EE sont composées de composants. Un composant J2EE contient des unités logicielles fonctionnelles qui sont assemblées dans une application J2EE avec ses propres classes et fichiers et qui communiquent avec d'autres composants. La spécification J2EE définit les composants J2EE suivants :

- Les composants clients et les applets sont des composants qui tournent sur le client.
- Les composants servlets et Java Server Pages (JSP) sont les composants web qui tournent sur un serveur.
- Les composants Enterprise Java Bean (EJB) sont les composants métiers qui tournent sur un serveur.

Les composants J2EE sont écrits en Java et sont compilés de la même manière que tous les autres langages de programmation. La différence entre une classe Java standard et un composant J2EE est que les composants J2EE sont assemblés dans une application J2EE. Ces composants sont vérifiés pour qu'ils soient conformes aux spécifications de J2EE. Ils sont déployés sur un serveur J2EE où ils tournent et sont gérés par ce même serveur.

2.2.2. *Clients J2EE*

Un client J2EE peut être un client web ou une application cliente.

2.2.2.1. Client web

Un client web est divisé en deux :

- Des pages web dynamiques contenant différents types de langages à balise tel que HTML, XML qui sont générées par les composants web tournant sur le niveau Web de l'architecture multitiers.
- Les navigateurs web qui affichent les pages reçues des serveurs.

Un client web est généralement appelé « client léger » du fait qu'il ne traite pas des règles métiers complexe, n'effectue pas des requêtes à la base de données, ou ne se connecte pas aux applications héritées.

2.2.2.2. Applet

Un applet est une petite application écrite en Java et qui s'exécute sur un « Java Virtual Machine» (JVM) installé sur un navigateur web. Cependant les systèmes clients peuvent avoir besoin des plug-ins Java et des politiques de sécurité "de fichier pour que les applets puissent fonctionner correctement.

Les composants web sont les APIs les plus utilisés pour la création des clients web, parce qu'aucun plug-in Java et aucune politique de sécurité de fichier n'est nécessaire sur le système client. Mais aussi les composants web séparent la programmation des applications de la mise en page des applications web.

2.2.2.3. Applications clientes

Une application cliente tourne sur une machine cliente et permet de traiter des tâches qui requièrent des interfaces utilisateurs plus riches que celle proposé par les langages à balises. Typiquement, il a une interface graphique (GUI) créé par les API « Swing » et « Abstract Window Toolkit (AWT) ».

Les applications clientes peuvent directement avoir accès au niveau métier de l'architecture multitiers de J2EE. Aussi, une application cliente peut avoir accès à une servlet située sur le niveau web par l'intermédiaire du protocole HTTP.

2.2.3. *Composants web*

Les composants web sont soit des servlets, soit des Java Server Page (JSP).

- Les Servlets sont des classes Java qui, dynamiquement, traitent les requêtes et construisent les réponses.

- Les JSP s'exécutent de la même manière qu'une servlet, mais permet une approche beaucoup plus naturelle pour la création du contenu statique.

Les pages HTML et les applets sont liés au composant web durant l'assemblage de l'application mais ne sont pas considérés comme des composants web selon la spécification J2EE. Les autres scripts côtés serveur sont aussi liés aux composants web, mais comme les pages HTML, ils ne sont pas considérés comme des composants web.

Comme illustrés sur la figure 2.01 ci-dessous, le niveau web, comme le niveau client, peut inclure des composants JavaBeans, qui sont des classes java spéciales, pour gérer les entrées utilisateurs et envoyer ces entrées vers 1' « enterprise beans » qui tournent sur le niveau métier.

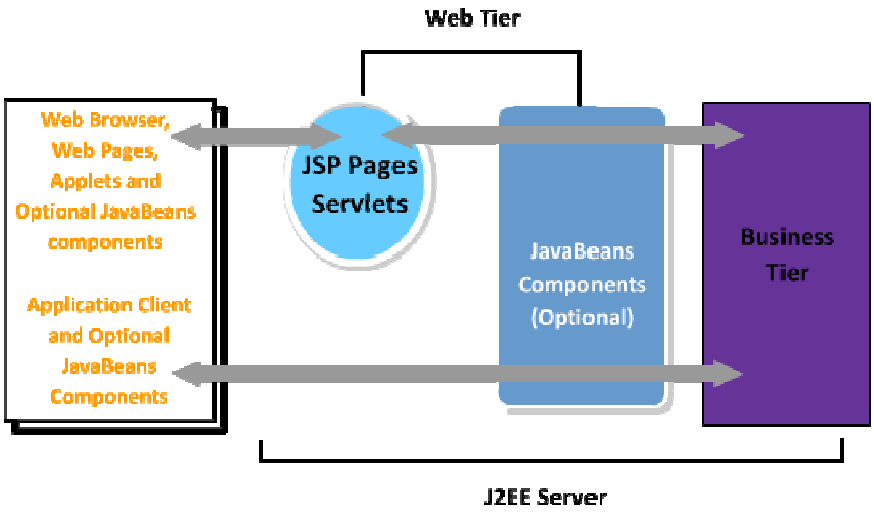


Figure 2.01 : Le niveau web et les Applications j2EE

Les serveurs et le niveau client peuvent aussi inclure des composants basés sur les composants JavaBeans pour gérer le flux de données entre les applications clientes et les applets et les composants tournant sur le serveur J2EE ou entre les composants serveurs et la base de données. Les composants JavaBeans ne sont pas considérés comme un composant J2EE par la spécification J2EE.

Les JavaBeans possèdent des propriétés et ont des méthodes « getproperty » et « setproperty » pour avoir accès à ces propriétés. Les composants JavaBeans utilisés à cet effet sont simples en conception et en implémentation mais devraient se conformer à la convention d'identification et de conception décrite dans l'architecture de composant JavaBean.

2.2.4. *Composants métiers*

Les codes métiers qui résolvent ou répondent au besoin d'un domaine d'activité particulier comme les banques, la finance ou la vente sont traités par 1' « enterprise bean ». Cette dernière tourne sur le niveau métier.

La figure 2.02 ci-dessous illustre comment l'enterprise bean reçoit les données des programmes clients, les traite si nécessaire, et les envoie au système d'information de l'entreprise (EIS) pour la sauvegarde. Une enterprise bean effectue aussi l'opération inverse, il rend les données sauvegardées, les traite si nécessaire, et les envoie au client qui les a demandé.

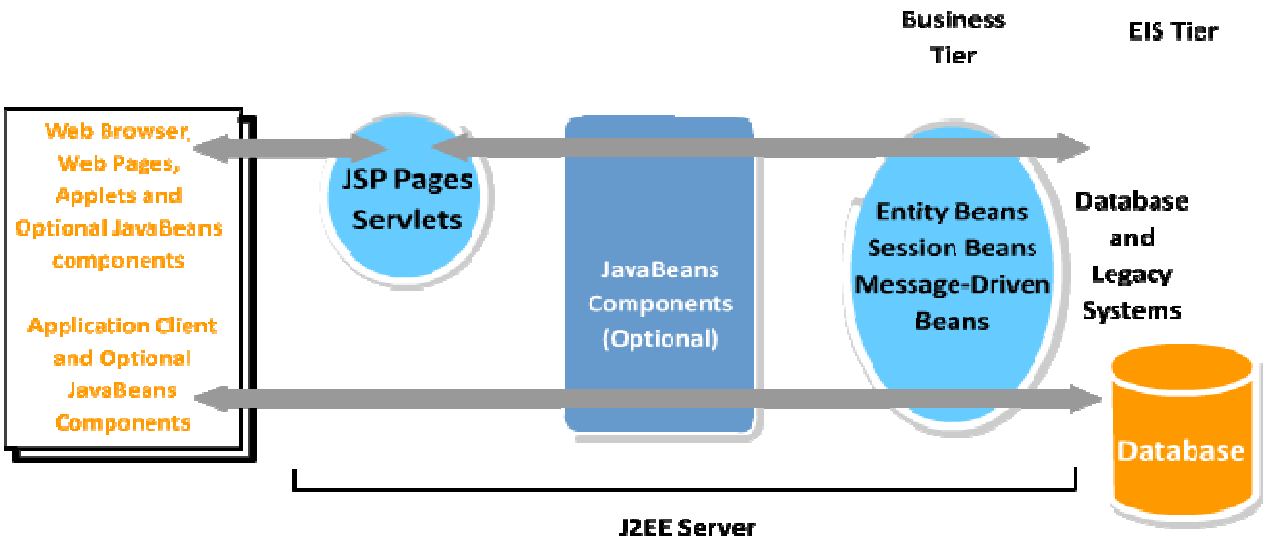


Figure 2.02 : Les niveaux métiers et EIS

Trois types d'entreprise bean ont été constatés:

- Beans session: un bean session représente une conversation transitoire avec un client. Lorsqu'un client fini l'exécution, le bean session et les données sont perdus.
- beans entity: un bean entity représente les données persistantes sauvegardées dans une table de la base de données. Si le client a terminé ou le serveur s'est arrêté, les sous-couches services assurent que les données de l'entité bean soient sauvegardées.
- beans message-driven : un bean message-driven combine les caractéristiques d'un bean session et d'un « message listener » de Java Message Service (JMS), permettant ainsi de recevoir des messages JMS de façon asynchrone.

2.2.5. *Niveau EIS*

Le niveau EIS traite les programmes EIS et renferme les infrastructures systèmes de l'entreprise comme les processus de transaction des mainframes, les systèmes de base de données et d'autres systèmes d'information hérités. Par exemple, une application J2EE peut avoir besoin d'accéder au système d'information de l'entreprise pour une connexion à la base de données.

2.3. Conteneurs J2EE [1] [4]

Normalement le développement d'application multitiers est difficile à réaliser parce qu'il implique beaucoup de code complexe à traiter comme la transaction et le multithreading. L'architecture de

la plate-forme J2EE produit des applications J2EE facile à programmer parce que la logique applicative est organisée de façon à ce que les composants soient réutilisables. De plus, le serveur J2EE produit une sous-couche service sous forme de conteneur pour chaque type de composant.

2.3.1. *Services conteneurs*

Les conteneurs sont les interfaces entre un composant et les fonctionnalités spécifiques de bas niveau de la plate-forme que supporte le composant. Avant qu'un composant web, ou un composant enterprise bean, ou un composant d'application client s'exécute, ils doivent être assemblés dans un module J2EE et déployés dans leur propre conteneur. Le processus d'assemblage implique des paramètres spécifiques des conteneurs dans l'application J2EE et pour l'application J2EE lui même. Les paramètres du conteneur personnalisent les supports sous-jacents fournis par le serveur J2EE incluant les services comme :

- la sécurité : le modèle de sécurité J2EE permet de configurer un composant web ou une enterprise bean, de ce fait les utilisateurs autorisés sont les seuls à avoir accès aux ressources systèmes.
- la gestion des transactions : le modèle de transaction J2EE permet de spécifier les relations entre les méthodes qui composent une simple transaction afin que toutes les méthodes dans une transaction soient traitées comme une seule unité.
- le Java Naming Directory Interface (JNDI) lookup : le service JNDI lookup fournit un service de nommage multiple et de répertoires dans l'entreprise afin que le composant d'application puisse avoir accès au service de nommage et de répertoire.
- la connexion à distance : le modèle de connexion à distance de J2EE gère la connexion de base entre le client et l'enterprise bean. Quand une enterprise bean a été créée, le client appelle une de ces méthodes comme s'il était dans la même machine virtuelle.

L'architecture J2EE fournit des services configurables. Les composants d'application à l'intérieur d'une même application J2EE peuvent se comporter différemment selon le cas où elle a été déployée. Par exemple une enterprise bean qui se connecte à une base de données à un niveau de sécurité spécifique pour un environnement de production et un autre niveau de sécurité pour un autre environnement de production.

2.3.2. *Types de conteneurs*

Le processus de déploiement installe les composants d'application J2EE dans les conteneurs J2EE illustrés par la figure 2.03 ci-dessous.

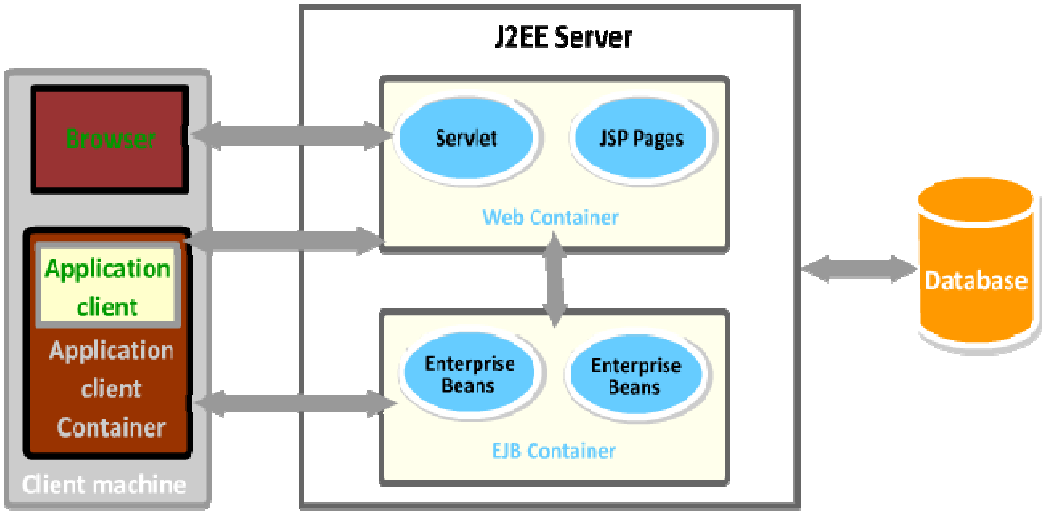


Figure 2.03 : Les serveurs et conteneurs j2EE

2.3.2.1. Serveur J2EE

Un serveur J2EE fournit un conteneur EJB et un conteneur web.

2.3.2.2. Conteneur EJB

Le conteneur EJB gère l'exécution des Enterprise beans pour les applications J2EE. L'Enterprise bean et son conteneur s'exécutent sur le serveur J2EE.

2.3.2.3. Conteneur web

Le conteneur web gère l'exécution des composants JSP et servlet pour les applications J2EE. Les composants web et son conteneur s'exécutent sur le serveur J2EE.

2.3.2.4. Conteneur d'application cliente

Le conteneur d'application cliente gère les composants d'application cliente. L'application cliente et son conteneur s'exécutent sur le client.

2.3.2.5. Conteneur d'applet

Le conteneur d'applet gère l'exécution des applets. Cela concerne l'exécution des navigateurs web et des plug-ins Java sur le client en même temps.

2.4. API de J2EE [1] [3] [4] [5]

La figure 2.04 ci-dessous illustre les API de la plate-forme J2EE 1.4 dans chaque conteneur J2EE

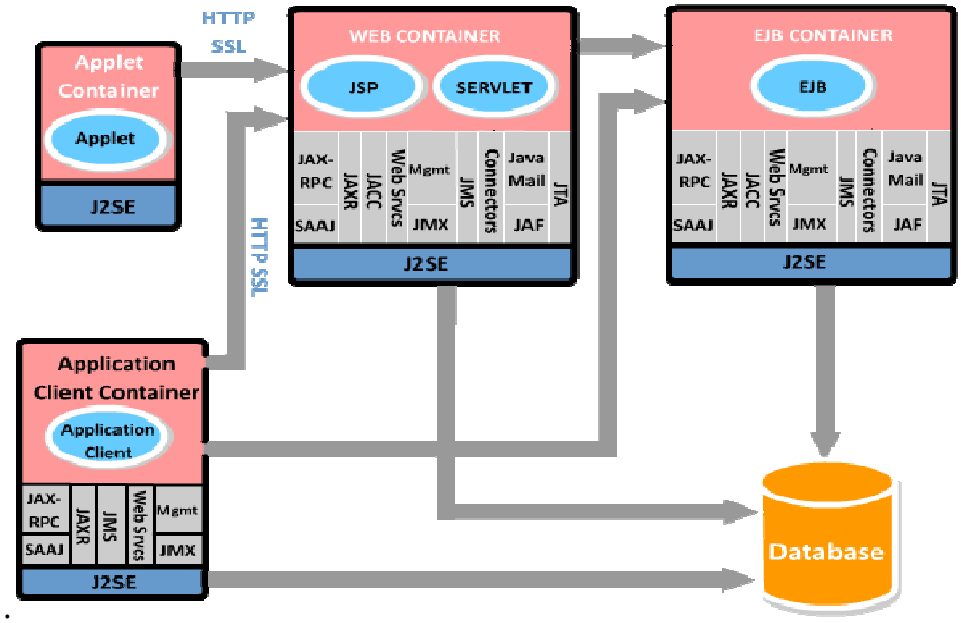


Figure 2.04 : Les APIs de la plate-forme J2EE

2.4.1. *Technologie Enterprise JavaBeans*

Un composant enterprise JavaBeans ou enterprise beans, est un ensemble de code possédant des champs et des méthodes pour implémenter le module du logique métier. Une enterprise bean est comme un bloc qui peut être utilisé seul ou avec d'autre enterprise beans qui exécutent le logique métier dans le serveur J2EE.

2.4.2. *Technologie Java Servlet*

La technologie Java servlet permet de définir les classes servlet spécifique à HTTP. Une classe servlet étend les capacités des serveurs qui hébergent les applications et qui sont accédées par le modèle de programmation requête-réponse. Quoique les servlets puissent répondre à n'importe quel type de requête, elles sont communément employées pour étendre les hébergeurs d'applications par les serveurs web.

2.4.3. *Technologie JavaServer Page*

La technologie JavaServer Page permet d'inclure des snippets de code servlet directement dans un document basé texte. Une page JSP est un document basé texte qui contient deux types de texte :

- Les données statiques qui peuvent être exprimées par n'importe quel format de document basé texte tel que l'HTML, le WML, le XML.

- Les éléments JSP qui détermineront le contenu dynamique de la page

2.4.4. *API Java Message Service*

L'API Java Message Service est un service de messagerie standard qui permet au composant d'application J2EE de créer, d'envoyer, de recevoir et de lire des messages. Cela permet la communication distribuée qui est sûre, et asynchrone.

2.4.5. *API JDBC*

L'API JDBC permet d'invoquer des commandes SQL à l'intérieur des programmes Java. On utilise L'API JDBC par les servlets et les JSP pour avoir un accès direct à la base de données sans passer par l'enterprise bean.

2.4.6. *API Java Transaction*

L'API Java Transaction permet de mettre en place une gestion des transactions dans des applications distribuées en partageant une même transaction entre plusieurs composants EJB, et/ou en partageant une même transaction entre différentes connexions de base de données

2.4.7. *JNDI*

Cet API permet l'accès à des services de nommage ou d'annuaire (LDAP) fournissant une implémentation de l'interface SPI.

2.4.8. *API JavaMail*

L'API JavaMail fournit des fonctionnalités de gestion de courrier électronique aux applications Java. L'API JavaMail a deux parties :

• L'interface niveau application utilisée par le composant d'application pour envoyer le mail,

• L'interface fournisseur de service.

La plate-forme J2EE inclut au JavaMail un fournisseur de service qui permet au composant d'application d'envoyer un mail par Internet.

2.5. Serveurs d'applications [2]

2.5.1. *Rôles d'un serveur d'application*

Le rôle d'un serveur d'application est de mettre en œuvre des applications distribuées, fabriquées à base de composant Java (servlet, JSP, EJB) et de les rendre accessible à des clients web (navigateur) et à des applications d'entreprises écrites en Java.

Le serveur doit prendre en charge la création et le chargement en mémoire des instances de composant et la gestion d'une file d'attente pour satisfaire aux requêtes des clients. De plus, pour satisfaire aux exigences des applications d'entreprises le serveur d'application doit être performant et fiable. Il est donc capable de gérer la disponibilité (scalability) des applications (gestion des montées en charge, tolérances de panne) en mettant en place des fermes (cluster) de serveur.

Le serveur s'occupe également de l'implémentation de différents services utiles aux composants et aux fonctionnements de l'application :

- Service de nommage
- Service de gestion des transactions
- Service de gestion de la disponibilité des applications (montée en charge, tolérance de panne)
- Service de sécurité
- Service d'administration
- Service d'accès aux données
- Service de gestion des messages

2.5.2. *Intégration dans l'architecture d'entreprise*

La plupart des serveurs d'application du marché sont implémentés partiellement voire totalement en Java. De ce fait il bénéficie des caractères de la plate-forme :

- Les différents serveurs sont disponibles pour de nombreux système d'exploitation Windows 2000-2003, AIX d'IBM, SOLARIS, HPUX, LINUX, NetWare de Nobel…
- Ils peuvent fonctionner avec les principaux serveurs web du marché : Apache Web Server, Netscape Enterprise Server, Lotus Domino, Microsoft Internet Information Service …

- Ils peuvent communiquer avec de nombreux serveurs de ressource de données SGBDR : ORACLE, IBM DB2, Microsoft SQL SERVER, et d'une manière générale tous les SGBDR disposant d'un pilote JDBC

- Annuaire LDAP, serveur de messagerie synchrone/asynchrone, SPP, serveur transactionnel, Mainframe

2.5.3. *Architecture externe d'un serveur d'application*

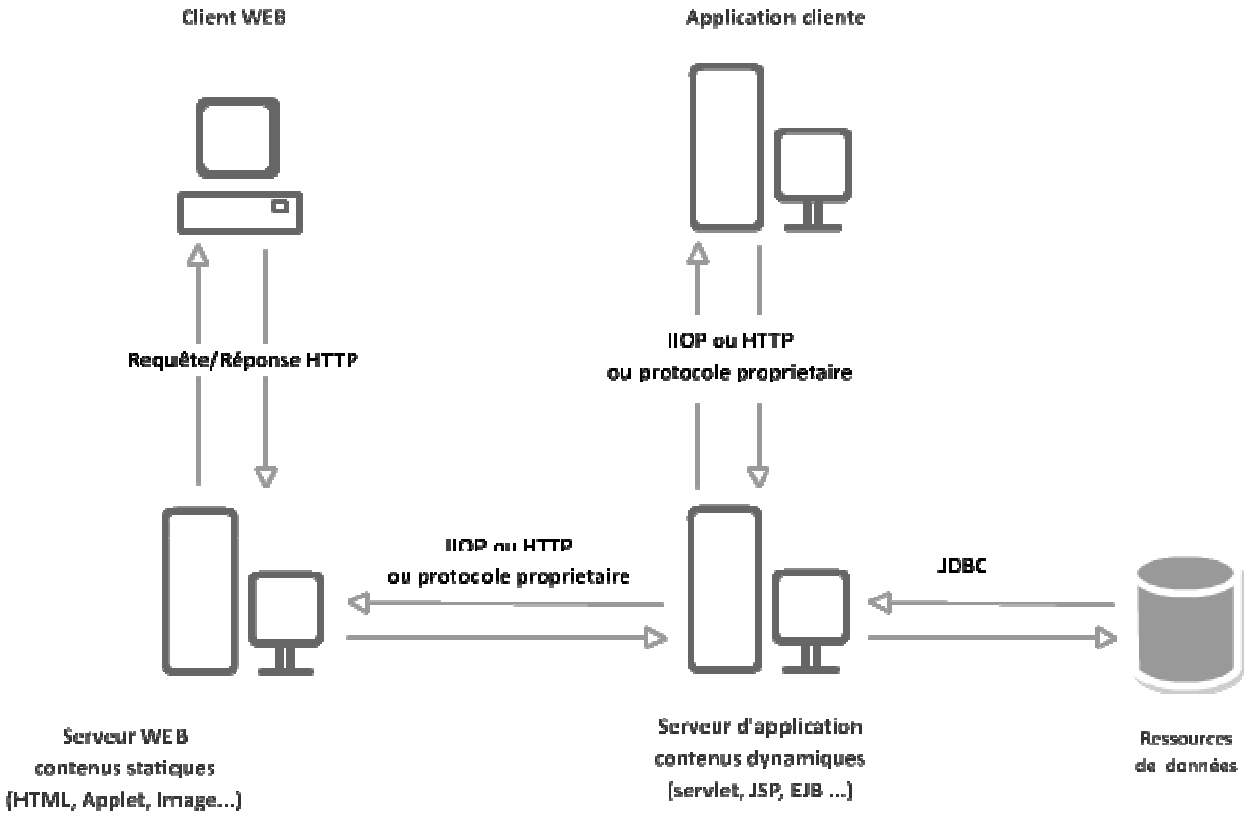


Figure 2.05 : Architecture externe d'un serveur d'application

Le schéma présente l'architecture matérielle typique à mettre en œuvre autour d'un serveur d'application ainsi que différentes relations possibles entre les tiers :

- Le serveur d'application contient et s'occupe de la mise en œuvre de la partie dynamique des applications composées de Servlet, JSP, EJB. Les composants peuvent accéder à différentes sources de données.

