

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministre De L'enseignement Supérieur Et
De La Recherche Scientifique

Université SAAD DAHLAB De BLIDA

Faculté des Sciences

Département d'informatique

Projet de fin d'études

En vu de l'obtention du
Diplôme d'Ingénieur d'état en informatique

Option : *Intelligence artificielle*

Thème

**Implémentation et test d'une
MIDlet complète**

Proposé et dirigé par :

Dr Mahieddine Mohamed

Exposé par :

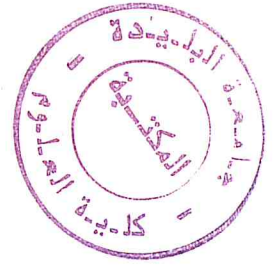
Mr MAZARI Redha



Promotion : *Octobre 2004*

MIG-004-30-1

Remerciements



Avant tout je remercie « Allah » qui m'a aidé à réaliser ce modeste travail.

Je tiens à remercier mon promoteur M^r. MAHIEDDINE Mohammed pour son aide, sa patience, sa disponibilité, et sa compréhensibilité.

Je remercie les membres du jury pour m'avoir fait l'honneur de juger mon travail.

J'adresse mes sincères remerciements à M^{me} Docteur OUKID pour nous avoir fourni un bon environnement de travail.

Je tiens aussi à exprimer ma profonde gratitude et mes vifs remerciements à toute ma famille surtout mes parents pour leur soutien durant toute ma scolarité.

Je remercie tous les enseignants de la faculté des sciences de BLIDA et surtout mes enseignants du département informatique.

Que tout le personnel du département d'informatique trouve ici ma reconnaissance pour son entière disponibilité.

Je remercie, de tout coeur, tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Dédicace

Je dédie ce modeste travail :

*A mes très chers parents pour leur soutien durant
toute ma carrière,*

*Pour leur bienveillance, leurs efforts constants dans mes études, et
Pour leurs encouragements,*

A tous mes frères et mes sœurs surtout le petit Ossama

et mes très chères nièces Hoda et Lina

A toute ma famille de proche et de loin

A mon promoteur M^{er}.Mahiedinne.

*A mes très chères amies : B. Amine, B. Bilelle, K. Abdelhake,
K. Ahmed, M. M^{ed}, L. M^{ed}, L. Sidali, A. Moustafa, B. Fayçale,
B. Othmen, H. Mohamed, O. Mhamed, T. Ahmed, T. Rodhouane,
S. Ahmed.*

A toute ma promotion.

A tous mes amis que j'aime et qui m'aiment.

Redha

Résumé

Le marché des télécommunications sans fil connaît depuis ces dernières années une croissance exponentielle. Le nombre d'abonnés au téléphone portable croît parallèlement au nombre d'utilisateurs d'ordinateurs personnels. Ces utilisateurs sont de plus en plus avides d'informations et de services, et demandent à les obtenir sur leurs appareils itinérants, sans fil.

Le but de ce projet est d'étudier et de mettre sur pied un environnement orientée objet apte à développer, héberger et déployer une application destinée à des téléphones portables.

L'application choisie consiste à développer une télé-consultation des notes au niveau de la scolarité par des étudiants à partir de leur téléphone mobile.

Mots clés :

MIDlet, Servlet, MIDP, J2ME, CLDC, JSP, JBean, communication MIDlet-Servlet, JDBC, MySql, TomCat Apache, programmation orientée objets, modélisation orientée objet, UML, Agile, XP.

Chapitre I : Introduction générale	1
Partie I : Généralités	4
Chapitre II : Modélisation orientée objets	5
1.Introduction	5
2.Les objets	5
3.Objets et classes	5
4.Messages entre les objets	5
5.Caractéristiques de la programmation orientée objets	6
6.Bénéfices de la P.O.O	6
7.Modélisation	6
8.Introduction au UML	7
9.Une vue générale des diagrammes UML	8
10.Pourquoi agile	9
11.Qu'est ce qu'une méthode Agile	9
12.Le Manifeste	10
13.Les valeurs	10
14.Les principes	10
15.La méthode agile XP (eXtreme Programming)	12
16.Conclusion :.....	13
Chapitre III : JAVA 2 Micro Edition et les applications MIDP	14
Introduction :.....	14
Partie I : La technologie J2ME	14
1. Définition	14
2. J2ME et les applications embarquées :	15
3. Particularités des applications destinées aux appareils mobiles	16
4. Exemples d'applications	17
5. L'architecture de j2ME	18
5.1. Les configurations	18
5.1.1. CDC: (Connected Device Configuration)	19
5.1.2. CLDC: (Connected Limited Device Configuration)	19
5.1.2.1. L'API CLDC 1.0 :	20
5.2. Les profils:	20
5.3. La machine virtuelle :	22
5.3.1 Compact Virtual Machine:	22
5.3.2. Kilo Virtual Machine :	22
5.4. Package optionnelle :	23
6. MIDP : Mobile Information Device Profile	23
Partie II : Les applications MIDP « les MIDlets »	25
1.Definition d'une MIDlet :	25
2. Création d'une MIDlet :	25
2.1. Ecriture de l'application :	25
2.2. Le cycle de vie des MIDlets :	26
2.3. Communication avec le gestionnaire d'application	27
2.3. Compilation et prévérification d'une MIDlet.....	28
2.4. Test de l'application :	28
2.5. Empaquetage de l'application :	28
2.6. Test de l'application empaquetée :	29
3. Distribution d'une MIDlet :	30
4. Exécution de MIDlet :	31

4.1. Exécution du <i>MIDlet</i> via le système de fichier :	31
4.2. Exécution du MIDlet depuis un serveur web :	31
5. Déploiement d'une MIDlet sur un terminal réel :	31
6. Programmation des interfaces utilisateur avec MIDP	32
6.1. Pourquoi ne pas réutiliser le AWT?	32
6.2. Les APIs MIDP du GUI	33
6.3. Le modèle MIDP des GUI	34
6.4. Le package LCDUI	35
6.5. Gestion des évènements	38
6.6. Exemple de <i>MIDlet</i>	38
7. La base de données légère	42
7.1 gestion des records store	42
8. Connexion réseaux :	42
8.1. Le GCF (Generic Connection Framework)	43
8.2. Utilisation	44
8.3. Programmation avec les sockets :	45
8.4. Programmation avec les Datagrammes	46
8.5. Programmation avec HTTP:	46
9. Les outils de développement	47
9.1. J2ME Wireless Toolkit	48
9.1.1. Obtenir le kit de développement	48
9.1.2 utiliser le Wireless Toolkit	48
10. Les concurrents de j2me	50
10.1. Microsoft Embedded Architecture	50
10.2. Binary Runtime Environment for Wireless (BREW)	51
11. Conclusion	53
Chapitre IV : Java est les applications web	55
1. Introduction	55
Partie I : Les applications Web	55
2. Présentation de l'architecture d'un système client/serveur	55
3. Avantages de l'architecture client/serveur	56
4. Inconvénients du modèle client/serveur	56
5. Fonctionnement d'un système client/serveur	56
6. Les types des applications client-serveur	57
6.1. Architecture à deux niveaux	57
6.2. Architecture à trois niveaux	58
6.2.1. Inconvénients de l'architecture trois-tiers	58
Partie II : Les Servlets	59
7. Définition	59
8. Avantages des Servlets par rapport aux CGI traditionnels	60
8.1. Efficacité :	60
8.2. Simplicité :	60
8.3. Puissance :	60
8.4. Portabilité :	61
8.5. Sécurité :	61
8.6. Élégance :	61
8.7. Economique :	61
9. Architecture des Servlets	62
10. Le cycle de vie d'une Servlet	62

11.Fonctionnement d'une Servlet	63
11.1.Initialisation de la Servlet	63
11.2.Traitement	64
11.2.1.Lecture des paramètres	64
11.2.2.Les objets requêtes (Request) et réponses (Response):	65
11.2.3.Ecriture du résultat	65
11.3.Destruction de la Servlet	66
Partie III : JDBC (Java Data Base Connectivity)	66
12.Introduction au JDBC	66
13.Qu'est-ce que JDBC ?	67
14.Architecture du JDBC :	67
15.Le pilote et la base de données :	68
16.Fonctionnement de JDBC	70
16.1.Chargement du Pilote :	70
16.2.Définir de l'URL de connexion :	70
16.3.Etablir de la connexion :	71
16.4.Création d'un Statement :	72
16.5.SQL préparé :	73
16.5.1.Instructions préparées :	74
16.6.Traitement des résultats :	75
16.7.Fermeture du connexion :	76
17.Conclusion	76
Partie II : Développement de l'application	78
Chapitre V : Conception et implémentation	79
1.Introduction	79
2.Spécification des besoins	80
2.1. Les Cas d'utilisation	80
2.2 Les Acteurs	80
2.3. Descriptions textuelles des cas d'utilisation	81
2.4. Diagramme de cas d'utilisation	81
2.5. Diagramme de séquence	82
2.6. Diagramme de collaboration	87
2.7. Diagramme d'activité	88
3.La Conception	90
3.2.Conception globale	90
3.2.1.Conception de l'architecture	90
3.2.2.L'architecture Trois-tiers	90
3.3.Conception détaillée	93
3.3.1.Côté Client	93
3.3.2.Côté Serveur	94
3.4.La base de données	96
3.5.L'architecture matérielle	96
4.Implémentation	97
4.2.Environnement de travail	98
4.3.Choix de l'environnement de développement :	98
4.3. Réalisation de l'application	99
4.3.1. Le MIDlet	99
4.3.2. Les Servlets :	101
4.3.3. Les pages JSP :	101

Table de matière

4.3.4.La base de données	104
5.Les tests	104
6.Conclusion	108
<i>Chapitre VI : Conclusion générale</i>	109

Chapitre I. Introduction générale

Le marché des télécommunications sans fil connaît depuis ces dernières années une croissance exponentielle. Les utilisateurs des appareils sans fil sont de plus en plus avides d'informations et de services, et demandent à les obtenir sur leurs appareils itinérants, sans fil. Ainsi, depuis quelques années, fabricants et opérateurs se penchent sur ce marché en pleine expansion.

Notre projet consiste en une recherche et l'implémentation des développements courants de la plateforme «*Java 2 Micro Edition (J2ME) platform*» développée par Sun Microsystems, qui est destiné aux appareils mobiles sans fil ayant des ressources système limitées et une connexion réseau.

Le projet a été proposé au laboratoire LDRSI et est supervisé par Monsieur Mahieddine Mohammed, responsable de recherche au laboratoire.

Peu de travaux abordent avec détail le domaine de la conception et réalisation orientée objets complète de MIDlets (applications sur des terminaux mobiles)[GIG 00],[DEL 02],[WHI 02].

La plupart des publications parues dans la littérature ne traitent que des exemples élémentaires ne prenant en compte la plupart du temps qu'un côté ou facette non achevée et non détaillée du domaine entier.

Notre travail consiste d'une part à étudier:

- les performances des nouvelles méthodologies de conception et modélisations orientées objets (UML, Agile, ..), et de
- la faisabilité de leur application réelle dans les technologies actuelles (terminaux mobiles), et, d'autre part,
- de satisfaire les besoins de la réalisation complète par des tests réels.

C'est donc dans cette double optique que nous nous sommes intéressés au problème du développement d'une MIDlet complète.

Le long de notre recherche nous avons travaillé Agile, c'est à dire de manière réduite et localisée, ou techniquement parlant par itération (petits incréments) pour synthétiser petit à petit les différents sous-domaines de notre étude, nous avons étudié les méthodes et les techniques de:

- mise en oeuvre du cycle de vie d'une MIDlet, de son Display et son Screen [RIG 01],
- l'élaboration de composants simples, telles que les Commands,

- la réalisation d'UI plus élaborées, telles que les Lists, et les Forms, ...
- l'étude et de l'intégration de la partie Servlet [JAS 98] et Conteneur de Servlets (Tomcat en l'occurrence ici),
- l'étude, le traitement, et l'intégration du JDBC et des bases de données.

Dans ce travail, nous avons préféré utiliser des méthodes de tests pratiques réels, qui reposent sur des produits logiciels que nous avons sélectionné parmi ceux présentant certaines qualités.

Les critères qui ont guidé notre choix de logiciels sont:

- la performance,
- la disponibilité (gratuité du produit entre autres),
- l'intégration (c.à.d. la possibilité de collaboration ensembles de manière "naturelle"),
- la simplicité relative de mise en oeuvre.

Dans cette optique, les deux objectifs de notre travail ont été :

1. d'étudier les méthodes de conception et de modélisation orientées objets, ainsi que la technique de programmation du J2ME [KNU 03],
2. de proposer, d'implémenter et de tester une méthode orientée objets comportant toutes les étapes pour la réalisation, tout en tenant compte des difficultés rencontrées lors de la mise en oeuvre sur des cas pratiques.

La première partie de ce mémoire répond à la première de ces préoccupations. Elle va présenter des généralités sur l'ensemble des technologies nécessaires pour réaliser le projet envisagé.

En particulier, nous mettons en évidence l'influence de la modélisation Agile dans le domaine d'apprentissage, et de prototypage rapide.

Dans le chapitre 2, nous rappelons quelques concepts fondamentaux sur les objets et la programmation orientée objets. Nous présentons ensuite les différentes méthodes que nous avons choisi d'utiliser, pour la conception et la modélisation orientées objets, par la suite dans la réalisation pratique du travail. L'accent sera mis surtout sur UML [MUL 97] et la méthode Agile.

Le chapitre 3 est consacré à la présentation de ce que c'est qu'une application sur terminal mobile (ou MIDlet); nous montrons les différences entre la programmation du J2SDK et celle du J2ME, et la

La terminologie ainsi différentes techniques de ce domaine sont alors introduites.

Toutes les étapes de l'élaboration d'une MIDlet, de sa construction à son test, en passant par sa compilation, sa prévérification ainsi que son archivage, ont été abordées

Une MIDlet sans interface utilisateur graphique (GUI) est presque un non-sens. Les techniques de programmation des GUI MIDP sont alors introduites en ce chapitre.

Nous pensons qu'il fallait introduire dans cette partie le plus de détails possibles, tout d'abord parce que le traitement des GUI MIDP est bien différent de celui du Java standard, et en même temps pour qu'on puisse bien comprendre la partie GUI de l'application réalisée.

Le chapitre 4 couvre la technologie Java au application *web*, nous allons étudier les différentes architectures des applications *web*, ainsi une partie JDBC permettant à un programmeur Java de lancer des requêtes SQL de création, de consultation et de mise à jour de bases de données, sans être contraint à passer par les exigences d'un quelconque SGBD.

Le terminal mobile avec ses contraintes, ne pourrait réaliser des tâches de recherche et de stockage d'information qu'en collaborant avec des ordinateurs comportant des périphériques de mémorisation de masse, et à travers un réseau bien sûr.

Dans la dernière partie de ce chapitre on va vous exposer la technologie de Servlet, permettant dans notre cas, à un terminal mobile de communiquer avec un serveur installé sur ordinateur.

Une procédure complète de réalisation est présentée, puis mise en oeuvre dans la deuxième partie de ce mémoire.

Ceci nous permet de mettre en évidence des difficultés techniques qu'on pourrait rencontrer lors de des phases d'implémentation et de tests.

Le dernier chapitre fournit une conclusion de ce travail. Il inclut des remarques sur la démarche suivie pour la réalisation de ce travail, ainsi que des suggestions pour les explorations et recherches ultérieures dans un domaine analogue.

Partie I : Généralités

Chapitre II. Modélisation orientée objets

1. Introduction

L'approche objet consiste à penser à un problème, ou à un système en termes des concepts orientés objets, à savoir : objet et encapsulation, héritage et polymorphisme, message et interaction entre les objets. Il faut penser objet non seulement lors de la programmation mais également lors de l'analyse et de la conception des systèmes informatiques. Ceci nécessite une notation unifiée afin de :

- représenter des concepts abstraits (graphiquement par exemple),
- limiter les ambiguïtés (parler un langage commun, au vocabulaire précis),
- faciliter l'analyse (simplifier la comparaison et l'évaluation de solutions).

Dans la suite de ce chapitre nous allons développer les concepts de la programmation orientée objets.

Nous allons aussi introduire le langage de modélisation UML [MUL 97], et la méthode de modélisation Agile que nous avons utilisé lors du développement de l'application.

2. Les objets:

Un objet est une collection de:

- champs (données membres, variables d'instance, ..) pour maintenir l'état d'un objet et de
- méthodes (traduisant le comportement des objets, ...).

Chaque objet possède sa propre mémoire propre pour maintenir son état (la valeur de ses champs).

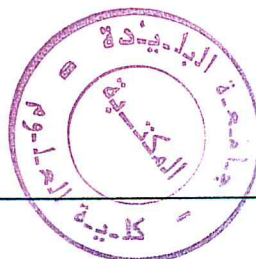
Tous les objets partagent le même code.

3. Objets et classes :

L'étude des formes de communication entre objets du domaine est de première importance dans la modélisation objet et, d'ailleurs, la grande différence entre l'approche fonctionnelle et l'approche objet réside précisément dans cette articulation qui réduit le couplage entre la structure et la fonction.

4. Messages entre les objets :

L'unité de communication entre objets s'appelle le message. Le message est le support d'une relation de communication qui relie, de façon dynamique, les objets qui ont été séparés par le processus de décomposition.



5. Caractéristiques de la programmation orientée objets:

L'abstraction dénote les caractéristiques essentielles d'un objet permettant ainsi de le distinguer des autres sortes d'objets, et fournit donc des frontières conceptuelles relativement à l'observateur.

L'encapsulation est un mécanisme utilisé pour comportementaliser la structure interne avec l'interface d'un objet dans une même structure.

La masquage de données « *data hiding* » est un mécanisme utilisé pour cacher la structure interne et les détails d'implémentation. L'interaction avec l'objet est faite à travers une interface publique.

L'héritage est une méthode de réutilisation à travers laquelle un objet avec une nouvelle fonctionnalité est obtenue par l'extension de l'implémentation d'un objet existant.

Le polymorphisme permet de créer du code s'intéressant à des types d'objets généraux, sans avoir à se préoccuper de connaître préalablement de quel type est l'objet spécifique.

Dans l'utilisation des objets formes géométriques, on voudrait être capable de traiter toute sorte de figure géométrique

6. Bénéfices de la P.O.O.:

Les bénéfices de la programmation orientée objets sont:

- l'efficacité dans la programmation induite par la réutilisation du code,
- la qualité du code (clarté, modularité, couplage faible, ...), induite par l'abstraction et l'encapsulation,
- l'efficacité et l'extensibilité, induites par l'héritage,
- la puissance induite par le polymorphisme

7. Modélisation

On est bien d'accord qu'aujourd'hui, pour construire un système informatique, il est impossible de s'attaquer directement à l'implantation, mais au contraire, il faut d'abord modéliser ce système.

La modélisation consiste tout d'abord à écrire un problème (les besoins à propos d'un système informatique à construire), puis à décrire la solution de ce problème (la conception du système à construire).

Dans la modélisation objet, on crée des modèles :

- modèles de besoins,
- modèles de classes,
- modèles d'interaction entre les objets, etc.

Un modèle d'un système est une abstraction (c'est à dire une simplification) de ce système. La construction des modèles d'un système permet d'identifier les caractéristiques intéressantes du système en vue d'une utilisation précise.

Les modèles construits doivent refléter les caractéristiques essentielles du système, on parle aussi du raffinement dans le développement.

On commence par un modèle très abstrait (c-à-d, très simple), on raffine ce modèle en ajoutant au fur et à mesure des détails observés du système. Le résultat du processus de raffinement sera le système souhaité.

Le caractère abstrait d'un modèle doit notamment permettre de faciliter la compréhension du système étudié. Il réduit la complexité du système étudié, permet de simuler le système, le représente et reproduit ses comportements. Concrètement, un modèle réduit (décompose) la réalité, dans le but de disposer d'éléments de travail exploitables par des moyens mathématiques ou informatiques.

8. Introduction à UML :

UML [MUL 97], qui peut être traduit en français comme le langage de modélisation objet unifié, est né de la fusion des notations proposées et utilisées par une cinquantaine de méthodes objets au milieu des années 90, notamment OMT, Booch et OOSE; il s'agit des trois méthodes qui ont plus influencé. Bien qu'il existe encore différents points de vue à propos de ce langage, mais comme il est issu « du terrain » et le fruit d'un travail d'experts reconnus, UML est le résultat d'un large consensus. De très nombreux acteurs industriels de renom ont adopté UML et participent à son développement :

- des nombreuses recherches autour de l'élaboration/évolution du langage est en cours, ceci se justifie par très grand nombre de conférences internationaux autour UML, Objets, ECOOP, OOPSLA, etc.
- des applications d'UML se présentent partout : les système d'information, les systèmes critiques, les systèmes à base de connaissance, etc.

La modélisation objet, UML a été conçu comme un ensemble de notations qui visent à être utilisées dans différentes étapes de la modélisation objet, de l'expression des besoins jusqu'à l'implantation. Il s'agit des notation synthétique et intuitive et accessible par différents intervenants de la modélisation objets : utilisateurs, analystes, concepteurs, développeurs des systèmes.

Dans la modélisation objet, UML est avant tout :

- un support de communication performant, qui facilite la représentation et la compréhension de solutions objet .
- Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions.
- On peut éviter certaines ambiguïtés et des incompréhensions si on utilise constamment les notations UML.
- Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus.

9. Une vue générale des diagrammes UML

Une autre caractéristique importante d'UML, est qu'il permet de représenter un système selon différentes vues complémentaires par les diagrammes. Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle (système). Construire un modèle objet en notations UML (dans la suite, nous allons appeler modèle UML) consiste à construire quelques diagrammes UML.

UML fournit neuf types de diagrammes :

- **Diagrammes de classes** : ils représentent la structure statique en termes de classes et de relations. Ce sont les diagrammes les plus fréquents dans une modélisation par objets.
- **Diagrammes d'objets** : ils représentent les objets et leurs relations qui sont les instances des éléments qui apparaissent dans les diagrammes de classes.
- **Diagramme de cas d'utilisation** : ils représentent les fonctions du système du point de vue de l'utilisateur.
- **Diagrammes de séquence et diagrammes de collaboration** : ils présentent les objets et leurs interactions. Les diagrammes de séquence mettent l'accent sur le classement chronologique des interactions alors que les diagrammes de collaboration mettent l'accent sur l'organisation structurelle des objets qui s'interagissent.
- **Diagrammes d'état-transition** : ils représentent le comportement des objets d'une classe en terme d'états.
- **Diagrammes d'activités** : ils représentent le comportement d'une opération en termes d'actions.
- **Diagrammes de composants** : ils représentent les composants physiques d'une application.

- **Diagrammes de déploiement** : ils représentent le déploiement des composants sur les dispositifs matériels.

Une caractéristique importante des diagrammes UML, est qu'ils supportent l'abstraction. Cela permet de mieux contrôler la complexité dans l'expression et l'élaboration des solutions objet. UML opte en effet pour l'élaboration des modèles, plutôt que pour une approche qui impose une barrière stricte entre analyse et conception. Les modèles d'analyse et de conception ne diffèrent que par leur niveau de détail, il n'y a pas de différence dans les concepts utilisés. UML n'introduit pas d'éléments de modélisation propres à une activité (analyse, conception...) ; le langage reste le même à tous les niveaux d'abstraction.

Cette approche simplificatrice facilite le passage entre les niveaux d'abstraction. L'élaboration encourage une approche non linéaire, les "retours en arrière" entre niveaux d'abstraction différents sont facilités et la traçabilité entre modèles de niveaux différents est assurée par l'unicité du langage.

UML :

- nous amène à favoriser donc le prototypage, et c'est là une de ses forces. En effet, modéliser une application n'est pas une activité linéaire. Il s'agit d'une tâche très complexe, qui nécessite une approche itérative, car il est plus efficace de construire et valider par étapes, ce qui est difficile à cerner et maîtriser, et
- nous a permis donc non seulement de représenter et de manipuler les concepts objet, il sous-entend une démarche d'analyse qui permet de concevoir une solution objet de manière itérative, grâce aux diagrammes, qui supportent l'abstraction.

10. Pourquoi agile :

Agile a pour but de satisfaire les clients tout en rendant le travail de développement plus facile.

11. Qu'est ce qu'une méthode Agile :

Agile comporte les deux caractéristiques fondamentales suivantes :

- Agile est une méthode adaptative plutôt qu'une méthode prédictive, c'est à dire qu'elle est favorable aux changements, et munie d'une planification souple,
- Orientée vers les personnes plutôt que vers les processus, elle permet de travailler avec les spécificités de chacun, et voudrait aussi de responsabiliser.

12. Le Manifeste :

Nous découvrons la meilleure manière de développer des logiciels en le faisant soi-même, et en aidant les autres à le faire.

Agile privilégie les éléments suivants :

- Les personnes et interactions,
- Application fonctionnelle,
- Collaboration avec le client,
- Acceptation du changement

Sur :

- les procédures et outils,
- la documentation abondante,
- la négociation de contrat,
- la planification.

13. Les valeurs :

Agile donne la priorité :

- aux personnes et aux interactions par rapport aux procédures et aux outils (travail en groupe, communication,..)
- aux applications fonctionnelles, par rapport à une documentation pléthorique (documentation succincte à jour, documentation permanente du code),
- la collaboration avec le client, par rapport à la négociation de contrat (feedback régulier du client, solution répondant réellement aux attentes, grande maturité du client, relation de confiance avec le client,
- l'acceptation du changement par rapport à la planification (planning flexible, modifications possibles après la première version du système.

14. Les principes :

Agile est caractérisé par valeurs déclinées sur principes généraux.

La plus grande priorité est de satisfaire le client en lui livrant très tôt et régulièrement des versions fonctionnelles de l'application source de valeur (le client peut décider de la mise en production de l'application).

Agile permet d'accueillir les demandes de changement de manière naturelle, même tard dans le processus de développement.

Les méthodologies agiles exploitent les changements pour apporter au client un avantage concurrentiel, cela permettrait de produire des systèmes flexibles.

Agile exige de livrer le plus souvent possible des versions opérationnelles de l'application, avec une fréquence comprise entre deux semaines et deux mois, avec une préférence pour l'échelle de temps la plus courte, son objectif est de livrer une application qui satisfasse aux besoins des clients.

Les clients et développeurs doivent coopérer quotidiennement tout au long du projet.

La construction des projets doit se faire autour d'individus motivés. Il faudrait leur donner donc l'environnement et le support dont ils ont besoin, et leur faire confiance pour qu'ils puissent remplir leur mission.

Agile estime que la méthode la plus efficace pour communiquer des informations à une équipe, et à l'intérieur de celle-ci reste la conversation face à face.

Pour Agile, le fonctionnement de l'application est le premier indicateur d'avancement du projet.

Les méthodes agiles recommandent que le projet avance à rythme soutenable, c'est à dire que les développeurs et utilisateurs devraient pouvoir maintenir un rythme constant indéfiniment. L'adaptation du rythme permet de préserver la qualité du travail sur toute la durée du projet.

Agile accorde une attention continue à l'excellence technique et à la conception améliorée. Il faudrait donc maintenir le code source propre, clair et robuste.

La simplicité est essentielle pour Agile. Il faudrait répondre le plus simplement aux besoins actuels pour que le code soit adaptable.

Agile voit que les meilleures architectures, spécifications et conceptions sont le fruits d'équipes qui s'auto-organisent. Le partage des responsabilités se fait par volontariat.

Agile demande à ce que l'ensemble de l'équipe s'interroge, à intervalles de temps réguliers, sur la manière de devenir encore plus efficaces pour ajuster son comportement en conséquence. L'environnement doit être mis en perpétuelle évolution

15. La méthode agile XP (eXtreme Programming) :

Elle a été initiée par Kent Beck en 1996.

Elle est caractérisée par les valeurs de base suivantes :

- La communication. Tout doit être prétexte à communiquer.
- La simplicité. Il faudrait faire simple avant tout.
- La rétroaction (feedback). Il faudrait utiliser les expériences passées et les remarques du client le plus possible.
- Courage. Il faudrait accepter de jeter le fruit de semaines de labeur pour suivre les nouvelles exigences du client.

Les douze principes de XP sont :

- La planification.
- De petites mises à jour.
- Métaphore système.
- Modélisation (Design) simple.
- Tests permanents.
- Factoriser.
- Programmation en binôme.
- Propriété collective du code.
- Intégration continue.
- 40 heures de travail/semaine.
- Client sur le site.
- Standards d'écriture.

XP est adaptée pour :

- Les solutions par petites équipes, et
- les besoins « très » évolutifs.

XP gère :

- le planning ,
- la gestion des besoins changeants,
- l'ingénierie du produit logiciel,

- la coordination inter-groupes,
- la revue par ses pairs,
- la focalisation organisationnelle sur les processus,

16. Conclusion :

La programmation orientée objets est devenue l'approche clé pour développer des logiciels. Cependant son utilisation reste très ardue.

Un certain nombre de méthodes de conception et de modélisation ont vu le jour, mais elles restent à tester sur des projets réels.

Dans notre travail on a expérimenté :

- UML comme langage de modélisation, et
- Agile-XP comme méthode de modélisation.

Agile semblait bien adaptée à notre travail, surtout en ce qui concerne

- la motivation de l'équipe,
- l'environnement non-critique,
- focalisation sur les bons buts,
- - l'intégration/test continu,
- la taille réduite de l'équipe (équipe légère),
- La faible complexité du projet.

Chapitre III. Java 2 Micro Edition et les applications MIDP

Introduction :

Le marché des télécommunications sans fil connaît une exponentielle ces dernières années et constitue un secteur qui maintient son accroissement. En 2003 le nombre d'abonnées aux téléphones portables dépassait la barre du milliard dont 30 à 50% sont connectés à Internet. En parallèle, l'Internet se développe à grands pas et de plus en plus de personnes veulent accéder et obtenir des informations sur un appareil mobile, sans fil pour vivre leur mobilité et liberté. En conséquence, les grands fabricants d'équipements de communication et opérateurs de téléphonie mobile se penchent dans ce domaine dès qu'un standard de développement naît de la volonté de plusieurs fabricants (Nokia, Motorola, Siemens, Palm Computing, NTT, DoCoMo) en l'années 1999 dans le but d'offrir une plateforme java qui soit compatible avec une large gamme de petits appareils connectés aux ressources systèmes très limitées, ce standard (CLDC puis MIDP) a permis :

- de définir les API du modèle d'application,
- de l'interface utilisateur, de la mise en réseau et
- du stockage persistant des données.

Dans ce chapitre :

- Nous allons exposer la technologie *J2ME* et qui est concerne comme architecture.
- En suite nous allons détailler les applications *MIDP* qui sont les *MIDlets*.
- Et la dernière partie est consacrée à la présentation du programmation des gui du MIDP.

Partie I. La technologie J2ME

1. Définition

Sun Microsystems définit J2ME comme une JRE (Java Runtime Environment) spécialement adaptée aux petits appareils, tant grand public qu'intégrées, à capacités en termes de processeur, mémoire, affichage et saisie [ZEN 03].

autre. Comme J2EE et J2SE, J2ME est constitué d'un ensemble d'API défini par un groupe d'experts de constructeurs, développeurs de logiciels et fournisseurs de service.