- Le serveur web (moteur web du serveur d'application) contient et s'occupe de la mise en œuvre de la partie statique des applications composées des pages HTML, des images, des Applets …

2.5.4. *Architecture interne d'un serveur d'application*

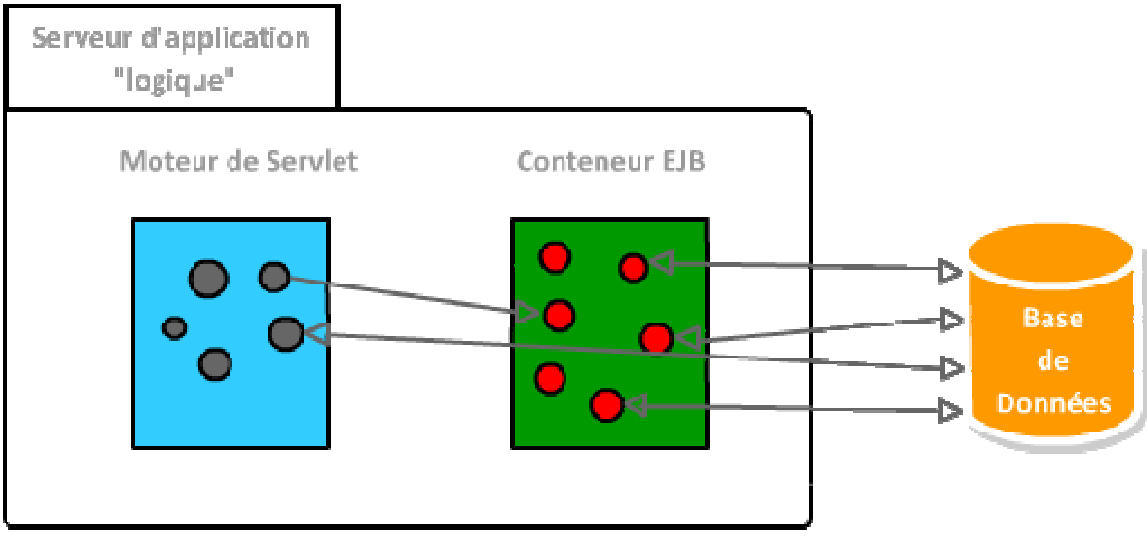


Figure 2.06 : Architecture interne d'un serveur d'application

Sur un serveur d'application (machine physique) il peut être installé et configuré plusieurs serveurs d'application logique. Chaque serveur d'application logique peut contenir des éléments : un conteneur web appelé moteur Servlet/JSP et un conteneur EJB.

Un serveur d'application est composé également d'un ensemble de service utile au fonctionnement des conteneurs et des composants. L'un des plus important est notamment le service de nommage JNDI. La plupart des serveurs d'application sont composés d'un conteneur web et d'un conteneur EJB. C'est notamment le cas des serveurs commerciaux IBM WebSphere Application Server et de BEA WebLogic.
Malgré tout, de nos jour des solutions open source ne proposent qu'un des deux types de conteneur tel que : le conteneur web Apache Tomcat ou le conteneur d'EJB JBOSS.

2.6. Communication entre le client et le serveur : le protocole HTTP [2]

Dans une application web, la communication entre l'application web et le serveur s'établit selon le protocole TCP/IP qui est chargé du routage de l'information d'un site à un autre. Le transite des informations (principalement des données au format HTML ou XML) s'effectue selon des protocoles http qui définissent comment les données sont transmises entre le client et le serveur via le protocole TCP/IP.

Le protocole http définit qu'une requête http est obligatoirement associée à une réponse http. Dans une application web si la Servlet/JSP ne renvoie pas explicitement au premier temps à la requête envoyée par le client, le client réagit en envoyant la réponse sous forme d'une page vierge.

2.7. Développement client [2]

Il existe deux types de clients et donc deux types de développement :

- Le développement du client léger

  Aussi appelé client web car le contexte d'exécution de l'application est un navigateur web. Il envoie une requête HTTP à destination du serveur. Idéalement les applications web cliente sont composées de page html contenant des images, des liens hypertextes, champ de formulaire et code JavaScript qui peut être utilisé pour contrôler des formulaires ou effectuer des traitements. Si la ressource concerne un élément dynamique, un filtre installé et configuré sur le serveur web s'occupe du transfert de la requête vers le serveur d'application.

  Java peut être également utilisée dans les applications web client, avec le développement d'Applet. L'applet présente des avantages et des inconvénients mais d'une manière générale celui-ci est utilisé comme dernier recours lorsque html n'apportera des solutions satisfaisantes. Ce qui est intéressant avec applet c'est qu'elle permet d'intégrer un client lourd au sein d'un client léger.

  Les fonctionnalités sont décuplées car on peut dans ce cas faire du client/serveur 2/3 en détournant les technologies web, mais en contre partie l'application consomme plus en ressource et en temps de traitement.

- Le développement du client lourd

  Le «client lourd » (application ou Applet graphique Java) peut communiquer avec des Servlets et JSP distant en utilisant une connexion http, mais ceci n'est pas très souvent mis en œuvre dans les développements. Le client lourd idéalement composé de l'interface graphique et de la gestion des événements ou de la commande utilisateur peut également déléguer tous les traitements à des composants métiers distants. Pour cela, il peut rechercher et invoquer des EJB contenus sur le serveur d'application par l'intermédiaire de service de nommage du serveur d'application

  Il peut être intéressant de savoir qu'il existe quelques passerelles permettant à des clients lourds autres que Java (VB, C++ …) de communiquer avec des EJB.

2.8. Développement serveur en Java [2]

Il consiste à écrire des composants web et des commandes composants métiers. Le développement d'application J2EE doit respecter un certain architecture afin d'optimiser la création et la maintenance des applications. SUN recommande de développer en respectant l'architecture MVC (Model View Controller) dans lequel, chaque composant à un rôle bien délimité, ce qui permet de séparer la partie logique de la partie présentation dans l'application L'architecture MVC proposé par SUN est aujourd'hui la seule solution de développement web côté serveur permettant de séparer la partie logique métier de la partie présentation dans une application web. Ceci est très important car elle permet au web développeur et au web designer de travailler séparément chacun sur ses fichiers composants au sein d'un projet.

Dans l'architecture MVC :

- Le modèle est représenté par les EJB et/ou les JavaBeans
- La vue est représentée par les JSP
- Le contôleur est représenté par les servlets

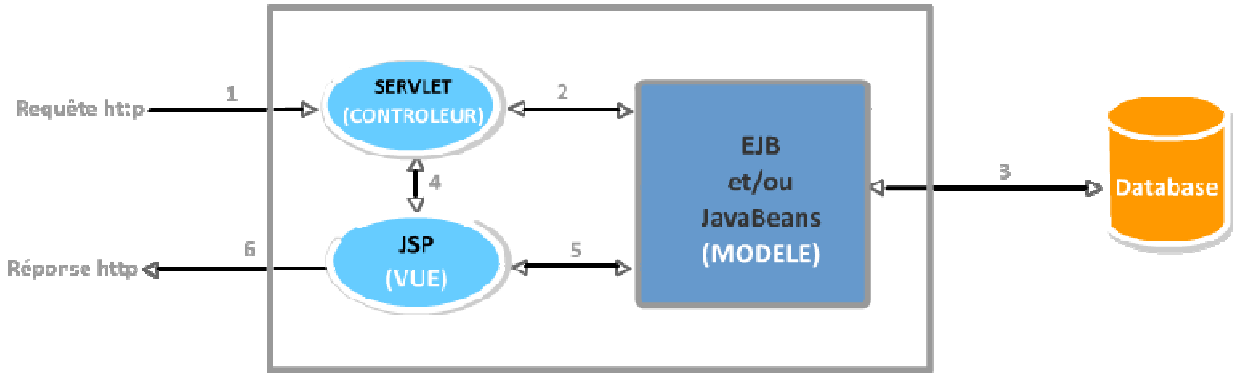


Figure 2.07 : Architecture MVC

2.9. Conclusion

Ce chapitre nous a permis de connaitre la solution offerte par SUN Microsystem pour le développement d'application d'entreprise. A la fois simple, fiable et sécuritaire, la plate-forme J2EE s'appuie sur l'architecture système multi niveau. Elle fournit des composants tournant dans des serveurs bien spécifiés: le serveur HTTP pour stocker les pages Web statiques, le conteneur Web pour exécuter les pages Web dynamiques grâce aux JSP et aux Servlets, le conteneur d'EJB pour exécuter le code métier et le niveau EIS traitant la base de données. Le but de cette répartition est de simplifier le développement des applications.

CHAPITRE 3 BASES DE DONNEES

3.1. Présentation générale [5] [6] [8]

3.1.1. *Introduction aux bases de données*

Une base de données est un ensemble structuré de données enregistrées avec le minimum de redondance pour satisfaire simultanément plusieurs utilisateurs de façon sélective en un même temps opportun. Dans cette approche la partie de structuration et de description des données sont unifiées et séparées par des programmes d'application.

Il semble plus facile de définir l'outil principal de gestion d'une base de données : le système de gestion de bases de données (SGBD).

- C'est un outil permettant d'insérer, de modifier et de rechercher efficacement des données spécifiques dans une grande masse d'informations.

- C'est une interface entre les utilisateurs et la mémoire secondaire facilitant le travail des utilisateurs en leur donnant l'illusion que toute l'information est comme ils le souhaitent. Chacun doit avoir l'impression qu'il est seul à utiliser l'information. Le SGBD est composé de trois couches successives :

- Le système de gestion de fichiers. Il gère le stockage physique de l'information. Il est dépendant du matériel utilisé (type de support, facteur de blocage, etc ...).

- Le SGBD interne. Il s'occupe du placement et de l'assemblage des données, la gestion des liens et gestion de la rapidité d’accès.

- Le SGBD externe. Il s'occupe de la présentation et de la manipulation des données aux concepteurs et utilisateurs. Il s'occupe de la gestion de langages de requêtes élaborées et des outils de présentation (états, formes, etc.).

3.1.2. *Objectifs et avantages*

Un système d'information peut toujours être réalisé sans outil spécifique. Quels sont les objectifs et avantages de l'approche SGBD par rapport aux fichiers classiques ?

La réponse tient en neuf points fondamentaux :

3.1.2.1. Indépendance physique.

Les disques, la machine, les méthodes d'accès, les modes de placement, les méthodes de tris, le codage des données ne sont pas apparents. Le SGBD offre une structure canonique permettant la représentation des données réelles sans se soucier de l'aspect matériel.

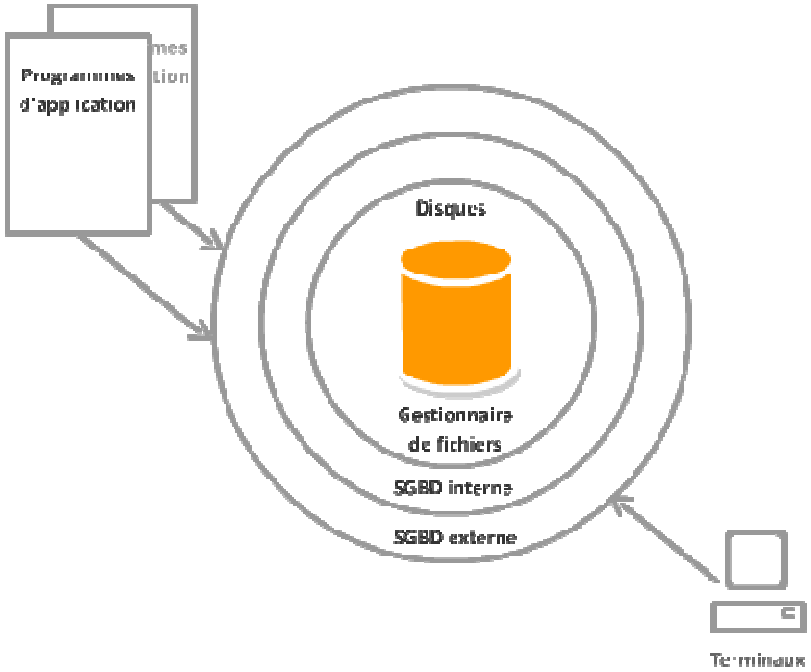


Figure 3.01 : Le modèle à 3 couches

3.1.2.2. Indépendance logique.

Chaque groupe de travail doit pouvoir se concentrer sur ce qui l'intéresse uniquement. Il doit pouvoir arranger les données comme il le souhaite même si d'autres utilisateurs ont une vue différente. L'administrateur doit pouvoir faire évoluer le système d'informations sans remettre en cause la vue de chaque groupe de travail.

3.1.2.3. Manipulable par des non-informaticiens.

Le SGBD doit permettre d'obtenir les données par des langages non procéduraux. Il suffit de décrire l'information voulue.

3.1.2.4. Accès aux données efficace.

Les accès disque sont lents relativement à l'accès à la mémoire centrale. Il faut donc offrir les meilleurs algorithmes de recherche de données à l'utilisateur.

Remarque: le système de gestion de fichiers y répond parfois pour des mono-fichiers (ISAM, VSAJV1 etc..) mais dans le cas d'intercroisements entre différents fichiers cela devient beaucoup plus complexe et parfois même contextuel à la recherche effectuée.

3.1.2.5. Administration centralisée des données.

Le SGBD doit offrir aux administrateurs des données des outils de vérification de cohérence des données, de restructuration éventuelle de la base, de sauvegarde ou de réplication. L'administration est centralisée et est réservée à un très petit groupe de personnes pour des raisons de sécurité.

3.1.2.6. Non redondance des données.

Le SGBD doit permettre d'éviter la duplication d'informations qui, outre la perte de place mémoire, demande des moyens humains importants pour saisir et maintenir à jour plusieurs fois les mêmes données.

3.1.2.7. Cohérence des données.

Cette cohérence est obtenue par la vérification des contraintes d'intégrité. Une contrainte d'intégrité est une contrainte sur les données de la base, qui doit toujours être vérifiée pour assurer la cohérence de cette base. Les systèmes d'information sont souvent remplis de telles contraintes, le SGBD doit permettre une gestion automatique de ces contraintes d'intégrité sur les données. Par Exemple :

- Un identifiant doit toujours être saisi.
- Le salaire doit être compris entre 4000F et 100000F.
- Le nombre de commandes du client doit correspondre avec le nombre de commandes dans la base.
- L'emprunteur d'un livre doit être un abonné du club.

Dans un SGBD les contraintes d'intégrité doivent pouvoir être exprimées et gérées dans la base et non pas dans les applications.

3.1.2.8. Partageabilité des données.

Le SGBD doit permettre à plusieurs personnes (ou applications) d'accéder simultanément aux données tout en conservant l'intégrité de la base. Chacun doit avoir l'impression qu'il est seul à utiliser les données.

3.1.2.9. Sécurité des données.

Les données doivent être protégées des accès non autorisés ou mal intentionnés. Il doit exister des mécanismes permettant d'autoriser, contrôler et enlever des droits d'accès à certaines informations pour n'importe quel usager. Par exemple un chef de service pourra connaître les salaires des personnes qu'il dirige, mais pas de toute l'entreprise. Le système doit aussi être tolérant aux pannes: si une coupure de courant survient pendant l'exécution d'une opération sur la base, le SGBD doit être capable de revenir à un état dans lequel les données sont cohérentes.

Remarque : ces neuf points, bien que caractérisant assez bien ce qu'est une base de données, ne sont que rarement réunis dans les SGBD actuels. C'est une vue idéale des SGBD. De plus les SGBD sont interfacés à l'aide d'un langage unique : le SQL. Ce langage permet d'effectuer l'ensemble des opérations nécessaires sur la base de données, et permet aussi la gestion de transaction. Une transaction est définie par 4 propriétés essentielles : ACID.

- Atomicité
- Cohérence
- Isolation
- Durabilité

Ces propriétés garantissent l'intégrité des données dans un environnement multiutilisateur :

- L'atomicité permet à la transaction d'avoir un comportement indivisible ; soit toutes les modifications sur les données dans la transaction sont effectuées, soit aucune n'est réalisée. L'intérêt de ce concept peut être expliqué par l'exemple d'une transaction débitant un compte A et créditant un compte B : il est clair que la transaction n'a de sens que si les deux opérations sont menées à leur termes.
- Cohérence : l'atomicité de la transaction garantit que la base de données passera d'un état cohérent à un autre état cohérent. La cohérence des données de la base est donc permanente.
- Isolation : l'isolation des transactions signifie que les modifications effectuées au cours d'une transaction ne sont visibles que par l'utilisateur qui effectue cette transaction. Au cours de la transaction l'utilisateur pourra voir des modifications en cours qui rendent la base apparemment incohérente mais ces modifications ne sont pas visibles par les autres et ne le seront qu'à la fin de la transaction si celle-ci est correcte.

- Durabilité : la durabilité garantit la stabilité de l'effet d'une transaction dans le temps, même en cas de problème grave tel que la perte d'un disque dur.

3.1.3. *Différents types de bases de données*

Il existe actuellement 5 grands types de bases de données :

- Les bases hiérarchiques.
  Ce sont les premiers SGBD apparus (notamment avec IMS d'IBM). Elles sont constituées d'une gestion de pointeurs entre les enregistrements. Le schéma de la base doit être arborescent.

- Les bases réseaux.
  Sans doute les bases les plus rapides, elles ont très vite supplanté les bases hiérarchiques dans les années 70 (notamment avec IDS II d' IBM). Ce sont aussi des bases navigationnelles qui gèrent des pointeurs entre les enregistrements. Cette fois-ci le schéma de la base est beaucoup plus ouvert.

- Les bases relationnelles. A l'heure actuelle les plus utilisées. Les données sont représentées en tables. Elles sont basées sur l'algèbre relationnelle et un langage déclaratif (généralement SQL).

- Les bases déductives. Les données sont aussi représentées en tables (prédicats), le langage d'interrogation se base sur le calcul des prédicats et la logique du premier ordre.

- Les bases objets.
  Les données sont représentées en tant qu'instances de classes hiérarchisées. Chaque champ est un objet. De ce fait, chaque donnée est active et possède ses propres méthodes d'interrogation et d'affectation. L'héritage est utilisé comme mécanisme de factorisation de la connaissance.

La répartition du parc des SGBD n'est pas équitable entre ces 5 types de bases. 75% sont relationnelles, 20% réseaux, les 5% restants étant partagés entre les bases déductives et objets.
Ces chiffres risquent néanmoins d'évoluer d'ici quelques années et la frontière entre les bases relationnelles et objets risque d'être éliminée par l'introduction d'une couche objets sur les bases relationnelles.

3.1.4. *Quelques systèmes existants*

- Oracle (http://www.oracle.com), DB2 (IBM, http://www.software.ibm.com), Ingres, Informix, Sybase (http://www.sybase.com), SQLServer (http://www.microsoft.com), O2, Gemstone

- Et sur micros: Access 97 (http://www.microsoft.com), Paradox 8.0 (http://www.corel.com), Visual Dbase (http://www.borland.com), FoxPro (http://www.microsoft.com), fileMaker 4.0 (http://www.claris.fr), 4D, Windev (http://www.pcsoft.fr). Il existe aussi quelques sharewares que l'on peut trouver sur Internet pour s'initier aux bases de données relationnelles comme MySQL (http://web.tryc.on.ca/mysql/), MSQL (http://Hughes.com.au/), Postgres (http://www.postgresql.org) ou InstantDB (http://www.instantdb.co.uk) qui est entièrement écrit en Java.

3.1.5. *Modéliser les données*

Avant de s'attaquer à tout problème, il est toujours nécessaire de réfléchir profondément aux tenants et aboutissants de ce que l'on veut réaliser. La phase de conception nécessite souvent de nombreux choix qui auront parfois des répercussions importantes par la suite. La conception de bases de données ne fait pas exception à la règle. Les théoriciens de l'information ont donc proposé des méthodes permettant de structurer sa pensée et présenter de manière abstraite le travail à réaliser. Ces méthodes ont donné naissance à une discipline : l'analyse, et un métier : l'analyste.

L'analyse est la discipline qui étudie et présente de manière abstraite le travail à effectuer. La phase d'analyse est très importante puisque c'est elle qui sera validée par les utilisateurs avant la mise en œuvre du système concret. Il existe de nombreuses méthodes d'analyse (AXIAL, OMT, Merise, UML, etc.). Merise sépare les données et les traitements à effectuer avec le système d'information en différents modèles conceptuels et physiques. Celui qui nous intéresse particulièrement ici est le MCD.
Le MCD (Modèle Conceptuel de Données) est un modèle abstrait de la méthode Merise permettant de représenter l'information d'une manière compréhensible aux différents services de l'entreprise. Il permet une description statique du système d'informations à l'aide d'entités et d'associations. Le travail de conception d'une base de données par l'administrateur commence juste après celui des analystes qui ont établi le MCD.

Commençons par quelques définitions propres au MCD:

- **La propriété** est une donnée élémentaire et indécomposable du système d'information. Par exemple une date de début de projet, la couleur d'une voiture, une note d'étudiant.

- **L'entité** est la représentation dans le système d'information d'un objet matériel ou immatériel ayant une existence propre et conforme aux choix de gestion de l'entreprise. L'entité est composée de propriétés. Par exemple une personne, une voiture, un client, un projet.

- **L'association** traduit dans le système d'information le fait qu'il existe un lien entre différentes entités. Le nombre d'intervenants dans cette association caractérise sa dimension :
  - réflexive sur une même entité
  - binaire entre deux entités,
  - ternaire entre trois entités,
  - n-aire entre n entités.

- **Les cardinalités** maximales sont nécessaires pour la création de la base de données, les cardinalités minimales sont nécessaires pour exprimer les contraintes d'intégrités. Remarque: il existe une notation "à l'américaine" dans laquelle on ne note que les cardinalités maximum.
  Un lien hiérarchique est un lien 1:n en notation américaine.
  Un lien maillé est un lien n:m en notation américaine.

  Des propriétés peuvent être attachées aux associations.
  Les cardinalités caractérisent le lien entre une entité et une association. La cardinalité d'une association est constituée d'une borne minimale et d'une borne maximale :

  - minimale : nombre minimum de fois qu'une occurrence d'une entité participe aux occurrences de l'association, généralement 0 ou 1.

  - maximale : nombre maximum de fois qu'une occurrence d'une entité participe aux occurrences de l'association, généralement **1 ou n.**

- **Identifiant** : l'identifiant d'une entité est constitué d'une ou plusieurs propriétés de l'entité telles qu'à chaque valeur de l'identifiant corresponde une et une seule occurrence de l'entité.

  L'identifiant d'une association est constitué de la réunion des identifiants des entités qui participent à l'association.

- **Les règles de gestion** : en réaction à 1 ou plusieurs événements externes ou à la faveur d'un événement temporel s'accomplit une action ou un ensemble d'action, Celle-ci représente les règles de gestion relatives aux activités. Les règles de gestion sont numérotées RG1, RG2 …

- **Les règles de calcul** sont les formules existantes dans l'entreprise. Une propriété calculée est souvent considérée comme redondante. De ce fait, toutes propriétés déductibles d'autres propriétés ne devraient pas être représentées et ainsi ne plus figurées dans un même objet.

3.2. Algèbre relationnelle [5] [6]

L'algèbre relationnelle a été introduite par Codd en 1970 pour formaliser les opérations sur les ensembles. C'est une méthode d'extraction permettant la manipulation des tables et des colonnes. Ces principes reposent sur la création de novelles tables à partir des tables existantes. Les nouvelles tables devenant des objets utilisables immédiatement. Les opérateurs de l'algèbre relationnelle permettant de créer les tables résultantes sont liés sur la théorie des ensembles.
Il existe deux familles d'opérations : les opérations ensemblistes et les opérations unaires.

3.2.1. *Opérations de base*

Trouver l'ensemble des opérations de base consiste à trouver un ensemble minimal d'opérations au sens où aucune d'entre elles ne peut s'écrire par combinaison des autres. Il existe plusieurs ensembles minimaux pour l'algèbre relationnelle. Celui que nous présentons se base sur 3 opérations ensemblistes et 2 opérations unaires.

- **L'union** de deux relations R et S de même schéma est une relation T de même schéma contenant l'ensemble des *tuples* appartenant à R, à S ou aux deux. Noter : T = R U S

- La **différence** entre deux relations R et S de même schéma dans l'ordre (R - S) est la

relation T de même schéma contenant les *tuples* appartenant à R et n'appartenant pas à S. Noter : T = R - S

- Le **produit cartésien** de deux relations R et S de schéma quelconque est une relation T ayant pour attributs la concaténation des attributs de R et de S et dont les *tuples* sont constitués de toutes les concaténations d'un *tuple* de R à un *tuple* de S. Noter T = R * S

- La **projection** d'une relation R de schéma (A_1, A_2, …, A_n) sur les attributs A_{i1}, A_{i2},…, A_{ip} (avec ij ≠ ik et p < n) est une relation R' de schéma (A_{i1}, A_{i2}, …, A_{ip}) dont les *tuples* sont obtenus par élimination des attributs de R n'appartenant pas à R' et par suppression des doublons. Noter T = Π x_i,… , x_n(R)

- La **restriction** (ou sélection) de la relation R par une qualification Q est une relation R' de même schéma dont les *tuples* sont ceux de R satisfaisant la qualification Q.
  La qualification Q peut être exprimée à l'aide de constantes, comparateurs arithmétiques (< ; ≤ ; > ; ≥ ; = ; ≠) et opérateurs logiques. Noter T = CJQ(R)

Les cinq opérations précédentes (union, différence, produit, projection, restriction) forment un ensemble cohérent et minimal. Aucune d'entre-elles ne peut s'écrire sans l'aide des autres. A partir de ces cinq opérations élémentaires, d'autres opérations peuvent être définies.

3.2.2. *Expressions de l'algèbre relationnelle*

Une fois les opérateurs relationnels identifiés il est alors facile de combiner ces opérations élémentaires pour construire des expressions de l'algèbre relationnelle. Ceci permet de fournir les réponses à des questions complexes sur la base.

3.2.3. *Quelques remarques sur l'algèbre relationnelle*

- L'algèbre relationnelle permet l'étude des opérateurs entre eux (commutativité, associativité, groupes d'opérateurs minimaux etc ...). Cette étude permet de démontrer l'équivalence de certaines expressions et de construire des programmes d'optimisation qui transformeront toute demande en sa forme équivalente la plus efficace.

- D'une manière pratique, les opérations les plus utilisées sont la projection, la restriction et la jointure naturelle.

- L'opération de jointure est en général très coûteuse. Elle est d'ailleurs proportionnelle au nombre de *tuples* du résultat et peut atteindre m * n *tuples* avec m et n les nombres de *tuples* des deux relations jointes.

- Avant d'avoir des requêtes efficaces en temps il est toujours préférable de faire des restrictions le plus tôt possible avant de manipuler des tables les plus réduites possibles.

- Les opérations de l'algèbre relationnelle sont fidèles à certaines lois algébriques : commutativité, associativité etc.... pour peu que l'ordre des colonnes soit sans importance. C'est pourquoi les colonnes sont toujours nommées.

3.3. Langage SQL [2] [5] [6] [7] [8] [9]

SQL est un langage de définition et de manipulation de bases de données relationnelles. Son nom est une abréviation de « Structured Query Language » (langage d'interrogation structuré). SQL est un standard qui a été normalisé par l'organisme ANSI. Il existe pour chaque produit SQL réalisé des différences ou des compléments par rapport à la norme.

Historiquement, après la découverte du modèle relationnel par E.F. Codd en 1970, plusieurs langages relationnels sont apparus dans les années suivantes dont SEQUEL d'IBM, puis SEQUEL/2 pour le System/R d'IBM en 1977, langage qui donna naissance à SQL. L'année 1981 a vu la sortie du premier SGBD relationnel implémentant SQL, le système Oracle. Ce n'est qu'en 1983 qu'IBM sortit DB2, héritier du System/R, lui aussi avec un langage de type SQL. Le langage a ensuite été normalisé en 1986 pour donner SQL/86 puis légèrement modifié en 1989 pour donner SQL/89, la version la plus utilisée actuellement. En 1992 de nombreuses améliorations ont été apportées à la norme pour donner SQL/92 que l'on nomme généralement SQL2, beaucoup plus verbeuse que la précédente. Actuellement une version SQL3 est en cours de rédaction. Elle intègre une couche objets supplémentaire mais n'est pas encore reconnue comme norme à la date d'aujourd'hui.

SQL contient un langage de définition de données, le **DDL**, permettant de créer, modifier ou supprimer les définitions des tables de la base par l'intermédiaire des ordres CREATE, DROP, et ALTER.

Il contient aussi un langage de manipulation de données, le **DML**, par l'intermédiaire des ordres SELECT, INSERT, UPDATE, DELETE. Il contient enfin un langage de gestion des protections

d'accès aux tables en environnement multi-utilisateurs par l'intermédiaire des ordres GRANT, REVOKE, le **DCL**.

DDL	DML	*DCL*
ALTER CREATE COMMENT DESCRIBE DROP RENAME	DELETE INSERT SELECT UPDATE	GRANT REVOKE

Tableau 3.01 : Ordres SQL principaux

Une requête SQL peut être utilisée de manière interactive ou incluse dans un programme d'application (quel que soit le langage).

Toutes les instructions SQL se terminent par un point-virgule (;). Un commentaire peut être introduit dans un ordre SQL par les signes /* et */ ou par le caractère % qui traite toute la fin de ligne comme commentaire.

3.3.1. *L'obtention des données*

L'obtention des données se fait exclusivement par l'ordre SELECT. La syntaxe minimale de cet ordre est :

SELECT <liste des noms de colonnes> FROM <Iiste des noms de tables> ;

Nous verrons qu'elle peut être enrichie, de très nombreuses clauses permettant notamment d'exprimer les projections, les restrictions, les jointures, les tris etc ...

FROM
WHERE
GROUP BY
HAVING
ORDER BY

Tableau 3.02 : Ordres des clauses du SELECT

3.3.1.1. Expression des projections

La projection d'une table en vue de l'obtention d'un ensemble de colonnes de cette table se fait simplement en spécifiant les noms des colonnes désirées. Pour obtenir toutes les colonnes d'une table le caractère '*' peut être utilisé avantageusement à la place de la liste des noms de colonnes.

3.3.1.2. Expression des restrictions

La restriction s'exprime à l'aide de la clause WHERE ajoutée à la requête juste après la liste des tables utilisées. D'autres prédicats peuvent encore être utilisés. Ils servent à définir des ensembles ou des valeurs approximatives. Ce sont les prédicats IN, BETWEEN, LIKE qui peuvent tous trois être préfixés par NOT pour exprimer la négation.

Le test de valeurs manquantes est aussi possible. Une valeur manquante est représentée par une valeur spéciale notée NULL. Le test d'une valeur NULL ne peut pas se faire avec l'égalité classique mais doit se faire obligatoirement avec les prédicats IS NULL ou IS NOT NULL.

3.3.1.3. Tri et présentation des résultats

SQL permet de trier les tuples obtenus par la requête selon différents critères grâce à la clause ORDER BY. Les mots clés ASC et DESC permettent de préciser si le tri est croissant ou décroissant.

Par défaut, les noms des colonnes de la table résultat de la requête sont les noms des colonnes spécifiées dans la liste des colonnes du SELECT. Il est néanmoins possible de changer le nom de la colonne à l'affichage en accolant au nom de colonne choisi le prédicat AS suivi du nouveau nom de colonne devant apparaître à l'affichage.

3.3.1.4. Expression des jointures

Les jointures se font en spécifiant les tables sur lesquelles la jointure sera faite dans la liste des tables utilisées et en spécifiant la qualification de jointure dans la clause WHERE. Rappellons qu'une jointure sans qualification est un produit cartésien et qu'une jointure avec égalité est dit équi-jointure.

Si les noms de tables qui servent de préfixes deviennent pénibles à taper, SQL offre de plus une possibilité de synonymes dans la clause FROM. Il suffit pour cela de préciser le synonyme juste après le nom de la table à l'aide du mot clé AS (facultatif) comme pour les synonymes de noms de colonne.

Les expressions de manipulation de données SQL permettent bien sûr d'exécuter des calculs arithmétiques à l'affichage des colonnes. Il suffit pour cela de placer l'expression à calculer comme champ d'affichage du SELECT. Ces valeurs calculées peuvent bien sûr être testées dans les clauses HAVING ou WHERE.

D'autres opérations sur chaque valeur sélectionnée peuvent être effectuées notamment sur les dates (YEAR, MONTH, DAY) et les chaînes de caractères (SUBSTRING, UPPER, LOWER, CHARACTER_LENGTH).

C'est ainsi par exemple que les responsables systèmes obtiennent les noms de login des utilisateurs en écrivant une requête affichant une combinaison des caractères du nom et du prénom de chaque utilisateur.

3.3.1.5. Fonctions statistiques

Les fonctions précédentes permettent de manipuler les données d'un tuple à la fois. Mais l'utilisateur a souvent besoin d'autres fonctions que celles présentées précédemment. Notamment, il lui faut parfois pouvoir manipuler des fonctions de regroupement « inter tuples ». SQL offre la possibilité de récupérer des données chiffrées sur les tables ou des parties de tables. Il est par exemple possible d'obtenir le nombre de tuples répondant à un critère, la valeur moyenne d'une colonne, la valeur maximum ou minimum et la somme d'une colonne.

AVG	Moyenne
COUNT	Nombre d'éléments
MAX	Maximum
MIN	Minimum
SUM	Somme

Tableau 3.03 : Fonctions statistiques principales

Il est important de noter que les opérations statistiques précédentes ne peuvent pas s'imbriquer.

Il est par exemple interdit d'écrire quelque chose comme AVG (SUM(montant)). Ce type de calcul ne pourra s'obtenir qu'avec des sous-requêtes ou des vues comme nous le verrons plus tard.

Les regroupements, il est souvent intéressant de regrouper les données d'une table en sous-tables pour y faire des opérations par groupes. Par exemple compter les commandes par fournisseur.

Ceci se fait avec la clause GROUP BY suivie de la colonne à partitionner. Remarque: si dans un SELECT certaines colonnes sélectionnées utilisent des fonctions de calcul et d'autres non, alors

toutes les colonnes sélectionnées qui n'utilisent pas de calcul doivent être spécifiées dans le GROUP BY.

La clause HAVING étroitement liée au GROUP BY permet d'exprimer une restriction sur les lignes de la table obtenue avec les fonctions de calcul. Les tests sur les colonnes simples se font dans la clause WHERE tandis que les tests sur les fonctions de regroupement se font dans la clause HAVING.

Les fonctions arithmétiques et statistiques testées dans une clause HAVING n'ont pas à être forcément sélectionnées comme colonne résultat. Il est possible d'utiliser la fonction uniquement dans la clause HAVING sans qu'elle soit projetée.

Les sous-requêtes SQL donne la possibilité d'utiliser des sous-requêtes afin de décrire des requêtes complexes permettant d'effectuer des opérations dépendant d'autres requêtes. La sous-requête est toujours placée dans la clause WHERE ou HAVING en lieu et place d'une constante et doit renvoyer soit une valeur unique (qui est alors utilisée avec les opérateurs classiques du WHERE comme n'importe quelle constante), soit une colonne unique (qui est alors utilisée avec les opérateurs IN et EXISTS, ou avec les comparateurs classiques suivis de ALL ou ANY). Attention : la sous-requête doit suivre obligatoirement l'opérateur de comparaison, le contraire est interdit.

3.3.2. *Mise à jour d'informations*

Trois types de mise à jour sont nécessaires pour le contenu d'une table relationnelle. L'ajout de nouveaux tuples, le changement de certains tuples et la suppression de certains tuples.

3.3.2.1. Insertion.

L'ordre INSERT permet d'ajouter des lignes dans une table. Dans sa forme la plus générale, SQL demande que les noms des colonnes soient explicitement cités. Les valeurs insérées sont alors en correspondance avec l'ordre dans lequel les colonnes sont citées. Si l'ordre INSERT contient une clause VALUE alors une seule ligne est insérée dans la table ; si l'ordre INSERT contient une clause SELECT alors plusieurs lignes peuvent être simultanément insérées dans la table. L'insertion de tuples peut donc se faire soit en extension par la clause VALUE soit en intension par un ordre SELECT imbriqué.

SQL accepte que tous les noms de colonne de la table dans laquelle les tuples seront insérés ne soient pas précisés. Dans ce cas, les colonnes non précisées sont alors automatiquement remplies avec la valeur NULL. Il est néanmoins conseillé de toujours préciser toutes les colonnes quitte à préciser les valeurs NULL manuellement, afin d'éviter de fâcheuses erreurs de programmation.

On notera que dans un ordre INSERT utilisant un SELECT, une constante peut être placée dans la requête en lieu et place d'un nom de colonne. Ceci permet d'insérer facilement des tuples dans la base avec des valeurs par défaut.

Une table citée dans le champ INTO de l'ordre INSERT ne peut pas être citée dans le champ FROM du sous-select de ce même INSERT. Il n'est donc pas possible d'insérer des éléments dans une table à partir d'une sous-sélection de cette même table.

3.3.2.2. Mise à jour

L'ordre UPDATE permet de modifier des lignes dans une table. L'expression caractérisant la modification à effectuer peut être une constante, une expression arithmétique ou le résultat d'un sélect imbriqué.
L'ordre UPDATE peut parfois poser des problèmes d'intégrité de la base. Par exemple, si une table possède comme clé unique un entier de 1 à n, que doit faire le SGBD lors d'une incrémentation de 1 de cette clé?

Certains SGBD vérifient la cohérence de la base uniquement après chaque ordre SQL. C'est le cas d'Oracle ou DB2. Dans ce cas cette requête ne pose aucun problème. D'autres en revanche vérifient la cohérence de la base après chaque modification de ligne. C'est le cas d'Access.
Dans ce cas il risque d'y avoir un problème d'unicité de la clé durant l'exécution de la requête.
Ce choix du mode de vérification de l'intégrité de la base n'est toujours pas tranché et vous trouverez selon les SGBD les deux modes d'exécution. D'une manière générale il n'est pas conseillé de faire des UPDATE sur des colonnes utilisées dans une clé primaire.

3.3.2.3. Suppression.

L'ordre DELETE permet de supprimer des lignes dans une table selon une qualification fixée.
L'ordre DELETE FROM <table>; permet de vider complètement une table. Néanmoins, dans ce cas, la table existe toujours bien qu'elle soit vide.

3.3.3. *Définition des données : DDL*

Avant de parler précisément des ordres de description de données (le DDL) précisons brièvement les types de données autorisés.

3.3.3.1. Types de données

Les types de données que l'on peut représenter dans un SGBD sont fortement dépendants de l'architecture de l'ordinateur sur lequel il tourne. Il existe donc de nombreuses variantes de types selon les SGBD. Néanmoins certains types se retrouvent dans la plupart des SGBD. Voici les principaux :

Le type alphanumérique :

CHAR (n) : Longueur fixe de n caractères, n_max = 16383 ;

VARCHAR (n) : Longueur variable, n représente le maximum ;

Le type numérique :

- NUMBER (n,d) : Nombre de n chiffres dont d après la virgule ;
- SMALLINT : Mot signé de 16 bits (-32768 à 32767) ;
- INTEGER : Double mot signé de bits (-2E31 à 2E31-1) ;
- FLOAT : Numérique flottant ;

Le type gestion de temps :

- DATE : Champ date (ex 31/12/2001) ;
- TIME : Champ heure (ex 18 :45 :21.86) ;
- TIMESTAMP : regroupe DATE et TIME ;

3.3.3.2. Création de tables

Une fois les types de données définis il est alors possible de créer les tables. La commande CREATE TABLE permet de définir des colonnes, de leur associer un type de données et d'y ajouter des contraintes à vérifier. La syntaxe la plus simple est la suivante :

CREATE TABLE <nom-de-table> (<nom-de-colonne> <Type de données>, ...);

Dans la plupart des SGBD, le nom de la table doit commencer par une lettre et on autorise 254 colonnes au maximum par table. Il est aussi possible de créer une table en insérant directement des lignes à la création. Dans ce cas, les colonnes existant ailleurs, il est inutile de les spécifier à nouveau. Les SGBG contiennent rapidement de très nombreuses tables. Il est alors difficile de connaître les structures de toutes les tables. La description d'une table peut être obtenue par l'ordre DESCRIBE qui affiche à l'écran la structure complète de la table choisie.

3.3.3.3. Expression des contraintes d'intégrité

Nous avons vus dans les chapitres précédents les différentes contraintes d'intégrité qu'il était possible d'exprimer dans un SGBD. Le DDL doit bien sûr offrir la possibilité d'exprimer ces

contraintes dans la requête de création de table. Associée à la définition d'une colonne, il est possible d'accoler différentes clauses gérant ces contraintes, bien qu'aucune ne soit obligatoire
Il est notamment possible de nommer une contrainte de colonne : CONSTRAINT nom permet de nommer une contrainte.

CONSTRAINT	permet de nommer une contrainte
DEFAULT	précise une valeur par défaut
NOT NULL	Force la saisie de la colonne
UNIQUE	vérifie que toutes les valeurs sont différentes
CHECK	vérifie la condition précisée

Tableau 3.04 : Contrainte de colonne

On peut alors à la création d'une table :

- Nommer une contrainte de table : CONSTRAINT nom.
- Définir une clé : PRIMARY KEY (liste de cols) permet de spécifier que la colonne définit la clé primaire de la table. La clé primaire peut porter sur plusieurs colonnes. Il n'y a bien sûr qu'une clause de ce genre par table. Automatiquement les colonnes qui constituent la clé primaire ne peuvent plus être nulles et chaque clé doit être unique. Dans la majorité des SGBD un index est automatiquement créé sur cette clé.
- Intégrité référentielle : FOREIGN KEY (liste-col 1) REFERENCES table (liste-col2) permet de spécifier que les colonnes liste-col 1 de la table en cours de définition référencent la clé primaire liste-col2 de la table étrangère spécifiée. La clé étrangère peut porter sur plusieurs colonnes. Il peut bien sûr y avoir plusieurs clés étrangères dans une même table. Les modifications automatiques à faire sur les clés étrangères en cas de changement de la clé primaire associée sont précisées par les clauses ON DELETE et ON UPDATE.

Une convention généralement admise consiste à nommer les contraintes d'intégrité référentielle par un nom préfixé par PK pour la clé primaire et FK pour les clés étrangères.

CONSTRAINT	Permet de nommer une contrainte
PRIMARY KEY	Déclare que la colonne est une clé primaire
FOREIGN KEY	Déclare que la colonne est une clé étrangère

Tableau 3.05 : Contrainte de table

3.4. Connexion aux bases de données [2] [6] [11] [12]

3.4.1. *API JDBC*

3.4.1.1. Qu'est ce que JDBC?

JDBC est une API fournie avec Java permettant l'accès à n'importe quelle base de données à travers un réseau. Le terme JDBC signifie Java DataBase Connectivity et permet d'écrire facilement des applications Java qui interagissent avec une base de données en respectant le principe de Java :

"Un seul code + une seule compilation = parfaite portabilité". JDBC permet notamment de résoudre le problème lié à tous les Intranet actuels : comment relier la base de données d'entreprise à des pages WEB et ainsi permettre aux clients de visualiser les informations qui les concernent à travers le réseau? Les avantages de JDBC par rapport aux autres approches (ODBC, appels systèmes etc....) sont avant tout des avantages liés à Java : portabilité sur de nombreux systèmes d'exploitation et de nombreuses bases de données, uniformité du langage de description des applications, des Applets et des accès bases de données et surtout une liberté totale vis à vis des différents constructeurs.