La plate-forme J2ME permet d'appliquer les avantages et la puissance de la technologie Java sur les appareils embarqués ou mobiles, et elle inclut une interface utilisateur flexible, un modèle de sécurité fort, une sélection de protocoles réseaux intégrée et une gestion des applications locales et distantes.

2. J2ME et les applications embarqués :

Un système embarqué, est un système informatique autonome interagissant en temps réel avec le monde physique de manière généralement invisible. Il comprend un ou plusieurs processeurs accompagnés de logiciels. Ce système fait partie intégrante de la fonction de l'appareillage. Il accepte en entrée des informations provenant de bases de données ou de capteurs et émet en sortie des informations vers une interface utilisateur ou vers un système de commande [DEL 02].

L'utilisation des systèmes embarqués a évolué de plus en plus dans notre vie, de la puce de carte de crédit à l'ordinateur d'aide à la conduite dans une voiture, pour cela on trouve plusieurs types d'applications qui tournent sur les systèmes embarqués en cite :

Application locale : sont des applications qui peuvent tourner sur des appareils sans couverture réseau, par exemple les jeux locaux, les calculatrices les agendas...

Application en réseau peer-to-peer : sont des applications réseau terminal à terminal sans serveur, comme le chat en direct, l'envoi des fichiers multimédias...

Application client-serveur : sont des applications légères dont les clients ont une logique adaptative limitée (navigateur WAP).

Application client-serveur intelligente : sont des applications dont le client a plus de logique adaptative et de traitement local, comme les applications sans fil.

Application distribuée : sont des applications qui tournent sur plusieurs appareils sans serveur comme les jeux multijoueurs, forum de discussions sans serveur.

De même on définit un terminal embarqué comme un appareil basé sur un microprocesseur fait une tâche précise et limitée, comme les téléviseurs numériques, les voitures, les téléphones mobiles, les robots ...

Sun Microsystems cible le marché embarqué par son produit J2ME qui s'adresse particulièrement aux terminaux embarqués qui spécifient des fonctionnalités limitées et qui sont dotés de processeurs 32 bits et d'une connectivité réseau.

L'utilisation de ce produit pour le développement d'applications embarqué donne des avantages est la mise en disposition d'outils faciles à utiliser comme un émulateur qui garantit un cycle rapide d'édition, de construction et de test, il reproduit les même performances qu'un vrai terminal afin de développer des applications adaptées aux performance réelles du terminal et non pas à celle de l'ordinateur sur lequel se tourne.

De coté utilisateur l'utilisation de J2ME fourni un haut niveau d'interactivité et une présentation beaucoup plus riche contrairement au plate-forme client léger qu'utilise le WML ou CHTML.



Figure III-1: Diversité des appareils embarqués

3. Particularités des applications destinées aux appareils mobiles :

Les applications destinées à s'exécuter sur des appareils mobiles nécessitent de nombreux pré requis pour s'exécuter correctement.

- **Ressources :**

Actuellement, les ressources proposées par les appareils fixes sont suffisamment généreuses pour ne poser aucun problème aux applications. Il n'en est pas de même pour les appareils embarqués, où les ressources doivent être gérées avec parcimonie.

- **Accès au réseau :**

Dans le cas des réseaux sans fil, l'accès au réseau n'est pas forcément garanti. Un tunnel, une position géographique défavorable, des conditions météorologiques particulières, et les transmissions peuvent être réduites ou tout simplement impossibles. L'application doit

donc être conçue pour pallier ces inconvénients. Elle doit être suffisamment intelligente pour permettre la re-transmission des données si le réseau est indisponible, pour permettre le stockage des informations entre temps ou l'annulation de leur transmission si les informations non traitées en temps utiles sont obsolètes.

- **Communications :**

Dans le cadre des communications sans fil, tout le monde partage la même largeur de bande, et même si celle-ci est importante, elle reste à prendre en considération.

Enfin, il faut prendre en compte les débits. Ils changent en effet en fonction du protocole utilisé, mais aussi à cause de l'effet Doppler. Par exemple, une voiture roulant à 120km/h peut perdre le contact avec le réseau GSM si sa direction de déplacement et la direction de propagation sont colinéaires. De manière similaire, le débit utile de l'UMTS baisse de manière significative en fonction d'un déplacement, même lent (homme au pas). Ces données sont donc à prendre en compte si on désire utiliser une application mobile durant des déplacements rapides.

- **Interface utilisateur :**

Les applications portables s'exécutent sur des appareils aux dimensions réduites, aussi bien pour les unités d'entrée que de sortie. Le programmeur doit donc prendre en compte ces aspects.

L'affichage est généralement limité à quelques centimètres d'écran LCD TFT, avec une résolution jusqu'à 640*480 (VGA) pour 5 pouces d'écran. Les conditions de lecture sont souvent défavorables, notamment à cause de la luminosité.

La saisie des champs sur les appareils portables est laborieuse, en raison de l'absence de clavier. Il faut réduire au minimum ces entrées et proposer des alternatives, comme des cases à cocher, des listes cliquables.

4. Exemples d'applications

A l'utilisation de J2ME on peut créer différentes sortes d'application pour les téléphones mobiles :

- **Divertissement:** jeux, animaux virtuels, animation, compositeur de musique.
- **Vie courant:** Listes de courses, météo, outils pour la comptabilité personnelle, calendrier....
- **Voyage:** Plan du métro, horaire de trains, convertissement de devises, dictionnaire, horloge ...
- **Information:** Informations financières, moteur de recherche optimisés.

- **Voyage:** Plan du métro, horaire de trains, convertissement de devises, dictionnaire, horloge ...
- **Information:** Informations financières, moteur de recherche optimises.

Ainsi, n'importe quel possédant de terminale sans fils supportant cette technologie peut télécharger et tourner des applications au gré de ses envies et de ses besoins.

5. L'architecture de j2me :

Les terminaux n'ayant pas les même capacités en terme de ressources que les ordinateur de bureau (mémoire, disque et puissance), la solution passe par la fourniture d'un environnement allégé a fin de s'adapter aux différentes contraintes d'exécution. Cependant, comment faire en sorte d'intégrer la diversité de l'offre à un socle technique dont la cible n'est pas définie à priori ? [PIR 02].

Donc une seule plate forme J2ME ne peut couvrir tous les besoins.

La solution proposée par J2ME consiste à regrouper par catégories certaines familles de produits tout en proposant la possibilité d'implémenter des routines spécifiques pour un terminal donné, l'architecture de j2me définit des *Configurations*, des *Profils* et des *Packages Facultatifs*.

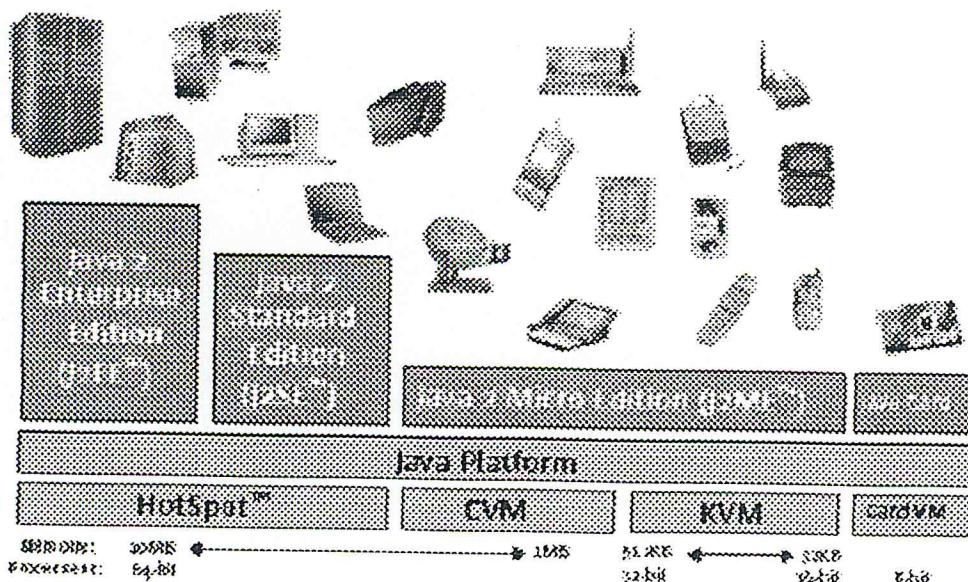
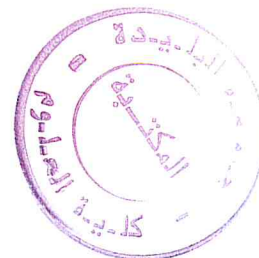


Figure III-2: La plate-forme Java 2.



5.1. Les Configurations :

Une configuration définit une machine virtuelle et un ensemble de bibliothèques pour un ensemble de terminaux qui possèdent des caractéristiques similaires (tailles mémoire, capacité du processeur, etc..).

5.1.1. CDC: (Connected Device Configuration).

La CDC fournit une machine virtuelle et des bibliothèques de classe de base aux applications java adresses aux appareils tels que les PDA avancés, les TV numérique interactives etc... Typiquement, ces dispositifs sont basés sur un processeur 32 bits, plus de 2 Mo de mémoire disponible et une connexion réseaux obligatoire (sans fil ou pas). La CDC contient la machine CVM (compact virtuel machine) qui est une machine virtuelle complète conçue pour des dispositifs ayant besoins de l'ensemble des fonctionnalités et services de Java 2, mais avec une taille réduite, cette configuration s'inscrit donc dans le cadre d'une architecture Java presque complète.

5.1.2. CLDC: (Connected Limited Device Configuration)

la CLCD est la plus petite des deux configurations, elle adresse les aux terminaux légers tels que les téléphones mobiles ou les assistants personnels ...

Ces périphériques sont limités en termes de ressources (processeur lent, une capacité mémoire limite aussi que des connexions réseau sans fil intermittentes). Ces dispositifs ont typiquement des processeurs 16 ou 32 bits et de 128 à 512 ko de mémoire disponible pour plate-forme Java et ses applications associées.

La CLDC concerne le langage java et les fonctionnalités d'une machine virtuelle même les bibliothèques noyau de java (java.lang.* , java.util.*), les entrées-sorties, le réseau, la sécurité et l'internationalisation. Par contre la CLDC n'intègre pas la gestion de la dures de vie de l'application, l'interface utilisateur, la gestion des évènements, les particularités de chaque terminal ou l'interaction entre l'utilisateur et l'application.

Cette configuration s'inscrit donc dans une logique d'économie de ressources avec une KVM(K Virtual Machine) de 40 à 80 ko s'exécute 30 à 80% moins vite qu'une JVM normale[ZEN 03].

Le but principal de cette machine virtuelle est d'être conforme à la spécification du langage Java et à la spécification de la Java VM tout en respectant les contraintes de mémoire, excepté ceci :

- pas de types de données à virgule flottante (float et double).

- pas de JNI™(Java Native Interface™).
- les chargeurs de classe ne peuvent pas être définis par le développeur.
- pas de groupes de Threads ni de Threads démons.
- pas de finalisation des instances de classes; la méthode `Object.finalize()` n'existe pas.
- pas d'optimisation de la mémoire avec les références faibles (weak references).
- la gestion des erreurs est limitée.

5.1.2.1 L'API CLDC 1.0 :

CLDC étant un environnement allégé de java, les bibliothèques qu'il fournit sont donc réduites mais néanmoins suffisantes pour le développement sur les terminaux cibles.

5.1.2.1.1 java.io

Le package `java.io` fournit la gestion des flux d'entrées/sorties contenant donc les classes et les méthodes nécessaires pour récupérer des informations des systèmes distants.

5.1.2.1.2 java.lang

Le package `java.lang` est un sous-ensemble des classes standards du package `java.lang` de J2SE telle que l'interface `Runnable` ou les classes `Boolean`, `Integer`, `Math`, `String`, `Thread`, `Runtime` et d'autres (15 au total).

Un absent de marque, toutefois : la classe `Float`. En effet, MIDP ne supporte pas les calculs en virgule flottante ! Vous devrez donc faire sans ou les émuler.

5.1.2.1.3 java.util

Le package `java.util` contient un petit sous-ensemble du package correspondant de J2SE, don voici les classes retenues : `Calendar`, `Date`, `TimeZone`, `Enumeration`, `Vector`, `Stack`, `Hashtable` et `Random`.

5.1.2.1.4 javax.microedition.io

Cette bibliothèque contient les classes permettant de se connecter via TCP/IP ou UDP. Le principal objet du package `javax.microedition.io` est la classe `Connector`. Cette classe fournit un moyen uniforme et pratique d'effectuer des entrées/sortie quelque soit le type de protocole. Il est possible d'ouvrir une connexion sur le système de gestion de fichier du terminal ou sur une page web via HTTP. Il est également possible d'utiliser des socket ou de contrôler l'éventuel port série du terminal.

caractéristiques communes telles que la gestion de l'affichage, des événements d'entrées/sorties (pointage, clavier, ...) ou des mécanismes de persistance (Base de données légères intégrées). L'objectif de profil est de donner plus de flexibilité pour maintenir la portabilité des applications entre les terminaux.

Sun propose deux profils de référence J2ME :

- Le profil **Fondation** qui est destiné à la configuration CDC est offert par conséquent toute la richesse d'une machine virtuelle presque identique à la machine virtuelle standard, cela signifie que les développeurs utilisant ce profil auront accès à une implémentation complète des fonctionnalités de J2SE.
- Le profil **MIDP** [PIR 02] destiné à la configuration CLDC, fournit la plateforme Java destinée aux appareils mobiles actuels. Il prend en charge un sous-ensemble limité de la bibliothèque de classes J2SE et définit des classes d'entrées/sorties et d'interface utilisateur personnalisées pour une utilisation sur une configuration CLDC.

Voici d'autres exemples de profils proposés ou spécifiés par JCP (Java Community Process).

Profil	Configuration a destiné	Domaine ciblé
PDA	CLDC	Les agendas électroniques du type <i>Palm Pilot</i> .
PersonalJava	CDC	Décodeurs numériques intégrant les fonctionnalités de courrier électronique et de navigation Web.
PersonalJava Basic	CDC	Domaines de l'automobile et de la télématique.
RMI	CDC	Applications distribuées.
Game	CDC	Consoles de jeu fixes ou mobiles.
J2ME/CLDC pour Jsamp	CLDC	Droid et Robot MindStorm.
STIP	CDC	Les terminaux de paiement.
...		

Tableau III-1 : Les différents Profils.

5.3. La machine virtuelle :

En fonction de la cible, la machine virtuelle pourra être allégée afin de consommer plus ou moins de ressources. On trouve de type KVM et CVM.

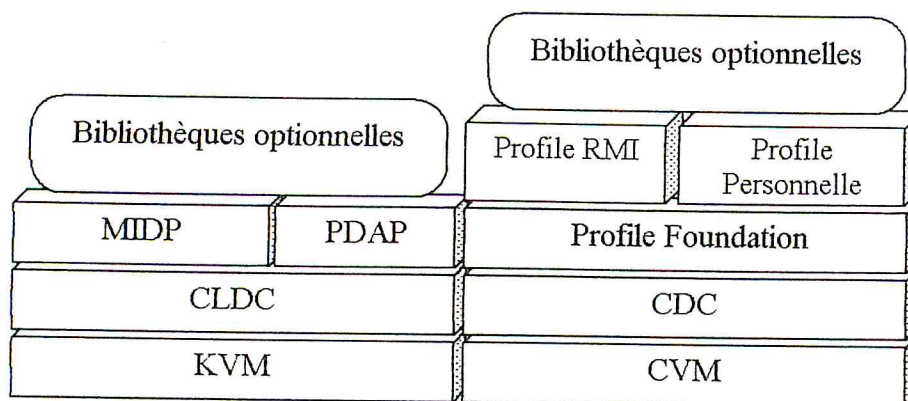


Figure III-3: Architecture de Java 2 Micro Edition (J2ME).

5.3.1 Compact Virtual Machine:

C'est une machine virtuelle Java2 conçue pour les terminaux ayant besoin de l'ensemble des fonctionnalités de JVM mais avec des capacités plus réduites. CVM est conçue pour répondre aux besoins du marché émergent des terminaux embarqués de nouvelle génération. Les terminaux utilisant CVM sont généralement des terminaux compacts et connectés, orientés consommateur [DEL 02].

5.3.2. Kilo Virtual Machine :

La KVM est une machine virtuelle Java portable et compacte prévue pour des dispositifs avec des ressources limitées tels que les téléphones portables, organisateurs personnels, distributeurs automatiques, etc.. L'objectif principal d'une telle machine virtuelle est d'être la plus petite possible (40 à 80 ko) tout en préservant l'aspect du langage de programmation Java. La KVM n'est pas implémentée que par Sun Microsystems et a déjà été portée sur plusieurs dizaines de dispositifs par des constructeurs agréés par Sun.

Pour les dispositifs ne possédant pas d'interface utilisateur capable de lancer des applications, il existe un gestionnaire d'applications Java (JAM Java Application Manager) qui joue le rôle d'interface entre le system d'exploitation hôte et la machine virtuelle [HAS 03].

5.4. Package optionnelle :

Il est important de remarquer sur la figure 3 qu'un dessus de profils se trouvent de bibliothèques optionnelles que chaque constructeur est libre d'ajouter à son produit. Parfois, les fonctionnalités ajoutées sont propriétaires et ne s'appliquent que pour les produits d'un fabricant. L'exemple type concerne les bibliothèques pour les jeux. Afin d'exploiter mieux les particularités matérielles des appareils, les constructeurs proposent leurs propres API dédiée aux jeux (par exemple : API Game de Siemens). Il faut donc être conscient que l'utilisation de telles APIs limitent la portabilité.

6. MIDP : Mobile Information Device Profile

Le MIDP est composé d'un groupe d'API Java qui, avec la configuration CLDC, forment un environnement d'exécution globale des informations pour des applications J2ME et fournissent à des coûts avantageux des informations sur les appareils mobiles. Le MIDP permet aux développeurs de créer des applications sur une vaste gamme de mobiles, tout en respectant les possibilités et les limites de chacun d'eux [HAS 03].

Aujourd'hui, il existe deux versions de MIDP. La nouvelle version 2.0 répond mieux aux appareils actuels et permet de mieux les exploiter. Des nombreux constructeurs de la téléphonie mobile se sont associés pour développer ensemble ce standard, qui permet d'améliorer les technologies Java sur les téléphones mobiles. Le standard MIDP 2.0 support les nouvelles et toutes dernières application de jeux, graphique, applications vidéo, vocales et sécuritaires, de même que bon nombre d'autres fonctions liées aux téléphones mobiles et aux organisateurs PDA.

Le MIDP 1.0 contient les trois bibliothèques suivantes :

- `Javax.microedition.lcdui` : Cette librairie fournit toutes les classes nécessaires à la gestion de l'interface utilisateur du terminal.
- `Java.microedition.midlet` : cette librairie s'occupe de la gestion du cycle de vie des *MIDlets*.
- `Java.microedition.rms` : contient toutes les classes permettent de gérer une base de données légère sur le terminal.

Le MIDP v2 offre la même chose que l'édition précédente avec quelques améliorations qui concernent principalement l'affichage. Les nouveaux packages qui apporté par le MIDP v2.0 sont :

- `Javax.microedition.lcdui.game` qui permet de gérer bien l'interface utilisateur pour construire des bons jeux.
- `Javax.microedition.media` contient des classes permettant d'utiliser facilement et de contrôler des fichiers multimédias (son, vidéo).
- `Javax.microedition.pki` cette bibliothèque destinée à la sécurité réseaux.

Dans la pratique, cet apport de nouvelles bibliothèques peut se traduire par les fonctionnalités suivantes :

- Support jeux : Le MIDP 2.0 ajoute un API jeux standard pour les jeux complexes, utilisant ainsi le potentiel graphique des appareils pour simplifier le développement et mieux contrôler les fonctions graphiques et la performance.
- Support média : Avec le MIDP 2.0, les développeurs peuvent exploiter toutes les fonctionnalités audio des appareils modernes. Les applications MIDP utilisant une plateforme standard sont capables d'intégrer les séquences de tonalités et les fichiers WAV.
- Connectivité élargie : Le standard MIDP 2.0 est compatible avec les principaux standards de connectivité, élargis au-delà de l'HTTP à l'HTTPS, au datagrammes, aux sockets et sockets serveurs, ainsi qu'à la communication par interface série, proposant ainsi différents moyens d'échanger des informations.
- Sécurité de bout en bout : Pour protéger le réseau, les applications et les mobiles, le standard MIDP 2.0 est assorti d'une sécurité de bout en bout, robuste et basée sur des standards ouverts. Ce modèle de sécurité supporte l'HTTPS et s'appuie sur des standards généraux de type SSL et WTLS pour transmettre des informations cryptées.
- Méthode OTA (Over The Air) : La méthode OTA fait désormais partie du standard MIDP. Elle permet de reconnaître, d'installer, d'actualiser et d'éliminer des suites *MIDlet* sur les mobiles. Un fournisseur de services peut donc identifier le type de suites *MIDlet* qui fonctionnent sur un appareil spécifique, et vérifier ensuite l'état de l'installation, de l'actualisation ou de l'élimination des suites *MIDlet* sur ce même appareil.
- Architecture *Push* : Le MIDP 2.0 intègre un modèle d'architecture serveur *push*, qui permet d'enregistrer les *MIDlets*, et de les activer dès qu'un appareil reçoit une information du serveur. Ainsi, on pourra consulter en temps réel sur un mobile les actualités, la bourse et les enchères en ligne.

II. Les applications MIDP « les *MIDlets* »

1. Définition d'une *MIDlet* :

- Les *MIDlets*TM sont des applications sur des mobiles à base de Java, avec une convention de nom similaire à celle de *Applets* et *Servlets*[KNU 03].
- Tout ce dont on a besoin de faire (ou d'apprendre) est :
 - Certains concepts de base (c.à.d. durée de vie d'une *MIDlet*, le packaging ou la compression des *MIDlets*, des suites de *Midlets*, ...)
 - Le cycle de compilation et d'exécution de base
 - Les outils de développement standards et spécifiques au vendeur et les émulateurs
 - Les grandes différences entre les classes et les interfaces du J2SE et ceux du MIDP
 - Les APIs des *MIDlets* et leur utilisation
 - L'intégration des *MIDlets* avec des applications côté serveur (c.à.d. *servlets*)

2. Création d'une *MIDlet* :

La création d'une *MIDlet* suit un cycle dont les étapes sont l'écriture du *MIDlet*, sa compilation, sa prévérification et le test du *MIDlet*, ensuite le packaging en vue de sa distribution puis le test du package complet ainsi réalisé

2.1. Ecriture de l'application :

La première étape du cycle de développement d'un *MIDlet* est l'écriture du code.

La *MIDlet* doit respecter un certain nombre de règles.

- La *MIDlet* doit importer le package *midlet* et *lcdui*. Le package *midlet* définit le MIDP et le package *lcdui* fourni les APIs de l'interface utilisateur.
- Une *MIDlet* ne doit pas avoir de méthode public static void main() .
- Chaque *MIDlet* doit étendre la classe *MIDlet*. Comme pour les *Applets* cela permet de démarrer, de suspendre et de terminer une *MIDlet* [DEL 02].

2.2. Le cycle de vie des *MIDlets*

Toutes les applications pour le profile MIDP doivent être dérivées d'une classe spéciale, *MIDlet*. La classe *MIDlet* gère le cycle de vie de l'application. Elle est localisée dans le package `javax.microedition.midlet`.

Les *MIDlets* sont représentées par des instances de la classe `javax.microedition.midlet`.

Chaque *MIDlet* possède son propre cycle de vie, qui s'exprime dans les méthodes et le comportement de la classe *MIDlet*.

Chaque terminal de *MIDlet* comporte un gestionnaire d'application un logiciel qui contrôle l'installation et l'exécution des *MIDlets*

Le cycle de vie général des *MIDlets* est représenté dans la figure 4, et qui pourrait être décrit par un diagramme d'états et de transitions :

- Une *MIDlet* peut se trouver en un des états suivants: loaded, active, paused, et destroyed.
- Quand une *MIDlet* est chargée dans le terminal et est lancée (en invoquant la classe de la *MIDlet*), un objet instance est créé. Tout d'abord le constructeur est exécuté, après cela, la *MIDlet* va se trouver dans un état « *paused* ». Cela pourrait arriver en tout moment avant que le gestionnaire de programme ne démarre l'application en appelant la méthode `startApp()`
- Une fois que le gestionnaire d'application appelle `startApp()`, la *MIDlet* va rentrer dans l'état active.
- Dans l'état actif, le gestionnaire d'application pourrait suspendre la *MIDlet* en invoquant `pauseApp()` ou `destroyApp()`. `pauseApp()` va changer l'état de la *MIDlet* à *paused*, et `destroyApp()` termine la *MIDlet*. D'autre part la *MIDlet* pourrait aussi se mettre soit-même à l'état *paused* en appelant `notifyPaused()`.
- Finalement, le gestionnaire d'application pourrait terminer l'exécution de la *MIDlet* en appelant `destroyApp()`, de telle façon que la *MIDlet* va se trouver à l'état *destroyed* et sera donc prête pour le processus du récupérateur de mémoire. De manière similaire la *MIDlet* pourrait aussi se détruire soit-même en appelant `notifyDestroyed()`.

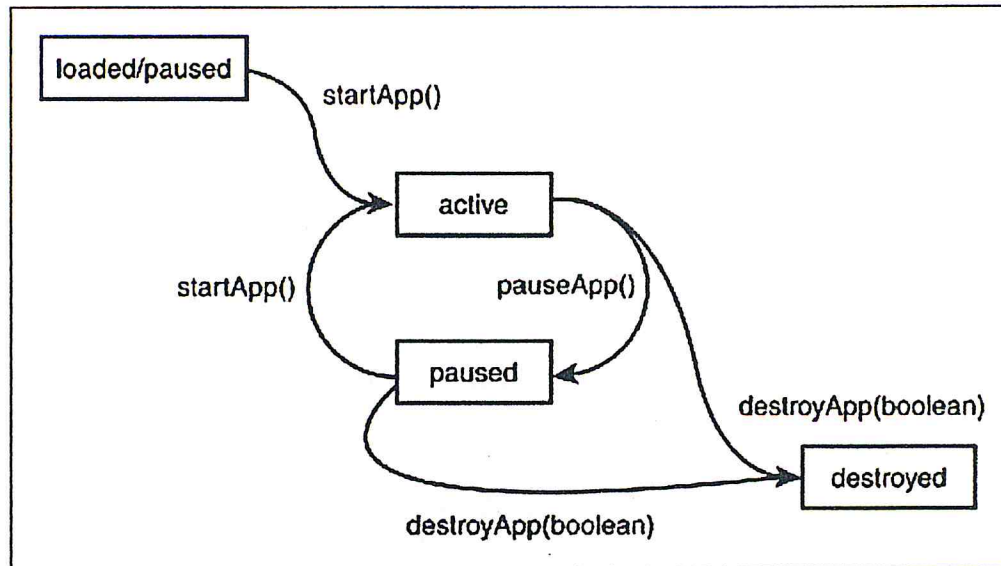


Figure III-4: Cycle de vie d'une MIDlet sur un terminal

2.3. Communication avec le gestionnaire d'application :

Le gestionnaire d'application (Application Manager) est un logiciel tournant sur le terminal mobile. Il est dépendant du terminal et est implémenté par le fabricant du terminal lui-même. Ce gestionnaire gère l'installation, l'exécution et le retrait des *MIDlets* sur le terminal. Il fournit également un mécanisme de gestion des erreurs. Il peut fournir, en option, une interface utilisateur permettant aux utilisateurs de démarrer, d'arrêter et de terminer des *MIDlets* [DEL 02].

Pour être efficace, la communication doit être bien évidemment bidirectionnelle et les applications MIDP ne font pas exception à la règle. La classe *MIDlet* fournit un ensemble de méthodes finales permettant de communiquer avec le gestionnaire d'applications :

- **notifyDestroyed()** indique au gestionnaire que l'application demande sa fermeture. L'appel de cette méthode n'entraînera pas l'exécution de la méthode *destroyApp()*, il faudra donc l'appeler manuellement.
- **notifyPaused()** notifie le gestionnaire que l'application demande sa mise en pause.
- **resumeRequest()** demande au gestionnaire d'effectuer la reprise de l'application si celle-ci a été mise en pause.
- **getAppProperty()** extrait des informations de configuration concernant une application, à partir d'un fichier « manifest » ou d'un fichier descripteur de l'application. Pour simplifier, on peut dire que cette méthode accède à un fichier d'initialisation privé [HAS 03].

2.3. Compilation et prévérification d'une *MIDlet*:

La compilation d'une classe MIDP est faite de la même manière que pour tout autre classe java `C:\>javac « chemin de Fichier de la MIDlet »`.

Comme déjà expliqué, un mobile a toujours une mémoire réduite, spécialement les terminaux comme les téléphones et les pagers. Pour résoudre cette question, CLDC (y compris MIDP) spécifie que la vérification du code pour toutes les *MIDlets* est faite en deux étapes. La première est faite pendant le développement de la *MIDlet* (ainsi appelée prévérification) et le reste est fait sur le terminal mobile.

En particulier, le terminal mobile nécessite de faire une deuxième vérification légère.

Il faut maintenant le traiter avec la commande prévérification. Ce traitement permet d'utiliser cette *MIDlet* sur la machine virtuelle KVM. Il faut fournir à la commande prévérification le nom de la classe et le répertoire de sortie si aucun répertoire n'est spécifié, le répertoire output est utilisé par défaut.

Voici donc comment pré vérification une *MIDlet* :

```
C:>preverify -classpath « chemin de répertoire de sortie » « Nom de la classe »
```

Si vous essayez de lancer une application dans la machine virtuelle KVM sans avoir lancé la prévérification, vous obtiendriez ce message d'erreur :

ALERT : Error verifying the class XXX.

2.4. Test de l'application :

Pour tester une application, il vous suffit d'utiliser l'exécutable midp. Aussi il suffit de lancer : `C:>midp « nom du MIDlet »`.

Cela suffit si vous n'avez qu'une seule *MIDlet* à lancer. Par contre si vous en avez plusieurs vous devrez packager votre application.

2.5. Empaquetage de l'application :

Si votre application comprend plusieurs classes, utilisez un fichier jar (Java archive) pour les regrouper. Cela facilitera la distribution et le déplacement.

On peut faire ça comme suite :

```
C:> jar cf «nom du package».jar «nom du classe à empaqueter».class
```

Nous allons maintenant créer un fichier descripteur d'application. Son extension doit être jad (Java Application Descriptor). Ce fichier fournit des informations sur le contenu du fichier

JAR. Il est utilisé par le gestionnaire d'application et par la *MIDlet* lui-même. Les attributs commençant par *MIDlet-* sont passés au *MIDlet* sur demande.