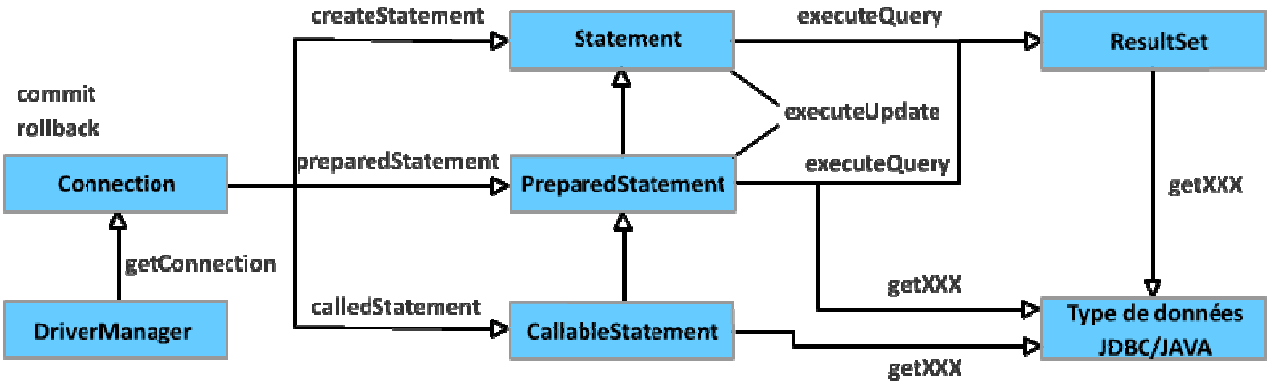


Figure 3.02 : Java et les bases de données

Le langage Java contient depuis la version 1.1 une API adaptée à la connexion avec les bases de données nommée JDBC. L'avantage de cette API est qu'elle a été construite indépendamment de toute base de données offrant ainsi une portabilité sans équivalent : Le langage Java qui est interprété peut s'exécuter sur toute architecture et l'API JDBC, étant indépendante du SGBD, s'utilise de la même manière pour toute base de données.

Si ODBC offre une couche d'abstraction universelle indépendante de la base de données, il n'est, par contre, pas indépendant de la plate-forme (MS Windows). JDBC par contre offre au développeur l'abstraction dans les deux directions : base de données et plate-forme système.
Ecrire une application Java de connexion à une base de données nécessite le JDK (Java Development Kit) développé par le constructeur SUN pour permettre la compilation des programmes Java, un serveur de bases de données pour répondre aux requêtes et un Driver particulier propre à chaque SGBD.

Pour permettre l'accès aux bases de données, le JDK fournit le paquetage java.SQL qui est mis à la disposition des développeurs pour formuler et gérer les requêtes aux bases de données.
Ce paquetage offre plusieurs interfaces définissant les objets nécessaires à la connexion à une base éloignée et à la création et exécution de requêtes SQL ainsi que des classes utilitaires.

Interfaces	Classes	Exceptions
Array	Date	BatchUpdateException
Blob	DriverManager	DataTruncation
CallableStatement	DriverPropertyInfo	SQLException
Clob	Time	SQLWarning
Connection	Timestamp	
DatabaseMetaData	Types	
Driver		
PreparedStatement		
Ref		
ResultSet		
ResultSetMetaData		
SQLData		
SQLInput		
SQLOutput		
Statement		
Struct		

Tableau 3.06 : Les classes utilitaires de l'API Java

3.4.1.2. Structure d'une application JDBC

Chaque programme Java souhaitant utiliser l'API JDBC doit tout d'abord inclure le paquetage java.sql : import java.sql.*;. Ensuite, comme pour toutes les entrées-sorties Java, des exceptions sont générées en cas d'erreur. Tous les ordres SQL doivent donc capter l'exception SQLexception qui est appelée dès qu'un ordre SQL ne se passe pas correctement. Cette classe contient notamment la méthode getMessage() qui renvoie le message en clair de l'erreur.

La philosophie de développement d'une application JDBC est alors la suivante :

a) Enregistrer le pilote JDBC.

Chaque base de données utilise un pilote (driver) qui lui est propre et qui permet de convertir les requêtes JDBC dans le langage natif du SGBD. L'enregistrement de ce pilote doit être effectué dans le DriverManager Java au début de chaque application.

Quand une classe de Driver est chargée, elle doit créer une instance d'elle même et s'enregistrer auprès du Driver Manager. Ceci se fait par l'Appel à la méthode suivante :

```
Class.forName("nom du driver");
```

Par exemple, le nom du pilote fourni par le constructeur SUN pour effectuer un pont JDBC-ODBC est sun.jdbc.odbc.JdbcOdbcDriver. L'ordre d'enregistrement aura donc la forme :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

On notera que certains compilateurs refusent cette notation et demandent l'invocation explicite de la méthode newInstance() de la manière suivante :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
```

b) Etablir la connexion avec la base de données

Une fois le pilote enregistré, il est alors possible d'établir la connexion avec la base de données.

Ceci se fait par la méthode getConnection du Driver Manager qui renvoie un objet de type Connection. Il est nécessaire de passer 3 arguments à cette méthode : l'URL de la base de données, le nom de l'utilisateur de la base et enfin son mot de passe. Ces trois arguments doivent être fournis sous forme de chaînes de caractères.

Par exemple, dans le cas du pont JDBC-ODBC, le nom de DSN à passer doit être de la forme jdbc:odbc:monDSN. Par exemple, si votre DSN se nomme compta accessible par l'utilisateur admin et le mot de passe ukvg la connexion se fera par l'ordre :

```
java.sql.Connection con = DriverManager.getConnection("jdbc:odbc:compta","admin","ukvg");
```

c) Créer une zone de description de requête

Pour définir une requête, il est ensuite nécessaire de créer un objet de type Statement. Cet objet non seulement une zone de description de la requête SQL par elle-même, mais aussi les ordres d'exécution de cette requête.

La création de l'objet Statement se fait par la méthode createStatement de l'objet Connection précédemment créé: java.sql.Statement stmt = con.createStatement();

d) Exécuter la requête

Une fois la zone de description créée, il est alors possible de la remplir avec une requête SQL puis d'exécuter cette requête.
Deux méthodes de l'objet Statement permettent respectivement de définir et exécuter la requête. Il s'agit de la méthode executeQuery qui permet d'Exécuter une requête du type SELECT et qui renvoie un objet de type ResultSet contenant les tuples résultant et de la méthode executeUpdate qui permet d'effectuer les requêtes du type CREATE, INSERT, UPDATE et DELETE et qui renvoie un entier (int) indiquant le nombre de tuples traités.
Ces deux méthodes nécessitent un argument de type chaîne qui indique la requête SQL à exécuter.
Par exemple, java.sql.ResultSet rs = stmt.executeQuery("SELECT * FROM clients");

```
int nb = stmt.executeUpdate("INSERT into clients values('Durand','Paul');
```

e) Traiter les données retournées

L'objet ResultSet retourné après l'exécution de la méthode executeQuery contient plusieurs méthodes permettant de manipuler l'ensemble résultant de la requête :
Pour parcourir l'ensemble résultat, il n'existe qu'une seule méthode : la méthode next. Initialement le pointeur est positionné avant le premier tuple, chaque appel à la méthode next fait avancer le pointeur sur le tuple suivant. next renvoie à chaque invocation un booléen permettant de savoir s'il y a encore des tuples disponibles.

```
while (rs.next())
  {…// traitement de chaque tuple
  }
```

Il n'est donc pas possible de revenir aux tuples précédents ou de parcourir l'ensemble résultat dans un ordre aléatoire; on commence par le premier, on passe au suivant et ainsi de suite jusqu'au dernier, sans possibilité de retour en arrière. Si un retour arrière est souhaité il est donc nécessaire de refaire un ExecuteQuery.
Pour accéder aux éléments d'un tuple, l'objet ResultSet fournit toute une série de méthodes de la forme getXXX() permettant de lire le type de données xxx dans chaque colonne du tuple courant. On utilisera par exemple les méthodes getInt, getFloat, getDate, getLong, getInt, getString etc ... La colonne est identifiée soit par son nom que l'on passe en paramètre sous forme de chaîne de caractère soit par son numéro relatif dans l'ordre des colonnes.

int pds = getInt(3); // accède à la 3ème colonne

int prod= getString("PRODUIT"); // accède à la colonne PRODUIT

Type SQL	Type Java	Fonction d'accès Java
CHAR	String	getString()
VARCHAR	String	getString()
NUMERIC	java.math.BigDecimal	getBigDecimal()
DECIMAL	java.math.BigDecimal	getBigDecimal()
BIT	Boolean	getBoolean()
INTEGER	Integer	getInt()
REAL	Float	getFloat()
DOUBLE	Double	getDouble()
BINARY	byte[]	getBytes()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

Tableau 3.07 : Conversions principales SQL-Java

Il faut cependant noter que Java ne fournit pas directement la possibilité de savoir si une colonne SQL est nulle ou pas. JDBC fournit pour cela la méthode wasNull() de la classe ResultSet qui permet de savoir a posteriori si une colonne était nulle ou pas.

```
int poids = rs.getInt("poids");
if (!rs.wasNull()) System.out.println("le poids est connu");
```

f) Fermer les différents espaces

Tout programme bien construit se doit de fermer les espaces ouverts durant son travail. La méthode close est prévue à cet effet. Elle est défnie pour les objets ResultSet, Statement et Connection.

```
rs.close();
st.close();
con.close();
```

Nous en savons assez pour écrire nos premières applications JDBC, mais auparavant, il reste à rappeler que toutes ces méthodes déclenchent des exceptions qui doivent impérativement être testées (ce qui n'est pas sans alourdir le code) :

```
String url = "jdbc:mySubprotocol:myDataSource";
Class.forName("myDriver.ClassName");
con = DriverManager.getConnection(url,"myLogin","myPassword");
```

```
stmt.executeUpdate(createString);
ResultSet rs = stmt.executeQuery(selectString);
while (rs.next()) ...
rs.close();
stmt.close();
con.close();
```

3.4.2. *Différents pilotes*

Sans rentrer dans les détails techniques nécessaires uniquement aux concepteurs de nouveaux pilotes, il est néanmoins important d'apporter quelques précisions sur les pilotes JDBC.
Il existe, selon la taxonomie de JavaSoft, 4 types de pilotes JDBC différents :

- Type 1. Ce sont les pilotes accédant à une base de données par l'intermédiaire de ponts.

L'exemple typique est le pilote fourni par SUN permettant le pont entre JDBC et ODBC.
C'est celui que nous avons utilisé dans nos différents exemples. Ce type de pilote ne peut être utilisé que par les applications Java. En effet le modèle classique de sécurité pour l'exécution des Applets (untrusted) interdit à une Applet de charger du code natif dans la mémoire vive de la plate-forme d'exécution.

- Type 2. Ce sont les pilotes d'API natifs. Ils sont fournis par les éditeurs de bases de données et gèrent des appels C/C++ directement avec la base. Ils sont en général payants. Comme précédemment, le modèle classique de sécurité pour l'exécution des Applets (untrusted) interdit à une Applet de charger du code natif dans la mémoire vive de la plate-forme d'exécution. Ce type de Pilote ne peut donc pas être utilisé dans une Applet.

- Type 3. Ce sont les pilotes qui interagissent avec une API réseau générique et qui communiquent avec une application intermédiaire (middleware) sur le serveur. C'est typiquement le cas de l'API RMI-JDBC fournie par l'INRIA ou du système DB Anywhere offert par Symantec Visual Café. Ils sont écrits en Java et peuvent donc être chargés sans aucun problème par la plate-forme d'exécution de l'Applet. Un seul de ces pilotes permet d'interagir avec plusieurs bases de données, ce qui rend cette approche très souple à l'utilisation.

- Type 4. Ce sont les pilotes qui interagissent avec la base de données via les sockets et encapsulent complètement l'interface cliente du SGBD. Ils sont généralement fournis par les

éditeurs de SGBD. Là encore, il n'y a aucun problème d'exécution que ce soit dans une application ou dans une Applet.

Une dernière contrainte est à noter en ce qui concerne JDBC et les Applets : Le modèle classique de sécurité de l'exécution des Applets (untrusted) ne permet d'ouvrir une connexion réseau qu'avec la machine sur laquelle elle est hébergée. Le serveur de base de données (ou le serveur middleware) doit donc être installé sur la même machine que le serveur HTTP.

3.4.3. *Requêtes précompilées*

Dans la majorité des applications d'accès aux bases de données, les requêtes SQL dépendent de paramètres du programme. Pour introduire une variable Java dans la requête SQL, deux méthodes sont possibles.

La première, très simple, consiste à remarquer que l'ordre SQL est décrit dans une chaîne de caractères, et que donc, il est possible de définir cette chaîne en utilisant des variables. Il est donc possible d'écrire :

```
String personnes[]={"Durand", "Dupond","Martin"};
String table="Clients";
String query;
for (int i=0; i< personnes.length; i+)
{
query = "select * from " + table
+ " where nom = '" + personne[i] + "'";
}
....
```

La seconde méthode consiste à utiliser une requête précompilée à l'aide de l'objet PreparedStatement.

Cet objet, qui hérite de l'objet Statement permet d'envoyer une requête sans paramètres à la base de données pour précompilation, puis à spécifier au moment voulu la valeur des paramètres.

La création d'un PreparedStatement s'obtient à l'aide de la méthode preparedStatement de l'objet Connection. Cette méthode nécessite un argument de type chaîne de caractères, décrivant la requête SQL à exécuter. Les arguments dynamiques doivent être marqués par un point d'interrogation.

```
PreparedStatement st = c.prepareStatement("SELECT * FROM ? WHERE nom = ?");
```

Chaque argument dynamique doit ensuite être positionné à l'aide des méthodes setInt, setFloat, setDate, setLong, setString, etc ... de l'objet PreparedStatement. Ces méthodes nécessitent deux arguments, le premier de type entier qui indique le numéro relatif de l'argument dans la requête, le second qui indique la valeur à positionner. Par exemple :

```
st.setString(1,"Clients");
st.setString(2,personne[1]);
```

Pour Exécuter la requête ainsi construite il suffit alors d'exécuter les méthodes executeQuery ou executeUpdate de l'objet PreparedStatement sans aucun argument. Comme pour la méthode classique la méthode executeQuery retourne un objet de type ResultSet contenant les tuples résultant du SELECT, tandis que la méthode executeUpdate retourne le nombre de tuples mis à jour.

```
java.sql.ResultSet rs = st.executeQuery();
```

Il est clair que si votre requête doit être exécutée plusieurs fois avec des arguments variables, la méthode PreparedStatement est bien plus rapide qu'avec un Statement classique. Assurez-vous néanmoins que votre SGBD accepte les requêtes précompilées, ce qui n'est pas le cas de tous.

3.4.4. *Commit et Rollback*

Par défaut, tout objet Connection fonctionne en mode auto-commit. Ce qui veut dire, qu'un Commit est effectué automatiquement après chaque ordre SQL. Il est néanmoins possible de repasser en mode manuel par l'appel à la méthode setAutoCommit de l'objet Connection. Cette méthode admet un seul argument booléen indiquant si oui ou non le mode est automatique.

```
con.setAutoCommit(false)
```

Une fois mis en mode manuel, c'est à l'application de solliciter le commit ou le rollback par les méthodes commit et rollback de l'objet Connection. Ceci est notamment très intéressant lorsqu'il est nécessaire de valider tout un groupe d'instructions à la fois : con.commit()

3.4.5. *Méta-base*

En plus des différents objets permettant d'Exécuter les requêtes, JDBC offre aussi la possibilité d'obtenir des renseignements sur les données à travers deux classes : DatabaseMetaData et ResultSetMetaData. La première donne des renseignements très complets sur la méta-base en général tandis que la seconde indique les renseignements sur le ResultSet obtenu après une requête.

3.5. Conclusion

Pour les applications d’entreprise, les systèmes de stockage généralement utilisés sont des bases de données. Ces données sont structurées, mises à jour et maintenues grâces à un système de gestion de base de données ou SGBD, c'est en fait l'interface entre la base de données et les utilisateurs ou leurs programmes. Pour accéder à ces données un langage standardisé nommé Structured Query Language (SQL) permet de lire et mettre à jour les informations contenues dans une base de données relationnelle. Avec la plate-forme J2EE une interface de programmation a été définie pour uniformiser les accès en Java aux bases de données relationnelles: Java DataBase Connectivity (JDBC). Ce chapitre nous a permis de voir l’architecture d’un système d’information et les spécifications de J2EE pour permettre l’accès à ce système.

CHAPITRE 4 SERVLETS ET JSP

4.1. Application web [1] [4] [6] [13] [16]

Une application web est une extension du serveur web ou du serveur d'application. Il existe deux types d'application web :

- Orientée présentation : une application web orientée présentation génère des pages web interactives contenant différent type de langage à balisage comme le HTML ou le XML et des contenus dynamiques qui se trouvent à l'intérieur des réponses aux requêtes.
- Orientée service : une application web orientée service implémente le point final du service web. Les applications orientées présentation sont souvent le client des applications orientées services.

Sur la plate-forme Java 2 Enterprise Edition, les composants web fournissent les capacités de l'extension dynamique des serveurs web. Les composants web sont soit les servlets, soit les JavaServer Pages, soit les services web.

L'interaction entre un client web et une application web est décrite dans le paragraphe suivant.
Le client envoie une requête HTTP au serveur web. Le serveur web qui implémente la technologie servlet et JSP converti la requête contenu dans un objet « HTTPServletRequest ».

Cet objet est envoyé au composant web, qui peut interagir avec un composant JavaBeans ou une base de données pour générer dynamiquement le contenu.

Le composant web peut ensuite générer un « HTTPServletResponse », ou il peut envoyer la requête sur un autre composant web. Eventuellement un composant web peut générer un objet « HTTPServletResponse ». Le serveur web convertit cet objet en une réponse HTTP et la renvoi au client.

La Figure 4.01 suivante schématise cette succession d'événements.

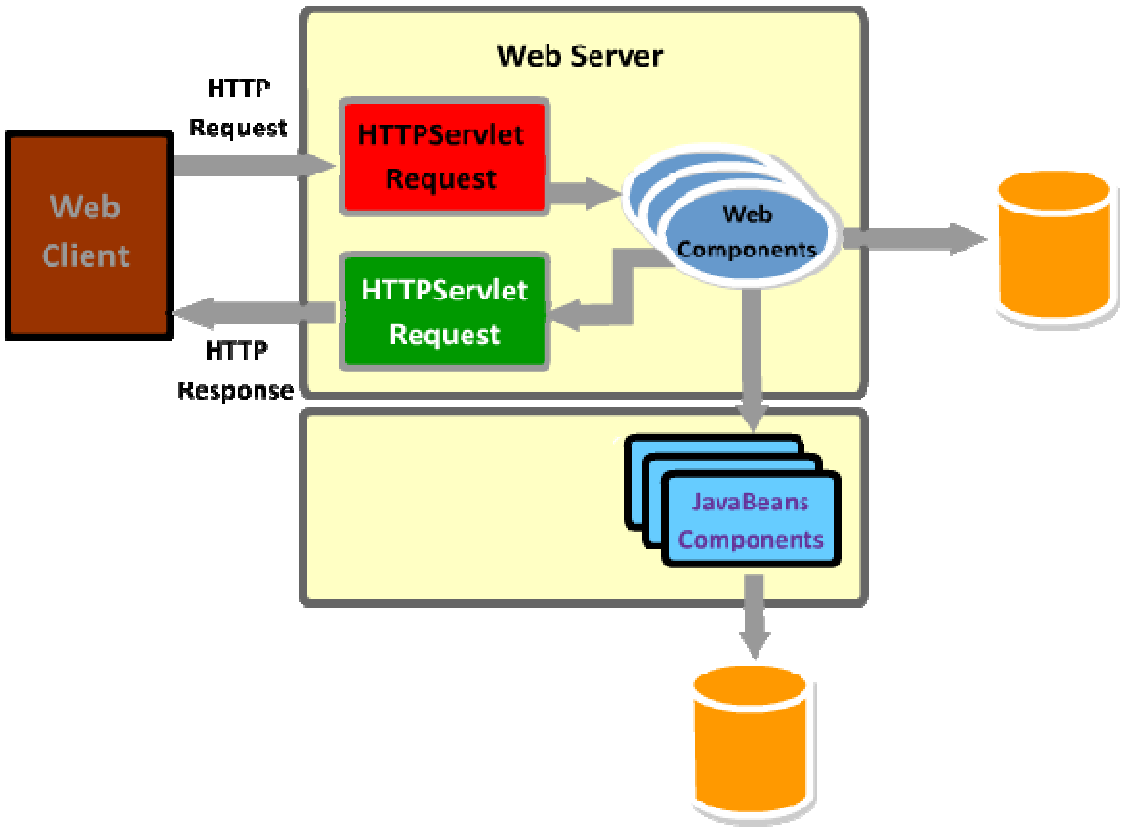


Figure 4.01 : Traitement d'une requête d'une application web Java

Les servlets sont des classes Java qui traitent dynamiquement les requêtes et construisent les réponses. Les pages JSP sont des documents basés texte qui s'exécutent comme un servlet mais permet d'avoir une approche beaucoup plus naturelle pour la création des contenus statiques. Quoique les servlets et les pages JSP puissent être utilisés de façons interchangeables, chacun a sa propre force. Les servlets sont meilleures pour convertir les applications orientées services (les services web sont implémentées en servlets) et pour le contrôle des fonctions des applications orientées présentation comme la répartition des requêtes et le traitement des données non textuelles. Les pages JSP sont plus appropriées pour générer des balises basées texte comme le HTML, le WML, le XML.

Depuis l'introduction des technologies servlets et JSP, d'autres technologies Java et Frameworks ont été développées pour construire des applications web interactives. Les interactions entre ces technologies sont illustrées par la figure 4.02 ci-après.

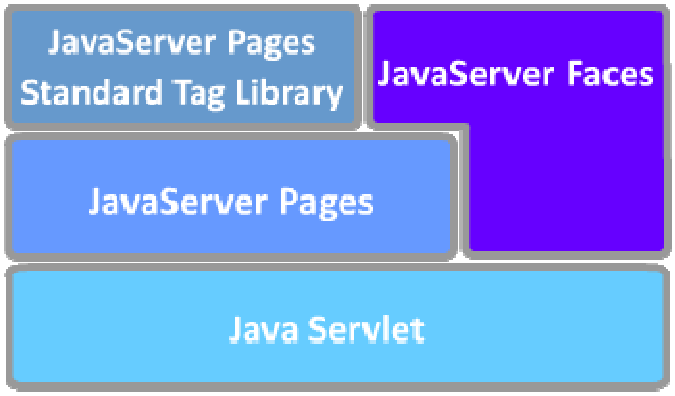


Figure 4.02 : Les technologies des applications web Java

Puisque la technologie servlet est la base des technologies des applications web, chaque technologie ajoute un niveau d'abstraction, qui fabrique des prototypes d'application web. Ces derniers accélèrent le développement, et permettent aux applications web d'être mieux entretenues, disponibles et robustes.

Les composants web sont supportés par les conteneurs web. Un conteneur fournit des services comme les requêtes réparties, la sécurité, la concurrence, la gestion du cycle de vie des applications web. Il permet aussi au composant web d'accéder aux API comme le nommage, la transaction, et l'e-mail. Certains comportements du composant web peuvent être configurés quand le composant est installé, ou déployé dans le conteneur web. L'information de configuration est conservée dans un fichier texte dans le format XML appelé : « descripteur de déploiement d'application web ». Un descripteur de déploiement d'application web doit être conforme au schéma décrit par la spécification de Java servlet.
Beaucoup d'application web utilise le protocole HTTP. Et pouvoir supporter le protocole HTTP est un aspect majeur des composants web.

4.1.1. *Cycle de vie des applications web*

Une application web est constituée de composant web, des fichiers statiques comme les images, des classes d'aide et des librairies. Le conteneur web supporte beaucoup de services qui mettent en valeur les capacités des composants web et les rend plus facile à développer. Cependant, puisque les applications web doivent prendre ces services en compte, le processus de création et de mise en marche d'une application web est différent des classes Java standard. Les processus de création, de déploiement, et d'exécution d'application web peuvent être récapitulés comme suit :

- Développer les codes des composants web
- Développer le descripteur de déploiement de l'application web
- Compiler les composants d'application web et les classes d'aide référencées par les composants
- Optionnellement empaqueter l'application dans une unité déployable
- Déployer l'application dans un conteneur web
- Accéder à un URL qui référencie l'application web

4.1.2. *Modules web*

Dans l'architecture J2EE, les composants web et les contenus web statiques sont appelés « les ressources web ». Un module web est la plus petite unité déployable et utilisable des ressources web. Un module web J2EE correspond à une application web comme défini dans la spécification du Java Servlet.
Avec les composants web et les ressources web, un module web peut contenir d'autres fichiers :

- Des classes utilitaires côté serveur (comme des bases de données beans). Souvent ces classes sont conformes à l'architecture des composants JavaBean ;
- Des classes côté client (comme les applets).

Le module web a une structure spécifique. Le niveau supérieur du répertoire d'un module web est « le document racine » de l'application. Le document racine est le lieu où les pages JSP, les classes et archives côtés client et les ressources web statiques sont enregistrés. Le document racine a un sous répertoire appelé « /WEB-INF/ » qui contient les fichiers et répertoires suivants :

- « web.xml » : descripteur de déploiement d'application web ;
- fichiers de descripteur des « tags librairies » ;
- classes : un répertoire qui contient les classes côtés serveur dont les servlets, les utilitaires de classes, et les composants JavaBeans ;
- lib : un répertoire qui contient les archives JAR des librairies appelés par les classes côtés serveur.

Un module web peut être déployé comme une structure de fichier déballé ou peut être empaqueté dans un fichier JAR connu comme un fichier WAR, vue que l'utilisation d'un fichier WAR est différent des fichiers JAR, un nom de fichier WAR utilise une extension « .war ». Le module web est portable ; il peut être déployé dans n'importe quel conteneur web conforme à la spécification du Java Servlet.

Pour déployer un fichier WAR sur un serveur d'application, le fichier doit aussi contenir un descripteur de déploiement. Le descripteur de déploiement est un fichier XML nommé « sun-web.xml » et se trouve dans le répertoire « /WEB-INF/ ».

La structure du module web qui peut être déployé dans un serveur d'application est montrée par la figure 4.03 ci-dessous.

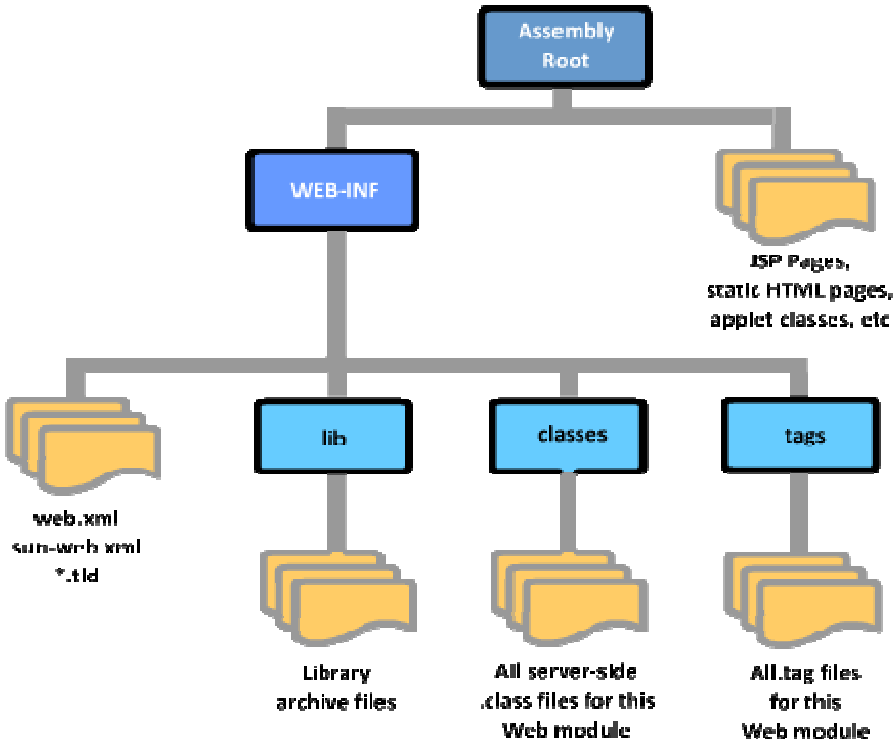


Figure 4.03 : Structure d'un module web

4.2. Servlets [6] [14] [17] [18]

4.2.1. *Définition*

Une servlet est un composant web conçu sous la forme d'une classe Java. Cette dernière existe au sein d'une application web et la mise en œuvre est gérée par un conteneur web du côté du serveur web. Une servlet interagit avec un client web par l'intermédiaire du protocole HTTP via un mécanisme de requête-réponse.

La servlet permet d'étendre les possibilités d'un serveur web en apportant la possibilité de générer du contenu dynamique en réponse à des requêtes clients. Au sein de l'architecture MVC, la servlet joue le rôle de contrôleur.

4.2.2. *Cycle de vie d'une servlet*

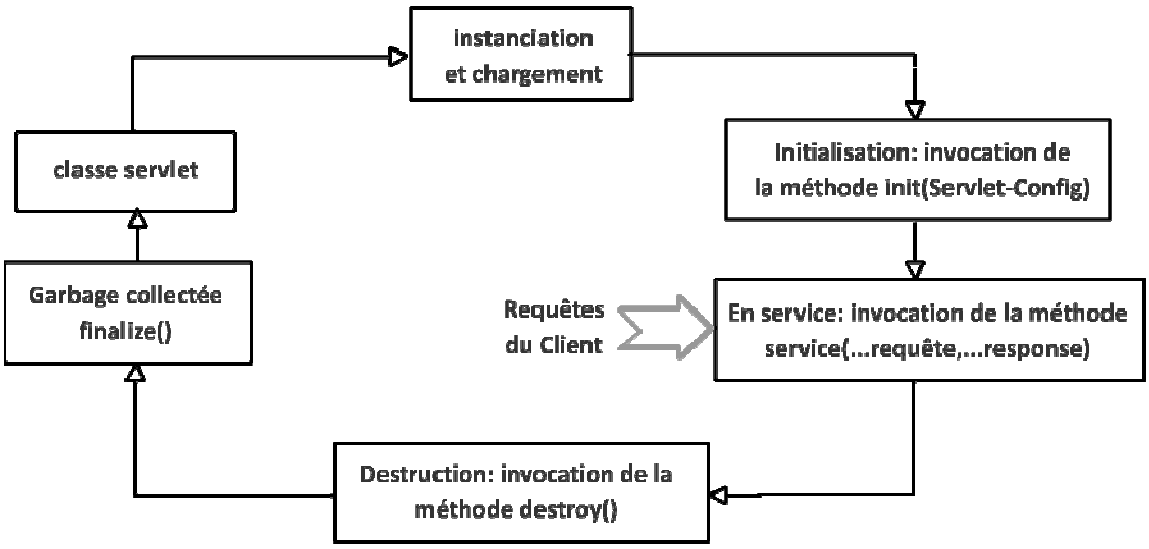


Figure 4.04 : Cycle de vie d'une servlet

4.2.3. *API servlet*

Cet API fournit un certain nombre de classe et d'interface permettant le développement de servlet ainsi que leur déploiement et leur mise en œuvre au sein d'un conteneur web. L'API servlet est contenu principalement dans deux packages :

- javax.servlet
- javax.servlet.http

4.2.4. *Structure de fonctionnement interne d'une servlet*

Figure 4.05 : Structure et fonctionnement interne d'une servlet

Lorsque le client invoque une servlet en émettant une requête HTTP, la méthode service est invoquée sur la servlet par le contenu web. La méthode reçoit deux paramètres :

- javax.servlet.ServletRequest

Un objet de ce type fournit principalement des méthodes permettant de récupérer ou d'insérer des données de la requête, ainsi que des méthodes permettant d'extraire les méta-informations contenues dans la requête (information client, information serveur,...)

- javax.servlet.ServletResponse

Ce type fournit des méthodes permettant d'obtenir des objets afin d'insérer des données (texte ou binaire) renvoyées au client ainsi que des méthodes permettant d'agir sur le buffer temporaire de données.

Le seul traitement effectué par la méthode service est l'invocation de la méthode service surchargée qui prend deux paramètres :

- javax.servlet.http.HttpServletRequest
- javax.servlet.http.HttpServletResponse

qui sont des objets qui fournissent des méthodes en plus de celles hérités spécifique aux protocoles HTTP (en têtes, cookies, authentification,...).

L'interface « javax.servlet.http.HttpServletRequest » hérite de l'interface «javax.servlet.ServletRequest » et l'interface «javax.servlet.http.HttpServletResponse » hérite de l'interface «javax.servlet.ServletResponse ».

Ensuite par défaut la méthode service surdéfinie invoque la méthode correspondant à la méthode HTTP utilisé par le client : doGet(...) ou doPost(...) ou doPut(...) ou doDelete(...) ou doHead(...) ou doOption(...) en lui transmettant les paramètres de type javax.servlet.http.HttpServletRequest et javax.servlet.http.HttpServletResponse.

4.2.5. *Implémentation d'une servlet*

Implémenter une servlet revient à écrire une classe qui est la classe abstraite : javax.servlet.http.HttpServlet. Celui-ci fournit de modèle de base pour la création de Servlet HTTP. Afin d'interagir avec le client, il faut redéfinir :

- Soit la méthode service et dans ce cas intercepter tous les types de requêtes HTTP.

```
import javax. servlet. *;
import javax.servlet.http.*;
import java, io.*;
public class Servlet 1 extends HttpServlet
{
    Public void  service (Http ServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        //implémentation
    }
}
```

- Soit une ou plusieurs méthodes doXXX() en correspondance avec le type de méthode HTTP que peuvent émettre le client :

```
import javax.servlet. *;
import javax.servlet.http.*;
import java.io.*;
public class Servletl extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        //implémentation
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {       //implémentation
    }
}
```

La classe javax.servlet.http.HttpServlet étend la classe javax.servlet.GenericServlet qui implémente les interfaces javax.servlet.Servlet et javax.servlet.ServletConfig lui fournit ainsi une

structure élaborée pour des applications web. Voici la liste des principales méthodes :

- public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException ;
- public void doDelete(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException ;
- public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException ;
- public void doHead(Http ServletRequest request, HttpServletResponse response) throws ServletException, IOException ;
- public void doOptions(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException ;
- public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException ,
- public void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException ;
- public void doTrace(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException ;

4.2.6. *Initialisation et destruction d'une servlet*

L'interface javax.servlet.Servlet fournit des méthodes correspondant au cycle de vie des servlets : initialisation, en service (traitement de requête) puis destruction.
L'interface javax.servlet.Servlet est implémentée par les classes javax.servlet.GenericServlet et javax.servlet.http.HttpServlet. Il est donc impératif d'implémenter ces méthodes dans toutes les servlets à développer.

4.2.6.1. Initialisation

Lorsque la servlet est chargée par le conteneur web, il exécute la méthode « init » de la servlet correspondant à sa phase d'initialisation. Par défaut, cette méthode ne fait rien. Il est conseillé de

la redéfinir dans le code de la servlet pour charger des ressources utilisées dans le reste du fonctionnement de la servlet, ou bien pour récupérer des paramètres d'initialisation renseigner par le descripteur de déploiement (web.xml) de l'application web, par l'intermédiaire de l'objet ServletConfig passé en paramètre.

```
public void init (ServleConfig config) throws ServletException ;
```

4.2.6.2. En service

C'est la méthode service qui est invoquée par le conteneur lors de la sollicitation des clients,

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException ;
```

Par défaut cette méthode est déjà redéfinie dans la classe javax.servlet.http.HttpServlet pour invoquer la méthode service spécifique au protocole HTTP, qui ensuite invoque la méthode doXXX correspondant à la méthode HTTP utilisée par le client pour émettre sa requête,

```
public void service (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException ;
```

4.2.6.3. Destruction

Lors de l'arrêt du conteneur web ou lors de la surcharge en mémoire, les instances de servlet en mémoire sont détruites, avant cela le conteneur web exécute la méthode destroy de chaque servlet.

```
public void destroy() ;
```

4.2.7. *Invocation d'une servlet à partir d'un client léger*

4.2.7.1. Invocation de la méthode doGet()

La méthode doGet() d'une servlet est invoquée principalement dans les cas suivants :

- Lorsque son URL est saisie directement dans la barre d'adresse du navigateur
- Lorsqu'on clique sur un lien hypertexte qui pointe sur l'URL de la servlet.

Saisie d'un URL sur la barre d'adresse du navigateur

```
import javax. servlet. *;
```

```
import javax. servlet.http. *;
import java.io.*;

public class DoGetServlet extends HttpServlet
{
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException
   {
     response.setContentType(CONTENT_TYPE);
     PrintWriter out = response. getWriter();

     out.println("<HTML><BODY>");
     out.println("<Hl> Texte Servlet DoGetServlet</Hl>");
     out.println("</BODY></HTML>");
  }
}
```

L'URL d'invocation de la servlet est par défaut :

http:// localhost:8080/<webapps>/servlet/DoGetServlet

L'utilisation d'un lien hypertexte d'une page HTML est comme suit :

TestDoGetServlet.html

```
<html>
  <head>
     <title> TestDoGetServlet </title>
  </head>

  <body>
     <a href="/<webapps>/servlet/DoGetServlet">Invocation de la méthode doGet() de
     DoGetServlet</a>
  </body>
</html>
```

4.2.7.2. Invocation de la méthode doPost()

La méthode doPost est invoquée principalement lors de l'envoie de donnée saisie dans un formulaire HTML par un clic sur un bouton de type submit.

```
import javax. servi et.*;
import javax.servlet.http.*;
import java.io.*;

public class DoPostServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response. setContentType("Text/html");
        PrintWriter out = response. getWriter();

        String prénom = request. getParameter("prenom");
        String nom = request.geParameter("nom");
        out.println("<HTML><BODY>");
        out.println("<Hl> Bonjour "+prenom+" "+nom+".</Hl>");
        out.println("</BODY></HTML>");
    }
}
```

Code de la page HTML qui appelle cette servlet.

```
 TestDoPostServlet.html
<html>
  <head>
    <title>Document sans titre</title>
    <meta http-equiv=" Content-Type" content="text/html; charset=iso-8859-l">
  </head>
```

```
    <body>
        <form name="forml" method="post" action="/<webapps>/serviet/DoPostServlet">
            <p>Pr&eacute;nom : <input name="prenom" type="text" id="prenom"> </p>
            <p>Nom :<input name="nom" type="text" id="nom"> </p>
            <p><input type="submit" name="Submit" value="Envoyer"> </p>
        </form>
    </body>
</html>
```

4.3. JSP [13] [14] [16] [18]

4.3.1. *Généralités*

La technologie JSP (JavaServer Pages) permet aux développeurs et concepteurs Web de développer rapidement et de maintenir facilement des pages Web, dynamiques et riches en information, qui tirent profit des systèmes d'entreprise existants. Faisant partie de la famille Java, la technologie JSP permet de développer rapidement des applications Web indépendantes des plates-formes.

Théoriquement, la technologie JSP sépare l'interface utilisateur de la génération de son contenu, ce qui permet aux concepteurs de modifier la disposition générale des pages sans altérer le contenu dynamique sous-jacent. En pratique, il faut un peu d'organisation et le respect de quelques règles de programmation pour que le HTML soit bien séparé du code Java dans la JSP, puisque les deux se trouvent dans le même fichier. Les concepteurs Web gérant la partie HTML doivent avoir une compréhension minimale des balises qui représentent le code Java incorporé, pour éviter les problèmes lors de la conception de l'interface utilisateur. La technologie JSP utilise des balises XML et des scriptlets écrits en langage de programmation Java pour encapsuler la logique qui génère le contenu de la page. De plus, la logique de l'application peut résider dans des ressources basées sur le serveur (comme l'architecture de composants JavaBeans), auxquelles la page accède par le biais des balises et des scriptlets. Toutes les balises de formatage (HTML ou XML) sont transmises directement à la page réponse. En séparant la logique de la page de sa conception et de son affichage, et en supportant une conception basée sur les composants réutilisables. La technologie JSP rend plus rapide et plus facile que jamais la construction des applications Web.

JSP est une technologie basée sur le langage Java et permet le développement de site web dynamique. Les fichiers JSP sont des fichiers HTML avec des tags spéciaux contenant des codes sources en Java qui fournit le contenu dynamique. Les pages JSP ont des extensions « .jsp ».

Les sources JSP s'exécutent dans le serveur web qui supporte la machine servlet-JSP. La machine servlet-JSP génère dynamiquement les pages HTML et les envoie vers les navigateurs clients.

4.3.2. *Comparaison entre JSP et servlet*

Une servlet est une classe Java qui fournit un service spécial côté serveur. Il est très difficile d'écrire des codes HTML avec les servlets. Dans les servlets, vous devez avoir des expressions « println » pour générer ces codes HTML.

La technologie JSP est une extension de l'API Servlet de Java. Elle apporte essentiellement une façon simplifiée de développer des servlets.

Comme démontré précédemment, les servlets sont des modules côté serveur indépendants des plates-formes et entièrement Java qui s'adaptent de façon transparente à une architecture de serveur Web et servent à augmenter les possibilités du serveur Web avec des temps système, une maintenance et un support minimaux. Au contraire des autres langages de script, les servlets ne demandent aucune attention ni aucune modification par rapport aux plates-formes. Ensemble, la technologie JSP et les servlets sont une alternative intéressante aux autres types de programmation ou d'écriture de scripts dynamiques pour le Web.

Les JSP ressemblent beaucoup aux ASP (Active Server Pages) de la plate-forme Microsoft. La principale différence entre JSP et ASP est la suivante : les objets manipulés par les JSP sont des JavaBeans et donc indépendants des plates-formes. Les objets manipulés par les ASP sont des objets COM, ce qui lie complètement les ASP à la plate-forme Microsoft. Tout ce qu'il faut à une JSP, c'est une page basée sur la technologie JSP. C'est une page qui contient des balises spécifiques à JSP, des déclarations et, éventuellement, des scriptlets, combinés avec du contenu statique (HTML ou XML). Une page basée sur la technologie JSP porte l'extension .jsp ; cette extension signale au serveur Web que son moteur JSP devra traiter les éléments qu'elle contient.

Une JSP peut aussi utiliser de manière optionnelle un ou plusieurs JavaBeans dans des fichiers .java séparés. Lorsqu'une JSP est compilée par le moteur de JSP sur le serveur Web, elle est compilée dans une servlet. En tant qu'utilisateur, nous ne verrons généralement pas le code de la servlet généré.

4.3.3. *Architecture*

Le moteur servlet-JSP analyse les « jsp » et crée un fichier servlet. Après il compile le fichier résultant en fichier classe, ceci est effectué en une seule fois et c'est pourquoi, JSP est un peu lent lors de son premier chargement. Et après, la servlet compilée est exécutée.

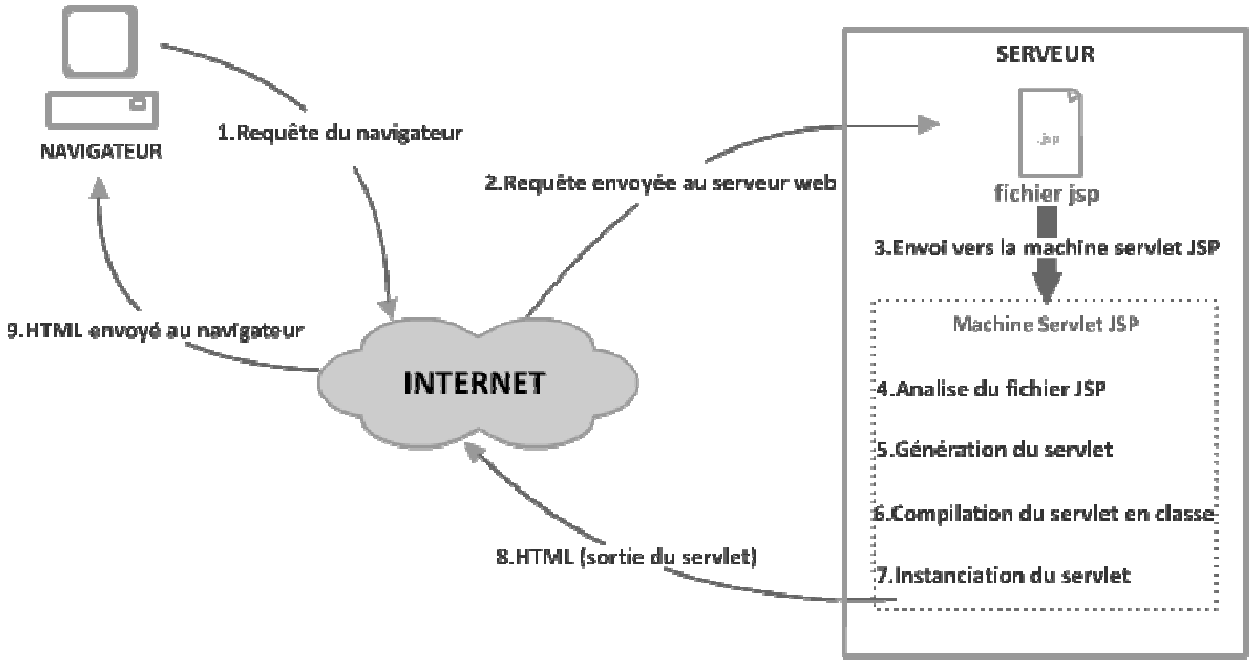


Figure 4.06 : Etapes sur une requête JSP

Si des messages d'erreur sont obtenus lors de la compilation des JSP, n'oublions pas qu'ils peuvent se référer aux lignes de code de la servlet générée. Il nous sera plus facile de déterminer ce qui se passe dans les JSP si nous avons une idée de la façon dont elles sont traduites en servlets. Mais, lorsque le serveur Web compile la JSP, il n'effectue pas cette traduction. Quand nous visualisons un suivi de pile dans le navigateur, la vue Web ou un suivi de pile dans les sorties du serveur Web, le nom de fichier et les numéros de ligne ne sont pas traduits.