Voici un exemple de fichier descripteur :

```
MIDlet-1: Scolarité, Scolarité.png, Scolarité
MIDlet-Jar-Size: 100
MIDlet-Jar-URL: Scolarité.jar
MIDlet-Name: Scolarité
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
Servlet1-URL: http://localhost/redapp/servlet/InscriptionServeur
Servlet2-URL: http://localhost/redapp/servlet/NoteServeur
Servlet3-URL: http://localhost/redapp/servlet/MessageServeur
```

L'attribut *MIDlet-1* a trois paramètres : nom, icône et classe.

Le nom est affiché dans la liste des applications par le terminal. L'éventuel icône figure après le nom. La classe est le nom de l'application appelée pour lancer la *MIDlet*.

Le nom d'attribut *MIDlet-1* indique qu'il s'agit du premier *MIDlet*. Si d'autres *MIDlets* sont utilisés, ils seront désignés *MIDlet-2*, *MIDlet-3* et ainsi de suite.

Comme on peut définir des propriétés personnelles pour l'utiliser dans la *MIDlet* et faciliter de les modifier sans toucher le code du *MIDlet*, comme dans notre cas les liens des servlets.

2.6. Test de l'application empaquetée :

Pour tester une application packagée, utilisez l'exécutable *midp* avec l'option *-descriptor* pour lancer *ScolaritéMidlet* dans l'émulateur :

```
C:> midp -descriptor Scolarité.jad
```

Vous pouvez maintenant déployer la *MIDlet* sur un serveur Web. Pour cela, stockez les fichiers JAR et JAD sur le serveur. Ajoutez maintenant le nouveau type MIME dans le fichier de configuration des types mime. Typiquement, cette ligne ressemble à ceci :

```
text/vnd.sun.j2me.app-descriptor jad
```

Puis relancez le serveur Web pour prendre en compte les modifications. La *MIDlet* est alors téléchargeable.

Pour télécharger la *MIDlet* et le lancer depuis le serveur Web, utilisez la ligne de commande:

```
c:> midp -transient http://localhost/path/first.jad
```

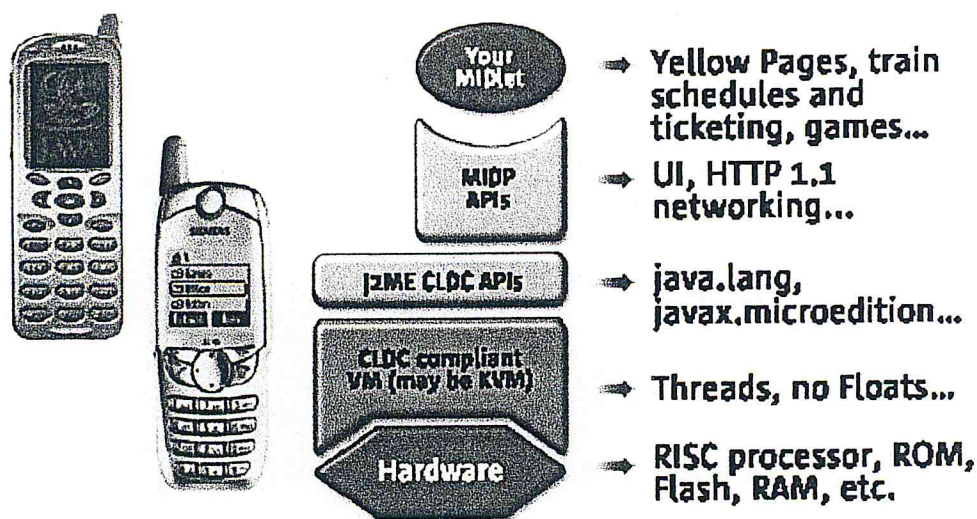


Figure III-5: Architecture de développement (couche logiciel)

3. Distribution d'une *MIDlet* :

Le fait de placer les *MIDlets* dans un fichier archive .Jar c'est le moyen le plus pratique de distribuer des applications J2ME-MIDP. On appelle ce regroupement de *MIDlet* une midlet-suit [DEL 02].

Une midlet-suit comprend donc le fichier .Jad, ainsi que les fichiers de classe java et les ressources associées comme les images, tous les contenus dans le fichier .jar.

Le fichier jar :

Un certain nombre de fichiers sont regroupés dans un fichier décrivant le contenu du fichier .Jar lui-même. Ce dernier fichier a le nom manifest.inf.

Le fichier jad :

Il est fortement conseillé d'inclure un fichier jad (java application descriptor) dans le fichier jar et ce pour deux raisons.

- Les informations fournies par un fichier permettent au gestionnaire d'application de déterminer si la *MIDlet* est bien adaptée au terminal. Par exemple, la fonction de l'attribut MIDlet data-size, le gestionnaire d'application peut détecter un manque de mémoire persistant sur le terminal.
- Le fichier .jad permet de modifier les paramètres fournis au MIDlet sans avoir à modifier le fichier .jar.

4. Exécution de MIDlet :

On peut lancer une *MIDlet* soit via le système de fichier, soit depuis un serveur *web*.

4.1. Exécution du MIDlet via le système de fichier :

Il suffit de saisir la ligne de command suivant :

```
C:\>midp -transient file://MIDlets.jad
```

4.2. Exécution du *MIDlet* depuis un serveur web :

Après avoir uploadé les fichiers *jar* et *jad* vous pouvez lancer la *MIDlet* en spécifiant l'URL :

```
C:\>midp -transient http://localhost/MIDlets.jad
```

5. Déploiement d'une *MIDlet* sur un terminal réel :

Lorsque notre application est écrite puis simulée grâce à un utilitaire adapté, il est temps de l'exécuter sur un téléphone portable, par exemple. Plusieurs manières de charger le code dans le dispositif sont envisageables.

La première qui ne coûte rien est d'utiliser les possibilités de connexion tels que l'infrarouge ou le bluetooth pour autant de disposer d'un utilitaire adéquat. Cette méthode peut être pratique mais elle est très restreinte par le fait qu'elle nécessite la proximité du serveur d'applications.

L'autre manière, qui est la plus utilisée et qui repousse toutes les limites géographiques, est de s'appuyer sur une connexion sans fil du type GPRS. Couplée à un protocole de communication. Afin que les fichiers téléchargés puissent être correctement transférés et interprétés par le terminal, il faut faire quelques manipulations du côté serveur.

En effet, il faut définir le type MIME (Multipurpose Internet Mail Extensions) pour les fichiers *.jad* et *.jar* sur le répertoire dédié à ces fichiers. A titre de rappel, le type MIME permet au destinataire de savoir de quelle manière interpréter l'information reçue.

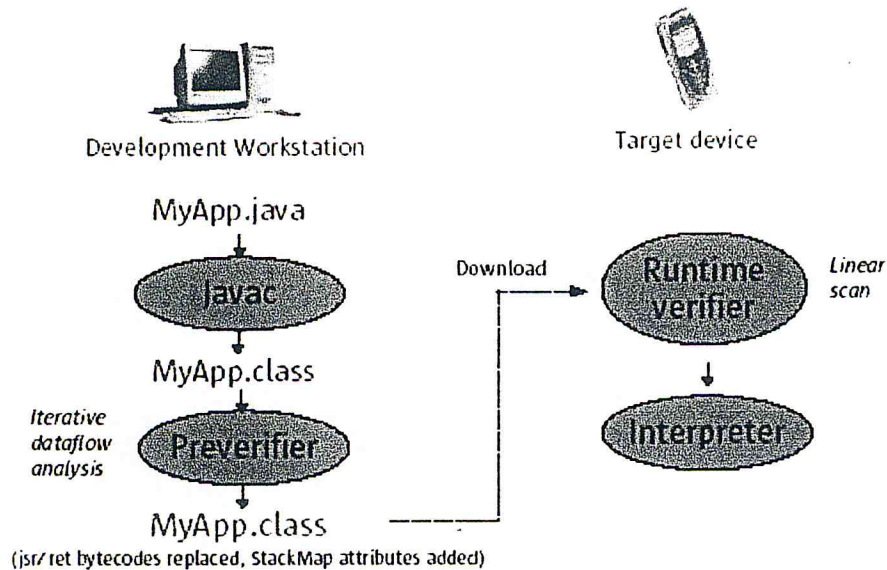


Figure III-6: Cycle de développement d'une MIDlet

6. Programmation des interfaces utilisateur avec MIDP

Les besoins de l'interface utilisateur pour les terminaux mobiles sont différents de ceux des ordinateurs. Par exemple, la taille des terminaux est plus petite, et l'entrée des terminaux ne comporte pas des outils de pointage tels que la souris. Pour ces raisons, on ne pourrait pas suivre les mêmes directives de programmation des interfaces utilisateur que ceux qu'on utilise pour les ordinateurs.

Le CLDC lui ne définit aucune fonctionnalité des GUI. Les classes GUI sont plutôt incluses dans des profiles tels que le MIDP. Dans ce chapitre nous allons commencer par montrer pourquoi les classes GUI incluses dans le MIDP ne sont pas basées sur l'Abstract Window Toolkit (AWT). Nous aborderons ensuite les différents écrans correspondants aux fenêtres de l'écran du AWT. Nous introduirons aussi tous les composants UI. Nous dirons quelques mots concernant la gestion des événements. Nous terminons ce chapitre par un exemple simple d'implémentation.

6.1. Pourquoi ne pas réutiliser le AWT?

Il a été décidé de ne pas prendre un sous-ensemble ni du AWT, ni du Swing pour les raisons suivantes:

- AWT est modélisé pour des ordinateurs et est optimisé pour ces machines.

- AWT suppose certains modèles d'interaction de l'utilisateur. L'ensemble des composants du AWT est modélisé des périphériques de pointage comme la souris, alors que les terminaux mobiles ne possèdent qu'un petit clavier de touches comme entrée utilisateurs.
- AWT possède un ensemble riche de caractéristiques riche, dont la plupart ne correspondent pas à ce qui est demandé par des terminaux mobiles. Par exemple la gestion des fenêtres, tel que la mise à la taille, et l'« overlapping » des fenêtres, le gestionnaire de positionnement, etc....
- Dans l'interaction de l'utilisateur dans une application basée sur le AWT, les objets événements sont créés dynamiquement, et ils continueraient à exister jusqu'à leur prise en compte dans un traitement utilisateur, avant qu'ils soient soumis au récupérateur. La mémoire limitée du terminal mobile ne pourrait permettre cela.

6.2. Les APIs MIDP du GUI

A cause de ce qui a été dit précédemment le MIDP contient ses propres GUI (Graphical User Interface), qui sont assez différents de ceux du AWT. Les GUI du MIDP consistent en:

- APIs de haut niveau, et
- APIs de bas niveaux.

Chaque classe possède son propre ensemble d'événements. L'API de haut niveau est modélisée pour des applications où la portabilité entre les terminaux mobiles est importante. Pour aboutir à cette portabilité, les APIs emploient des abstractions de haut niveau et ne permettent qu'un faible contrôle sur la présentation. Par exemple, on ne peut pas définir l'apparence visuelle (forme, couleur, et font) des composants de haut niveau. La plupart des interactions sont encapsulées par les implémentations, et ne sont pas touchés au niveau de l'application. Par conséquent, c'est l'implémentation qui fait l'adaptation nécessaire aux différents matériels et styles d'interface utilisateur. Les classes qui implémentent l'API de haut niveau héritent toutes de la classe `javax.microedition.lcdui.Screen`.

L'API de bas niveau fournit une abstraction légère. Elle est modélisée pour des applications qui ont besoin du placement et du contrôle précis des éléments graphiques, aussi bien que l'accès aux événements d'entrée de bas niveau. Cette API donne à l'application le

plein contrôle sur ce qui est dessiné à l'écran. L'API de bas niveau est implémentée par les classes `javax.microedition.lcdui.Canvas` et `javax.microedition.lcdui.Graphics`

6.3. Le modèle MIDP des GUI

Nous allons vous expliquer comment fonctionne le modèle MIDP. Pour pouvoir montrer quelque chose sur un terminal MIDP, on a besoin d'obtenir le display du terminal, qui est représenté par classe `javax.microedition.lcdui.Display`. La classe `Display` est instanciée pour chaque MIDlet et fournit des méthodes pour récupérer des informations sur les possibilités de l'écran du terminal.

L'abstraction la plus intéressante est le *Screen*, qui encapsule et organise les objets graphiques et coordonne l'entrée de l'utilisateur à travers le terminal. Les Screens sont représentés par l'objet `javax.microedition.lcdui.Screen` et sont vus par l'objet `Display` au moyen de l'appel de sa méthode `setCurrent()`. Il pourrait y avoir plusieurs écrans (screen) dans une application, mais il y'a un seul qui est visible (ou curen) en même temps sur le `Display`, et l'utilisateur n'a accès qu'aux éléments de ce `Screen`.

La figure 7 montre la relation un-à-plusieurs entre le Display et les Screens.

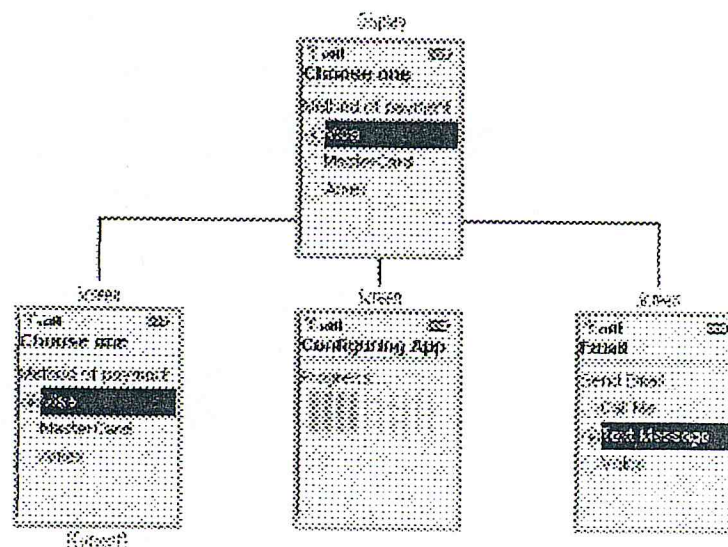


Figure III-7: Relation entre le Display et les Screens.

Il y'a trois types de screens dans le GUI MIDP:

- Les Screens qui encapsulent entièrement un composant interface utilisateur complexe, telle qu'un composant List ou un TextBox (le composant List est vu en figure 8 et le composant TextBox en figure 9). La structure de ces écrans est prédéfinie et l'application ne peut pas rajouter d'autres composants à ces Screen.



Figure III-8: Une List de choix XCLUSIVE



Figure III-9: Un exemple de TextBox

- Les écrans génériques qui utilisent un composant Form. L'application peut rajouter du texte, des images, et un ensemble simple de composants UI en relation avec le Form, qui agit comme un conteneur.
- Les Screens utilisés dans le contexte d'une API de bas niveau, telle qu'une sous-classe de la classe Canvas ou de la classe Graphics.

6.4. Le package lcdui

Toutes les classes GUI du MIDP sont contenues dans le package `javax.microedition.lcdui`.

Ce package contient trois interfaces et vingt et une classes, comme données en table 2 et table 3.

Interface	Description
Choice	Définit une API pour une interface utilisateur qui implémente une sélection à partir d'un nombre prédéfini de choix.
CommandListener	Utilisé par des applications qui ont besoin de recevoir des événements de haut niveau à partir des implémentations.
ItemStateListener	Utilisé par des applications qui ont besoin de recevoir des événements indiquant des changements dans l'état interne des éléments interactifs.

Tableau III-2: Les interfaces lcdui

Classe	Description
Alert	Un Screen qui montre des données à l'utilisateur et attend pendant une certaine période de temps avant de passer au Screen suivant.
AlertType	Une classe utilitaire qui indique la nature de l'Alert.
Canvas	La classe de base pour écrire des applications qui ont besoin de traiter des événements de bas niveau et d'effectuer des Appels graphiques pour dessiner sur le Display.
ChoiceGroup	Un groupe d'éléments sélectionnables pour être placé dans un Form.
Command	Une construction qui encapsule l'information sémantique d'une action.
DateField	Un composant éditable pour présenter des données du calendrier et de l'information sur le temps qui pourrait être placé dans un Form.
Display	Un utilitaire pour représenter le gestionnaire de l'écran et les terminaux d'entrée du système.
Displayable	Un objet ayant la possibilité d'être placé sur le Display.
Font	Un utilitaire représentant les fonts et les métriques des fonts.
Form	Un Screen qui contient un mélange d'éléments (images, texte, Textfields ou Choicegroups par exemple).
Gauge	Un utilitaire qui implémente un affichage d'un histogramme à barres d'une certaine valeur à utiliser dans un Form.
Graphics	Un utilitaire qui fournit des possibilités de rendu d'éléments géométriques bidimensionnels.
Image	Un utilitaire qui maintient des données image graphique.
ImageItem	Un utilitaire qui fournit un contrôle de positionnement quand les objets sont rajoutés à un Form ou une Alert.
Item	Une super classe pour tous les composants pouvant être mis dans un Form ou une Alert.
List	Un Screen contenant une liste de choix.
Screen	La super classe des classes interface utilisateur de haut niveau.
StringItem	Un élément qui peut contenir une String.
TextBox	Un Screen permettant à l'utilisateur d'entrer et d'éditer du texte.
TextField	Un composant texte éditable pouvant être placé dans un Form.
Ticker	Une pièce de texte qui s'affiche par un déplacement continu le long de l'écran. Il pourrait être attaché à tout type de Screen sauf le Canvas.

Tableau III-3: Les classes de lcdui

Le diagramme de classes de la figure 10 montre les principales classes et les relations entre elles.

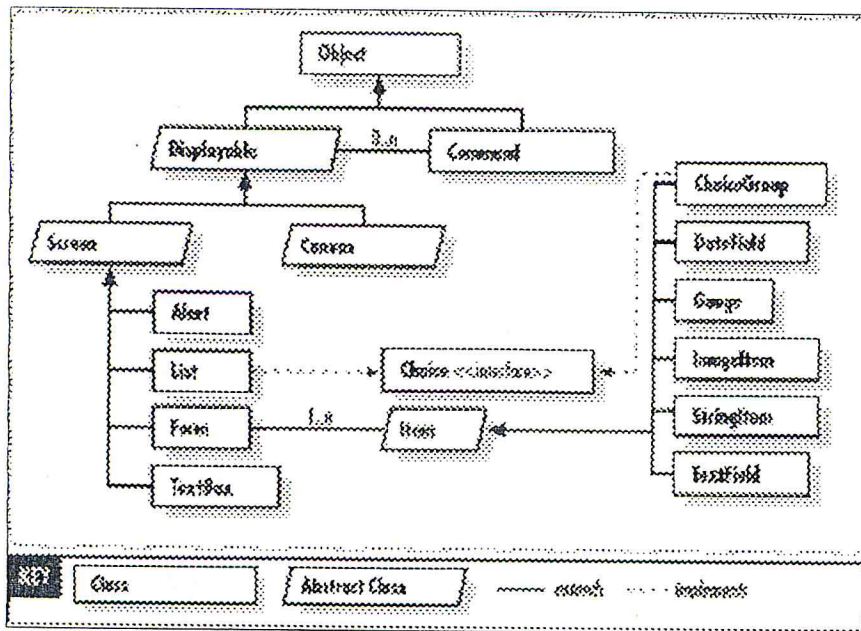


Figure III-3: Diagramme de classes des principales classes du package *lcdui*.

6.5. Gestion des événements

Le modèle d'événement des composants `javax.microedition.lcdui` ressemble au système utilisé par AWT (Abstract Windowing Toolkit) et Swing. Chaque type d'événement dispose d'une interface définie pour représenter cet événement, et chaque composant appellera une méthode donnée sur cette interface afin de signaler l'occurrence de cet événement à un ou plusieurs objets d'écoute. Comme vous pouvez vous y attendre, la différence tient au fait que les choses sont simplifiées dans le système `javax.microedition.lcdui`.

Le nombre d'événements à gérer est réduit et chaque composant ne peut avoir qu'un seul module d'écoute enregistré à la fois, d'où le passage de `addXListener` dans les composants AWT à `setXListener` dans les classes `javax.microedition.lcdui`.

6.6. Exemple de MIDlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

// Une première MIDlet avec un texte simple et quelques commandes.
public class PremiereMIDlet extends MIDlet
    implements CommandListener {

    //Les commandes exit, info, et buy
    private Command exitCommand;
    private Command infoCommand;
    private Command acheteCommand;

    //Le display pour la MIDlet
    private Display display;

    public PremiereMIDlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Exit", Command.Exit, 1);
        infoCommand = new Command("Info", Command.SCREEN, 1);
        acheteCommand = new Command("Acheté", Command.SCREEN, 2);
    }

    // Démarrer la MIDlet par la création d'un TextBox et
    //l'association des commandes et du listener.

    public void startApp() {
        TextBox t = new TextBox("PremiereMIDlet",
            "Bienvenu a la Programmation MIDP ", 256, 0);
        t.addCommand(exitCommand);
        t.addCommand(infoCommand);
        t.addCommand(acheteCommand);
        t.setCommandListener(this);
        display.setCurrent(t);
    }

    // Pause est vide parce-qu'il n'y a pas d'activité en arrière plan
    // ou de record store à fermer.

    public void pauseApp() { }
```

```
// La destruction doit effacer toute chose non prise en compte
// par le récupérateur.
// Dans cette application il n'y a rien à nettoyer.

public void destroyApp(boolean unconditional) { }

// Répond aux commandes. Ici on n'a implémenté que la commande exit.
// Dans la commande exit, il y'a le nettoyage et l'action d'informer que
//la MIDlet a été détruite.

public void commandAction(Command c, Displayable s) {
    if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    }
}
}
```

Remarque :

- Nous importons les packages `midlet` et `lcdui`. Le package `package midlet` définit le MIDP, et le package `midlet` fournit l'API de l'interface utilisateur pour implémenter les interfaces utilisateur pour les application MIDP.
- Chaque MIDlet doit étendre la classe `MIDlet`, qui permet dans l'ordre de démarrer, stopper, et nettoyer la MIDlet.
- Une MIDlet ne devrait pas avoir une méthode `public static void main()`.

Dans `PremiereMIDlet`, la classe `Command` est utilisée pour créer des commandes. Cette classe encapsule l'information sémantique d'une action. La commande elle-même contient seulement de l'information sur la commande, mais rien sur l'action actuelle qui se produit quand une commande est activée. L'action est définie dans un `CommandListener` associé au `Screen`. Regardons de près une des commandes du listing 1:

```
infoCommand = new Command("Info", Command.SCREEN, 2);
```

Comme on le voit, une commande contient trois pièces d'information: une étiquette « label », un type, et une priorité. Le label (qui est une `String`) est utilisé pour la représentation visuelle de la commande. Le type de la commande spécifie son rôle. Et la

Dans `PremiereMIDlet`, la classe `Command` est utilisée pour créer des commandes. Cette classe encapsule l'information sémantique d'une action. La commande elle-même contient seulement de l'information sur la commande, mais rien sur l'action actuelle qui se produit quand une commande est activée. L'action est définie dans un `CommandListener` associé au `Screen`. Regardons de près une des commandes du listing 1:

```
infoCommand = new Command("Info", Command.SCREEN, 2);
```

Comme on le voit, une commande contient trois pièces d'information: une étiquette « label », un type, et une priorité. Le label (qui est une `String`) est utilisé pour la représentation visuelle de la commande. Le type de la commande spécifie son rôle. Et la valeur priorité décrit l'importance de cette commande relativement aux autres commandes du `Screen`. La valeur priorité de 1 indique que la commande est la plus importante, les valeurs de priorité hautes indiquent que les commandes auxquelles elles sont associées sont de moindre importance.

Quand une application est exécutée, le terminal choisit le placement des commandes sur la base du type de la commande, et place les commandes similaires sur la base de leur priorité. Dans l'exemple précédent, il y'a les trois commandes suivantes:

```
exitCommand = new Command("Exit", Command.Exit, 1);  
infoCommand = new Command("Info", Command.SCREEN, 1);  
acheteCommand = new Command("Acheté", Command.SCREEN, 2);
```

Dans cet exemple, le gestionnaire de l'application applique la commande `exit` au `Screen`, et comme cela est vu dans les figures 11.2 et 11.3, il crée une commande menu pour contenir la commande `info`, et la commande `Acheté`.



Figure III-11.1: Démarrage initial

La figure 11.1 montre à quoi ressemble le Screen quand l'application vient juste d'être démarrée.



Figure III-11.2: Les Command Menu et Exit

Cliquer sur le bouton de sélection nous amène à la figure 11.2.

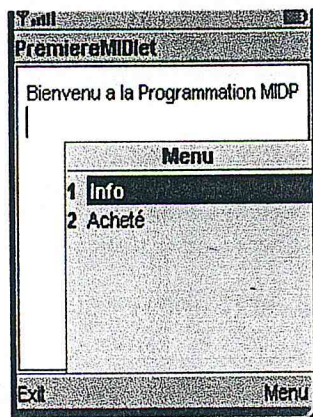
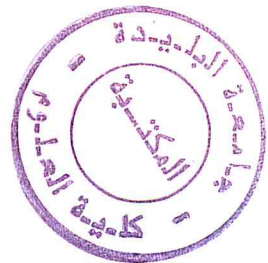


Figure III-4: les Commands Acheté et Info

Et cliquer sur le bouton soft sous Menu nous amène à la Figure 11.3.

7. La base de données légère

RMS (Record Management System) est une API de stockage persistant sur le terminal. C'est en quelque sorte une base de données indépendante du terminal. Chaque enregistrement est représenté sous forme de tableau d'octets. La mise à jour est dite atomique : l'enregistrement entier est réécrit à chaque fois.

Les enregistrements sont stockés dans ce que l'on appelle un Record store. Si l'on veut faire un parallèle avec les SGBD relationnels, RMS correspond au SGBD lui-même et le Record store à la table. D'ailleurs, le parallèle de la notion de clé primaire des bases de données relationnelles est le « recordID ». Il s'agit de l'identifiant de l'enregistrement. C'est un nombre entier. La valeur de l'ID du premier enregistrement est 1 et chaque nouvel enregistrement a une valeur ID augmentée de un.

7.1 Gestion des records store

Plusieurs méthodes permettent de gérer les Records store.

"openRecordStore" et "closeRecordStore" permettent respectivement d'ouvrir et de fermer un Record store. La liste de tous les RecordStores peut être obtenue par "listRecordStore". "deleteRecordStore" en supprime un. Le nombre d'enregistrements dans un Record store est retourné par "getNumRecords".

Les opérations de base sur les enregistrements sont assurées par ces méthodes : "addRecord", "deleteRecord", "getRecord", "setRecord", "getRecordSize".

L'API RMS dispose cependant de quelques particularités supplémentaires, concernant la sélection des enregistrements. La première est l'utilisation de la méthode "RecordEnumeration" pour lister tous les enregistrements du Record store. La seconde est la possibilité de définir un filtre avec la méthode "RecordFilter". Enfin, l'interface "RecordComparator" doit être implémentée pour que des enregistrements puissent être comparés et donc triés.

8. Connexion réseaux :

Les principaux atouts d'un terminal sans fil sont sa connectivité et son accessibilité, qui permettent de rester connecté avec le monde entier à tout instant et en tout lieu. Un certain nombre d'améliorations, telles que la couverture du réseau, la bande passante et les technologies sans fil, ont achevé de mettre ces terminaux au goût du jour.

La connectivité est assurée par les outils de connexion au réseau et de communication. La principale difficulté pour l'API réseau de J2ME consiste à intégrer les spécificités de

connectivité de chaque famille de terminal. Par exemple, MIDP n'impose que l'implémentation du protocole HTTP, alors qu'un terminal donné peut implémenter d'autres protocoles.

Une autre spécification très importante pour les réseaux est liée aux différents types de réseaux utilisés : réseau à commutation de paquets ou à commutation des circuits. Où le premier utilise une communication à base de datagramme, comme UDP (User Datagram Protocol) et le seconde à base de socket, comme TCP (Transport Control Protocol) [DEL 02]. Pour cela, le CLDC fournit un framework, le GCF (Generic Connection Framework).

8.1. Le GCF (Generic Connection Framework)

J2ME doit pouvoir supporter un certain nombre de protocoles, dont le plus répandu est HTTP. Il doit donc à la fois être assez générique pour supporter tous les terminaux et assez spécifique pour supporter les terminaux particuliers. Le GCF est conçu pour relever ce défi.

Le GCF est un ensemble de classes et d'interfaces modélisant six types de connexion différents, allant d'une simple connexion uni-directionnelle à une connexion à base de datagrammes.

Le GCF repose sur l'utilisation de classes abstraites. Il définit un « moule » générale, et il revient à chaque fabricant de remplir ce moule. Pour garantir à la fois l'extensibilité et la flexibilité, le CGF utilise sept interfaces de Connection « figure12 ».

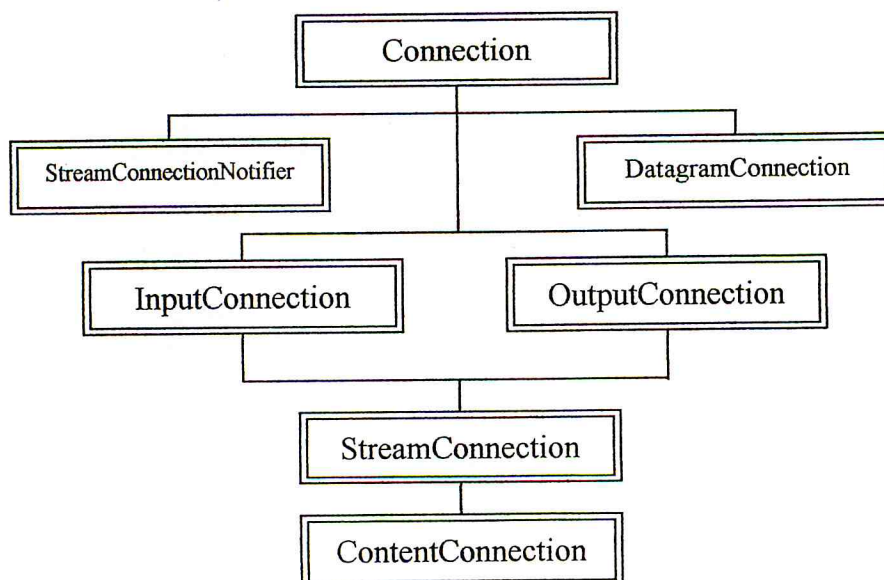


Figure 12: L'arborescence des classes du GCF

8.2. Utilisation

Pour spécifier un type de connexion, vous devez utiliser les méthodes statiques de la classe `Connector`. Ainsi, la méthode `open`, qui accepte en argument un URI (Uniform Resource Indicator) retourne un objet `Connection`.

Par exemple, pour ouvrir une connexion HTTP, le code serait :

```
import javax.microedition.io.*;
URLConnection conn = (URLConnection) Connector.open
    ("http://www.webforyoo.com");
```

Par contre, pour ouvrir un fichier en lecture seule, le code serait :