4.3.4. *Tags JSP*

Il y a cinq principaux tags :

- Déclaration tag
- Expression tag
- Directive tag

- Scriptlet tag
- Action tag

4.3.4.1. Déclaration tag (<%! %>)

Ce tag permet au développeur de déclarer les variables ou les méthodes.
Avant la déclaration, on doit avoir <%!
A la fin de la déclaration, on doit avoir %>.
Les codes entre ces deux signes doivent se terminer par un « ; ».
Les déclarations ne génèrent pas des sorties, mais ils sont utilisés seulement avec les expressions ou les scriptlets.

4.3.4.2. Expression tag (<%= %>)

Ce tag permet au développeur d'inclure n'importe quelle expression Java dans la page HTML. Les codes entre ces deux signes doivent se terminer par un « ; ». Prenons l'exemple pour l'affichage de la date actuelle : Date : <%=new java.util.Date(); %>

4.3.4.3. Directive tag (<%@ directive ... %>)

Un « directive tag » donne les informations nécessaires sur la page JSP au moteur JSP Il y a trois principaux types de directives :

- page - information de traitement pour la page
- Include - les fichiers à inclure
- Tag library - librairie de tag utilisé par la page.

Ce tag ne produit pas de sortie visible, mais change le mode de traitement de la page par le moteur JSP.

4.3.4.3.1. Page directive

Cette directive possède 11 attributs optionnels qui fournissent au moteur servlet-JSP les informations de traitements nécessaires.

Langage	Quel langage les fichiers utilisent-ils ?	<%@ page language= "java"%>
extends	Les superclasses que le moteur JSP utilise pour la création de la servlet correspondant	<%@ page extends = "com.taglib..." **%>**
Import	Importe toutes les classes dans les paquets Java dans la page actuelle. Les paquets suivants sont importés implicitement : java.lang.*; javax. servlet.jsp.*;	<%@ page import = "java.util.*" %>
Session	Utilisation ou non de session	Par défaut, sa valeur est à « true »
Buffer	Contrôle l'utilisation du buffer de sortie. Par défaut, il est à 8 kb	<%@ page buffer = "none" %>
autoFlush	Vidage du buffer de sortie quand il est plein	<%@ page autoFlush = "true" %>
isThreadSafe	Utilisation ou non de multiple requête par la servlet générée	
Info	Information sur le développeur	<%@ page info = "infol,info2" %>
errorPage	Les pages traitées avec des erreurs	<%@ page errorPage = "/error/error.jsp" %>
isErrorPage	Il est mis à « true » pour créer une Page d'Erreur spécial.	
content Type	Cela spécifie le type MIME de sortie. Par défaut, on a text/html.	

Tableau 4.01 : Page directive

4.3.4.3.2. Include directive

Cette directive permet au développeur JSP d'inclure d'autres fichiers. Typiquement, les fichiers inclus sont utilisés pour la navigation, et les tables qui sont commun à d'autres pages.

Exemple :

<%@ include file = "navigation.jsp" %>

4.3.4.3.3. Tag Lib directive

Ce tag est une collection de tags que peut utiliser la page.

<%@ taglib uri= "tag library URI" prefix = "tag Prefix"%>

4.3.4.4. Scriptlet tag (<% … %>)

Entre les tags « <% » et « %> », tous les codes Java valides sont appelés scriptlet. Ces codes peuvent avoir accès à tous les variables et les beans déclarés.

<% String username = "nom": Out.println(username); **%>**

4.3.4.5. Action tag

Il y a trois rôles principaux des « actions tags » :

- Permet l'utilisation des Javabeans côté serveur
- Transfert des contrôles entre les pages
- Supports indépendants des navigateurs pour les applets

4.3.4.6. Javabeans

Un Javabean est une classe spéciale. Les pages JSP peuvent appeler les méthodes de ces Javabeans.

Voici le syntaxe :

<jsp : usebean id = " ..." scope = "application" class="com..." />

La liste suivante donne les valeurs possibles pour « scope » :

- page
- request
- session
- application

4.3.4.7. Commentaire

Les commentaires JSP ne sont pas visibles dans la source de la page générée.
Syntaxe : <%— commentaire --%>

Les JSP tendent à ne plus contenir de code Java et à n'utiliser que des bibliothèques de balises. Par exemple, la bibliothèque JSTL (JavaServer Pages Standard Tag Library) possède des balises

pour les conditions et les boucles, qui remplacent du code Java. L'utilisation du Framework Struts sépare également le code Java du code HTML.

Les Frameworks et les bibliothèques de balises JSP facilitent le développement des JSP en fournissant des fonctionnalités et des balises JSP qui ne sont pas disponibles dans l'API JSP. Une bibliothèque de balises est une collection de code Java qui peut être appelée à partir de marqueurs d'un fichier JSP.

Un Framework définit une procédure de développement d'applications. En échange de l'acceptation du paradigme de développement du Framework, vous bénéficiez d'une aide de développement et obtenez une application plus nette et plus standardisée. Une bibliothèque de balises JSP comprend un fichier TLD (descripteur de bibliothèque de balises) et des classes gestionnaires qui implémentent ou appellent les fonctionnalités requises par les balises. Elle peut également faire partie d'un Framework. Un Framework peut contenir une ou plusieurs bibliothèques de balises. Il peut également exister d'autres types de Frameworks, dont certains ne sont pas conçus pour être utilisés avec les JSP. Trois Frameworks répandus sont utiles pour le développement des JSP. Ils sont décrits ci-après :

- Struts - Le Frameworks à source libre Struts offre une approche Model *2,* ou Model-View Controller, de la conception logicielle.

- JSTL (JavaServer Pages Standard Tag Library) - JSTL fournit une méthode standard pour accomplir les tâches de codage courantes à l'aide de balises simples.

- InternetBeans Express – C'est une bibliothèque de composants Borland qui facilite la création des servlets et des JSP orientés données.

Il est important de comprendre que la majorité du code Java contenu dans les balises JSP devient partie intégrante de la méthode service () de la servlet lorsque la JSP est compilée en servlet. Cela ne s'applique pas au code contenu dans les balises de déclaration qui est transformé en déclarations autonomes complètes de méthodes ou de variables. La méthode service() est appelée chaque fois que le client fait un GET ou un POST.

4.3.5. *Avantages et inconvénients des JSP*

Les JSP présentent des avantages et des inconvénients par rapport aux servlets.

Les avantages sont notamment les suivants :

- Moins de code à écrire.
- Facilité d'intégration des JavaBeans existants.
- Facilité de déploiement. Prise en charge automatique d'un nombre supérieur de problèmes de déploiement, car les JSP établissent la correspondance avec un serveur web comme le font les fichiers HTML.
- Utilisation directe des balises HTML ainsi le développeur n'est pas obligé d'écrire en Java dans le JSP pour incorporer du code HTML. Si le travail est bien planifié, le code HTML peut être clairement séparé du code Java, ce qui rend le JSP plus lisible.
- Un développeur ou une équipe de développeurs peuvent créer une application d'entreprise permettant aux utilisateurs de navigateurs Web d'effectuer des requêtes sur une base de données. Cette application peut contenir les composants suivants :

Bloc de création	**Rôle dans l'application**
Page HTML	Collecte les requêtes utilisateur
Servlet	Prépare les requêtes en vue de leur traitement par un Enterprise Bean
Enterprise Bean	Se connecte à la base de données et extrait des informations en fonction des requêtes
Deuxième page HTML	Présente les résultats de la requête

Tableau 4.02 : Composants et leur rôle dans l'application

Les programmeurs d'application peuvent également omettre l'Enterprise Bean et laisser la servlet s'occuper de l'accès aux données. Enfin, un ou deux fichiers JSP peuvent remplacer les fichiers HTML et la servlet. Parmi les inconvénients des JSP, citons :

- La non-visibilité du code de la servlet générée peut engendrer une certaine confusion.
- Le code HTML et le code Java n'étant pas dans des fichiers séparés, un développeur Java et un concepteur web travaillant ensemble sur une application doivent faire très attention à ne pas écraser mutuellement leurs modifications.

4.4. Conclusion

Ce chapitre décrit en particulier les principes des applications Web qui est un programme au cœur de la spécification J2EE puisqu'il met en œuvre deux types de composants J2EE: les servlets et les JavaServer Pages ou JSP. Ces spécifications définissent des moyens de générer des contenus de pages Web dynamiquement. Ils sont appelés des composants Web.

Un conteneur Web, en reposant principalement sur la gestion du protocole HTTP pour l'envoi de documents HTML, est un outil destiné à la gestion de la présentation des applications d'entreprise. La génération des données à présenter peut elle-même être effectuée directement par des servlets ou des JSP, car ce sont des programmes Java à part entière, disposant de toute l'interface de programmation de base de J2EE.

CHAPITRE 5 MISE EN ŒUVRE DE L'APPLICATION

5.1. Etat de l'art

Notre but est de développer une application fournissant la possibilité d'acheter, soit par carte bancaire soit par le crédit du téléphone, et de télécharger via le réseau internet des services utilisés pour les téléphones mobiles. Afin de garantir la sécurité de l'application, il s'avère nécessaire de restreindre l'accès à certaines ressources. Une authentification est nécessaire avant toute transaction. Pour cela, le programme parcours la base de données stockant la liste des membres inscrits ayant le privilège d'accéder à la page de transaction. C'est à l'application donc de vérifier dans la base de données si le demandeur est un membre ou non. Pour l'accès à la base de données, Java utilise l'API JDBC. L'accès à la base ne se fait donc pas nativement mais ait recours à un middleware ou intermédiaire.

5.2. Choix du Langage Java

Java est un langage de Programmation Orienté Objet. Dérivé du C++, il jouit des avantages de ce dernier sans en posséder ses défauts. L'esprit du ″write once, run everywhere″ gouverne le monde de Java. En effet, Java est portable, robuste et parfaitement fiable. La gratuité et la bonne structuration de Java fait de lui un langage très populaire auprès des développeurs.
Tous ces paramètres ont conforté le choix du langage sans négliger qu'il est appris depuis la classe de 3è année.

5.3. Choix de la base de données

Les divers SGBD utilisent tous le langage non procédural SQL. Ce langage permet d'effectuer des requêtes sur une base de données telles la manipulation (mis à jour, suppression d'enregistrement, extraction), la création (table, vue, base de données). Citons quelques noms comme : Oracle, PostGre, MySql, DB2, MS SQL SERVER etc. MySql a été choisi du fait que ce SGBD est gratuit, de plus il propose un système de gestion assez simplifié des bases de données.

5.4. Outils de développement

5.4.1. *MERISE*

La démarche de Merise se fait en trois cycles :

- Cycle d'abstraction : Merise utilise trois formalismes pour modéliser un système d'information : le niveau conceptuel qui permet de définir ce que l'on veut faire en

identifiant des informations des règles de gestion et l'enchainement des traitements ; le niveau organisationnel (ou logique) qui permet de définir qui fait quoi, quand et où ? à ce niveau sont traités l'organisation des données, la répartition géographique et fonctionnelle ; au niveau le plus bas, la méthode répond à la question : avec quel moyen, en décrivant les traitements, les données, les états et écrans.

- Cycle de vie : Il comporte 3 grandes périodes : la conception ou période d'étude de l'existant puis du système à mettre en place, la réalisation qui recouvre la mise en œuvre et l'exploitation, la maintenance.

- Cycle de décision : Il définit les instances et les points de décision se rapportant au projet étape par étape.

Le MCD est un modèle abstrait de la méthode Merise permettant de représenter l'information d'une manière compréhensible aux différents services de l'entreprise.
Il permet une description statique du système d'informations à l'aide d'entités et d'associations.
Le travail de conception d'une base de données par l'administrateur commence juste après celui des analystes qui ont établi le MCD.
Les entités du MCD doivent vérifier les règles suivantes :

- Première forme normale : dans une entité, toutes les propriétés sont élémentaires et il existe au moins une clé caractérisant chaque occurrence de l'objet présenté.

- Deuxième forme normale : toute propriété d'une entité doit dépendre de la clé par une dépendance fonctionnelle élémentaire c'est-à-dire toute propriété de l'entité doit dépendre de tout l'identifiant

- Troisième forme normale : dans une entité, toute propriété doit dépendre de la clé par une dépendance fonctionnelle élémentaire directe

- Forme normale de Boyce-Codd (BCFN) : si une entité à un identifiant concaténé, un des éléments composants cette identifiant ne doit pas dépendre d'une autre propriété

La figure 5.01 illustre le MCD de l'application

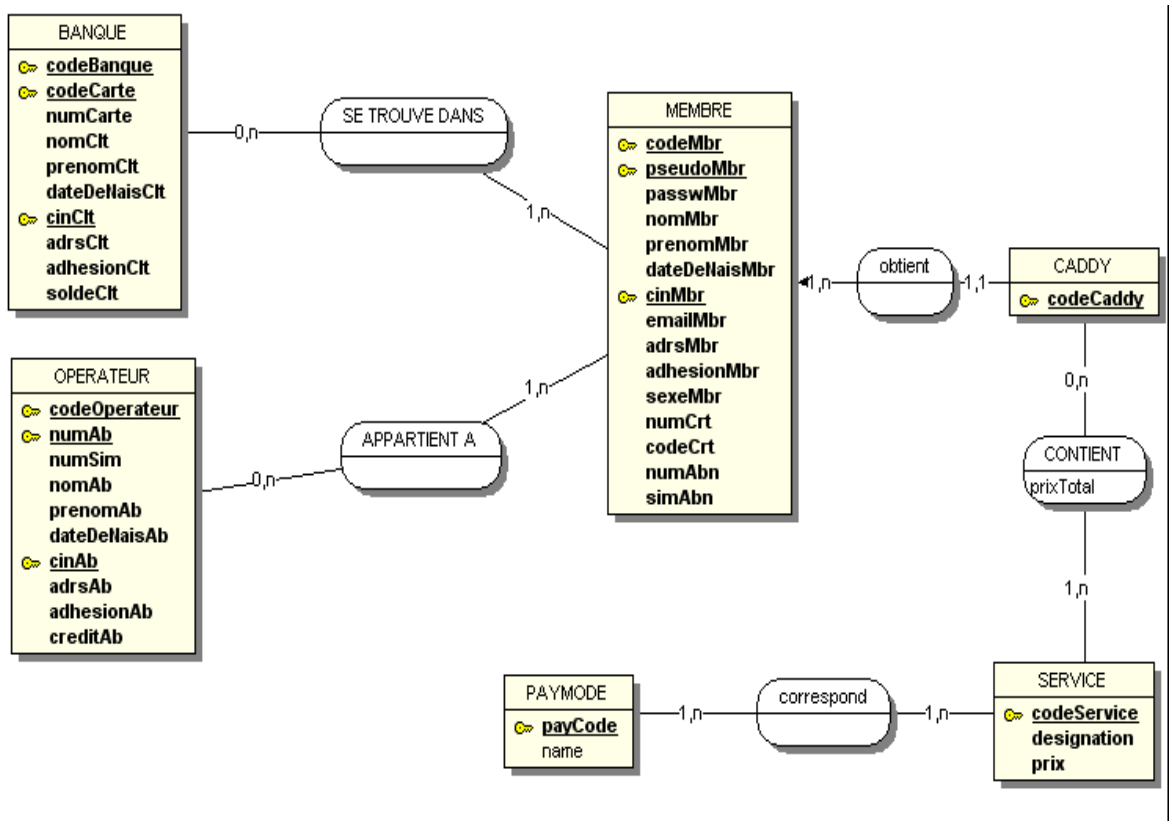


Figure 5.01 : MCD

5.4.2. *ECLIPSE et DREAMWEAVER*

Pour mettre œuvre l'application il nous a fallu deux logiciels, qui sont gratuits sur le marché. Ils représentent une interface aisée, très accessible et facile à utiliser.

Avec ECLIPSE et DREAMWEAVER le développement d'un site web et tout autre système d'information n'attend plus que le savoir faire et le design du développeur.

5.4.2.1. Configuration d'ECLIPSE

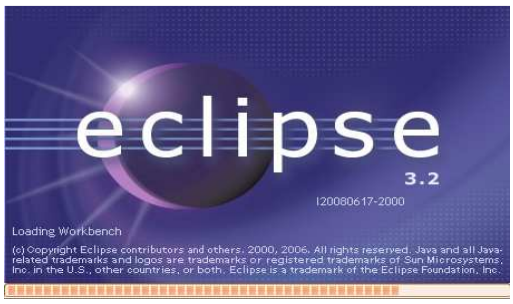


Figure 5.02 : Chargement d'ECLIPSE

Pour installer le logiciel il suffit de décompresser le fichier .zip portant le nom eclipse dans un répertoire de votre choix. Dans notre cas eclipse est répertorié dans Program Files.

Comme nous allons faire une application JSP il faut donc ajouter dans le dossier « plugins » les plugins, avec extension .jar, appropriés comme org.apache, org.eclipse.tomcat et autres.

Une fois l'espace de travail créé on configure Tomcat dans : Window/Preference puis on crée un nouveau projet : File/New/Project…/Java/Projet Tomcat et on suit la procédure de création du projet.

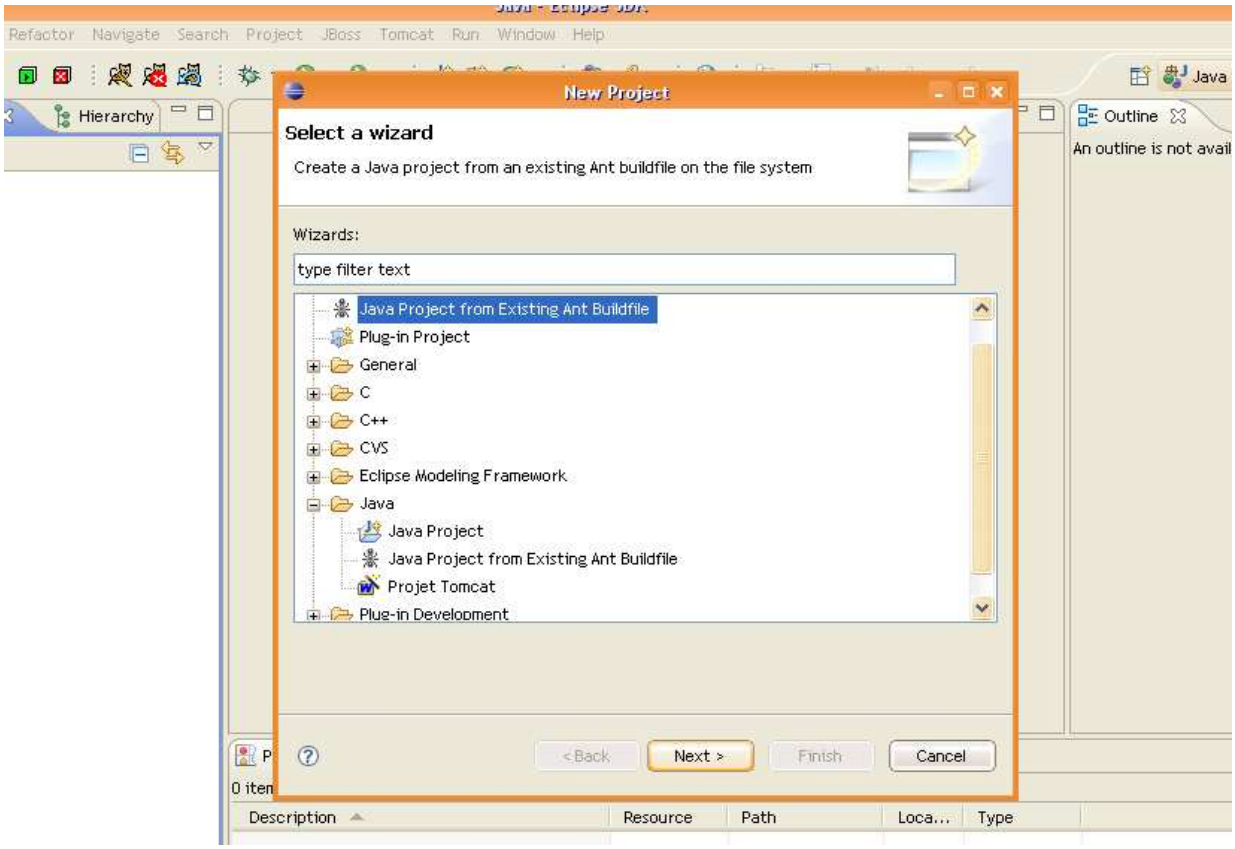


Figure 5.03 : Création d'un nouveau projet Tomcat

5.4.2.2. Configuration de DREAMWEAVER

Figure 5. 04 : Chargement de DREAMWEAVER

Après la création du projet, eclipse génère automatiquement le document racine incluant les fichiers et les dossiers utiles comme WEB-INF et le descripteur de déploiement d'application web : web.xml.

A présent nous pouvons choisir comme outil de développement soit Dreamweaver soit Eclipse. Dans notre cas, il est plus aisé de développer sous Dreamweaver mais au cours du projet, confronté à divers problèmes il a fallu utiliser en même temps les deux logiciels.

La configuration de DREAMWEVER commence par la création du site où l'on définit comme Dossier racine local la racine du projet créé sous Eclipse.