```
import javax.microedition.io.*;
URLConnection conn = (URLConnection) Connector.open
    ("file:/autoexec.bat", Connector.READ);
```

Pour gérer les flux en entrée et en sortie, utilisez les méthodes `openInputStream`, `openDataInputStream`, `openOutputStream` et `openDataOutputStream`.

La syntaxe d'une URI est la suivante :

Protocole:Adresse;paramètres

Les paramètres sont :

- protocole indique comment se connecter,
- adresse indique où se connecter,
- paramètres indiquent les paramètres de connexion.

Protocole est une chaîne de caractères du type "http" ou "mailto". Par contre, la syntaxe du reste de l'URI dépend du protocole utilisé. Donc chaque connexion (n'importe ce qu'il est la forme de communication) est créée en appelant la méthode statique `Connector.open()`:

```
Connector.open("http://www.blida.com");
```

```
Connector.open("socket://localhost:8080");
```

```
Connector.open("datagram://www.webnow.com");
```

```
Connector.open("comm.:0;baudrate=9600");
```

```
Connector.open("file:/monText.txt");
```

Vous devez bien comprendre que le CLDC ne définit que les classes et les interfaces du GCF, par l'implémentation elle-même. Ce sont les profils qui implémentent les protocoles adéquats. Par exemple, MIDP implémente le protocole HTTP. Mais tous les protocoles peuvent ne pas être implémentés.

Pour qu'une application soit portable, elle ne doit utiliser que les protocoles définis par le profile du terminal sur lequel tourne l'application.

L'interface `Connection` définit une méthode `close`, qui ferme la connexion.

L'interface `InputConnection` permet de lire les flux en entrée, d'une manière analogue à la lecture d'un fichier. Elle ajoute deux méthodes au `Connection` : `InputStream` et `DataInputStream`.

Si vous utilisez à la fois des flux en entrée et en sortie, comme avec le protocole socket, vous devrez utiliser l'interface `StreamConnection`, qui étend `InputConnection` et `OutputConnection`. Afin de bien fermer la connexion, vous devez fermer tous les flux utilisés.

L'interface `ContentConnection` étend `StreamConnection` et lui ajoute trois méthodes d'accès à certaines données stockées dans les réponses HTTP : `getType`, `getEncoding` et `getLength`.

L'interface `StreamConnectionNotifier` permet d'écrire une application serveur. Elle ne comprend qu'une seule méthode, `acceptAndOpen`. Cette méthode attend qu'un client se connecte au serveur et retourne alors un `StreamConnection` qui peut être utilisé pour communiquer avec le client. Le protocole `serversocket` utilise cette interface pour ouvrir des sockets serveur.

L'interface `DatagramConnection` permet d'envoyer et de recevoir des datagrammes, grâce à l'interface `Datagram`. Cette interface est modélisée d'après la classe `java.net.DatagramSocket` de la plate-forme J2SE.

8.3. Programmation avec les sockets

Une socket est un lien de communication point à point entre deux programmes fonctionnant dans un réseau. Il s'agit du mécanisme de communication le plus basique. Cependant, ce type de communication n'est pas supporté par tous les fabricants.

Avant que l'émetteur et le récepteur puissent communiquer, l'émetteur doit établir une communication via les sockets. Pour cela, le récepteur doit être en mode d'écoute des requêtes de connexion. L'émetteur émet alors une requête de connexion au récepteur, qui accepte ou refuse cette requête.

La première étape consiste à établir une connexion à base de socket avec le serveur distant à l'aide de la méthode `Connector.open()`.

Pour la réception de données, il est nécessaire de créer un `InputConnection` puis de lire les données entrantes dans un `InputStream`. Pour l'émission de données, il faut créer un `OutputConnection` puis écrire les données sortant dans un `OutputStream`.

La dernière étape est la fermeture de la connexion.

8.4. Programmation avec les datagrammes

Un datagramme est un message indépendant envoyé sur le réseau. Il est utilisé dans le cadre d'une communication à base de paquets, comme avec le UDP. Il n'y a aucune garantie que les paquets arrivent à destination, et aucune connexion n'est ouverte pour cette communication. De plus, les datagrammes ne sont pas supportés par tous les terminaux.

Les datagrammes sont particulièrement utiles pour le temps-réel, qui supporte la perte de paquets, et pour les réseaux à commutation des paquets.

La programmation à base de datagrammes en J2ME repose sur deux interfaces : `DatagramConnection` et `Datagram`.

`DatagramConnection` propose des méthodes de communication fondées sur le protocole UDP, tandis que `Datagram` fournit un stockage pour un message de datagramme. Un message `Datagram` peut être envoyé ou reçu par une connexion `DatagramConnection`.

8.5. Programmation avec HTTP

L'interface `HttpConnection` simplifie la communication avec le protocole HTTP en proposant des méthodes spécifiques pour ce protocole.

Il existe d'autres avantages à utiliser cette interface :

- Tous les terminaux MIDP ne supportent pas les sockets ni les datagramme, tandis que HTTP est obligatoirement implémenté. Il permet donc une plus grande portabilité.
- HTTP est indépendant du réseau, contrairement aux sockets et aux datagrammes.

HTTP peut être implémenté à partir de différents protocoles : IP (comme TCP/IP) ou non-IP, comme WAP ou i-Mode. Toutes les implémentations de MIDP doivent supporter le

protocole HTTP. Il est de la responsabilité de celui qui propose l'implémentation sur le terminal de supporter ou non les connexions à base de data gramme. Il est recommandé de n'utiliser que les protocoles supportés par MIDP afin de garantir la portabilité sur tous les terminaux mobiles [DEL 02].

En utilisant le protocole HTTP, vous pouvez développer une application pouvant tourner sur tout terminal MIDP, que ce soit un téléphone WAP, un téléphone i-Mode, un PDA ou un terminal Bluetooth.

9. Les outils de développement

Afin de pouvoir développer des applications J2ME vous devez posséder un minimum d'outils.

- Vous devez tout d'abord avoir une version du SDK java 2 (au moins la version 1.3 aujourd'hui en version 1.5), installé sur votre poste, et
- un éditeur de texte et
- un simulateur pour testez votre application, ce dernier est de but de facilité la création, la compilation et le test de votre application J2ME.

Il existe pas mal d'émulateurs J2ME. Presque chaque constructeur de téléphones portables en offre un avec les modèles de simulation identique à leur gamme de produits réels (Nokia developer's Suit , IBM Visual Age Micro Edition , Motorola MADK 2.0 ...).

L'éditeur Borland fournit également une extension (Mobil Set) à son logiciel Jbuilder afin de s'adapter à la technologie J2ME.

Nous nous concentrerons malgré tout sur le Wireless Toolkit de Sun, car il est largement suffisant, simple à installer et à utiliser, neutre de toute implémentation et indépendant de l'éditeur de code Java.

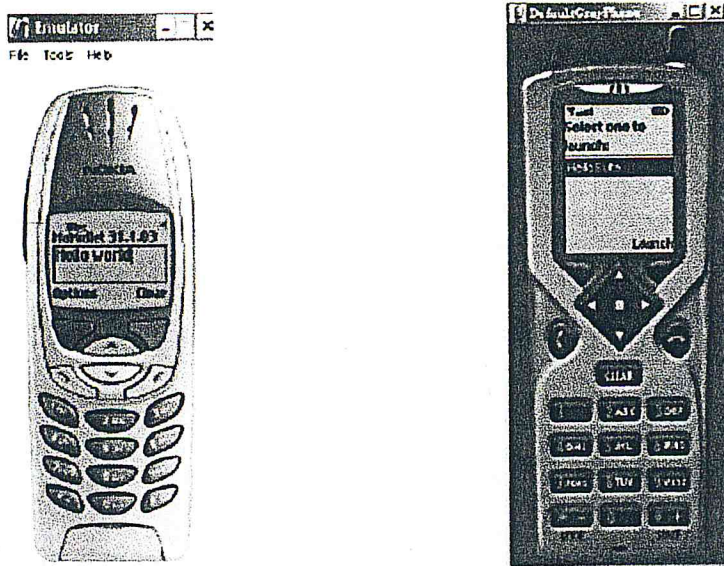


Figure III-13: Les émulateurs des kits Nokia et Sun

9.1. J2ME Wireless Toolkit

J2ME Wireless Toolkit est un émulateur qui supporte le développement des applications Java tournant sur des appareils supportant le Mobile Information Device Profile (MIDP), tels que les téléphones cellulaires, les pagers bi-directionnels et les communicateurs.

Il existe des versions de WTK dont chacune pour une propre version de profile MIDP pour implémenter ses nouveautés. Par exemple, le WTK v 2.0 ajout le OTA qui est arrivé avec le MIDP 2.0.

9.1.1. Obtenir le kit de développement

Pour ce faire, il suffit de se rendre sur le site de Sun Microsystems et de télécharger une des deux versions disponibles du *Wireless Toolkit*. En effet, la première version, c'est-à-dire la 1.04, exploite le MIDP 1.0 et la seconde le MIDP 2.0. C'est donc principalement sur la version du profil J2ME fourni que se différencient ces deux éditions de l'utilitaire. Cette plateforme de développement existe aussi bien pour Linux que pour Windows ou Solaris SPARC. Il faudra néanmoins des droits d'administrateur pour l'installer.

Une fois cette boîte à outils installée, tout ce dont on a besoin pour compiler, pré-vérifier et simuler des applications J2ME se trouve désormais sur le disque.

9.1.2 Utiliser le Wireless Toolkit

Les méthodes de création et de déploiement d'applications J2ME sont différentes par rapport aux applications standards. En effet, il n'existe pas de vérificateur de classe

d'exécution pour l'implémentation MIDP. Une fois que le code est compilé et avant qu'il puisse être exécuté sur un appareil MIDP, il doit être pré-vérifié à l'aide de l'outil `Preverify.exe`. De plus, un ensemble de bibliothèques, de classes et de fichiers de démarrage doivent être inclus via un commutateur de ligne de commande avec le compilateur Java. Le Wireless Toolkit est capable d'effectuer ces actions supplémentaires et permet donc de ne pas s'en inquiéter. La création d'une application avec le Wireless Toolkit passe par ces étapes de base :

1. Lancer le toolkit (*KToolbar*).

(*KToolbar* c'est l'outil de Wireless Toolkit capable de vous aide à créer, modifier et tester votre *MIDlets*).

2. Créer un projet à l'aide du bouton *Create Project*. Saisir le nom du projet ainsi que le nom de la classe *MIDlet* principale (sans le `.class`) qui devra être exécutée pour démarrer l'application. Le toolkit crée alors un répertoire nommé avec le nom du projet. Ce répertoire se situe dans `<répertoire d'installation>\apps\<le nom du projet>`
3. Rédiger le code source Java dans l'éditeur de votre choix, puis copier le fichier correspondant dans le sous-répertoire *src* du projet.

Ensuite, il faut construire l'application à l'aide du bouton *Build* et elle sera exécutée en appuyant sur le bouton *Run* avec l'émulateur d'un appareil mobile. A l'aide du menu *Project - Package - Create package*, l'outil construit les fichiers `.jar` et `.jad` qui seront nécessaires pour exporter l'application vers un terminal réel.

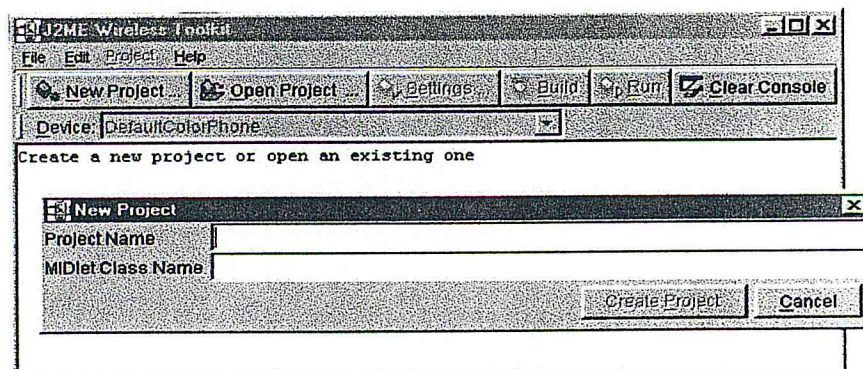


Figure III-14: KToolbar et la fenêtre New Project

On peut, à l'utilisation de *KToolbar*, simuler l'exécution d'une *MIDlet* sur diverse appareils « figure14 ».

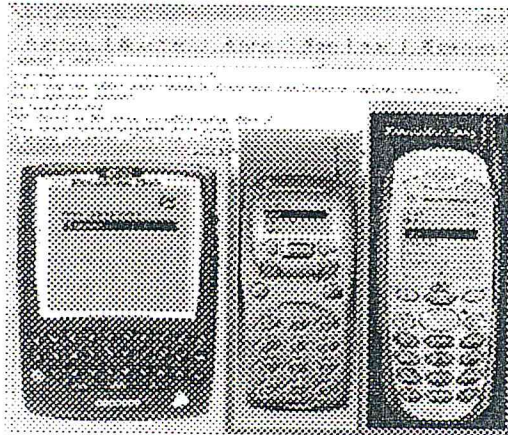


Figure III-9: KToolbar avec divers simulateur

10. Les concurrents de J2ME

Il est évident qu'avec l'ampleur que cette technologie a pris dans le monde des terminaux sans fils, Sun se retrouve en concurrence directe avec deux plates-formes que nous allons décrire ici.

10.1. Microsoft Embedded Architecture [SAM 02]

L'offre de Microsoft est essentiellement basée sur son système d'exploitation embarqué « Windows CE ». Microsoft propose des composants système en tout genre et adaptés à tout type de besoin, charge ensuite à l'intégrateur de définir sa propre configuration. Cette approche comparée à Java est donc c'est différente.

Avec l'arrivée du Framework .NET, la partie embarquée de l'offre de Microsoft a subi quelques évolutions majeures. Ainsi, le .NET Compact Framework en version bêta a fait son apparition au sein de l'architecture avec une philosophie proche de J2ME pour Java : Un ensemble d'API allégés permettant de répondre aux exigences des périphériques en terme de ressources. Pour l'heure, .NET CF se destine aux plate-formes Windows CE bien qu'il ait été conçu pour être portable. A l'avenir, on pourrions très bien voir apparaître des versions Windows CE destinées à d'autre système d'exploitation [SAM 02].

Il est à noter qu'il existe des machines virtuelles Java permettant d'exécuter une application J2ME sur Windows CE. Ainsi, MicroJBlend, une extension de la KVM de Sun, propose nativement ce genre de solution. L'architecture embedded de Microsoft n'intègre pas de notion de configuration ou de profil telle qu'il en existe dans Java.

J2ME et Microsoft embedded Architecture ont une approche totalement différente dans leur manière d'aborder le développement embarqué. Si Java privilégie l'aspect Multi-périphériques et Multi-OS, Microsoft préfère concentrer ses efforts sur Windows CE, son système d'exploitation maison. Concernant les environnements de développement, les deux prétendants proposent une large palette d'outils aussi puissants les uns des autres avec Visual Studio .NET d'un côté et J2ME Toolkit de l'autre. Les développeurs habitués à l'environnement Visual Studio ne pourront qu'apprécier Smart Device Extensions, quant aux développeurs préférant Java, ils trouveront leur bonheur dans le large choix d'IDE proposé sur le marché.

Les API mises en oeuvre, elles, se basent essentiellement sur les architectures implémentées de part et d'autre avec un allègement des classes jugées trop gourmandes en terme de ressources.

Sur ce marché de l'embarqué, Sun possède des alliés de poids en la personne d'IBM ou d'Oracle. Quant aux constructeurs, les alliances se forment à tout va avec Nokia, Motorola, Siemens ou Sony. Il faudra beaucoup d'énergie à Microsoft pour arriver à remporter des batailles sur un marché hautement stratégique et promis à un bel avenir (gprs, umts, ...). Demain sera riche en enseignements et les technologies nous réserveront sûrement encore quelques surprises.

Il est très important de comprendre que J2ME cible énormément de catégories de produits. Le MIDP/CLDC qui présente des caractéristiques moins puissantes en terme d'affichage et d'exécution que l'équivalent .NET. La pauvreté de l'affichage de cette configuration est due en partie à la nature même des API mises en oeuvre. Un PocketPC 2000 ou 2002 avec Windows CE.NET offre énormément plus de capacités en terme de traitement qu'un Nokia 6600. Afin de mieux juger les différentes offres, il aurait fallu prendre en compte le Personal Profile proposé par Sun, qui interprète le nombre d'applications développées sur la base de cette technologie.

10.2. Binary Runtime Environment For Wireless (Brew) [Qualcomm]

Ce "Binary Runtime Environment for Wireless" est un système d'exploitation développé par Qualcomm, comparable à Symbian ou à PalmOS, dédié aux téléphones mobiles. Pour le moment essentiellement disponible pour les téléphones CDMA (équivalent à GSM en Asie et Amérique), Qualcomm a néanmoins fait la démonstration au GSM WorldCongress que son environnement logiciel était disponible pour les réseaux GPRS.

BREW est destiné spécifiquement aux applications sans fil pouvant être téléchargées et exécutées sur des appareils mobiles. L'environnement d'exécution BREW est mis gratuitement à la disposition des fabricants d'appareils CDMA. Il ressemble beaucoup à la machine virtuelle de Java, à la différence près qu'il n'est pas conçu pour assurer la portabilité d'un appareil à l'autre. Cette plate-forme possède des capacités avancées, notamment les technologies GPS, VOIP, Bluetooth 1.1, MP3 et MIDI. Comme la plate-forme d'exécution BREW fonctionne au niveau du jeu de composants (chipset), elle peut mieux que J2ME tirer profit des capacités de communication et multimédia des appareils mobiles.

Les services BREW incluent les communications réseau TCP/UDP, le support HTTP, la messagerie SMS, des services de téléphonie avancés et la prise en charge de l'accès au système de fichiers. BREW permet le développement de logiciels en C/C++, et le kit SDK BREW C/C++ s'intègre aisément à l'environnement de développement Visual C++ de Microsoft. Dans le cadre de la plate-forme d'application BREW, Qualcomm offre un kit de portage BREW, un kit SDK BREW qui comprend des outils de développement, une documentation, des API, etc., et un système de distribution BREW

BREW, compte deux machines virtuelles différentes pour faire fonctionner les applications java. D'un point de vue conceptuel, BREW agit comme une couche placée en dessous de la JVM, il est alors possible de développer des applications Java qui peuvent être téléchargées et exécutées sur des appareils compatibles BREW au moyen de BDS (BREW Distribution System). BDS 2002 permet de télécharger des *MIDlets* Java et même une JVM, qui, dans le cas présent, n'est rien d'autre qu'une application BREW. Bien qu'aucune évaluation concrète des performances de J2ME exécuté sur BREW ne soit disponible, des exigences de traitement supplémentaires apparaîtront certainement du fait de la présence d'une autre couche (BREW) dans l'exécution d'applications J2ME. A en croire certaines sources de Qualcomm, cela ne devrait pas poser problème, car la nouvelle gamme de puces disposera d'une puissance de traitement suffisante.

Globalement, BREW n'est pas très différent de Java. Pour l'utilisateur, il présente une similitude dans l'exécution des applications; pour le développeur, un modèle de développement plus ou moins similaire. Toutefois, certaines distinctions doivent être faites entre ces deux technologies :

BREW, à l'inverse de J2ME qui vise les appareils grand public et intégrés, cible exclusivement les appareils sans fil. En conséquence, il est davantage polyvalent dans le domaine de la téléphonie sans fil mais, en contrepartie, manque de portabilité.

A l'heure actuelle, BREW ne possède pas l'éventail d'outils de développement dont disposent les développeurs Java.

Comme J2ME est un langage interprété, il devrait être plus lent (au moins en théorie) que BREW, qui est plus proche du processeur.

Les technologies J2ME et BREW offrent toutes deux aux développeurs des possibilités intéressantes pour tirer profit de la convergence du sans fil et de l'internet. Si les développeurs Java peuvent considérer que J2ME est la meilleure option pour un large éventail de petits appareils grand public, BREW présente des avantages certains pour les téléphones sans fil.

Dans de nombreux cas, bien entendu, le choix d'une technologie par rapport à l'autre sera dicté par la plate-forme que l'opérateur de réseau décide d'adopter [Qualcomm].

11. Conclusion

La mise en place d'une plate-forme technique commune à tous les périphériques est une opération très délicate. Chaque appareil possède des spécificités qui impliquent aux applications de s'adapter à différentes caractéristiques d'affichage ou de pointage. Toute la difficulté de concevoir des technologies pour l'embarqué réside dans cette complexité inhérente à la diversité de l'offre. Java 2 Micro Edition est une architecture technique dont le but est de fournir un socle de développement aux applications embarquées. L'intérêt étant de proposer toute la puissance d'un langage tel que Java associé aux services proposés par une version bridée de J2SE. Ce qu'il faut retenir de l'architecture J2ME c'est qu'elle se compose:

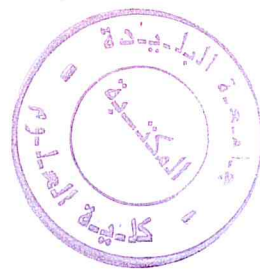
- d'une configuration contenant la machine virtuelle et les fonctionnalités minimales nécessaires à accéder au matériel et au réseau.
- d'un profil qui contient le reste des fonctionnalités exploitables sur une famille d'appareils aux caractéristiques semblables.
- et des bibliothèques optionnelles que les constructeurs sont libres d'implémenter.

Les applications écrites à partir du profil MIDP se nomment les *MIDlets*. Ces applications doivent tenir compte de la configuration limitée des dispositifs mobiles. Ce ne sont en effet pas des stations de calcul et ont des éléments de pointage souvent très simples ou en sont carrément dépourvus. Ainsi, une *MIDlet* ne doit pas, si possible, demander plus d'une interaction à la fois de la part de l'utilisateur.

Malgré que le plus grand nombre des *MIDlets* actuelles s'inscrive dans le cadre des jeux ou des petits utilitaires tels que les agendas ou convertisseurs d'unités, ce n'est de loin pas le

principal intérêt de cette technologie. Avec le standard MIDP 2.0, les opportunités de créer des services mobiles dans des domaines très variés sont grandes. Ainsi, on peut imaginer des applications de m-commerce (m pour mobile) d'achats en ligne et de consultation de la bourse, d'enchères ou de d'informations en temps réel. C'est justement ce genre de domaines qui présente un grand intérêt.

Pour développer ce genre des applications nous avons besoin des technologies côté serveur, Java propose les *Servlets* et le JDBC qui nous vous exposera dans le chapitre suivant.



Chapitre IV Java est les applications *web*

1. Introduction

L'arrivée des applications Java sur le serveur, de la *Servlet* autonome à la plate-forme J2EE (Java 2 Enterprise Edition) complète, a été l'une de plus existantes orientations de la programmation de Java. Le langage Java été à l'origine destiné à une utilisation dans des petits périphériques embarquables. Il a d'abord été utilisé pour le développement de contenu élaboré de côté client sous la forme d'Applets [HEI 00]. Mais jusqu'à ces dernières années, le potentiel de Java comme plat-forme de développement de côté serveur a été très largement négligé. Maintenant Java est devenu un langage parfaitement adapté aux développements côté serveur.

Dans ce chapitre nous allons présenter premièrement des généralités sur les applications *web*, en suite nous allons présenter les *Servlet* et détailler comment on peut l'utiliser pour développer une application web évoluer.

Partie I : Les applications *Web*

2. Présentation de l'architecture d'un système client/serveur

De nombreuses applications fonctionnent selon un environnement client/serveur, cela signifie que des **machines clientes** (des machines faisant partie du réseau) contactent un **serveur**, une machine généralement très puissante en terme de capacités d'entrée-sortie, qui leur fournit des **services**. Ces services sont des programmes fournissant des données telles que l'heure, des fichiers, une connexion, ...

Les services sont exploités par des programmes, appelés programmes clients, s'exécutant sur les machines clientes. On parle ainsi de client FTP, client de messagerie, ..., lorsque l'on désigne un programme, tournant sur une machine cliente, capable de traiter des informations qu'il récupère auprès du serveur (dans le cas du client FTP il s'agit de fichiers, tandis que pour le client messagerie il s'agit de courrier électronique).

Dans un environnement purement Client/serveur, les ordinateurs du réseau (les clients) ne peuvent voir que le serveur, c'est un des principaux atouts de ce modèle.

3. Avantages de l'architecture client/serveur

Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux atouts sont:

- des ressources centralisées: étant donné que le serveur est au centre du réseau, il peut gérer des ressources communes à tous les utilisateurs, comme par exemple une base de données centralisée, afin d'éviter les problèmes de redondance et de contradiction.
- une meilleure sécurité: car le nombre de points d'entrée permettant l'accès aux données est moins important.
- une administration au niveau serveur: les clients ayant peu d'importance dans ce modèle, ils ont moins besoin d'être administrés.
- un réseau évolutif: grâce à cette architecture on peut supprimer ou rajouter des clients sans perturber le fonctionnement du réseau et sans modifications majeures.

4. Inconvénients du modèle client/serveur

L'architecture client/serveur a tout de même quelques lacunes parmi lesquelles:

- un coût élevé dû à la technicité du serveur.
- une maillon faible: le serveur est le seul maillon faible du réseau client/serveur, étant donné que tout le réseau est architecturé autour de lui! Heureusement, le serveur a une grande tolérance aux pannes (notamment grâce au système RAID).

5. Fonctionnement d'un système client/serveur

Un système client/serveur fonctionne selon le schéma suivant:

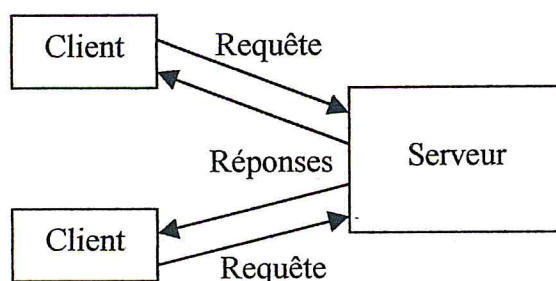


Figure IV-1: Le modèle client-serveur

- Le client émet une requête vers le serveur grâce à son adresse et le port, qui désigne un service particulier du serveur
- Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine client et son port

6. Les types des applications client-serveur

D'après l'architecture on peut distinguer deux types des applications client-serveur :

6.1. Architecture à deux niveaux

L'architecture à deux niveaux (aussi appelée architecture deux-tiers, tier signifiant étage en anglais) caractérise les systèmes clients/serveurs dans lesquels le client demande une ressource et le serveur la lui fournit directement. Cela signifie que le serveur ne fait pas appel à une autre application afin de fournir le service.

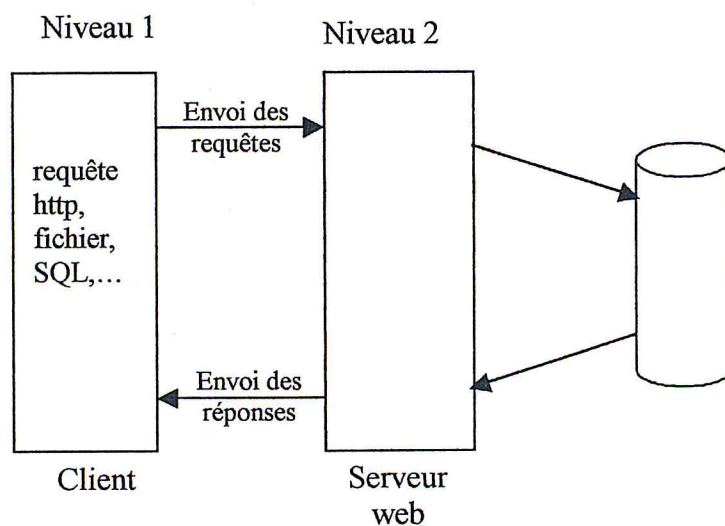


Figure IV-2: Architecture client-serveur à deux niveaux

Cette architecture est très limitée, elle nous permet pas d'un traitement parfait des données. Cette solution est parfaite si nos traitements se limitent à une simple validation des données, si le traitement est plus compliqué le client doit être plus complexe qui nous donne un client lourd ce qui fait que la machine cliente doit être plus puissante ce qui n'est pas toujours possible.

Pour résoudre ces problèmes on propose l'architecture trois-tiers.

6.2. Architecture à trois niveaux

Dans l'architecture à 3 niveaux (appelées *architecture 3-tier*), il existe un niveau intermédiaire, c'est-à-dire que l'on a généralement une architecture partagée entre:

1. Le client: le demandeur de ressources
2. Le serveur d'application (appelé aussi **middleware**): le serveur chargé de fournir la ressource mais faisant appel à un autre serveur.
3. Le serveur secondaire (généralement un serveur de base de données), fournissant un service au premier serveur.

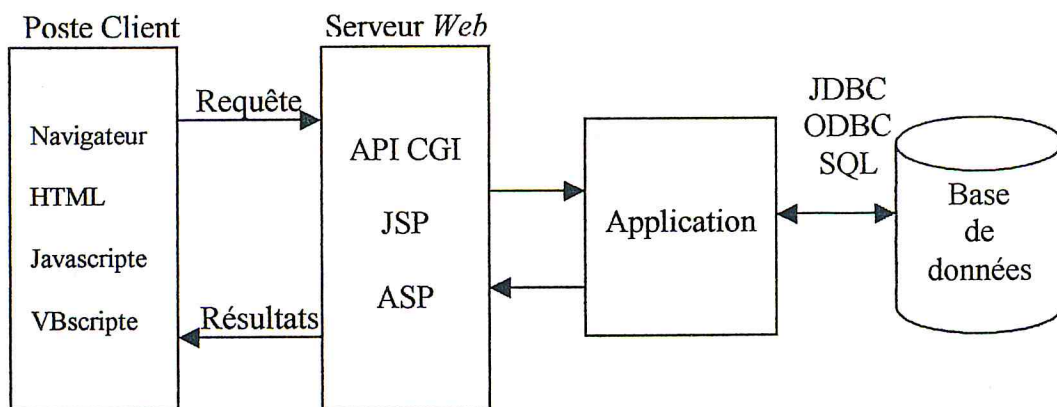


Figure IV-3: Architecture des applications à trois niveaux

Cette architecture assure :

- une connectivité à la base de données isolée : seul le serveur qui accède à la base de données, le client ne comunique que avec une application.
- Le traitement métier centralisés : le serveur d'application prend en charge les problèmes de traitement des données qui n'ont place dans les autres tiers.

6.2.1. Inconvénients de l'architecture trois-tiers

Même si l'architecture trois-tiers présente certains avantages importantes, elle n'est pas sans inconvénients. Le plus important est le niveau de complexité qu'elle ajoute à un système. Le système est composé d'un ensemble de composants distincts plus important, il sont donc plus nombreux à gérer.

Partie II : Les Servlets

7. Définition

Les *Servlets* présentent l'alternative de la technologie java à la programmation avec CGI (Common Gateway interface). Il s'agit de programmes exécutés sur un serveur Web, qui servent de couche intermédiaire entre une requête provenant d'un navigateur web ou un autre client http, et une base de données ou des applications serveur http[HEI 00].

La *Servlet* Java est une composante clé de développement Java côté serveur. Une *Servlet* est une petite extension intégrable à un serveur afin d'en étendre les fonctionnalités. Les *Servlets* permet au développeur d'étendre et de personnaliser tout serveur web ou tout serveur d'application supportant Java avec un degré de portabilité, de complexité et de facilité. Leur tâche est de :

1. Lire toutes les données envoyées par l'utilisateur : ces données sont typiquement saisies dans un formulaire sur une page Web, mais elles peuvent également provenir d'une applet Java ou d'un programme client HTTP particulier.
2. chercher d'autres informations sur la requête, à l'intérieur de cette requête http : Ces informations contiennent des détails sur les capacités du navigateur, sur les cookies, sur le nom de l'hôte du programme envoyant la requête, etc.
3. générer les résultats : Ce processus peut nécessiter une communication avec la base de données, en exécutant un appel RMI ou CORBA, ou en invoquant une ancienne application, ou encore en calculant directement la réponse.
4. Formater le résultat dans un document : Dans la plupart des cas, cela implique l'incorporation des informations dans une page HTML.
5. définir les paramètres de la réponse http appropriés : Cela signifie qu'il faut indiquer au navigateur le type de document renvoyé (c'est-à-dire HTML), définir les cookies, mémoriser les paramètres, ainsi que d'autres tâches.
6. Renvoyer le document au client : Ce document peut être envoyé au format texte (HTML), au format binaire (comme pour des images GIF) ou même dans un format compressé comme gzip, qui est en fait une couche venant recouvrir un autre format sous-jacent.

Les raisons expliquant la nécessité de construire des pages en temps réel :

- La page *web* est fondée sur des données envoyées par l'utilisateur.
- La page *web* est calculée à partir d'informations qui sont fréquemment modifiées.
- La page *web* se sert d'information provenant de bases de données appartenant à des entreprises ou à d'autres sources situées au niveau d'un serveur.

La *Servlet* possède néanmoins certaines particularités :

- Elle réside sur le réseau (pas forcément sur le serveur web).
- Elle est exécutée par le moteur de *Servlet* quand une URL est mappée dessus.
- Elle construit alors une page HTML dynamiquement et la renvoie au navigateur.

8. Avantages des *Servlets* par rapport aux CGI traditionnels

Les *Servlets* Java sont plus efficaces, plus simples à utiliser, plus puissantes, plus portables, plus sûrs, et moins chers que les CGI traditionnels et que plusieurs autres technologies comparables aux CGI [HEI 00].

8.1. Efficacité :

Avec un CGI traditionnel, un nouveau processus est exécuté pour chaque requête HTTP. Avec les *Servlets*, la machine virtuelle java est exécuté en permanence et traite chaque requête grâce à un thread Java plus simple, et non selon un processus complexe du système d'exploitation, s'il existe n requêtes simultanées demandant le même programme CGI, le code de programme sera chargé n fois dans la mémoire, avec une *Servlet*, il fondrait n *threads* mais uniquement une seule copie de la classe du *Servlet*.

8.2. Simplicité :

Les *Servlets* disposent d'une infrastructure complète pour

- Analyser,
- Décoder automatiquement les données d'un formulaire HTML,
- lire et définir des entêtes http, gérer les cookies,
- conserver une trace des sessions,
- ainsi que bien d'autres outils de haut niveau.

8.3. Puissance :

Les *Servlets* peuvent communiquer directement avec le serveur Web, ce qui est impossible pour des CGI classiques, du moins sans avoir recours à une API spécifique au serveur. Plusieurs *Servlets* peuvent également partager des données, ce qui permet de réunir facilement plusieurs connexions de basses de données et d'implémenter d'autres types d'optimisations fondées sur le partage de ressources. Les *Servlets* peuvent aussi conserver des informations d'une requête à l'autre, cette technique simplifie l'implémentation de l'enregistrement des sessions et la conservation en mémoire de calculs précédemment effectués.

8.4. Portabilité :

Les *Servlets* sont écrites en langage de programmation Java et respectent l'API standard. Par conséquent, une *Servlet* est écrite une seule fois, compilé un seul fois et exécuté partout sans modification. Les *Servlets* sont supportés soit directement soit par l'intermédiaire d'un composant logiciel fournit pour tous les principaux serveur web.

8.5. Sécurité :

Les *Servlets* supportent des pratiques de programmation sûre à plusieurs niveaux. Parce qu'elles sont écrites en Java et héritent donc du de typage forte de ce langage [JAS 98].

Les *Servlets* peuvent gérer les erreurs de façon sûre, grâce au mécanisme de gestion des exceptions de Java, qui évite la plantation du serveur en cas d'un erreur provoqué d'une opération illégale (Par exemple : division par zéro). Elles peuvent aussi garantir une communication sûre par l'utilisation de cryptage des données à envoyer.

8.6. Elégance :

Le code d'une *Servlet* est propre, orienté objet, modulaire et étonnamment simple. Une des raisons de cette simplicité réside dans l'API *Servlet* elle-même, qui inclut des méthodes et des classes pour gérer les opérations avancées, comme la gestion des cookies et des sessions, qui sont prises en charge par des classes utilitaires [JAS 98].

8.7. Economie :

Une fois que vous possédez un serveur web, quel qu'en soit le prix (gratuit ou cher), l'ajout des fonctionnalités permettant de supporter les *Servlets* est très économique, il se peut même que ce support soit directement intégré dans le serveur web.

Cette solution contraste avec les options offertes par le CGI, qui nécessitent un investissement initial non négligeable pour acheter des bibliothèques propriétaires.

Un inconvénient néanmoins, le recours à un élément supplémentaire, le moteur de *Servlet* ne supporté que par les serveurs dédiés au *Servlets*, alors que CGI est supporté nativement par les différents serveurs Web.

Néanmoins, il en existe plusieurs moteur de *Servlet*, tant propriétaires (Resin, Web Server), que libres (ApacheJServ, Tomcat). C'est TomCat qui a été choisi par Sun comme implémentation de référence des *Servlet* et JSP.

9. Architecture des *Servlets*

Une *Servlet* doit implémenter l'interface `javax.servlet.Servlet`, soit directement, soit via une classe l'implémentant déjà, comme `javax.servlet.http.HttpServlet`. Cette interface (ou plutôt ses implémentations) fournit un certain nombre de méthodes qui vont gérer la *Servlet* et la communication avec le client. Comme d'habitude, c'est en surchargeant ces méthodes que l'on va définir le comportement de la *Servlet*.

Lorsqu'une *Servlet* répond à une requête d'un client, elle reçoit deux objets :

- une `ServletRequest`, et
- une `ServletResponse`.

Chacun de ces deux objets encapsule une des deux parties de la communication (client-serveur et serveur-client).

- `ServletRequest` va donc permettre d'avoir accès aux paramètres de la connexion (type, origine, arguments, etc...), tandis que
- `ServletResponse` va permettre de définir la réponse (type, contenu, etc...).

Les versions spécifiques de *Servlet* fournissent un certain nombre de services supplémentaires. Ainsi, `HttpServlet` permet de gérer les sessions utilisateur.

10. Le cycle de vie d'une *Servlet*

Le cycle de vie d'une *Servlet* est assuré par le conteneur de *Servlet*. Ainsi afin d'être à même de fournir la requête à la *Servlet*, récupérer la réponse ou bien tout simplement démarrer/arrêter la *Servlet*, cette dernière doit posséder une interface (un ensemble de méthodes prédéfinies) afin de suivre le cycle de vie suivant :

1. Le serveur crée un pool de *threads* auxquels il va pouvoir affecter chaque requête.
2. La *Servlet* est chargée au démarrage du serveur ou lors de la première requête.
3. La *Servlet* est instanciée par le serveur.
4. La méthode `init()` est invoquée par le conteneur.
5. Lors de la première requête, le conteneur crée les objets `Request` et `Response` spécifiques à la requête.
6. La méthode `service()` est appelée à chaque requête dans une nouvelle thread. Les objets `Request` et `Response` lui sont passés en paramètre.

7. Grâce à l'objet *Request*, la méthode `service()` va pouvoir analyser les informations en provenance du client.
8. Grâce à l'objet *Response*, la méthode `service()` va fournir une réponse au client.
9. La méthode `destroy()` est appelée lors du déchargement de la *Servlet*, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La *Servlet* est alors signalée au garbage collector (communément francisé en ramasse-miettes).

Attention ! Mis à part `init()`, appelée une seule fois, toutes les autres méthodes doivent tenir compte des problèmes de concurrence, à cause du caractère multi-thread de l'exécution des servlets ! Ce mécanisme peut être bloqué en faisant implémenter l'interface `javax.servlet.SingleThreadModel` à la *Servlet*, néanmoins, ceci ne concerne que l'exécution de `service()`, `destroy()` doit donc toujours être impérativement thread-safe.

Exemple abstrait

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

public class SkeletonServlet implements Servlet {
    public void init(ServletConfig config) throws ServletException {
        ...
    }
    public void service(ServletRequest request, ServletResponse response) throws
    ServletException, IOException {
        ...
    }

    public void destroy() {
        ...
    }
}
```