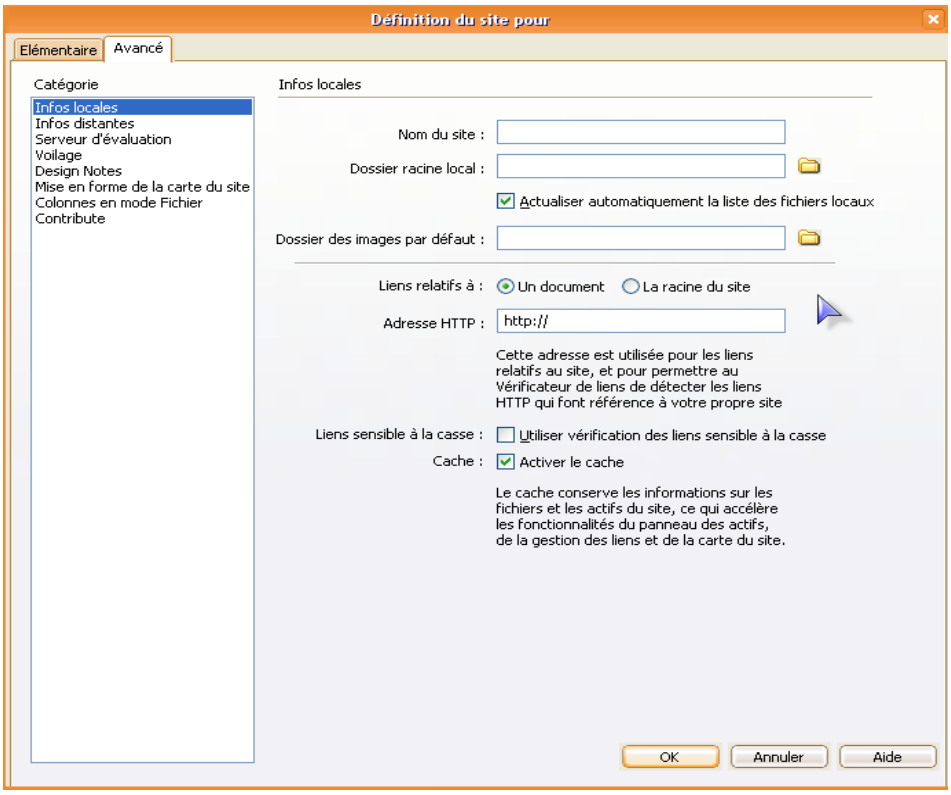


Figure 5.05 : Création d'un nouveau site

5.4.3. *Win' Design et EASYPHP*

Le logiciel Win' Design sert à élaborer le MCD ou le Model Conceptuel de Données.

Figure 5.06 : Présentation de Win' Design

Notre MCD a été mis en œuvre suivant quelques règles de gestion que nous citons après.

Ainsi à partir du MCD on peut générer un script SQL qu'on peut importer dans n'importe quel logiciel traitant les bases de données.

Le logiciel EasyPHP implémentant les serveurs Apache et MySql. Il est utilisé ici en tant que conteneur web Tomcat et pour la gestion des bases de données sous PhpMyAdmin. Le script SQL est alors importer dans ce dernier. Une fois importé on peut ajouter de l'enregistrement, consulter et modifier les tables.

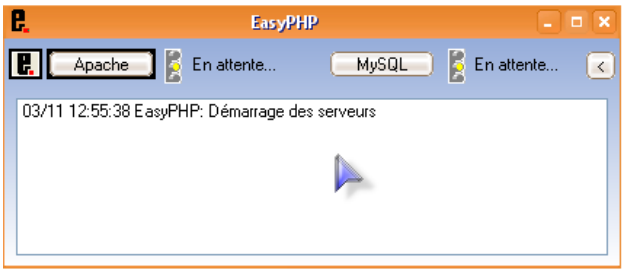


Figure 5.07 : Démarrage d'EasyPHP

5.4.4. *TOMCAT*

L'API JSP peut être classé parmi les scripts côté serveur. Pour le mettre en œuvre, il faut que la machine, qui sera donc considérée comme serveur, possède un conteneur JSP. Il en existe beaucoup sur le marché, mais Apache Tomcat se démarque du fait de sa fiabilité et de sa gratuité. Apache Tomcat est gratuit et une fois installé, il suffit de placer les fichiers *.jsp dans la racine du site.

5.4.4.1. Configuration de Tomcat

Le conteneur Tomcat impose certaines règles. Pour pouvoir déployer une application développée en JSP, il faut suivre certaines procédures.

5.4.4.2. Modification de contexte

Tomcat propose une administration centralisée par un site web. Ainsi, pour effectuer une quelconque modification il suffit de lancer le navigateur web et de taper comme URL : http://localhost:8080 où 8080 est le port d'écoute de Tomcat. Remarquons que cette page correspond au répertoire : $_CATALINA_HOME/webapps/ROOT/index.html où $_CATALINA_HOME est le répertoire d'installation de Tomcat.

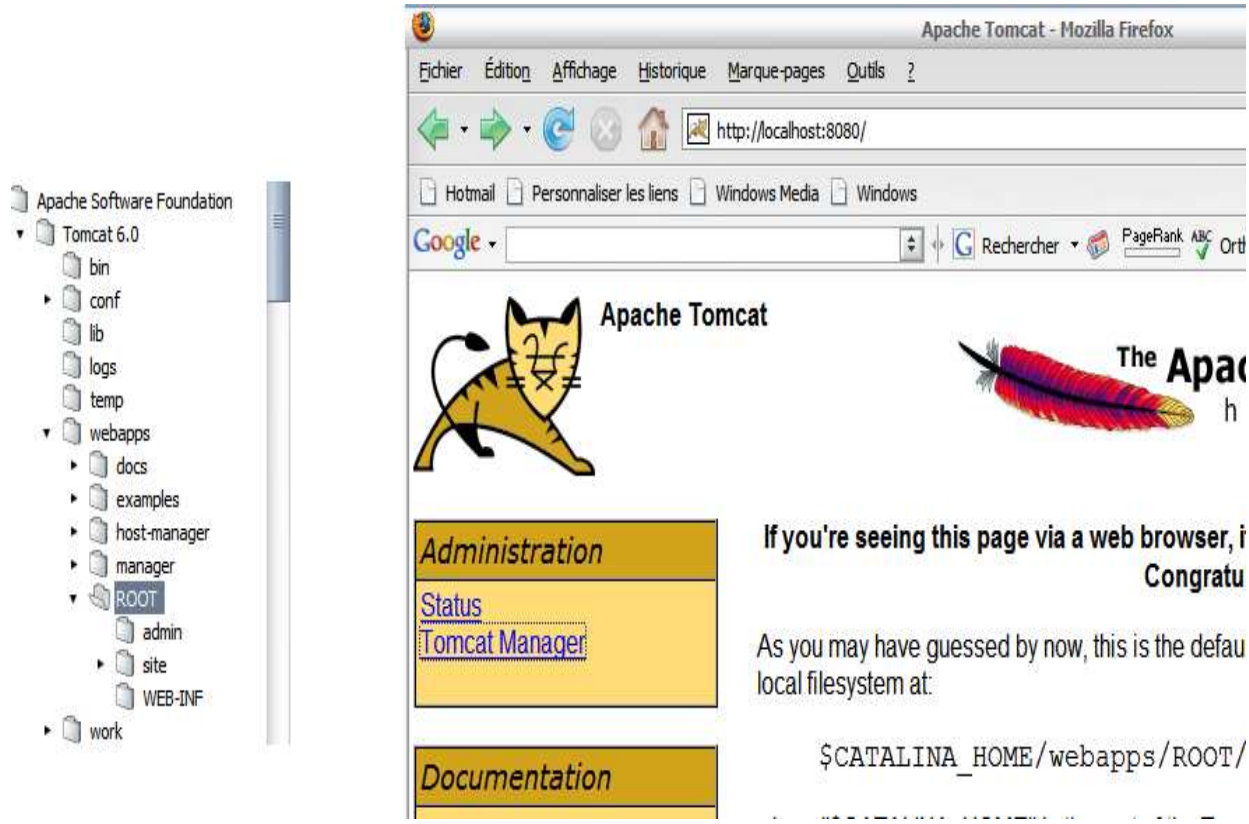


Figure 5.08 : Correspondance entre l'adresse URL et le répertoire ROOT

En fait, pour déployer un projet Tomcat il faut lancer le Tomcat Manager et de spécifier le contexte d'accès à l'URL ainsi que le chemin absolu du répertoire où se trouve le projet Tomcat.

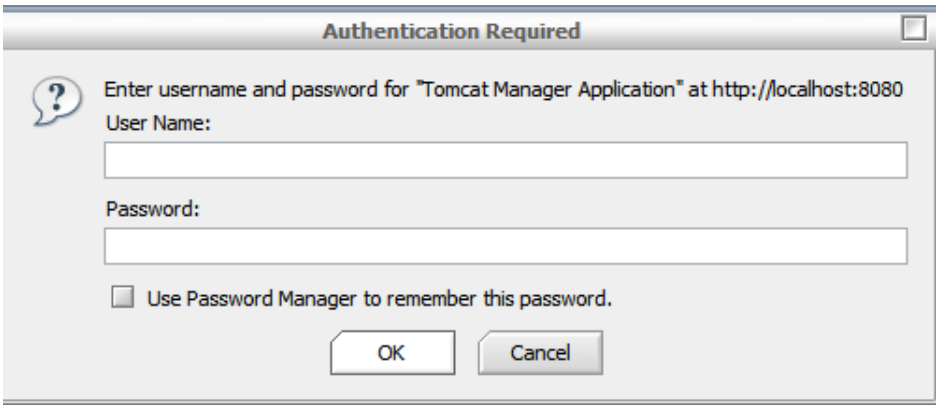


Figure 5.09 : Authentification de l'utilisateur

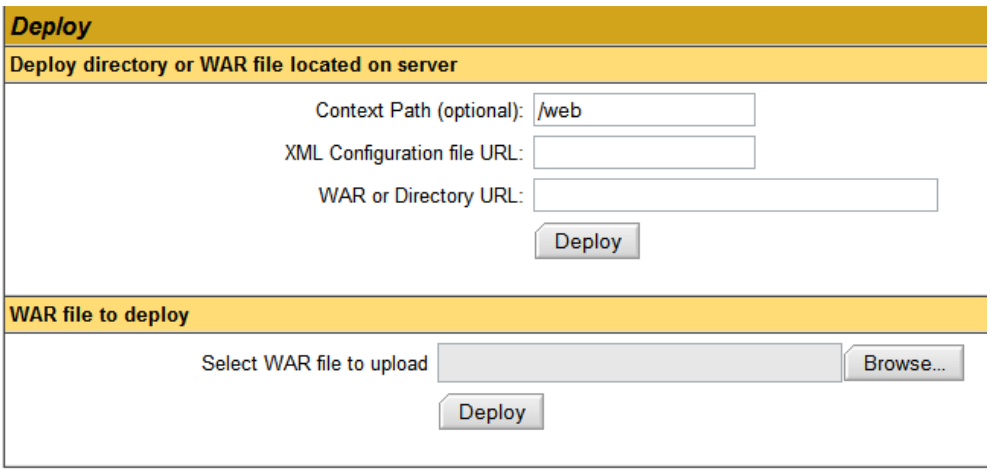


Figure 5.10 : déploiement du projet Tomcat en utilisant Tomcat Manager

Si l'opération a réussi il suffit alors de lancer l'application en appuyant sur le lien /web, toujours dans Tomcat Manager dans le tableau des applications.

/host-manager	Tomcat Manager Application	true	0	Démarrer Arréter Expire sessions
/manager	Tomcat Manager Application	true	1	Démarrer Arréter Expire sessions
/web		true	0	Démarrer Arréter Expire sessions

Figure 5.11 : liste des applications déployées dans Tomcat

5.5. Mise en œuvre de l'application

Ce paragraphe explicite la mise en œuvre effective de l'application. Il sera question de l'interface utilisateur qui comprendra les interactions accessibles à l'utilisateur, à savoir l'inscription, pour les nouveaux membres, l'authentification, pour les membres déjà inscrits et l'accès à l'application, pour les membres authentifiés.

5.5.1. *Règles de gestion*

Nous allons citer les règles de gestion et la règle de calcul régissant décrites à partir de la description du fonctionnement de l'application:

RG1 : Les internautes désirants achetés des services doivent être un membre

RG2 : Les membres voulant effectuer une transaction doit d'abord s'authentifier

RG3 : Un membre doit être client d'une banque et/ou abonné à un opérateur de téléphonie mobile

RG4 : Tout visiteur du site peut s'inscrire et devenir membre

RG5 : Une inscription doit contenir les renseignements du membre :

- Code membre
- Pseudo membre
- Mot de passe
- Nom
- Prénom
- Date de naissance
- Numéro de CIN
- E-mail
- Adresse
- Date d'adhésion
- Sexe
- Numéro carte bancaire
- Code carte bancaire
- Numéro de téléphone (mobile)
- Numéro de la carte SIM

RG6 : Les renseignements concernant le membre, sa banque et/ou l'opérateur doivent être égaux.

RG7 : Un membre authentifier obtient une session jusqu'à déconnexion

RG8 : Un membre peut acheter 0 ou plusieurs services

RG9 : Un service peut être acheté par l'un des deux modes de payement

RG10 : Le mode de payement se fait soit par carte bancaire soit par crédit téléphonique

RG11 : Un service comporte :

- Un code
- Une désignation
- Une catégorie
- Un prix

RG12 : Le mode de payement comporte

- Un code
- Un nom (bancaire ou crédit)

RG13 : Un membre une fois authentifié possède un caddy à sa disposition

RG14 : Un caddy comporte :

- Un code
- Le code service
- Le mode de payement

RG15 : Toute transaction doit être enregistrée

RG16 : Les services mis dans un caddy peuvent être supprimés et ajoutés avant validation de la transaction

RG17 : Le total des services a acheté doit être inférieur ou égal au solde (respectivement au crédit) dans la banque (respectivement à l'opérateur mobile)

RG18 : Un achat doit faire l'objet de :

- Code membre
- Pseudo membre
- Numéro CIN
- Code caddy
- Mode de payement
- Prix total
- Date de l'achat

RG19 : Une transaction n'est effectuée qu'après une validation (commit)

RC : Prix total = $\sum$ prix d'un service

5.5.2. *Organigramme*

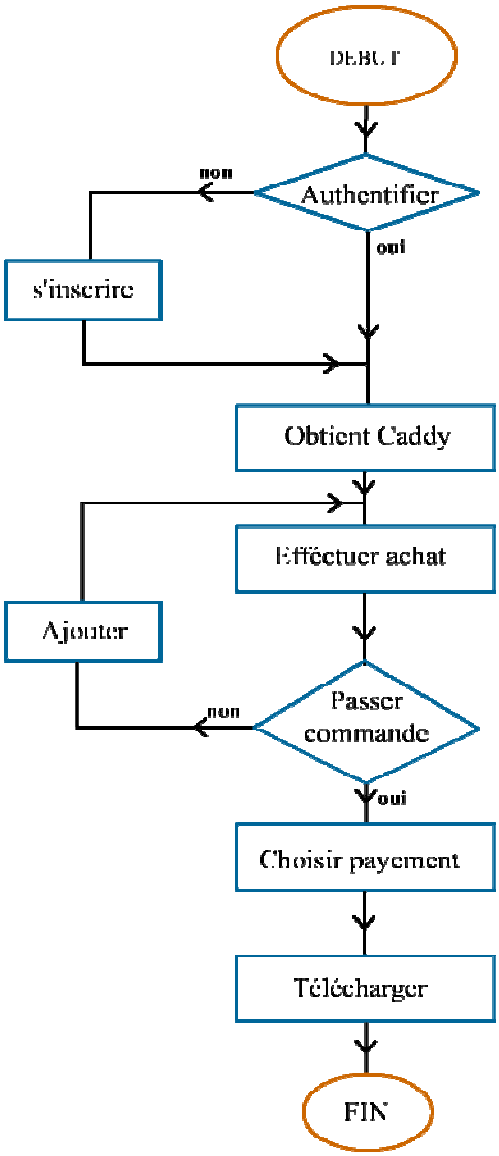


Figure 5.12 : Organigramme de l'application

5.5.3. *Exemples de pages de l'application*

5.5.3.1. Inscription

La mise en commun d'une ressource requiert paradoxalement la restriction de l'accès à cette ressource. En effet, pour protéger l'application des agressions extérieures (piratage) il est d'usage de restreindre l'accès aux ressources sensibles. Cette pratique est d'autant plus vérifiée dans le cas d'internet où chaque page web est visible et accessible par tout un chacun. Pour combler ce manque, on fait appel à un SGBD qui stocke la liste des utilisateurs pouvant accéder à la

ressource. Ainsi, on doit soumettre une inscription pour pouvoir figurer dans cette liste. Une fois inscrit, on pourra ultérieurement accéder à la ressource.

La page d'inscription peut se présenter comme suit :

Figure 5.13 : Page d'inscription

Le formulaire d'inscription comprend treize (13) champs à remplir. Premièrement : le pseudo du demandeur et son mot de passe, qui vont lui servir d'authentification pour sa prochaine visite. Deuxièmement, son nom, prénoms, date de naissance, CIN et son adresse actuelle qui doivent être identique aux renseignements enregistrés dans la base de données de sa banque ainsi que dans la base de données de l'opérateur mobile où il est abonné. Troisièmement, son adresse e-mail pour l'informer des nouveautés et finalement, son numéro de compte en banque, le code de sa carte, son

numéro de téléphone, et le numéro de la carte SIM pour l'identifier d'une part et d'autre part pour permettre une transaction.

Le système doit garantir l'unicité de chaque couple {pseudo, mot de passe}, c'est-à-dire que deux utilisateurs différents ne peuvent posséder le même login et le même mot de passe.

5.5.3.2. Authentification ou login

Chaque utilisateur qui veut accéder à la ressource doit s'authentifier par rapport à la liste des utilisateurs de la base de données. L'accès est permise dans le cas où l'utilisateur y est identifié et il est redirigé vers une autre page sinon.

La page de login se présente comme suit :

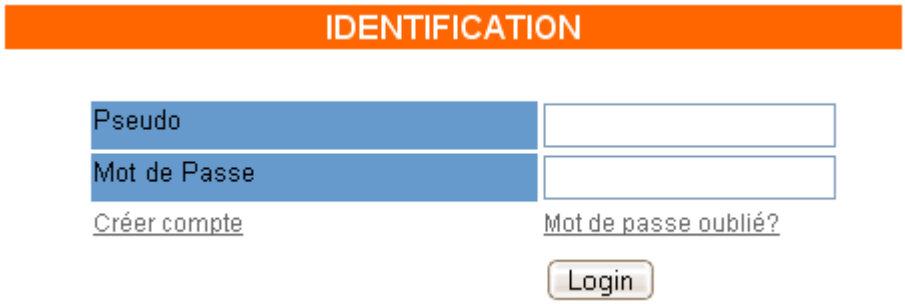


Figure 5.14 : Fenêtre d'identification

Si un enregistrement non vide est obtenu, une variable session est attribuée à l'utilisateur.

5.5.3.3. Page représentant la liste des différents services

Les services à vendre sont : les musiques (sonneries et mp3), les jeux, les vidéos et les photos (fond d'écran et animations). Un membre peut acheter un ou plusieurs de ces services à chaque session.

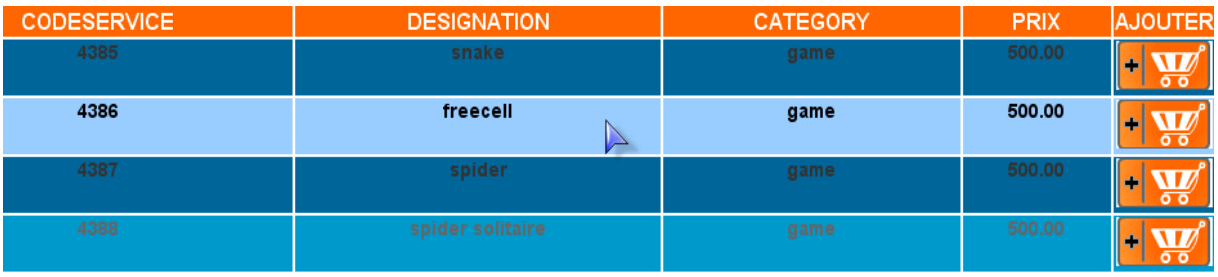


Figure 5.15 : Exemple de service

Si l'utilisateur finit de choisir les services qu'il veut acheter il peut voire la liste de ces services

contenus dans son caddy. Devant cette page il peut encore supprimer ou ajouter des services dans le caddy ou bien passer la commande tout en choisissant le mode de payement.

Le système traite donc la transaction en commençant par calculer si le prix total des achats est inférieur ou égal au compte enregistré dans sa banque et/ou le crédit restant dans son téléphone. Si la transaction est permise alors il a « commit » sinon il est redirigé vers la page mentionnant la liste des achats qui lui contraint de supprimer un ou plusieurs de ses choix.

Figure 5.16 : Page d'achat

Une fois la transaction terminée il peut tout de suite obtenir les services ainsi achetés par téléchargement. Et pour quitter il doit terminer sa session par une déconnexion

5.6. Conclusion

Ce chapitre concrétise les théories vues dans les chapitres précédents. En effet, cette application mise en œuvre avec le composant web JSP répond aux spécifications de J2EE. L'accès à la page est sécurisé par une authentification de chaque utilisateur. Les pages nous permettent de choisir des services et de les acheter, en ligne via internet, avec notre compte bancaire et/ou avec notre crédit de téléphone.

CONCLUSION GENERALE

La plate-forme J2EE est architecturée selon un modèle multi-niveau mettant en jeu *du côté client* le langage HTML, les applets et les applications JAVA, puis du *côté serveur* un serveur web incluant les JSP, les Servlets JAVA et un serveur pour les composants métiers, et sur un autre niveau le système d'information de l'entreprise ou la base de données.

Alors que la plateforme J2EE gagne en importance (jusqu'à représenter, selon certaines études, 60% du marché des serveurs d'applications), le nombre et la qualité des outils sous licence Open Source rendent leur utilisation réaliste en entreprise. Un bon exemple est Tomcat qui est maintenant largement utilisé sur des sites d'entreprise.

Cet ouvrage nous a démontré la portabilité, la robustesse et la fiabilité des produits et APIs offerts par Sun Microsystem. En effet, nous avons pu voir que le développement d'application web fait avec les technologies servlets et JSP offrent aux développeurs des solutions de facilité tout en restant sûr et fiable.

Ce type d'application requiert la maitrise de différents types de langage, de script et aussi un sens artistique au niveau du design. Il ne faut donc pas négliger ni prendre à la légère chaque étape pour mettre en œuvre l'application. Elle suit des parcours depuis la conception de la base de données à partir de la méthode Merise jusqu'à la création des pages jsp et html. Chaque étape constitue un maillon nécessaire pour l'optimisation du site, sa bonne marche et surtout sa sécurité.

Ce mémoire nous permet de voir le fonctionnement ainsi que la conception des sites web utilisant les systèmes de paiement en ligne avec une gestion de session. Il est à noter qu'en pratique les banques, après négociation, nous donnent une portion de code crypté qui va nous permettre d'accéder à leur base de données.

ANNEXES

ANNEXE 1 : ARCHITECTURE DES RESEAUX

L'architecture de base des réseaux à sept couches est un modèle dit d'interconnexion des systèmes ouverts, appelé encore OSI (Open Systems Interconnection) a été défini par l'ISO (International Standards Organization). Le modèle OSI répartit les protocoles utilisés selon sept couches, définissant ainsi un langage commun pour le monde des télécommunications et de l'informatique. Il constitue aujourd'hui le socle de référence pour tous les systèmes de traitement de l'information. Chaque couche regroupe des dispositifs matériels (dans les couches basses) ou logiciels (dans les couches hautes). Entre couches consécutives sont définies des interfaces sous forme de primitives de service et d'unités de données rassemblant les informations à transmettre et les informations de contrôle rajoutées. En principe, une couche ne peut communiquer qu'avec les couches de rangs immédiatement supérieur et inférieur

Couche 1 - Physique : rassemble les moyens électriques, mécaniques, optiques ou hertziens par lesquels les informations sont transmises. Les unités de données sont donc des bits 0 ou 1.

Couche 2 – Liaison : gère la fiabilité du transfert de bits d'un nœud à l'autre du réseau, comprenant entre autres les dispositifs de détection et de correction d'erreurs, ainsi que les systèmes de partage des supports. L'unité de données à ce niveau est appelée trame.

Couche 3 – Réseau : aiguille les données à travers un réseau à commutation. L'unité de données s'appelle en général un paquet.

Couche 4 – Transport : regroupe les règles de fonctionnement de bout en bout, assurant ainsi la transparence du réseau vis-à-vis des couches supérieures. Elle traite notamment l'adressage, l'établissement des connexions et la fiabilité du transport.

Couche 5 – Session : réunit les procédures de dialogue entre les applications établissement et interruption de la communication, cohérence et synchronisation des opérations.

Couche 6 – Présentation : traite les formes de représentation des données, permettant la traduction entre machines différentes.

Couche 7 – Application : rassemble toutes les applications qui ont besoin de communiquer par le réseau : messagerie électronique, transfert de fichiers, http, etc.

ANNEXE 2 : RAPPELS HTML

1. Le langage

HTML est un langage de Tags (balises) qui sont des délimiteurs de début et fin d'action. On trouvera par exemple le tag HTML <html> </html>, le tag HEAD <head> </head> et le tag BODY <body> </body> dans la plupart des pages.

La page Minimum permettant d'afficher des informations est réduite aux tags suivants:

<html>
<head>
<title> ... </title>
</head>
<body>
</body>
</html>

Le texte mis dans la section <title> n'est pas affiché directement sur la page HTML sur le poste client. Il est utilisé pour nommer la page dans les bookmarks ou la barre d'état des différents Browsers.

2. Formatage de texte

Afin de gérer les titres et sous-titres il existe 6 niveaux prédéfinis de titres <h1>...</h1> à <h6>...</h6>.

De nombreux tags servent à enrichir le texte :

italique ...
gras ...
clignotant <blink>...</blink>
centré ...
renforcé ...
taille d'une police ...
couleur d'une police ...

D'autres tags permettent d'indenter le texte affiché :

paragraphe <p>
retour ligne

filet <hr>

épaisseur du filet <hr size=n>

longueur du filet <hr align=center width=50%>

3. Listes et énumérations

Listes standard :

 item1

 item2

Listes numérotées :

 item1

 item2

Les sous-listes sont obtenues par imbrication de tags <ul>...

4. Les liens hypertextes

Un lien représente un saut vers un autre endroit. Il est caractérisé par un texte visible par l'utilisateur et une adresse de destination appelée quand on clique sur le lien.

texte affiché

On peut créer un lien externe vers une autre page du domaine local :

 cliquer ici

ou même sur un autre serveur :

 chez Amazon

Le lien peut être un simple texte ou même une photo (tag <img>).

L'attribut target dans le lien permet de spécifier la fenêtre de destination de la page à charger.

Ceci est notamment utile dans les sites qui contiennent des frames puisque cet attribut permet de préciser dans quelle sous-fenêtre du frame la page sera chargée. On notera que si la chaîne spécifiée dans l'attribut target n'est pas connue dans les pages, le Browser créera une nouvelle instance du Browser pour afficher la nouvelle page.

5. Les ancres

Le lien peut pointer sur une autre page HTML mais aussi sur un endroit précis de la même page.
Définition d'une ancre <a name="nomAncre">
lien interne vers l'ancre de la même page cliquer ici
Un lien externe peut aussi contenir une ancre vers un endroit précis de l'autre document:
cliquer ici

6. Rappel sur l'URL

Une URL permet d'identifier précisément une ressource sur l'internet
forme : ressource://host.domaine:port/pathname
ressource : file, http, news, gopher, telnet, wais, mailto
Un lien peut donc pointer sur le composeur de mail courrier PM ou sur la récupération d'un fichier par ftp un soft
Par défaut, le serveur HTTP est placé sur le port 80, la machine locale se nomme localhost et est accessible via l'adresse IP 127.0.0.1

7. Les images

Formats standard reconnus : GIF ou JPEG
<img src="nomImage">
<img alt="texte" src="nomImage">
<img src="nomImage" width=100 height=100>

8. Les tableaux

Le tableau est délimité par <table>...</table> les lignes sont délimitées par <tr>...</tr> les colonnes sont délimitées par <td>…</td>
La taille des cases est fixée par leur contenu

9. Les frames

Les Frames permettent de diviser une page HTML en plusieurs fenêtres indépendantes. Généralement l'un des Frames contient l'index avec les liens sur les pages concernées, tandis que l'autre présente la page en cours d'affichage.

Page principale de définition des Frames (sans tag <body>)

```
<html>
…
<frameset cols = "25%, 50%, *">
<frame src = "pageIndex.html" name = "index">
<frame src = "pageMain.html" name = "main">
<frame src = "pageNotes.html" name = "notes">
</frameset>
</html>
```

L'attribut name est très important puisque c'est lui qui permet d'identifier la sous-fenêtre du frame dans laquelle les futures pages sont chargées. C'est l'attribut target d'un lien hypertexte qui permet de spécifier le nom de la sous-fenêtre de destination. Par exemple le lien

affichage du d_etail permettra d'afficher la page detail.html dans la colonne notes du frame précédente.

10. Les Forms

Les Forms permettent de définir des interfaces d'échange d'informations entre le serveur http et l'utilisateur (CGI, Servlets)

```
<form action="url" method="post">
entrez votre nom <INPUT TYPE="text" SIZE="20" NAME=entree>
<br> <INPUT TYPE="submit"><INPUT TYPE="reset">
</form>
```

Le champ URL contient l'adresse à laquelle le formulaire sera envoyé. Le champ method contient le type de méthode utilisé pour envoyer les paramètres. Il en existe 2: GET et dans ce cas le contenu de la Form est concaténé à l'URL et POST et dans ce cas le contenu est passé par un message de requête. Le champ TYPE contient le type d'objet graphique que l'on souhaite afficher.
Il peut contenir notamment les valeurs text, hidden, password, checkbox, radio, submit et reset en fonction de l'objet que l'on souhaite afficher.
L'attribut name est lui aussi très important puisque c'est lui qui permettra d'identifier l'objet dans une Servlet ou un cgi. C'est la méthode getParameter("entree") d'une Servlet qui permet par exemple de récupérer la valeur saisie dans le champs texte nommé entree.

ANNEXE 3 : CODE D'AUTHENTIFICATION

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*" %>
<%@ include file="../Connections/connection.jsp" %>
<%
// *** Validate request to log in to this site.
String MM_LoginAction = request.getRequestURI();
if (request.getQueryString() != null && request.getQueryString().length() > 0) {
String queryString = request.getQueryString();
String tempStr = "";

for (int i=0; i < queryString.length(); i++) {
if (queryString.charAt(i) == '<') tempStr = tempStr + "&lt;";
else if (queryString.charAt(i) == '>') tempStr = tempStr + "&gt;";
else if (queryString.charAt(i) == '"') tempStr = tempStr +  "&quot;";
else tempStr = tempStr + queryString.charAt(i);
}
MM_LoginAction += "?" + tempStr;
}
String MM_valUsername=request.getParameter("idPseudo");
if (MM_valUsername != null) {
String MM_fldUserAuthorization="";
String MM_redirectLoginSuccess="../unused/string.jsp";
String MM_redirectLoginFailed="../utilities/login.jsp";
String MM_redirectLogin=MM_redirectLoginFailed;
Driver MM_driverUser = (Driver)Class.forName(MM_connection_DRIVER).newInstance();
Connection MM_connUser =
DriverManager.getConnection(MM_connection_STRING,MM_connection_USERNAME,MM_connection_PASSWORD);
String MM_pSQL = "SELECT *";
if (!MM_fldUserAuthorization.equals("")) MM_pSQL += "," + MM_fldUserAuthorization;
MM_pSQL += " FROM simulation.membre WHERE PSEUDOMBR=\'" + MM_valUsername.replace('\'', ' ') + "\'
AND PASSWMBR=\'" + request.getParameter("idPass").toString().replace('\'', ' ') + "\'";
PreparedStatement MM_statementUser = MM_connUser.prepareStatement(MM_pSQL);
ResultSet MM_rsUser = MM_statementUser.executeQuery();
boolean MM_rsUser_isNotEmpty = MM_rsUser.next();
if (MM_rsUser_isNotEmpty) {
// username and password match - this is a valid user
session.putValue("MM_Username", MM_valUsername);
session.putValue("codecaddy", codecaddy);

if (!MM_fldUserAuthorization.equals("")) {
session.putValue("MM_UserAuthorization", MM_rsUser.getString(MM_fldUserAuthorization).trim());
} else {
session.putValue("MM_UserAuthorization", "");
}

// insert datas in achat table

int codembr=MM_rsUser.getInt("CODEMBR");
String pseudombr=MM_valUsername;
int cinmbr=MM_rsUser.getInt("CINMBR");

String ins="insert into simulation.achat (CODEMBR, PSEUDOMBR, CINMBR,CODECADDY) values
("+codembr+","+"\""+pseudombr+"\""+","+cinmbr+", "+codecaddy+");";
MM_statementUser.executeUpdate(ins);

if ((request.getParameter("accessdenied") != null) && false) {

MM_redirectLoginSuccess = request.getParameter("accessdenied");
}

MM_redirectLogin=MM_redirectLoginSuccess;
}
MM_rsUser.close();
MM_connUser.close();
response.sendRedirect(response.encodeRedirectURL(MM_redirectLogin));
return;
}
%>
```

ANNEXE 4: CODE SOURCE JEUX.JSP

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*" %>
<%@ include file="Connections/connection.jsp" %>
<%
String Recordset1__MMColParam = "GAME";
if (request.getParameter("MM_EmptyValue") !=null) {Recordset1__MMColParam =
(String)request.getParameter("MM_EmptyValue");}
%>
<%
Driver DriverRecordset1 = (Driver)Class.forName(MM_connection_DRIVER).newInstance();
Connection ConnRecordset1 =
DriverManager.getConnection(MM_connection_STRING,MM_connection_USERNAME,MM_connection_PASSWORD);
PreparedStatement StatementRecordset1 = ConnRecordset1.prepareStatement("SELECT * FROM
simulation.service WHERE CATEGORY = '" + Recordset1__MMColParam + "'");
ResultSet Recordset1 = StatementRecordset1.executeQuery();
boolean Recordset1_isEmpty = !Recordset1.next();
boolean Recordset1_hasData = !Recordset1_isEmpty;
Object Recordset1_data;
int Recordset1_numRows = 0;
%>
<%
int Repeat1__numRows = 10;
int Repeat1__index = 0;
Recordset1_numRows += Repeat1__numRows;
%>
<%String classe="tabnormal";
int test=0;  %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Document sans titre</title>
<link href="stylesheet/mystyle.css" rel="stylesheet" type="text/css" />
</head>

<body>

<table border="0" align="center">
  <tr>
    <td class="titre">CODESERVICE</td>
    <td class="titre">DESIGNATION</td>
    <td class="titre">CATEGORY</td>
    <td class="titre">PRIX</td>
    <td class="titre" width="75">AJOUTER</td>
  </tr>
  <% while ((Recordset1_hasData)&&(Repeat1__numRows-- != 0)) { %>
        <tr align="center" class="<%
                  if(test==0)
                  {
                        classe="tabnormal";
                        out.print(classe);
                        test=1;
                  }
                  else
                  {
                        classe="tabnormal_1";
                        out.print(classe);
                        test=0;
                  }  %>" onmouseover="this.className='tabover';" onmouseout="this.className='<%
out.print(classe); %>';">

      <td><%=(((Recordset1_data = Recordset1.getObject("CODESERVICE"))==null ||
Recordset1.wasNull())?"":Recordset1_data)%></td>
      <td ><%=(((Recordset1_data = Recordset1.getObject("DESIGNATION"))==null ||
Recordset1.wasNull())?"":Recordset1_data)%></td>
      <td ><%=(((Recordset1_data = Recordset1.getObject("CATEGORY"))==null ||
Recordset1.wasNull())?"":Recordset1_data)%></td>
      <td ><%=(((Recordset1_data = Recordset1.getObject("PRIX"))==null ||
Recordset1.wasNull())?"":Recordset1_data)%></td>
```

```
      <td width="75" height="35"><object  classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=7,0,19,0"
width="75" height="35">
        <param name="movie" value="flash/caddy.swf?codeservice=<%=(((Recordset1_data =
Recordset1.getObject("CODESERVICE"))==null ||
Recordset1.wasNull())?"":Recordset1_data)%>&codecaddy=<%= session.getValue("codecaddy") %>" />
        <param name="quality" value="high" />
        <param name="allowScriptAccess" value="always" />
        <embed  src="flash/caddy.swf?codeservice=<%=(((Recordset1_data =
Recordset1.getObject("CODESERVICE"))==null ||
Recordset1.wasNull())?"":Recordset1_data)%>&codecaddy=<%= session.getValue("codecaddy") %>"
"width="75" height="35" quality="high" pluginspage="http://www.macromedia.com/go/getflashplayer"
type="application/x-shockwave-flash" allowScriptAccess="always"></embed>
      </object></td>
    </tr>
    <%
  Repeat1__index++;
  Recordset1_hasData = Recordset1.next();
}
%>
</table>
</body>
</html>
<%
Recordset1.close();
StatementRecordset1.close();
ConnRecordset1.close();
%>
```

ANNEXE 5 : CODE SOURCE SHOWCADDY.JSP

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"
errorPage="" %>
<%@ include file="Connections/connection.jsp" %>

<%
Driver DriverRecordset2 = (Driver)Class.forName(MM_connection_DRIVER).newInstance();
Connection ConnRecordset2 =
DriverManager.getConnection(MM_connection_STRING,MM_connection_USERNAME,MM_connection_PASSWORD);
PreparedStatement StatementRecordset2 = ConnRecordset2.prepareStatement("SELECT * FROM
simulation.service");
ResultSet Recordset2 = StatementRecordset2.executeQuery();
boolean Recordset2_isEmpty = !Recordset2.next();
boolean Recordset2_hasData = !Recordset2_isEmpty;
Object Recordset2_data;
int Recordset2_numRows = 0;
%>
<%
int Repeat1__numRows = 10;
int Repeat1__index = 0;
Recordset2_numRows += Repeat1__numRows;
String classe="tabnormal";
int test=0;
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Document sans titre</title>
<link href="stylesheet/mystyle.css" rel="stylesheet" type="text/css" />
</head>

<body>

<table border="0">
<form action="" method="get" name="formulaire" target="_parent">
   <tr class="titre">
    <td>codeservice</td>
    <td>designation </td>
    <td>prix</td>
    <td>payement</td>
        <td>remove</td>
  </tr>
  <% while ((Recordset2_hasData)&&(Repeat1__numRows-- != 0)) { %>
    <tr  align="center" class="<%
                if(test==0)
                {
                     classe="tabnormal";
                     out.print(classe);
                     test=1;
                }
                else
                {
                     classe="tabnormal_1";
                     out.print(classe);
                     test=0;
                }  %>" onmouseover="this.className='tabover';" onmouseout="this.className='<%
out.print(classe); %>';">
      <td><%=(((Recordset2_data = Recordset2.getObject("codeservice"))==null ||
Recordset2.wasNull())?"":Recordset2_data)%></td>
      <td><%=(((Recordset2_data = Recordset2.getObject("designation"))==null ||
Recordset2.wasNull())?"":Recordset2_data)%></td>
      <td><%=(((Recordset2_data = Recordset2.getObject("prix"))==null ||
Recordset2.wasNull())?"":Recordset2_data)%><input name="price" type="hidden"
value="<%=(((Recordset2_data = Recordset2.getObject("prix"))==null ||
Recordset2.wasNull())?"":Recordset2_data)%>" /></td>
      <td>
      <label></label><label></label>

      <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
```

```
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=7,0,19,0"
width="175" height="25" class="tabnormal">
        <param name="movie" value="flash/a effacer/radio.swf?rang= <%=Repeat1__index
%>&price=<%=(((Recordset2_data = Recordset2.getObject("PRIX"))==null ||
Recordset2.wasNull())?"":Recordset2_data)%>" />
        <param name="quality" value="high" />
        <embed src="flash/a effacer/radio.swf?rang=<%=Repeat1__index
%>&price=<%=(((Recordset2_data = Recordset2.getObject("PRIX"))==null ||
Recordset2.wasNull())?"":Recordset2_data)%>" width="175" height="25" quality="high"
pluginspage="http://www.macromedia.com/go/getflashplayer" type="application/x-shockwave-
flash"></embed>
      </object>
      <td><input name="vider" type="image"  title="vider"src="flash/corbeille.png" />

      </td>
    </tr>
    <%
  Repeat1__index++;
  Recordset2_hasData = Recordset2.next();
 // j++;
}
%>
<tr> <td colspan="5"><input name="Submit" type="button" id="Submit" value="commit" />
      <label>
      <input type="button" name="Submit" value="rajouter" />
      </label>
<div align="right">
        <p>Total        
          <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=7,0,19,0"
width="20" height="20">
            <param name="movie" value="flash/a effacer/send.swf?total= <%=Repeat1__index %>" />
            <param name="quality" value="high" />
            <embed src="flash/a effacer/send.swf?total= <%=Repeat1__index %>" quality="high"
pluginspage="http://www.macromedia.com/go/getflashplayer" type="application/x-shockwave-flash"
width="20" height="20"></embed>
          </object>

               :
          <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=7,0,19,0"
width="300" height="100">
            <param name="movie" value="flash/a effacer/label.swf" />
            <param name="quality" value="high" />
            <embed src="flash/a effacer/label.swf" quality="high"
pluginspage="http://www.macromedia.com/go/getflashplayer" type="application/x-shockwave-flash"
width="300" height="100"></embed>
          </object>

        </p>
        </div></td></tr>
</form>
</table>
</body>
</html>
<%
Recordset2.close();
StatementRecordset2.close();
ConnRecordset2.close();
%>
```

BIBLIOGRAPHIE

[1] S.Bodoff, D.Green, E.Jendrock, M.Pawlan, B.Stearns « The J2EE Tutorial », Sun Microsystems, 2001

[2] E.Randriarijaona, *Développement d'application d'entreprise*, cours 5è Année, Dép. Tél-E.S.P.A, A.U. : 2007-2008

[3] B.Shannon, « Java ™ 2 Platform Enterprise Edition Specification, v1.4 », Novembre 2003

[4] J.Couch, D.H.Steinberg, « JAVA 2 Enterprise Edition BIBLE », Hungry Minds, 2002

[5] J.L.Maitre, *Les bases de données relationnelles et leurs systèmes de gestion*, Département d'Informatique UFR des Sciences et Techniques Université de Toulon et du Var

[6] P.Mathieu, *Base de données (De Merise à JDBC)*, Université Des Sciences et Technologies de Lille, Mai 1999

[7] L.Rabeherimanana, *Système d'information*, cours 5è Année, Dép. Tél-E.S.P.A., A.U. : 2007-2008

[8] S.Miranda, J.M.Busta, *L'art des bases de données, volume 1 et 2*. Eyrolles,Paris, 1990.

[9] G.E Randria, *Développement Web et Service Web Xml*, cours 5ème année, Dép. Tél.-E.S.P.A. A.U. :2007-2008

[10] http://www.developpez.com/

[11] G.Reese, *JDBC et Java, guide du programmeur*, Edition O'Reilly, Paris, France, 1997

[12] L.R.Sébastien, *Mise en place d'une authentification via un formulaire sous Tomcat*, Octobre 2005

[13] H.Bergsten, « JavaServer Pages, 3rd Edition », Sun Microsystem, 2003

[14] I.Patrick, *Java côté Serveur, Servlet et JSP*, INRIA - Sophia Antipolis, Août 2000

[15] R.Ghurbhurn, *Introduction aux Web Services*, Ecole Nationale Supérieure des Mines Saint-Etienne, Août 2004

[16] B.Basham, K.Sierra, B.Bates, « Heads First Servlets and JSP, 2nd Edition », Edition O'Reilly, 2007

[17] S.Tahe, *Introduction à la programmation WEB en Java, Servlet et pages JSP*, ISTIA-Angers, Septembre 2002

[18] S. Allamaraju, K. Avedal et R.Browett, *Programmation avec J2EE, Conteneurs J2EE, servlets, JSP et EJB,* Eyrolles, 2003

FICHE DE RENSEIGNEMENT

Nom : RAKOTONDRAINA

Prénoms : Tahina Ezéchiel

Adresse de l'auteur : Lot 21-B-40 Sud Voirie
110 Antsirabe
MADAGASCAR
Tel : (261)33 11 761 10 / (261)32 43 935 41
E-mail : tahinaezechiel@yahoo.fr

Titre de mémoire *: **J2EE : MISE EN ŒUVRE D'UN SITE WEB, PAR PAIEMENT EN LIGNE, DE TELECHARGEMENT DE SERVICE D'APPLICATION POUR TELEPHONE MOBILE***

Nombre de pages : 110

Nombres de tableaux : 10

Nombre de figures : 46

Mots clés : J2EE, Application web, JAVA, Merise, Système d'information, payement en ligne, Base de données, JSP

Directeur de mémoire : RANDRIARIJAONA Lucien Elino

RESUME

La plate-forme Java 2 Enterprise Edition (J2EE) est un standard en matière de développement d'un système à objets distribués dans le cadre d'une application web. Avec ses APIs, il a été conçu pour simplifier les problèmes complexes avec le développement, le déploiement et la gestion de l'architecture multi-tier.

Le langage Java permet à l'application d'être déployée sur n'importe quel ordinateur (Serveur, PC, MAC …) équipé des systèmes d'exploitation usuels tels Windows, Linux, MacOS, …

ABSTRACT

The Platform Java 2 Enterprise Edition (J2EE) is a standard in case of developing the system of distributed computing in web application. J2EE was designed to simplify complex problem with the development, deployment and management of multi-tier enterprise solutions.
Java allows the application to be deployed in a variety computer (Server, PC, MAC …) running on a variety of operating system such Windows, Linux, MacOS …