11. Fonctionnement d'une *Servlet*

Le fonctionnement d'une *Servlet* suit les étapes suivantes

11.1. Initialisation de la *Servlet*

Au cours de son initialisation, la *Servlet* reçoit un objet `ServletConfig`, qui peut par exemple servir à récupérer le contexte d'exécution de la *Servlet* ou ses paramètres d'initialisation.

Si ces objets doivent être accédés ultérieurement, une référence doit obligatoirement en être conservée. Ceci étant fait automatiquement dans la méthode `init()` de `GenericServlet`, il convient

d'appeler celle-ci explicitement par `super.init()` si cette méthode est surchargée, ou alors d'établir manuellement cette référence.

L'initialisation réussie d'une servlet conditionne le traitement des requêtes émanant des clients. Si une ressource ne peut être accédée, il convient donc de lancer une exception `UnavailableException` plutôt que de tout bloquer.

11.2. Traitement

La classe `javax.servlet.http.HttpServlet`, dérivant de `javax.servlet.GenericServlet`, fournit les services de bases nécessaires à la gestion du dialogue HTTP, donc des appels Web. En particulier, sa méthode `init()` dispatche les requêtes qu'elle reçoit en fonction de leur nature :

- `doGet()` pour les requêtes GET, conditional GET et HEAD.
- `doPost()` pour les requêtes POST.
- `doPut()` pour les requêtes PUT.
- `doDelete()` pour les requêtes DELETE.

Ainsi, ce n'est pas la méthode `init()` qu'on doit surcharger, mais (au moins) un de ces méthodes, généralement `doGet()` ou `doPost()`.

11.2.1. Lecture des paramètres

L'objet `HttpServletRequest`, encapsulant la connexion du client au serveur, permet de recueillir les paramètres de la requête, de plusieurs façon :

- pour la méthode GET, on peut également parer manuellement la liste des arguments, que l'on obtient avec la méthode `getQueryString`.
- pour les méthodes POST, PUT, et DELETE, on peut recueillir le flux de données dans un `BufferedReader` retourné par la méthode `getReader()` pour du texte, ou dans un `ServletInputStream` retourné par la méthode `getInputStream()`.
- dans tous les cas, on peut utiliser les fonctions suivantes, qui offrent l'avantage de la généralité :
 - `getParameterNames()` retourne une énumération de tous les noms des paramètres.
 - `getParameterValues()` retourne un tableau de chaîne correspondant aux différentes valeurs d'un paramètre donné.
 - `getParameter()` retourne une chaîne correspondant à la valeur d'un paramètre donné. S'il y en a plusieurs, elle est choisie arbitrairement.

11.2.2. Les objets requêtes (Request) et réponses (Response):

Les méthodes `service()`, `doGet()` et `doPost()` ont deux paramètres : `HttpServletRequest`, et `HttpServletResponse`. Ces deux paramètres nous donnent le plein accès à toute l'information au sujet de la requête et nous laissent contrôler la sortie à envoyer au client comme réponse à la requête.

L'objet `HttpServletRequest` fournit l'information sur les variables d'environnement dans une manière standardisée. Il fournit des méthodes pour l'extraction des paramètres `http` à partir de la chaîne « `query` » ou du corps de la requête. Et cela dépendant de la requête (`GET` ou `POST`). Comme développeur de requête on accède aux paramètres de la même façon pour les deux types de requêtes.

Au lieu d'écrire la réponse sur `stdout` on obtient un `OutputStream` ou un `PrintWriter` à partir de la `HttpServletResponse`:

- Le `OutputStream` est destiné aux données binaires, comme les images.
- `PrintWriter` pour les sorties texte.

11.2.3. Ecriture du résultat

L'objet `doPost()``HttpServletResponse`, encapsulant la connexion du serveur au client, permet de construire la réponse en deux temps :

- écrire les entêtes `HTTP`, grâce aux différents accesseurs de la classe. Néanmoins, les valeurs par défaut sont suffisantes dans la plupart des cas, et seul le type de contenu est à faire explicitement, grâce à la méthode `setContentType()`.
- écrire le contenu, grâce à un `PrintWriter` retourné par `getWriter()` pour du texte, ou un `ServletOutputStream` retourné par `getOutputStream()` pour des données binaires.
-

Exemple (méthode GET) :

Voici un exemple simple d'une servlet qui générer une simple page web qui affiche
Exemple simple d'une Servlet :

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

public class ExempleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
```

```
// positionnement du type de contenu
response.setContentType("text/html");

// écriture du contenu
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("\t<HEAD>");
out.println("\t\t<TITLE>ExempleServlet</TITLE>");
out.println("\t</HEAD>");
out.println("\t<BODY>");
out.println("\t\t<H1 align=\"center\"> Exemple simple d'une Servlet </H1>");
out.println("\t</BODY>");
out.println("</HTML>");
out.close();
}
}
```

11.3. Destruction de la servlet

Il convient de libérer l'ensemble des ressources mobilisées par la servlet quand celle-ci est déchargée de la mémoire. Généralement, il s'agit de celles qui ont été initialisées au début du cycle de vie : connexions réseau, base de données, etc...

Attention néanmoins aux problèmes de concurrence : fermer une connexion alors qu'elle est en cours d'utilisation par un autre thread est une très mauvaise idée...

Partie III : JDBC (Java Data Base Connectivity)

12. Introduction au *JDBC*

En ce qui concerne la programmation avec les bases des données on se trouve actuellement en face de dizaines de produits de systèmes de gestion de base de données. Chaque gestionnaire de base de données (ou système de gestion de bases de données (SGBD)) communique avec l'application dans son propre langage. Donc si notre application devrait communiquer avec un nouveau moteur de base de données, nous serions obligés d'apprendre une nouvelle « langue ».

Mais avec Java, les choses sont bien différentes, nous devrions « écrire une fois, compiler une fois et exécuter partout », ce qui l'est aussi pour la programmation avec les bases de données. L'API JDBC de Java fournit aux applications Java l'accès à la plupart des systèmes de bases de données par l'intermédiaire du langage SQL.

Le JDBC inclut un ensemble de classes qui peuvent traiter la plupart des besoins d'accès aux bases de données, et cela de simple SELECT à des traitements des résultats des procédures stockées, jusqu'à accéder à plus d'une base de données en même temps.

Dans ce chapitre nous allons tout d'abord vous familiariser avec le JDBC. Nous allons vous montrer ensuite comment ouvrir une connexion avec votre SGBD, puis, tout ce que fait JDBC, comme comment envoyer du code SQL à votre SGBD. En vue de cela, nous utiliserons et expliquerons quelques instructions SQL. Après cela, nous vous montrerons avec quelle facilité d'utilisation JDBC passe ces instructions SQL à votre SGBD et comment on pourrait récupérer des résultats.

13. Qu'est-ce que JDBC ?

Pour rendre les bases de données familières au programmeur Java Sun a développé une API pour l'accès aux bases de données.

L'API JDBC définit des classes Java pour représenter:

- des connexions aux bases de données,
- des statements SQL,
- des resultsets,
- des metadata bases de données,

Le JDBC assure les trois points suivant :

- Le JDBC doit être une API de niveau SQL : Autrement dit, on utilise SQL comme langage de manipulation des bases de données.
- Le JDBC doit exploiter les réussites et l'expérience acquises avec les autres APIs de bases de données existantes.
- Le JDBC doit être simple : cela devrait aboutir à la lisibilité du code des applications accédant à une base de données, et à la simplicité de manipulation. Les tâches les plus complexes sont gérées par des interfaces fournies par le JDBC.

L'avantage majeur du JDBC reste néanmoins son indépendance vis à vis de la base de données. Avec une application écrite pour accéder à une base de données sous Oracle, et avec une modification de deux ou trois lignes de code, on pourrait facilement accéder à une base de données avec un autre moteur de base de données.

14. Architecture du JDBC :

JDBC est un ensemble d'interfaces Java, chacune implémentée de façon différente par chaque fournisseur. L'ensemble des classes qui implémentent ces interfaces, et assure la

communication à un moteur de base de données spécifique est appelé un *pilote JDBC*. Donc pour chaque moteur de base de données spécifique il faut son propre pilote.

Pendant la construction d'une application de bases de données on n'a pas besoin de connaître la spécificité de cette dernière car le JDBC la cache et nous laisse préoccupé uniquement de notre application.

La Figure 1 représente un diagramme de classes UML pour les classes et les interfaces JDBC de base.

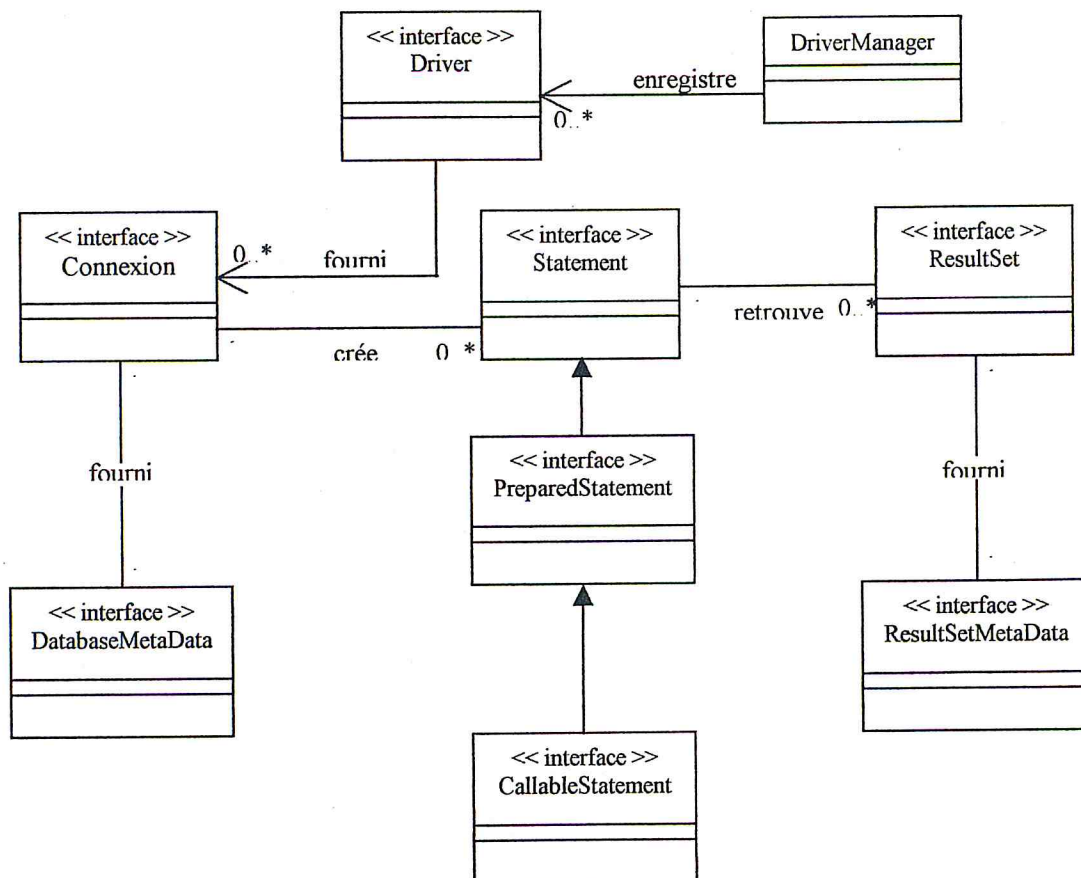


Figure IV-4: Diagramme de classes des interfaces de JDBC

15. Le pilote et la base de données :

Une fois le moteur de base de données installé, et la base de données créée, on a besoin d'un pilote JDBC pour ce moteur. La plupart des pilotes JDBC sont disponibles gratuitement sur le réseau, il suffit de les télécharger et de les installer sur votre machine virtuelle Java.

L'installation se fait par la décompression de fichier téléchargé dans le sous-répertoire `\jre\lib\ext\`, où est le répertoire d'installation de Java.

Les moteurs de bases de données commerciales, comme Oracle, ont des pilotes JDBC payants.

Les différents pilotes JDBC sont écrits dans des styles différents.

Pour les classifier Sun a défini le système de classification des pilotes représenté par la figure 2.

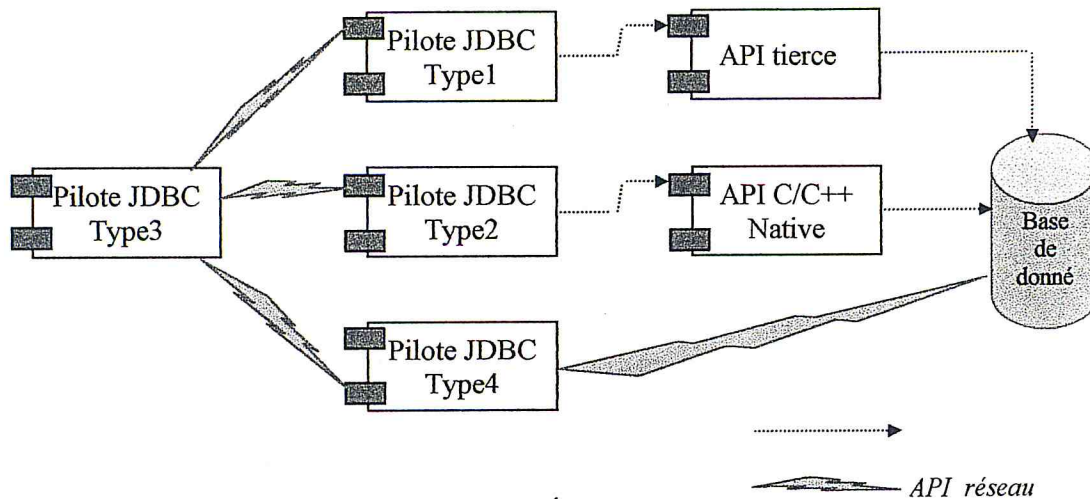


Figure IV-5 : Classification des pilotes du JDBC.

Type 1 :

Ces pilotes utilisent une technologie de pont pour accéder à la base de données. En trouve un bon exemple de se type c'est le pont JDBC-ODBC livrée avec le JDK1.2. il fournit une passerelle à l'API ODBC pour accéder à les base de données supportées par ODBC. Les solutions de pont nécessitent généralement l'installation d'un logiciel sur les systèmes client, dans le cas du pont JDBC-ODBC en a besoin d'installer l'ODBC sur la machine cliente. Elles ne sont donc pas adaptées aux applications qui ne nous permettent pas d'installer un logiciel sur le client.

Type 2 :

Les pilotes de ce type sont des pilotes natifs. Autrement dit ces pilotes contiennent du code Java qui appelle des méthodes en C ou en C++ implémentées par chaque fournisseur de bases de données et réalisant les accès à la base. Cette approche impose aussi l'installation d'un logiciel sur le système client.

Type 3 :

Les pilotes de type 3 donnent au client une API réseau générique qui est ensuite traduite en accès spécifiques à la base de données sur le serveur. Autrement dit, le pilote JDBC sur le client utilise des sockets pour appeler une application intermédiaire sur le serveur qui traduit les

requêtes du client en une API spécifique au pilote voulu. Ce type de pilote est extrêmement souple, puisqu'il ne demande l'installation d'aucun code sur le client et un seul pilote peut en fait fournir l'accès à de multiples bases de données.

Type 4 :

En utilisant les protocoles réseau intégrés au moteur de bases de données, les pilotes de type 4 conversent directement avec la base de données via des sockets Java. C'est la solution purement Java la plus directe. Ce type proviendra le plus souvent uniquement du fournisseur de base de données.

16. Fonctionnement de JDBC

Les parties majeures du fonctionnement du JDBC sont :

- Le chargement du pilote,
- la définition de l'URL de connexion,
- la connexion à la base de données,
- l'exécution de quelques SQL, et
- la récupération des résultats.

16.1. Chargement du Pilote :

La première chose qu'il faut faire dans une application qui utilise JDBC est le chargement du pilote JDBC pour que notre application sache parler au serveur de bases de données actuel, pour le faire, il suffit de charger une statique classe sans création d'une instance de celle-ci par l'instruction `Class.forName()`. Cette méthode prend comme paramètre une chaîne de caractères, pour notre cas c'est le nom de la classe du pilote. Vous devez obtenir un pilote JDBC spécifique à la base de données qui vous utilise, vous aurez besoin de vérifier sa documentation pour obtenir le nom de la classe pour l'utiliser.

16.2. Définition l'URL de connexion :

Pour se connecter à une base de données nous avons besoin de spécifier l'emplacement du serveur de la base de données par une *URL*, cette dernière est de la forme *jdbc:driver:database*. Dans certains cas on a besoin d'ajouter le *host* et le *port* du serveur à l'URL. Pour chaque base de données on a une URL spécifique, par exemple le pilote *JDBC-Thin* d'*Oracle* utilise une URL de la forme *jdbc:oracle:thin:@site:port:databasename*.

16.3. Etablissement de la connexion :

Ceux qui correspondent à l'URL fournie. S'il n'en trouve aucun, il lève une Exception. En utilisant l'URL de notre base de données et les propriétés nécessaires à l'emplacement de notre pilote JDBC (généralement un ID d'utilisateur et un mot de passe), l'application va commencer par demander le *DriverManager* à une connexion. Le *DriverManager* va ensuite rechercher parmi tous les pilotes qui sont chargés

Quand un Driver reconnaît notre URL, il crée une connexion à la base de données en utilisant les propriétés données. Le *DriverManager* retourne ensuite un objet *Connection*. Le processus complet de chargement d'un pilote (Driver), défini par l'URL de l'établissement d'une connexion peut être donné comme suit :

```
String userID = "redha";
String passwd = "labobila";
// définition de l'URL de connexion:
String URL = "jdbc:mysql://localhost/lrdsi";
// chargement du pilote:
Class.forName("com.mysql.jdbc.Driver");
// établissement de la connexion:
Connection con = DriverManager.getConnection(URL, userID, passwd);
```

Problèmes de Connexion :

Si vous avez un problème dans l'établissement de la connexion, regardez s'ils ne fait pas partie des suivants :

- La connexion échoue avec le message *Class not found*. Ce message est souvent obtenu quand le pilote JDBC ne se trouve pas dans le *CLASSPATH*. Vous devez simplement mettre explicitement le fichier *.zip* ou *.jar* dans un *CLASSPATH*.
- La connexion échoue avec le message *Driver not found*. Vous n'avez pas chargé votre pilote JDBC correctement, vérifiez le nom du pilote.

Une application utilisant le JDBC peut parler à une ou plusieurs bases de données sans connaître les détails d'implémentation des pilotes.

L'application utilise le JDBC comme une interface à travers laquelle elle passe ses requêtes.

La figure 3 montre la connexion avec plusieurs pilotes.

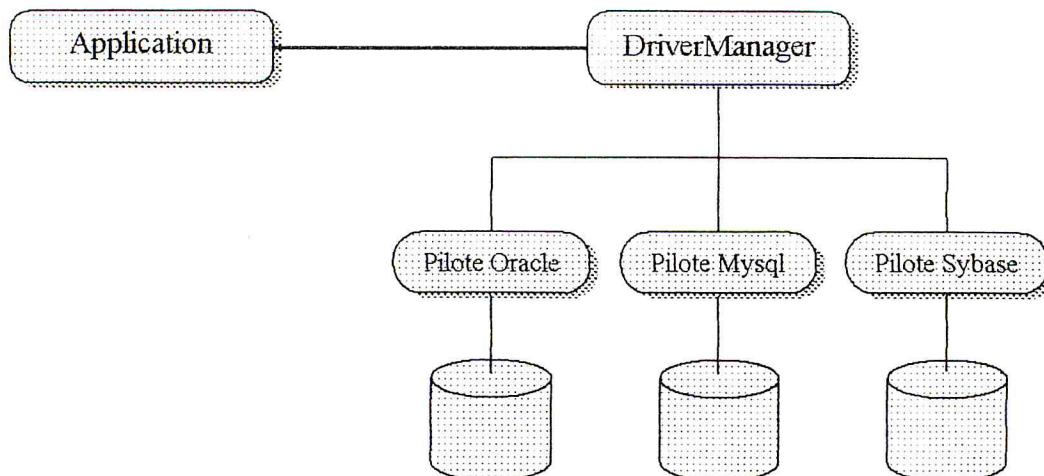


Figure IV-6: JDBC cache à l'application les spécificités d'implémentation de chaque base de données

16.4. Création d'un Statement :

Maintenant que nous sommes connectés à la base de données, nous pouvons commencer à effectuer des mises à jour et des requêtes SQL. Cela est fait par l'objet *Statement*, que nous utilisons pour envoyer des requêtes ou commandes à la base de données.

Il y a trois genres des Statement dans le JDBC:

- le *Statement* : Représente l'instruction SQL de base.
- le *PreparedStatement* : Représente une instruction SQL pré-compilé qui peut offrir une performance améliorée.
- le *CallableStatement* : Permet aux applications JDBC d'utiliser des procédures stockées.

Pour obtenir un objet *Statement*, appelez la méthode `createStatement()` de l'objet *Connection* comme suit :

```
Statement stm = con.createStatement();
```

Une fois vous créez un *Statement*, utilisez-le pour exécuter des instructions SQL.

Une *Statement* peut être une requête qui rend des résultats ou une opération de mise à jour qui manipule la base de données dans certaine manière.

Pour obtenir les résultats, utilisez:

- la méthode `executeQuery()` pour des requêtes select ou Cette méthode retourne un *ResultSet* qui reçoit le résultat de la requête qui a été exécutée,

- `executeUpdate()` pour des mises à jour (`INSERT` , `UPDATE` , `DELETE`,...). Cette méthode retourne un `int` indiquant le nombre des lignes qui ont été changées dans la base de données.

```
ResultSet rs = stmt.executeQuery( Requête SQL );
int lignes = stmt.executeUpdate( Mise à jour SQL );
```

Si vous ne savez pas si une instruction SQL va rendre des résultats (quand l'instruction SQL crée dynamiquement), vous pouvez utiliser la méthode `execute()` de `Statement` qui retourne un `boolean`. Cette méthode retourne `true` s'il y a un résultat associé à l'instruction. Dans ce cas le `ResultSet` peut être retrouvé en utilisant la méthode `getResultSet()`, et le nombre des lignes mis à jour peuvent être retrouvé en utilisant la méthode `getUpdateCount()`:

```
Statement SQLInconnu = con.createStatement( );
if(SQLInconnu.execute( stringsql ) ) {
    //affiche les résultats
    ResultSet rs = SQLInconnu.getResultSet( );
}
else {
    System.out.println("Lignes mises à jour :"+
        SQLInconnu.getUpdateCount());
}
```

Remarques :

1. Un appel de `executeQuery()`, `executeUpdate()` ou `execute()` implicitement ferme toutes les `ResultSet`s actifs associés à la `Statement`. Si votre application a besoin d'exécuter plus qu'une requête simultanée, vous avez besoin d'utiliser des multiples `PreparedStatement`.
2. Le traitement des résultats multiples d'une instruction SQL support par l'utilisation de la méthode `getMoreResults()`. L'appel de cette méthode ferme implicitement tous les `ResultSet` existants et passe aux prochains résultats de cette instruction. La méthode `getMoreResults()` retourne `True` s'il y a d'autre `ResultSet` disponibles à retrouver par `getResultSet()`. Pour être sûr que vous avez traité tous les résultats, il faut vérifier que `PreparedStatement.getMoreResults()` retourne `False` et que `PreparedStatement.getUpdateCount()` rende `-1`.

16.5. SQL préparé :

Les bases de données fournissent deux sortes de SQL préparé :

- les instructions préparées, et

- les procédures stockées.

L'avantage du SQL préparé par rapport aux SQL simples vues c'est qu'une base de données peut prendre le code SQL par avance et créer un plan de requête pendant que nous nous occupons d'une autre partie de l'application.

Cela signifie que notre code SQL va s'exécuter plus vite et que nous disposons d'une référence générique à une instruction que nous pouvons réutiliser au lieu de recréer à chaque fois de nouvelles instructions SQL pour chaque nouvel accès à la base de données.

16.5.1. Instructions préparées :

L'interface `PreparedStatement` étend l'interface `Statement`.

Elle permet de définir une instruction SQL contenant des paramètres, comme dans la définition d'une fonction, que nous pouvons exécuter plusieurs fois en changeant la valeur des paramètres. On peut utiliser une instruction préparée pour la mise à jour d'un groupe des lignes dans la même table. Par exemple si nous mettons à jour plusieurs notes des étudiants, nous pouvons utiliser une boucle :

```
Statement stmt = con.createStatement( );
for(int i = 0; i < etudiants.length; i++)
stmt.executeUpdate(" UPDATE etudiant " +
    " SET note = " + etudiants[ i ].getNote( ) " +
    " WHERE matricule = " + etudiants[ i ].getMat( ) );
con.commit( );
stmt.close( );
```

Cette instruction crée le même plan à chaque itération de la boucle.

Au lieu d'appeler plusieurs fois la même instruction avec des entrées différentes, nous pouvons utiliser un `PreparedStatement` :

```
PreparedStatement pstmt = con.prepareStatement(" UPDATE etudiant " +
    " SET note = ? " +
    " WHERE matricule = ? " );
for(int i = 0; i < etudiants.length; i++){
    pstmt.setFloat(1, etudiants.getnote( ));
    pstmt.setInt(1, etudiants.getmat( ));
    pstmt.execute( );
    pstmt.clearParameters( );
}
con.commit( );
pstmt.close( );
```

Lorsque l'objet `PreparedStatement` est créé à l'aide de la méthode `prepareStatement()` le code SQL a été envoyé à la base de données. On exécute cette instruction préparée plusieurs fois

dans la boucle `for()`, mais on construit le plan de requête une seule fois. Afin de faire rentrer les valeurs des paramètres de l'instruction, `PreparedStatement` fournit les méthodes `setXXX(int index, TypeXXX valeur)` pour spécifier types différents des paramètres (comme `setInt()`, `setFloat()`, `setString(),...`), cette méthode a deux paramètres, la première représente l'index du paramètre et la deuxième représente la valeur affecté à la paramètre. `setXXX()` lient les paramètres de la gauche vers la droite dans l'ordre où en les a placées dans l'instruction préparée, donc le « 1 » dans le premier paramètre de `setXXX()` associé à la première « ? » dans l'instruction préparée.

16.6. Traitement des résultats :

Le JDBC rend des résultats dans un objet `ResultSet` sous une forme qui peut être accédée par l'application, l'objet `ResultSet` représente des lignes de données qui se retournés par une requête. Si la requête n'a pas de résultats l'objet `ResultSet` ne représente aucune lignes. Pour traiter les résultats, on le fait par ligne, et pour faire passer ce qu'on appelle un curseur d'une ligne à l'autre on utilise la méthode `next()` de `ResultSet` qui retourne `true` s'il y a encore des lignes ou `false` sinon. Initialement le curseur est placé juste avant la première ligne. Le premier appel a la méthode `next()` déplace le curseur à la première ligne. Pour retrouver les valeurs des colonnes du résultat, nous utilisons la méthode `PreparedStatement.getXXX()` qui prend le nom de la colonne (dans la table à la base de données) ou son index (dans le résultat) comme paramètre et rend le résultat comme une variété des différents types Java. Par exemple nous utilisons `getInt()` si la valeur devrait être un `int`, `getString()` pour une chaîne de caractères. Ces méthodes sont toutes de la forme : `type getType(int | String)` où `int` pour le numéro de la colonne ou bien `String` pour son nom.

```

ResultSet rs = stmt.executeQuery( "SELECT nom , prenom FROM etudiant" );
While( rs.next( ) ){
    System.out.println( "Nom:" + rs.getString("nom") );
                        // ou rs.getString(1)

    System.out.println( "Prenom:" + rs.getString("prenom") );
                        // ou rs.getString(2)
}

```

Remarques :

1. Si vous utilisez l'index de la colonne comme un argument de la méthode `getXXX()` note que la première colonne dans une ligne de `ResultSet` a l'index 1, pas 0

2. Il est possible de retrouver chacun des types SQL de base avec la méthode `getString()` si vous voulez seulement afficher les résultats.
3. Vous pouvez utiliser la méthode `findColumn` ("Nom du colonne") pour obtenir l'index de la colonne nommée comme ici:

```
String nom ;
nom = results.getString( results.findColumn( "Nom" ) );
```

4. La méthode `getMetaData()` de `ResultSet()` rend un objet `ResultSetMetaData` qui vous permet de déterminer le nombre, les noms et les types des colonnes dans le `ResultSet` à travers les méthode `getColumnCont()` qui donne le nombre des colonnes dans la résultat, `getColumnName(int index)` (l'index du colonne commencer à 1) qui donne le nom du colonne associé a cet index dans le resultat et `getColumnTypeName(int index)` qui donne le nom du type de la colonne associée a cet index. Ces méthodes permettent d'être capable de traiter proprement les tables.

16.7. Fermeture de la connexion :

A la fin d'une application JDBC il faudrait libérer toutes les *Connexions*, *Statements*, et *ResultSets* qui ont été utilisés dans l'application au moyen de l'appel de la méthode `Close()`.

17. Conclusion

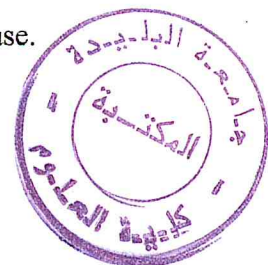
Dans ce chapitre nous vous avons exposé les principales parties d'une application serveur basé sur Java. Premièrement nous avons présenté les application *Web* et nous avons vu que les application trois-tiers répondent de plus en plus à ses dernières

Dans la deuxième partie nous avons présenté l'un des principaux module de Java côté serveur qui consiste en la partie *Servlets*. Ceux sont ces *Servlets* que nous avons utilisé pour développer des application serveur très évolué et compatible avec le reste de projet.

Et en dernière nous vous avons exposé, avec certains détails, le *JDBC* de base.

Vous avez vu comment:

- créer des tables,
- insérer des valeurs à dedans,
- faire des requêtes,
- accéder à un résultat, et
- mettre à jour une table.



Dans le but d'être clairs nous avons utilisé des requêtes très simples dans nos exemples.

Ce chapitre pourrait servir à suivre les détails de la partie serveur dans le chapitre suivant, y compris les parties JDBC et bases de données.

Partie II : Développement de l'application

Chapitre V Conception et implémentation

1. Introduction

Le plus grand nombre des MIDlets actuelles s'inscrit dans le cadre des jeux ou des petits utilitaires tels que les agendas ou convertisseurs d'unités, ce n'est de loin pas le principal intérêt de cette technologie. Avec le standard MIDP 2.0, les opportunités de créer des services mobiles dans des domaines très variés sont grandes. Ainsi, on peut développer des applications de m-commerce (m pour mobile) d'achats en ligne et de consultation de la bourse, d'enchères ou de d'informations en temps réel. C'est justement ce genre de domaines qui présente un grand intérêt.

Dans notre projet nous développons une application client-serveur basée sur une MIDlet et une *Servlet*. La conception d'une telle application passe par toutes les étapes de celles du développement de logiciels.

Comme tout le monde le dit [MUL 97], [SOM 83] le développement de logiciels est difficile à mener à bien, et le développement orientée objets est encore bien plus difficile.

Un certain nombre de méthodes de développement de logiciels ont été définies, ces méthodes permettent de mieux organiser, d'avoir une meilleure compréhension, de réduire la complexité des applications, et permettent une de réduire la complexité de l'interprétation des concepts logiciels.

Une méthode de développement de logiciel est définie pour représenter le processus de développement, elle comprend [MUL97] :

- Des éléments de modélisation qui sont les briques conceptuelles de base.
- Une notation dont l'objectif est d'assurer le rendu visuel des éléments de modélisation.
- Un processus qui décrit les étapes à suivre lors du développement du système.

La notation unifiée UML que nous avons utilisée est basée sur les méthodes d'analyse et de conception : BOOCH, OMT et OOSE, de ce fait, elle permet de couvrir le cycle de vie d'un logiciel depuis l'analyse du besoin jusqu'au codage. Cette notation est d'une très grande richesse.

Elle permet de couvrir à peu près toutes les phases du développement :

- les besoins des utilisateurs du futur système, exprimés à l'aide de cas d'utilisation

- la spécification complète du système, sous forme de diagrammes décrivant les parties statiques et dynamiques du système.
- la conception détaillée, jusqu'à un niveau très proche du code en langage objet de l'implémentation.
- les suites de tests permettant de s'assurer qu'une implémentation candidate est effectivement conforme à la spécification élaborée en phase amont, sous forme de diagrammes de séquences.

2. Spécification des besoins

La spécification des besoins est une étape essentielle au début du processus de développement. Son but est d'éviter de développer un logiciel non adéquat.

Cette étape a pour objectif de répondre à des questions de type :

- « quelles sont les fonctions du système ? »,
- « Les utilisateurs du système qui sont-ils ? »
- « Qu'attendent-ils du système ? ».

Ces questions doivent trouver la réponse à la fin de cette étape par l'étude du comportement du système comme des cas d'utilisations, le contexte du système les acteurs et les scénarios.

2.1. Les Cas d'utilisation

L'analyse détermine-le quoi faire, c'est à dire les besoins de l'utilisateur. L'expérience montre que la technique des cas d'utilisation (use cases) se prête bien à la détermination des besoins d'utilisateurs [MUL97].

L'étude des cas d'utilisation commence par la détermination des acteurs du système.

2.2 Les Acteurs

Un acteur représente un rôle joué par une personne ou par une chose qui interagit avec le système.

Par définition les acteurs sont à l'extérieur du système.

Mon application est destinée aux étudiants, ce qui implique qu'on a un premier acteur qui est l' Etudiant.

Et comme cette application implique aussi le côté administration pour la consultation et la réponse par exemple aux messages des étudiants, on a un second acteur qui est l(Administrateur .

2.3. Descriptions textuelles des cas d'utilisation

L'utilisation de notre application est très simple et limitée, elle est exprimée par les cas d'utilisation suivant :

- Inscription au service : pour que l'étudiant puisse utiliser notre *MIDlet*, il devrait préalablement être inscrit dans ce service.
Pour le faire, il suffit qu'il saisisse son matricule, son nom et prénom, et les envoyer au serveur.
- La consultation des notes : Quand l'étudiant est inscrit au service, il pourrait consulter ses notes à partir de la base de données de l'institut.
- L'envoi de messages à l'administration : L'étudiant peut également envoyer des messages à l'administration, comme des recours, des demandes,
- Consultation de messages : L'administrateur peut consulter les messages qui avaient été envoyé par les étudiants.

2.4. Diagramme de cas d'utilisation

Les différentes fonctionnalités offertes par notre application forment ainsi un ensemble de *cas d'utilisation* ("Use Case"), présentés dans la section précédente.

Afin de les formaliser, nous avons utilisé les diagrammes de cas d'utilisation proposés par UML.

La figure suivante illustre ce diagramme.

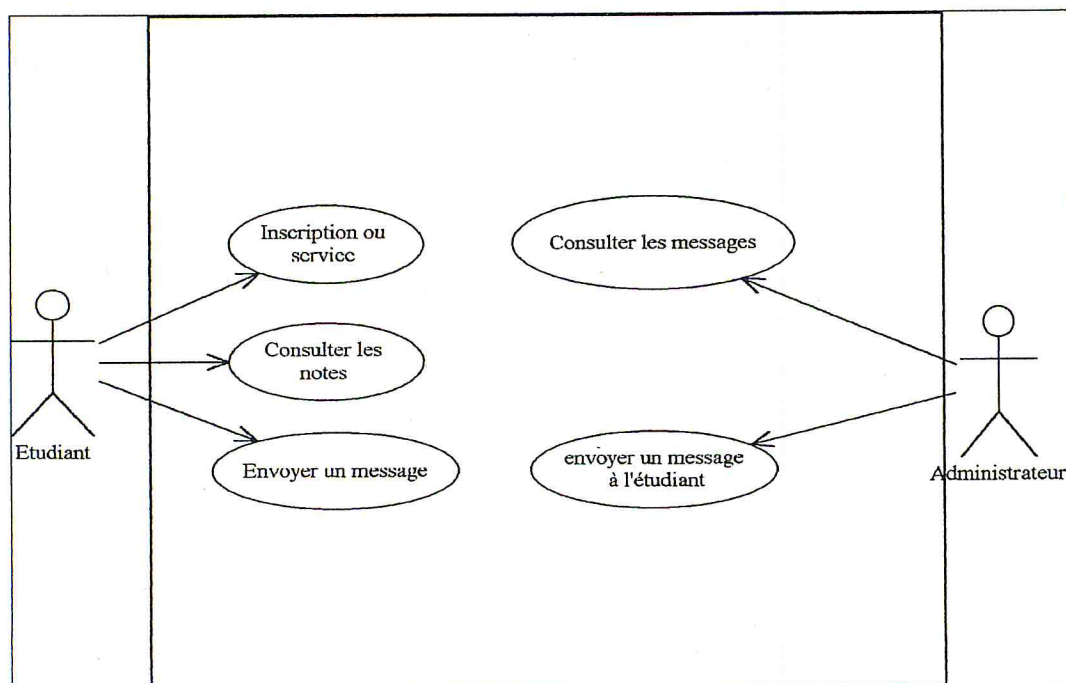


Figure V-1: Diagramme de cas d'utilisation

2.5. Diagramme de séquence

Les cas d'utilisation de UML ont certes l'avantage d'être graphiquement très simples et donc faciles à appréhender.

Malheureusement, cette simplicité ne va pas sans une certaine pauvreté sémantique [MUL97], cependant les diagrammes de séquence nous permettent de bien schématiser les scénarios des cas d'utilisation en montrant les interactions entre plusieurs objets.

a. Diagramme de séquence pour l'inscription au service.

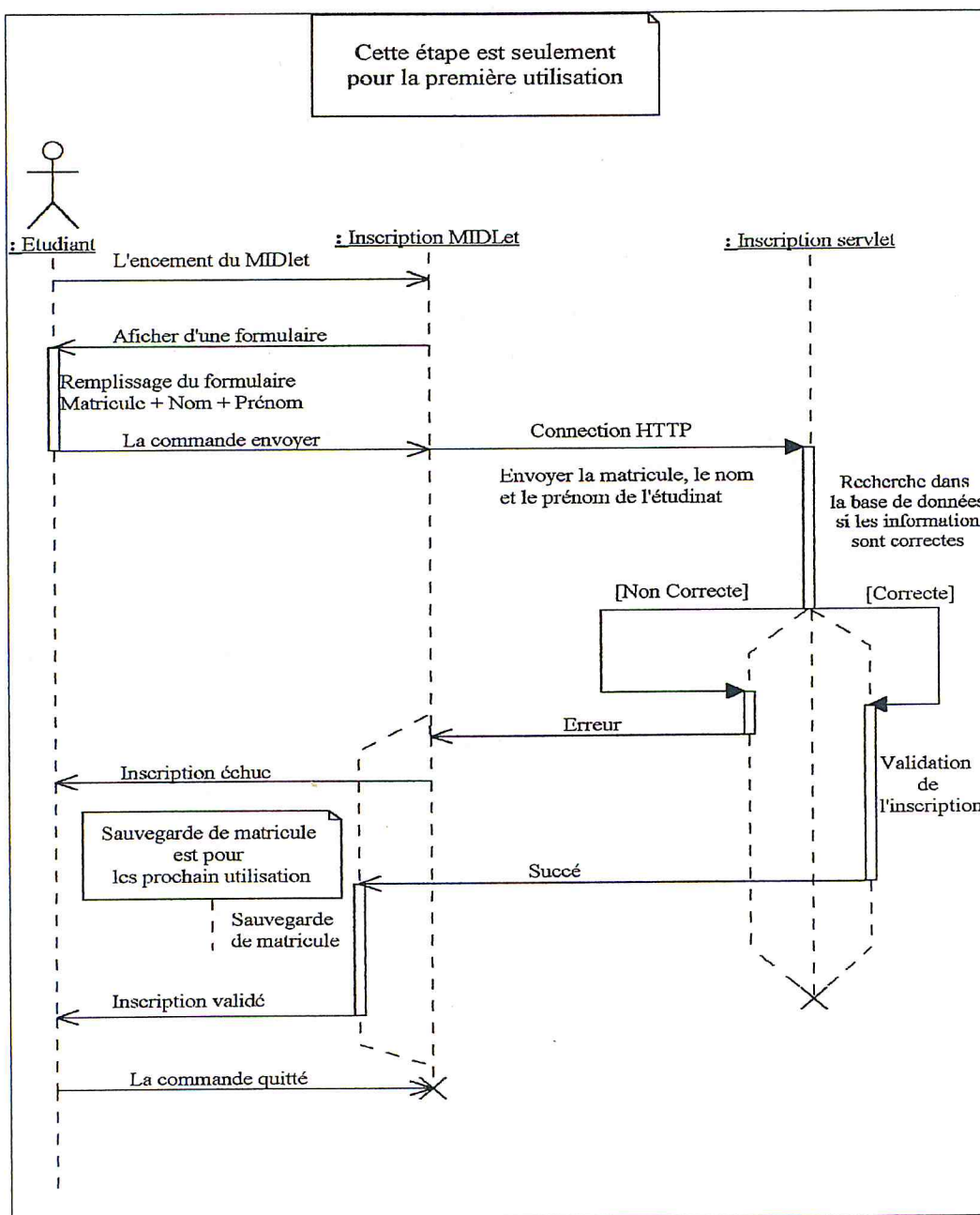


Figure V-2: Diagramme de séquence de cas d'inscription au service

Scénario :

1. Lors de la première utilisation la *MIDlet* affiche un formulaire.
2. L'étudiant remplit ce formulaire par en enregistrant son matricule, son nom et son prénom.
3. Ensuite l'étudiant appuie sur la commande *envoyer*.
4. La *MIDlet* se connecte alors à la *Servlet*, elle va donc la lancer, avec les données du formulaire comme arguments de la requête à la *Servlet*.
5. La *Servlet* utilise ses données pour effectuer une recherche à partir de la table étudiant de la base de données de la scolarité.
6. Si les données existent :
 - a) Alors valider l'inscription est validée et un message de succès est envoyé à la *MIDlet*.
 7. a) la *MIDlet* sauvegarde profite pour sauvegarder le matricule pour les prochaines utilisations, et envoie un message de succès à l'utilisateur.
 - b) Sinon la *Servlet* envoie un message d'erreur à la *MIDlet*.
 7. b) la *MIDlet* affiche le message d'erreur.
8. A la fin l'étudiant peut arrêter le *MIDlet* en appuyant sur la commande *quitter*.

b. Diagramme de séquence pour la consultation des notes :

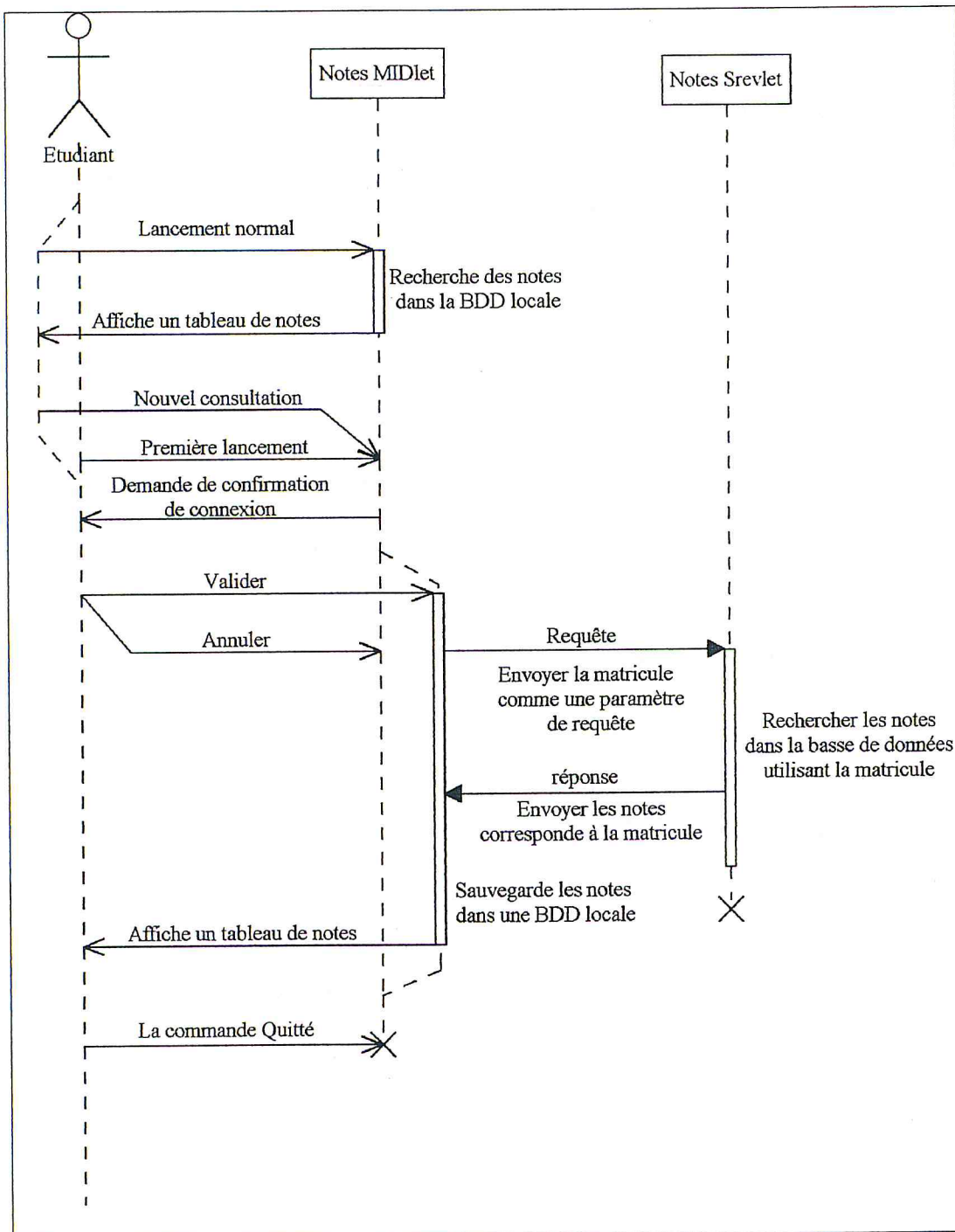


Figure V-3: Diagramme de séquence pour le cas de consultation des notes

Scénario :

1. Quand l'étudiant lance la consultation des notes la première fois il doit forcément se connecter au serveur pour charger ses notes à sa *MIDlet*.
S'il avait déjà effectué une consultation auparavant, la *MIDlet* lui affiche les notes enregistrées à la base de données locale de la *MIDlet* et leur date d'enregistrement.

2. l'étudiant peut aussi faire une nouvelle consultation à partir de la base de données de la scolarité qui nécessite une mise à jour à la base de données locale.
3. l'étudiant peut quitter l'application en appuyons à la commande Quitter.

Remarque : La *Servlet* utilise la matricule de l'étudiant qui a été envoyé par le *MIDlet* (qui est déjà sauvegardé lors de l'inscription) pour sélectionner les notes à partir de la base de données.

c. diagramme de séquence pour le cas d'envoi de message :

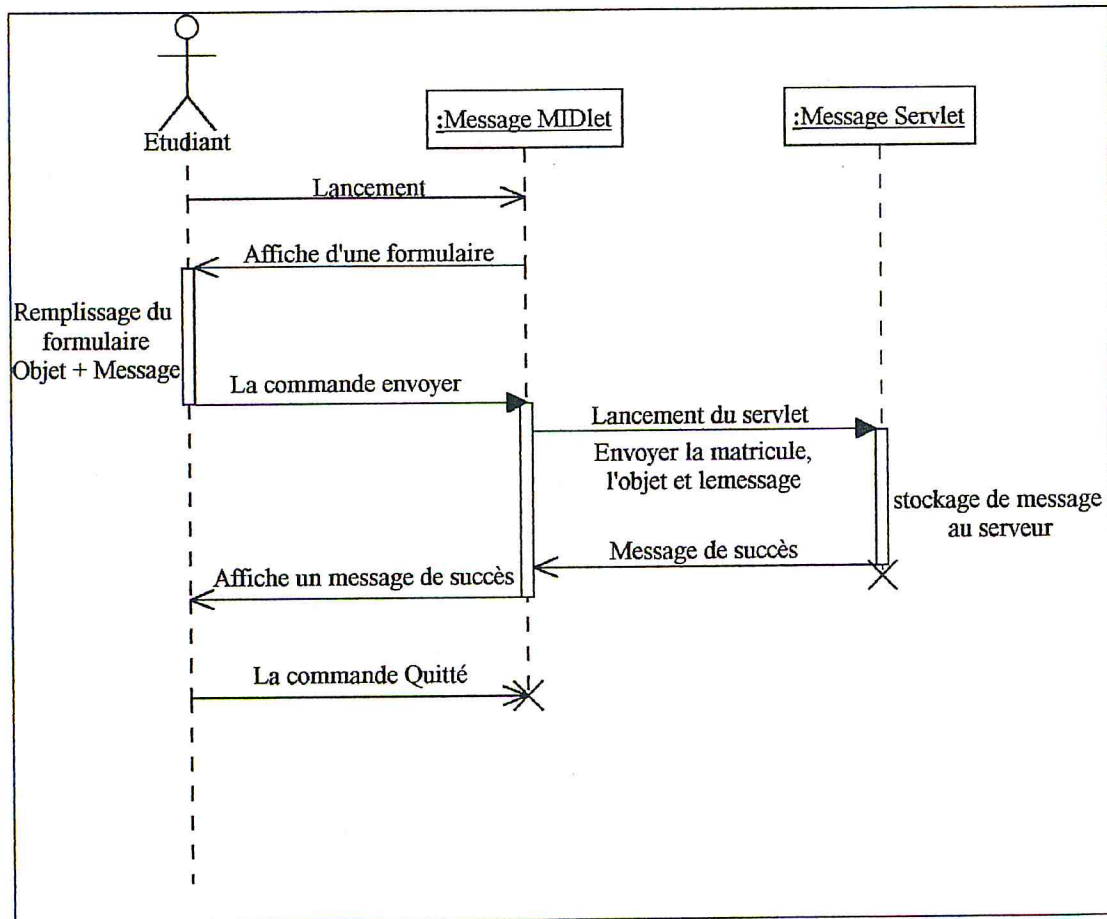
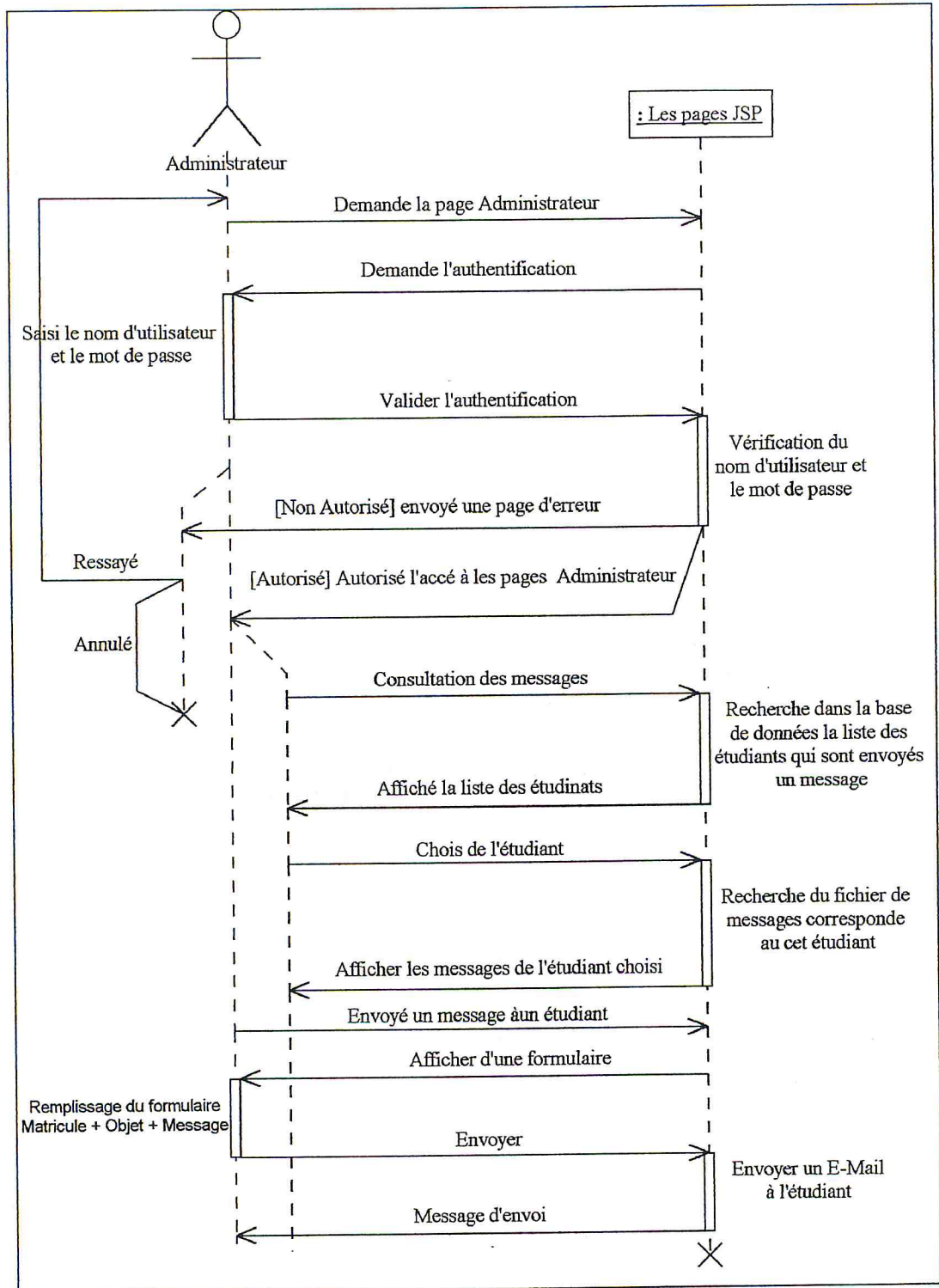


Figure V-4: Diagramme de séquence pour le cas envoi de message

Scénario :

1. Au lancement de l'application l'étudiant remplit un formulaire (Objet + Message).
2. Quand l'étudiant envoie le formulaire, la *MIDlet* fait une connexion au serveur et envoie la matricule, l'objet et le message.
3. Dans le serveur les messages sont stockés dans des fichiers.

d. Diagramme de séquence pour l'administrateur :



e.

Figure V-5: Diagramme de séquence pour l'Administrateur

Scénario :

1. Quand l'administrateur veut accéder aux pages Administrateur, le serveur envoie une page d'authentification.

2. L'administrateur saisit son nom d'utilisateur et son mot de passe et valide l'authentification.
3. Le serveur vérifie le nom et le mot de passe et autorise l'accès si ces derniers sont corrects.
4. Quand l'administrateur est autorisé, il peut faire une consultation des messages et pourrait aussi envoyer un message à l'étudiant.
5. Si l'administrateur voudrait consulter les messages, le serveur va lui afficher la liste des étudiants ayant envoyé des messages.
6. L'administrateur choisit un étudiant.
7. Le serveur affiche alors les messages de l'étudiant choisi.
8. Si l'administrateur veut envoyer un message à l'étudiant le serveur va lui afficher un formulaire de E-Mail pour qu'il le remplisse et l'envoyer.

2.6. Diagramme de collaboration

Les fonctionnalités décrites par les cas d'utilisation sont réalisées par des collaborations d'objets du domaine [MUL97], d'où il est envisageable, d'employer les diagrammes de collaborations bien que ce dernier ne soit qu'une variante des diagrammes de séquences et exprime de ce fait la même sémantique.

J'ai choisi comme un exemple la consultation des notes

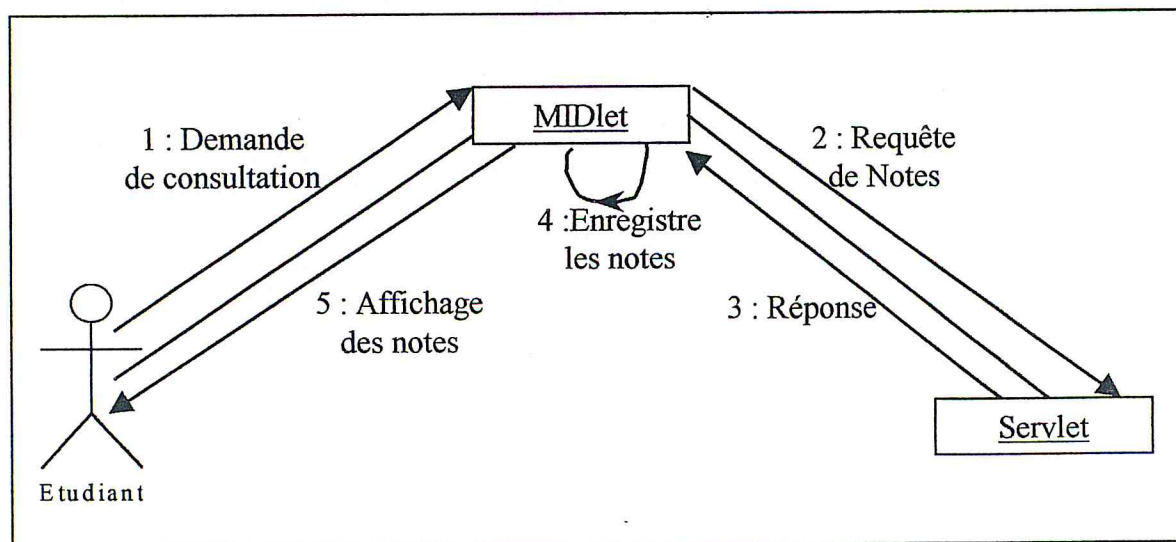


Figure V-V-6: Diagramme de collaboration pour la consultation des notes

2.7. Diagramme d'activité

Ce diagramme permet de décrire le déroulement d'un cas d'utilisations particulier.

Il est possible de décrire les acteurs responsables de chaque activité par l'utilisation des «couloirs d'activités» qui permettent de répartir graphiquement les différentes activités entre les acteurs opérationnels [MUL97].

Chaque activité est placée dans le «couloir» correspondant à l'acteur qui assume cette activité.

Nous avons utilisé ce formalisme pour présenter le diagramme d'activité du cas de consultation des notes.

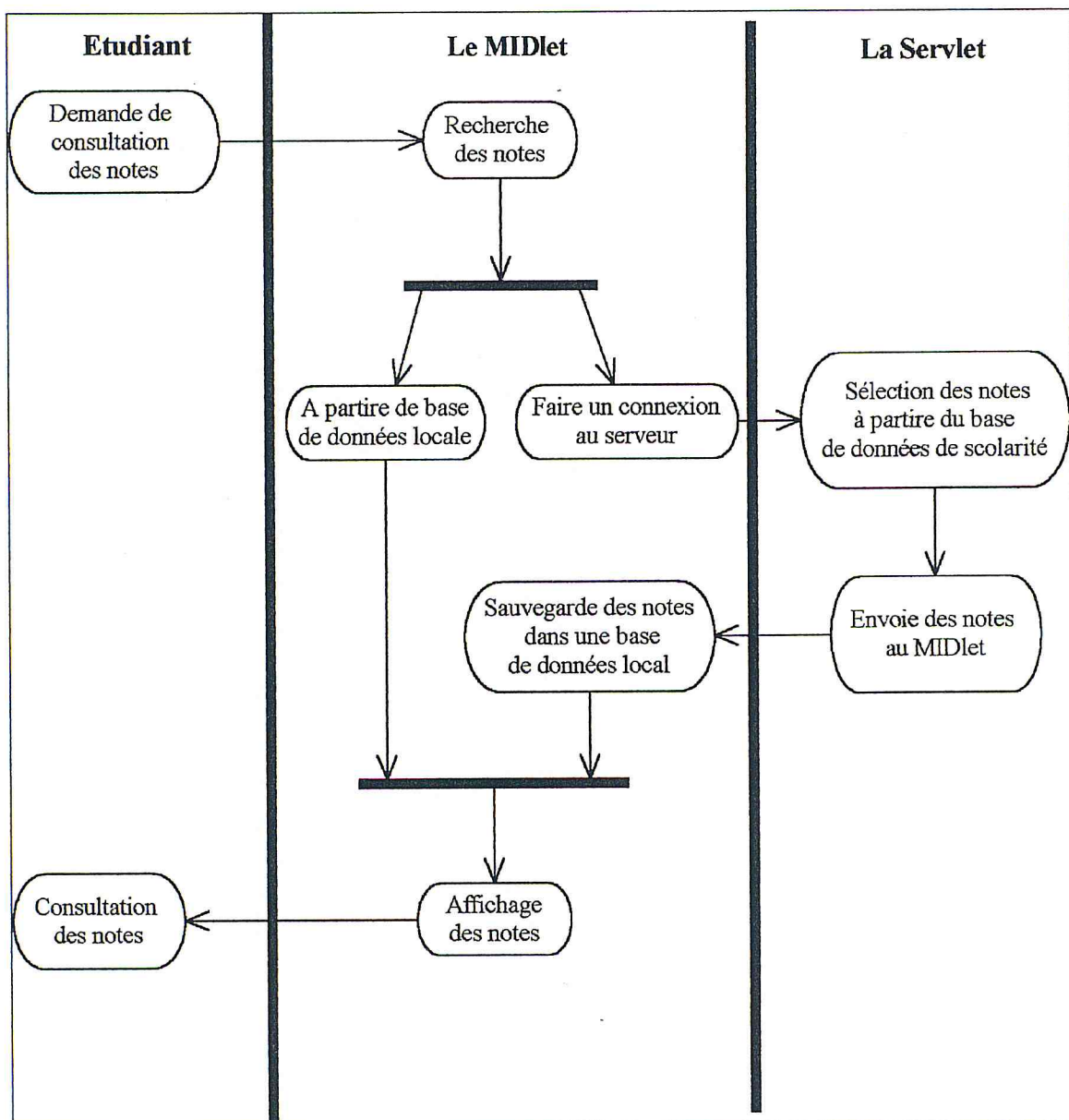


Figure V-7: Diagramme d'activité de cas de consultation des notes

Voilà le diagramme d'activité du côté administrateur :

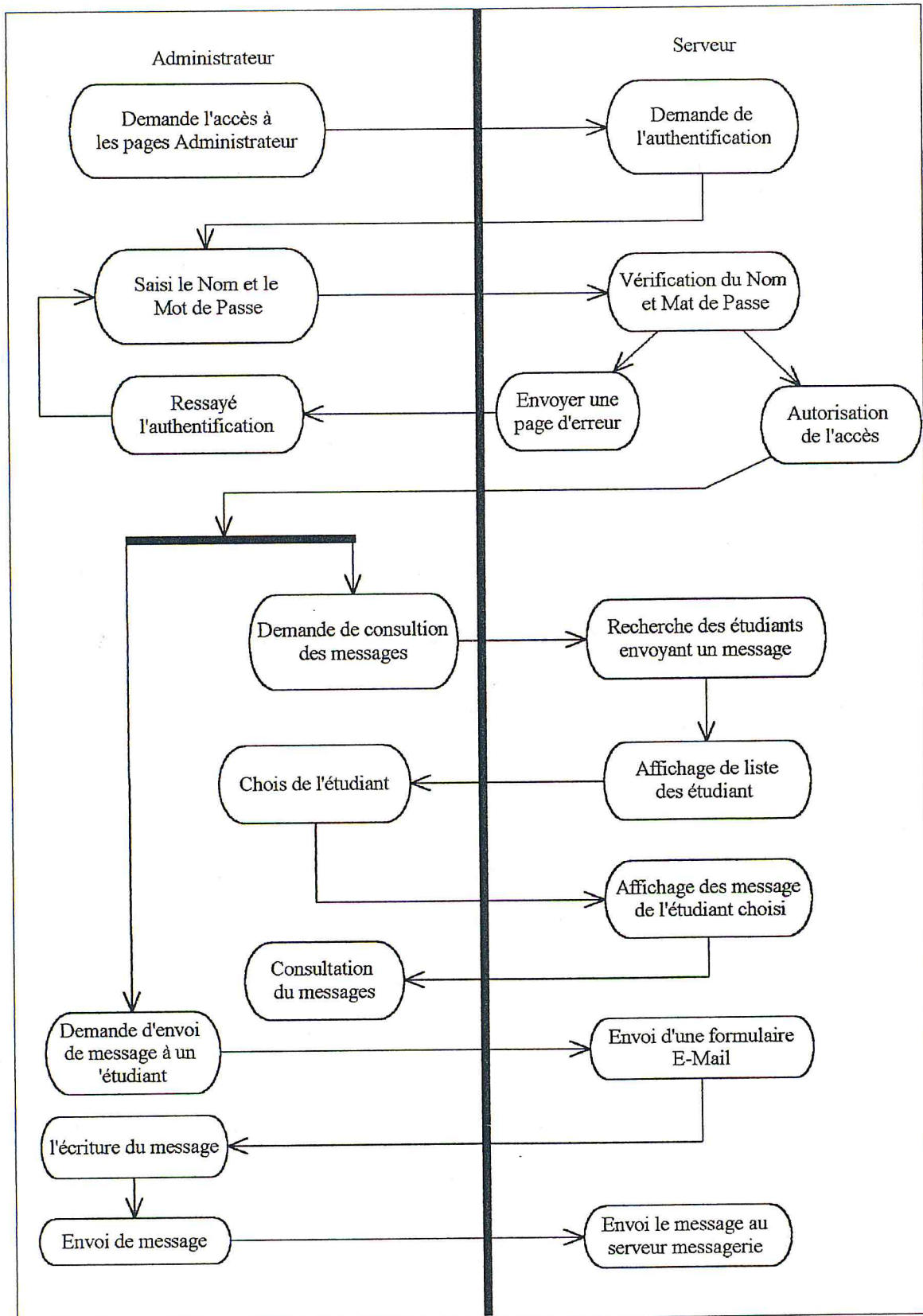


Figure V-8: Diagramme d'activité de l'Administrateur

3. La Conception

La conception s'intéresse d'abord au « comment », à savoir la solution du problème énoncé.

Elle commence par une conception dite « globale » qui décrit l'architecture de système, elle est suivie par une conception détaillée.

3.2 Conception globale

La conception globale a pour but de décomposer le logiciel en modules et de préciser les interfaces et les fonctions de chaque module. A l'issue de cette étape, on obtient une description de l'architecture du logiciel et un ensemble de spécifications de ses divers composants.

3.2.1 Conception de l'architecture

Cette section présente les avantages de la conception d'une architecture Trois-tiers pour le développement d'applications sur les mobiles.

Ce type d'architecture permet de séparer :

- les données,
- la mise en forme, et
- la logique.

Il devient ainsi possible de développer une mise en forme par type de terminal ciblé.

3.2.2 L'architecture Trois-tiers

L'architecture Trois-tiers est fondée sur une structure reposant sur trois tiers : le tiers de données, le tiers intermédiaire et le tiers client. Une autre vision de cette structuration en trois tiers est la décomposition en trois couches : la couche des données, qui est implémentée dans le tiers de données, et les couches métier et de présentation, implémentées dans le tiers intermédiaire. La source de données comprend les données issues d'une base de données. Le tiers intermédiaire comprend les couches métier et de présentation. La couche métier est typiquement implémentée dans des containers *JBeans*, tandis que la couche présentation peut être implémentée dans le serveur *web* et dans des pages *JSP* et des *servlets*. Enfin, le tiers client tourne sur les ordinateurs de bureau, sur les PDA ou sur les téléphones mobiles.

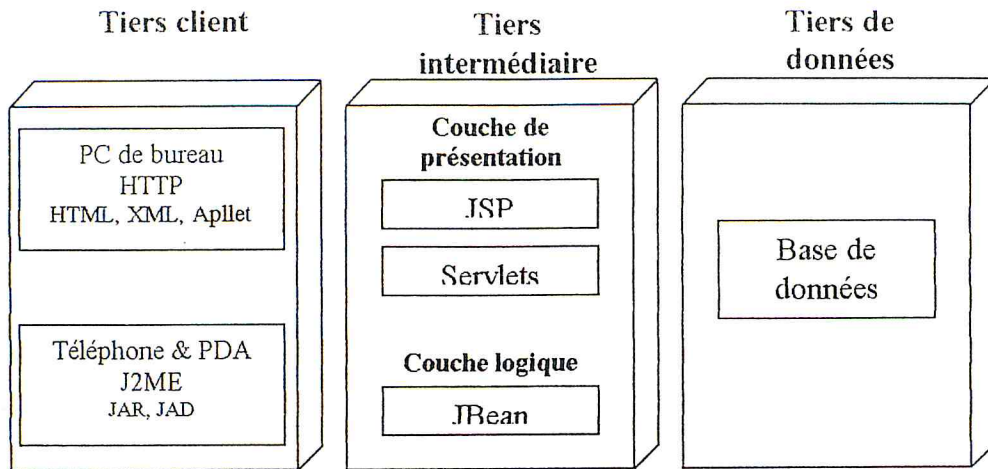


Figure V-9: L'architecture Trois-tiers

- Les *JBeans* (Les Java Bean) : Sont des composantes Java qui comprennent la logique métier et assurent la distribution et l'accès à ces éléments.
- Les *Servlets* : Sont des classes Java qui étendent les fonctionnalités d'un serveur d'application ou web.
- Les pages *JSP* (Java Server Page) : sont des templates utilisées pour générer dynamiquement des documents *web*.

Voici l'architecture générale de notre application :

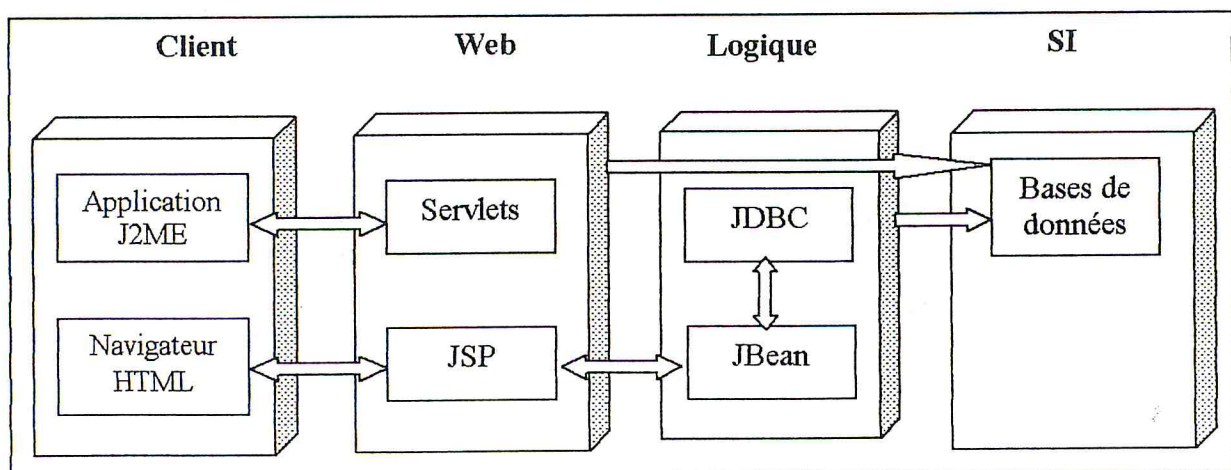


Figure V-V-10 : Architecture générale de l'application

- Les *Servlets* sont utilisées pour assurer la communication avec la base de données de la scolarité à partir de la *MIDlet* de l'étudiant.
- Les *JSP* sont utilisées pour générer des pages *web* dynamiques à l'administrateur afin de présenter :
 - des formulaires,

- des informations, et
- des résultats.
- Les JBean sont utilisés pour :
 - effectuer des requêtes SQL,
 - traiter les réponses, et
 - envoyer les résultats aux pages JSP.

Donc voilà un schéma global de mon application

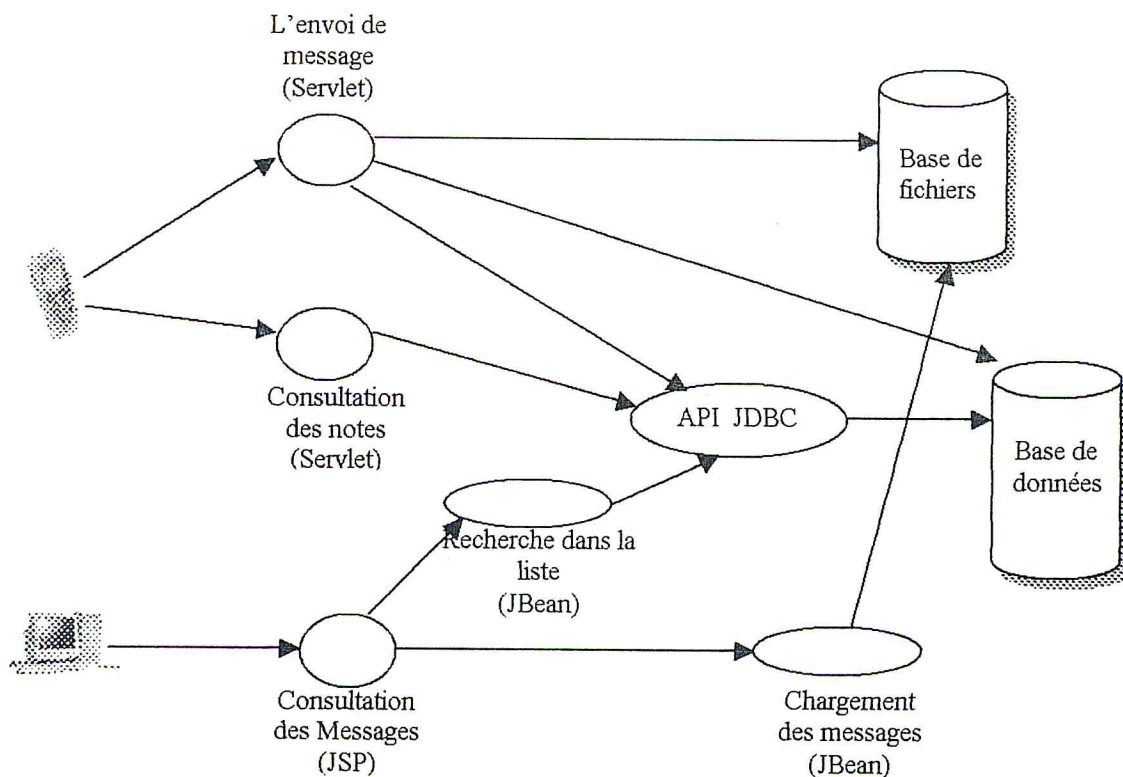


Figure V-10: Schéma global de l'application

Donc on peut décomposer notre application en trois modules :

- Module1 : le côté client.
- Module2 : le côté serveur.
- Module3 : la base de données.

- Le module client représente le MIDlet d'une part et représente le navigateur de l'administrateur d'autre part.
- Le module serveur représente les *Servlets* et les pages *JSP* et les *JBean*.

- La base de données c'est la base de données de scolarité plus une table pour le stockage les étudiant ayant envoyé un message.

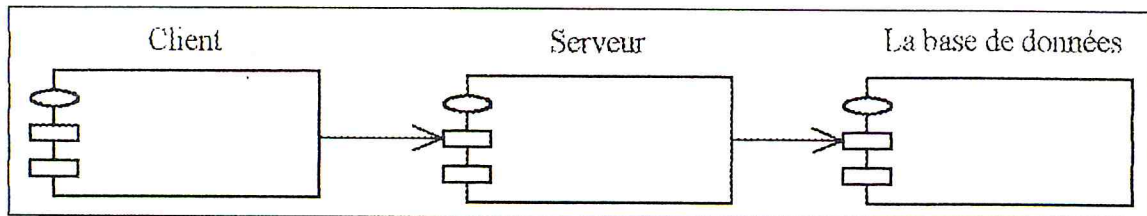


Figure V-11: Diagramme de composants pour les modules de système

3.3 Conception détaillée

La conception détaillée fournit, pour chaque module, une description détaillée de la manière dont les fonctions du composant sont réalisées: algorithmes, représentation des données [SOM88].

3.3.1 Côté Client

Cette application est constituée de le MIDlet et s'exécute sur le téléphone portable.

Lors d'une première utilisation, le MIDlet permet à l'étudiant de s'identifier : il doit fournir son matricule, son nom et son prénom.

Lors des utilisations ultérieures, ces champs ne seront plus à saisir, elle les sauvegarde dans une base de données légère de type *RecordStore*.

A la prochaine utilisation, la *MIDlet* affiche un menu au moyen d'une *Form* et d'une liste de choix permettant à l'étudiant de faire une consultation ou un envoi de message.

Si l'étudiant choisit la consultation des notes la *MIDlet* va faire une connexion http, au moyen de la classe *Connection*, au serveur de la scolarité et télécharger les notes de l'étudiant à partir de la base de données de la scolarité en utilisant le matricule saisi lors de la première utilisation.

Elle stocke les notes dans une *RecordStore*, et à la fin la *MIDlet* affiche une table de notes à l'aide de *Table*.

Si l'étudiant choisit l'envoi de messages, la *MIDlet* affiche un formulaire à l'étudiant pour qu'il le remplisse et l'envoyer.

A l'envoi, la *MIDlet* va faire une connexion au serveur et envoyer le message et le matricule de l'étudiant.

L'Administrateur peut consulter les messages qui ont été envoyer par les étudiants, à partir de sont navigateur web.

On représente les déférentes classes et ses relations par ce diagramme de classe

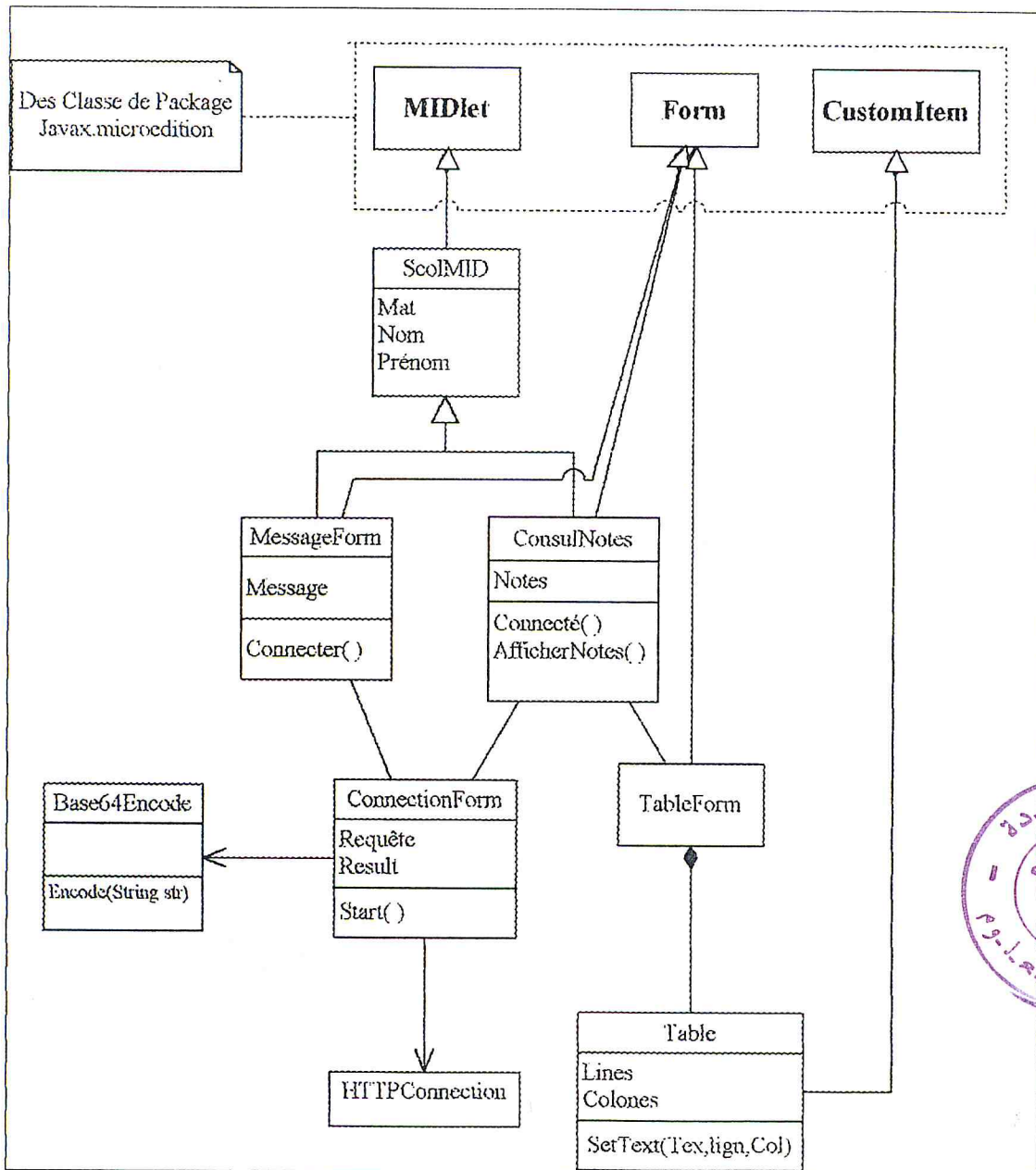
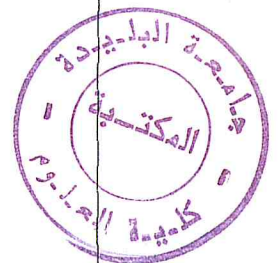


Figure V-12: Diagramme de classes du côté client

3.3.2 Côté Serveur

Cette application permet le stockage et la consultation des informations de la base de données.

Quand l'étudiant veut consulter ses notes, c'est la *Servlet* qui reçoit son matricule à partir de la *MIDlet*.



A partir de ce matricule la Servlet *NoteServeur* sélectionne les notes depuis la base de données et les envoie à l'étudiant.

Quand l'étudiant voudrait envoyer un message, la Servlet *MessageServeur* reçoit le message et le matricule de l'étudiant, le stocke et insère l'étudiant dans la table *Etd_Msg*.

C'est au moyen de pages JSP qu'on affiche à l'administrateur la liste des étudiants qui ayant envoyé des messages.

Les pages JSP embarquent des *JBeans* (*ListeBean* et *MessageBean*) pour accéder à la base de données.

Quand l'administrateur choisi un étudiant, une page JSP charge les messages de s'étudiant et les affichés.

Et voilà les diagrammes de classes de côté serveur.

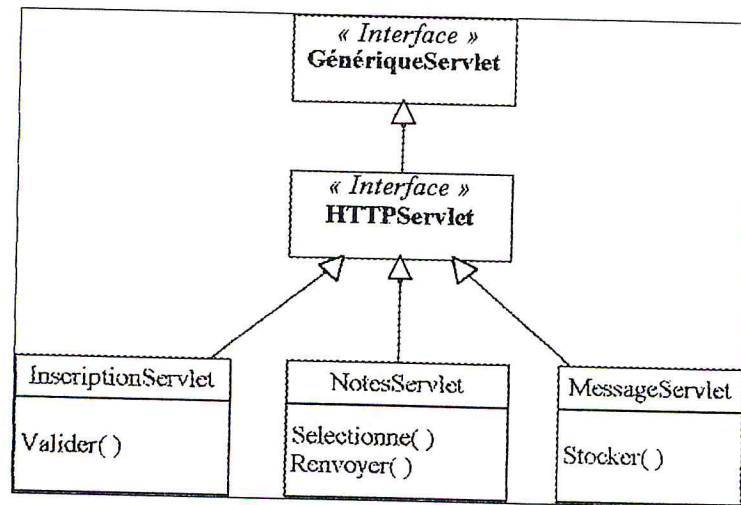


Figure V-13: Diagramme de classes du côté serveur (Les Servlets).

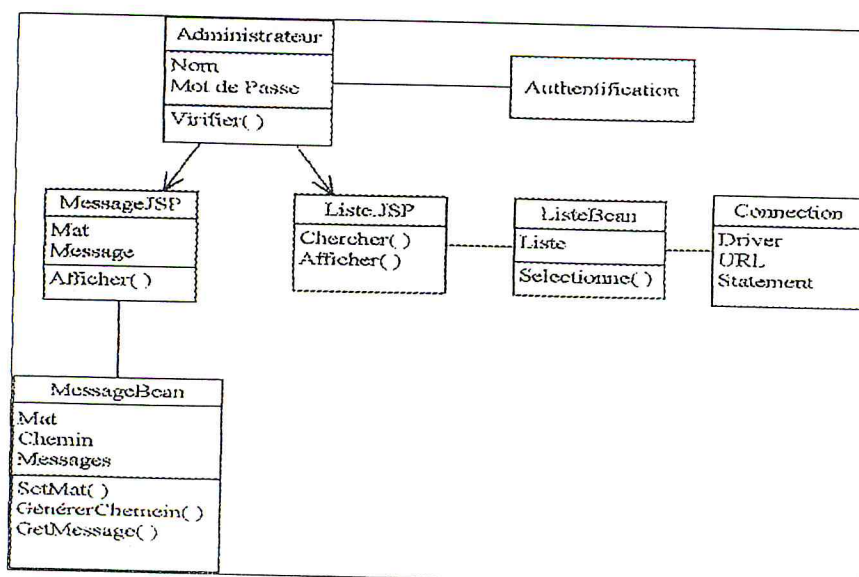


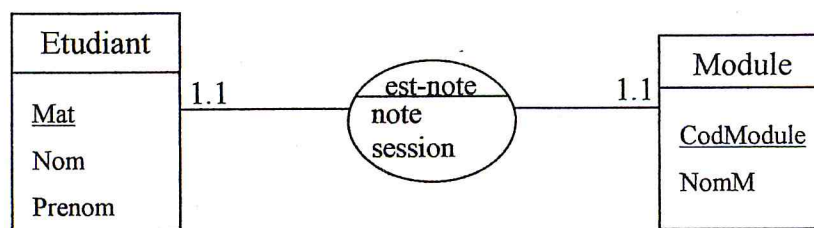
Figure V-14: Diagramme de classes du côté serveur (Pour l'Administrateur)

3.4 La base de données

Comme notre base de données, en réalité, est une partie de base de données de scolarité nous allons utiliser les table suivant :

- **Etudiant** qui représente les étudiants.
- **Module** la table des module.
- **Note** pour stocker les notes.
- **Message** indique les étudiants qui envoyés des messages.

Nous avons le schéma suivant :



A partir de ce schéma en peut construire les trois tables Etudiant, Note et Module.

Etudiant : [Mat, Nom, Prenom].

Note : [Mat, CodModule, Note, Sission].

Module : [CodModule, NomM].

3.5 L'architecture matérielle

Le système est constitué d'élément logiciels et matériels, interchangeable dans une large mesure [MUL 97].

Notre *MIDlet* a la possibilité de tourner :

- sur un terminal mobile supportant Java (terminal MIDP),
- sur un PDA, ou sur
- un téléphone cellulaire.

La partie serveur tourne sur le serveur *web* de l'université de BLIDA.

L'administrateur peut accéder à sa partie par son PC de bureau.

Le terminal mobile peut se connecter au serveur par l'intermédiaire d'une passerelle *Wap* qui peut être assurée par l'opérateur de téléphone.

Ce dernier doit supporter le GPRS pour assurer la connectivité au réseau Internet.

Voilà le diagramme de déploiement concernant notre architecture matérielle.

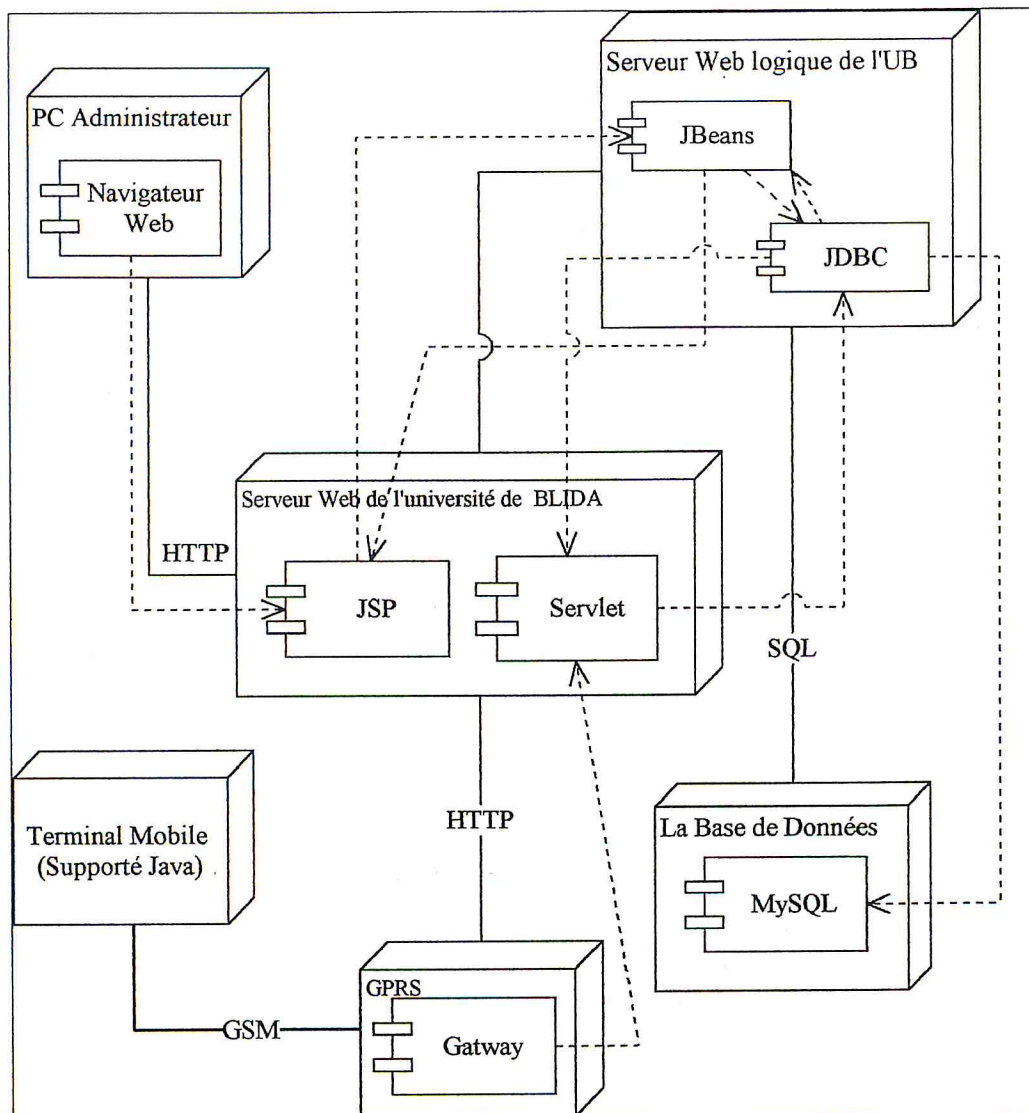


Figure V-15 Diagramme de déploiement

4. Implémentation

Notre développement consiste :

1) à mettre en place, c'est-à-dire :

- A choisir,
- A acquérir, et
- A installer

tout environnement de travail nécessaire, et

2) à développer trois applications :

- L'application côté client, constituée de la *MIDlet*.
- L'application côté serveur, constituée de *Servlets*, de pages *JSP* et de *Java Beans*, et de

- La base de données.

4.2 Environnement de travail

L'ensemble du travail a été réalisé sous *Windows 98*.

La *MIDlet* a nécessité la mise en place d'un environnement de développement Java *Java 2 SDK, Standard Edition, version 1.4*, disponible sur le site de *Sun Microsystems*.

L'émulateur de téléphone portable *J2ME™ Wireless Toolkit 2.0* a également été installé, ainsi que *JCreator*, comme éditeur de texte.

Le serveur *Tomcat 4.0.6* de Apache a été mis en place pour satisfaire la partie serveur.

Les *Servlets* et les *JavaBeans* ont également été éditées au moyen de *JCreator*.

Les pages *JSP* ont été éditées par *WebEditor* pour le côté *HTML* et *JCreator* pour le côté Java.

Enfin, la base de données a été mise en œuvre sous *MySQL*, qui est disponible sous licence gratuite pour *Windows*.

4.3 Choix de l'environnement de développement :

4.3.1 Côté client

La conception d'une *MIDlet* nécessite un environnement de programmation Java, est c'est *Java™ 2 SDK, Standard Edition, version 1.4.1*, disponible sur le site de *Sun Microsystems*, qui a été mis en place.

Il a fallu aussi installer un émulateur de téléphone portable pour tester les applications.

Là encore, *Sun Microsystem* qui s'est trouvée favorite, et c'est son émulateur *J2ME™ Wireless Toolkit 2.0* qui a été choisi.

De nombreux fabricants de téléphones portables offrent des émulateurs de leurs produits, mais nous avons préféré recourir au produit de Sun, très utilisé et profitant de bonnes critiques dans la littérature technique.

4.3.2 Côté serveur

Dans cette partie, il est question :

- de pouvoir recevoir des informations,
- de les traiter, et
- de communiquer avec une base de données.

Le choix d'un langage de programmation et d'un serveur Web approprié s'est ainsi posé.

Les langages *Java, JSP, ASP, PHP* et *CGI* figuraient parmi les candidats.

Java a été choisi pour plusieurs raisons :

- Sécurité.

- Communication avec la base de données facilitée grâce à l'API *Java Data Base Connectivity (JDBC)*.
- Cohérence avec l'ensemble du projet (la *MIDlet* a aussi été écrite en Java).

En ce qui concerne le serveur *Web*, *Tomcat* a été choisi, qui a également été écrit en Java et est dédié spécialement aux *Servlets* et *JSP*.

4.2.3. La base de données

Cette base de données étant relativement simple, notre choix s'est porté sur le *SGBD MySQL*.

Le nombre de tables étant restreint et les requêtes simples, aucune interface graphique n'a été installée, et nous avons opéré par lignes de commande.

4.3. Réalisation de l'application

Notre application se décompose en trois parties :

4.3.1. La MIDlet

La *MIDlet* assure :

- l'interface utilisateur,
- la connexion au serveur, plus
- le stockage des notes dans une base de données légère (*RecordStore*).

La classe *ScolMIDlet* est la *MIDlet* principale de notre application, elle étend la classe *MIDlet*, qui caractérise les applications se déployant sur les téléphones portables.

Elle implémente également l'interface *CommandListener*.

Comme c'est expliqué au chapitre 4, l'interface *CommandListener* permet la mise sur écoute du *Screen* de l'interface utilisateur.

A chaque action de l'utilisateur on pourrait associer une suite d'événements.

Comme interface utilisateur, *ScolMIDlet* affiche un menu *List* implicite

La classe *ConsultiNotes* étend la classe *Form* qui est un *Screen* et un conteneur des composants d'interface utilisateur.

Elle implémente également l'interface *CommandListener*.

Elle utilise un *TableForm* pour afficher les notes dans une table.

Pour la connexion au serveur, elle utilise une instance de la classe *ConnectionForm*.

Elle utilise aussi un *RecordStore* pour stocker les notes.

La classe `ConnectionForm` utilise une connexion `HTTP` à la méthode `Post` :
`conn.setRequestMethod(HttpConnection.POST);`

Pour plus de sécurité du trafic de données sensibles, on peut utiliser le protocole `Https` (`http + SSL`) pour envoyer des données cryptées à base de l'algorithme `RSA`.

La classe `MessageForm` également étend la classe `Form` pour afficher un formulaire. Elle implémente l'interface `CommandListener` pour gérer les `Commands`.

Voilà les différents écrans de notre MIDlet.



Figure V-16: Ecran d'accueil de ScolMidlet



Figure V-17: Formulaire de l'envoi d'un message

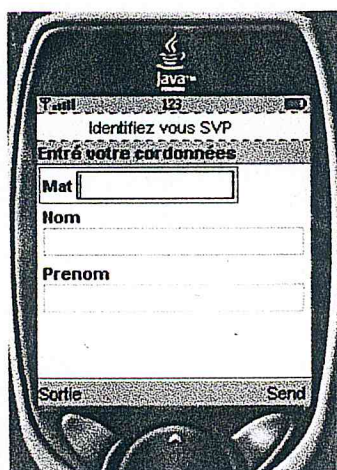


Figure V-18 : Formulaire d'identification



Figure V-18: Menu principal

4.3.2. Les Servlets :

La *Servlet*, comme cela est décrit précédemment, sert de partie intermédiaire pour la *MIDlet* afin d'accéder à la base de données dans le serveur.

Dans notre application nous avons développé trois *Servlets* :

- la première pour assurer l'identification de l'étudiant.

Cette *Servlet* reçoit le matricule, le nom et le prénom de l'étudiant, et les utilise pour vérifier leur existence dans la table *Etudiant* de la base de données de la scolarité.

Pour la communication http, la *Servlet* utilise la méthode POST pour traiter les requêtes http, donc elle utilise la fonction *doPost()* pour traiter les requêtes http.

Pour interroger la base de données la *Servlet* utilise l'API *JDBC*.

- La deuxième *Servlet* pour assurer la sélection des notes et les envoyer à l'étudiant.
- La troisième pour stocker les messages envoyés par les étudiants.

On met les classes des *Servlets* dans le sous répertoire *classes* de répertoire *WEB-INF* de répertoire de notre application.

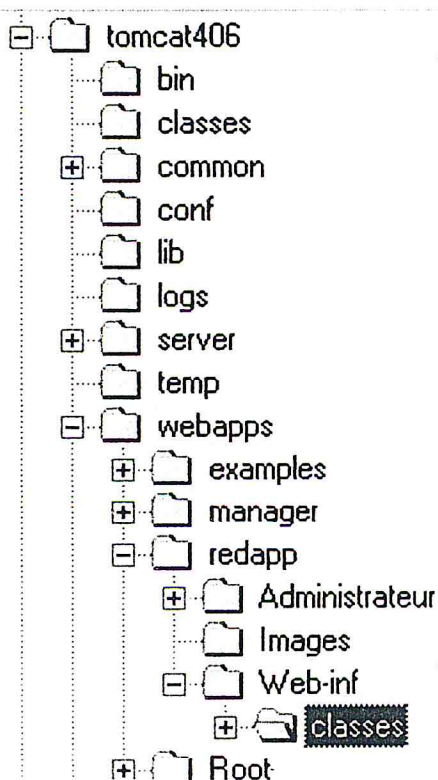
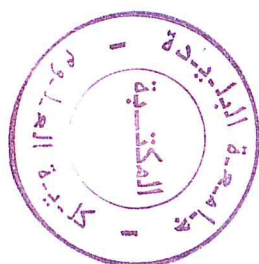


Figure V-19: L'hierarchie des répertoire au serveur TomCat

4.3.3. Les pages JSP

Les pages JSP sont des pages *web* dynamique, autrement dit, sont des pages HTML englobant du code Java pour assurer la partie dynamique.

Elle peut embarquer du *JavaBean* (des composants Java) pour diminuer la lourdeur de code Java dans les pages *JSP*.

Les *JBeans* sont utilisés pour se connecter à la base de données et au système de fichiers.

Les pages Administrateur sont protégées par le serveur, donc si on veut les accéder, le serveur envoie une page d'authentification.

Si l'authentification est correcte l'administrateur accède à la page Administrateur, cette dernière inclue deux liens, le premier vers la page consultation et le deuxième vers un formulaire de E-Mail pour envoyer un message à un étudiant.

Lors d'une mauvaise saisie de son nom ou mot de passe, l'administrateur est redirigé vers la page précédente.

La partie du fichier descripteur du serveur *TomCat web.xml* pour assurer l'authentification a été réalisée par le formulaire suivant :

```
<!-- Define a global security constraint, all pages are protected -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Redhapp</web-resource-name>
      <url-pattern>/Administrateur/*</url-pattern>
      <!-- Le dossier Administrateur est protégé -->
      <http-method>GET</http-method>
      <http-method>POST</http-method>
    </web-resource-collection>

    <!-- Users in the role "Adm" can read or write messages only -->
    <auth-constraint>
      <role-name>Adm</role-name>
      <!-- Sauf les utilisateur dont le rôle "Adm" peut lire ou écrire dans le
dossier Administrateur -->
    </auth-constraint>
  </security-constraint>

  <!-- Use BASIC authentication -->
  <login-config>
    <auth-method>
      FORM
    </auth-method>
    <form-login-config>
      <form-login-page>
        /loginpage.htm <!-- la page d'authentification -->
      </form-login-page>
      <form-error-page>
        /error.htm <!-- la page d'erreur -->
      </form-error-page>
    </form-login-config>
  </login-config>
```

```

    </form-error-page>
  </form-login-config>
</login-config>

```

les utilisateurs autorisés sont décrits dans le fichier *tomcat-users.xml* comme suit :

```

<tomcat-users>
  <user name="tomcat" password="tomcat" roles="tomcat" />
  <user name="role1" password="tomcat" roles="role1" />
  <user name="redha" password="mazari" roles="Adm" />
  <user name="redha" password="lrdsi" roles="users" />
  <user name="mazari" password="lrdsi" roles="users" />
  <user name="both" password="tomcat" roles="tomcat,role1" />
</tomcat-users>

```

BIENVENEU sur l'espace Administrateur
BIENVENEU sur l'espace Administrateur

IDENTIFICATION

Nom d'utilisateur: redha

Mot de Passe: *

VALIDER

Figure V-20: Formulaire d'authentification

Voici le script html de ce formulaire :

```

<FORM METHOD=POST ACTION=j_security_check>
  <p align="right"><b>Nom d'utilisateur:</b><input type="text" name="j_username" />
  <p align="right"><b>Mot de Passe:</b><input type="password" name="j_password" />
  <p align="center"><input type="submit" value="VALIDER" />
</FORM>

```

4.3.4. La base de données

Pour réaliser la base de données on a utilisé le SGBD MySQL. Pour créer notre base de données sous MySQL nous avons suivi les étapes suivantes :

1. Lancez le serveur de base de données MySQL.
2. Dans une fenêtre DOS, créez la base de données à l'aide de cette commande :

```
mysqladmin create Scolarite
```

```
C:\mysql-4.0.18\bin> mysqladmin create Scolarite
C:\mysql-4.0.18\bin>
```

Figure V-20: Création de la base de données

3. Dans une nouvelle fenêtre DOS, lancez le client MySQL en entrant cette ligne : `mysql`
4. Connectez-vous à la base de données Scolarite par cette commande :

```
connect Scolarite
```

```
C:\mysql-4.0.18\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.0.18

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> connect scolarite;
Connection id:      5
Current database:  scolarite

mysql>
```

Figure V-21: lancement du client et connexion à la base de données

5. Créez les tables de la base de données utilisant SQL. Par exemple pour la table Etudiant

```
mysql> create table 'Etudiant'
(
    'Mat' VARCHAR(25) not null primary key,
    'Nom' VARCHAR(10),
    'Prenom' VARCHAR(10)
);
```

5. Les tests

Les tests permettent de réaliser des contrôles pour la qualité du système. Il s'agit de relever les éventuels défauts de conception et de programmation (revue de code, tests des composants,...) [SOM 88].

Le test d'une *MIDlet* demande un appareil réel, pour le télécharger

On peut la télécharger sur le PC puis la transférer sur le mobile ou la télécharger directement sur le mobile depuis un serveur web. Dans ce cas on aurait besoin d'un téléphone compatible Java compatible *WAP*, ou d'un câble (appelé câble de données) servant à relier le téléphone au PC.

En ce qui nous concerne, pour le côté tests, nous avons utilisé le *J2MEWTK* de Sun qui peut simuler toutes les fonctionnalités d'un appareil mobile.

Le *WTK 2.0* simule le téléchargement d'une application depuis un serveur *web* par son utilitaire *OTA Provisioning (On The Air)* qui peut télécharger des *MIDlet* par le *http*.



Figure V-23: la première étape de OTA Provisioning



Figure V-24: Lancement de recherche

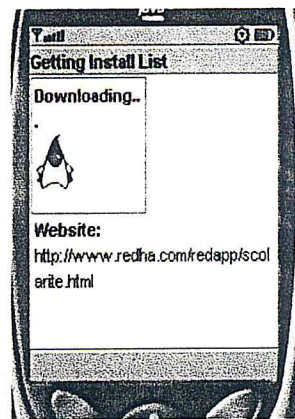


Figure V-22: Recherche des fichiers Jad

Quand il trouve la page il va chercher le lien vers des fichiers *.jad* et les affichées dans liste à l'écran.

L'utilisateur pourrait alors choisir la *MIDlet* à installer et lancer l'installation.



Figure V-25: La liste de fichier .jad



Figure V-26: L'installation de fichier .Jad

Comme on l'avait déjà décrit précédemment le fichier Jad est un fichier descripteur de le MIDlet

Donc l' *OTA Provisioning* charge les informations et les affiche à l'écran, tout en questionnant l'utilisateur sur l'installation.

Après le téléchargement la JVM de l'appareil mobile vérifie l'application avant l'installation.

Dès que l'application est installée, on peut la lancer, lui faire faire une mise à jour ou la supprimer.



Figure V-32: Téléchargement du fichier .jad



Figure V-31: Confirmation de l'installation



Figure V-27: Téléchargement de l'application (.jar)



Figure V-30: Vérification de l'application



Figure V-29: L'installation se termine avec succès



Figure V-28: Lancement de l'application après l'installation

Après lancement de notre application le formulaire d'identification s'affiche, si j'ai oublié un champ son remplissage une *Alert* d'erreur s'affiche et me retourne vers le formulaire.

Dans le cas de consultation de notes, à la connexion une *Alert* de confirmation s'affiche et me demande la confirmation de la connexion.

Après le téléchargement des notes, elles seront stockées dans une base de données légère *RecordStore* et s'affiche dans une table.

Lors d'une consultation ultérieure, le *MIDlet* affiche les notes qui sont stockées dans le RecordStore.

L'étudiant pourrait alors, s'il le désirait, faire une nouvelle consultation par l'appuie sur la command MAJ.

Au-dessus de la table, il y'aurait un *Ticker* affichant la date de dernière consultation à partir de la base de données de scolarité.



Figure V-33:
L'Identification

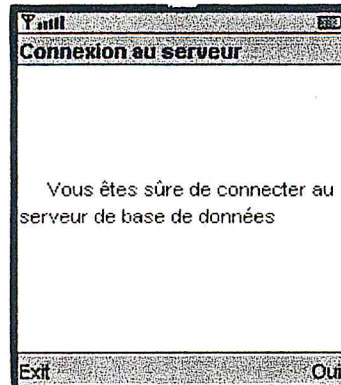


Figure V-35: Alert de confirmation

Module	EMD1	EMD2	MOYN
Algo	10.0	12.0	11.0
Annalys	15.0	14.0	14.5
Algèbre	0.0	0.0	0.0
BDD	12.0	10.5	11.75
Réseau	8.0	17.0	12.5
GL2	8.0	10.5	9.25

Figure V-34: Table des Notes avec un Tiker affiche la dernière consultation

L'envoi de message se fait à l'aide d'un formulaire comme suit

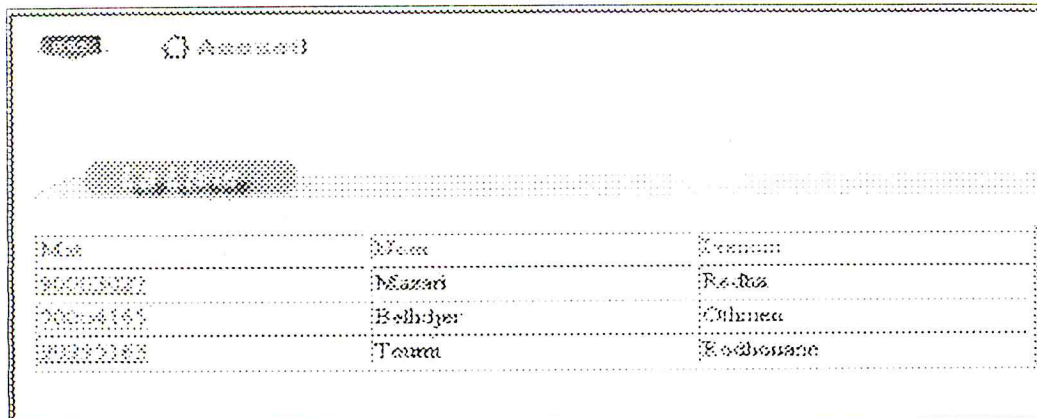


Figure V-36: L'envoi de message



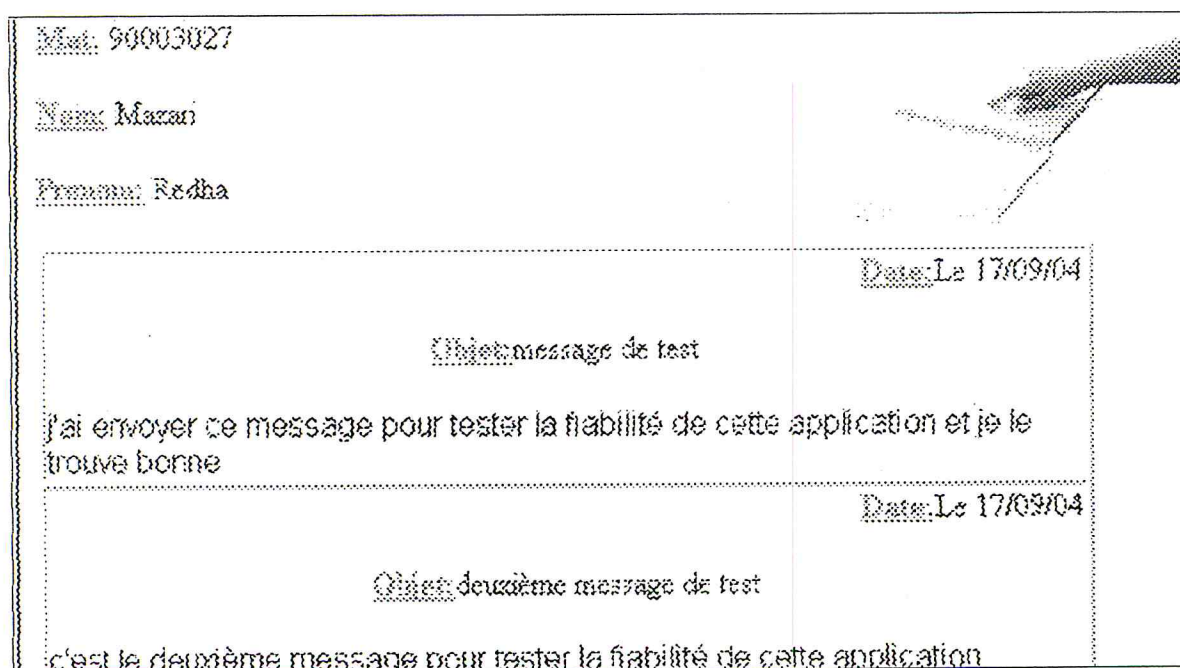
Figure V-37: Alert de confirmation de l'envoi de message

Après l'authentification de l'administrateur il peut accéder à la liste des étudiants qui ont envoyé des messages, et sélectionner un pour consulter ses messages.



Mat	Nom	Prénom
90003027	Mazari	Redha
90003151	Belhajjer	Chirine
90003183	Touma	Redouane

Figure V-40: La liste des étudiants envoyant des messages



Mat: 90003027	Nom: Mazari	Prénom: Redha
		Date: Le 17/09/04
Objet: message de test		
j'ai envoyer ce message pour tester la fiabilité de cette application et je le trouve bonne		
		Date: Le 17/09/04
Objet: deuxième message de test		
c'est le deuxième message pour tester la fiabilité de cette application		

Figure V-381: La liste des messages de l'étudiant de matricule 90003027

6. Conclusion

Nous avons présenté dans ce chapitre la mise en œuvre d'une *MIDlet* qui communique avec une *Servlet*.

Pour concevoir notre application, nous avons choisi l'architecture *trois-tiers* étant donnée sa grande utilisation dans les applications *web* [JAS 98], et nous avons utilisé une modélisation et une conception orienté objet basée sur UML comme un langage de modélisation.

La *MIDlet* qu'on a développée est facile à utiliser, l'interface utilisateur développée est assez riche (utilisation des *Commands*, *TextFields*, *Form*, *Tickers*, *Alerts*, ...), et elle pourrait avoir l'avantage d'être facile à utiliser et représente assez bien l'application.

Chapitre VI Conclusion générale

Dans ce mémoire, nous avons mis en oeuvre la démarche proposée dans le chapitre 2 pour l'implémentation des différentes étapes de notre réalisation.

Nous nous sommes heurtés à certaines difficultés lors de cette mise en oeuvre, qui tiennent essentiellement à trois facteurs :

- l'utilisation de logiciels de travail , dont le mode d'emploi est peu expliqué et qui ne sont pas toujours suffisamment appliqués ou testés dans le domaine que nous avons choisi d'étudier, comme celui du fonctionnement de Tomcat.
- les problèmes d'assimilation des techniques de modélisation diverses, comme celui ayant trait à la communication MIDlet-Servlet ou la manipulation de suites Screens pour la réalisation de « fenêtres ».
- le problème pour la recherche, l'étude et l'assimilation théorique du domaine de notre étude, en raison surtout à la diversité des champs qu'il englobe.

Nous avons estimé que les méthodes Agiles sont bien adaptées au domaine de la programmation J2ME.

Les méthode de modélisation des UI que nous avons mises en oeuvre s'avère intéressante pour un début d'expérience, même si elles ne conduisent pas encore systématiquement à un résultat général, net, et définitif.

Les résultats obtenus peuvent être améliorés par quelques modifications de techniques de modélisations, qui prennent en considération les problèmes d'un point de vue plus général, « aux Patterns », par exemple. Néanmoins, ils restent assez valables.

Le but de notre application est de montrer les possibilités et les limites de J2ME et l'intérêt de son utilisation pour le développement des applications très variées, du simple agenda jusqu'aux applications client-serveur basées sur une communication crypté.

Enfin on dit que notre application sert à un prototype pour les applications client serveur basé sur une *MIDlet* et *Servlet*, pour aider les développeurs qui veulent rentrer dans ce domaine.

Du point de vue personnel, ce projet a été très enrichissant par le fait qu'il permet d'ouvrir les portes de monde des applications mobiles avec tous qu'elles porter comme technologies.

Recherches futures :

Une fois cette application achevée, nous avons trouvé qu'il reste bien des domaines à défricher, comme :

- l'utilisation des patterns dans la modélisation,
- le développement d'une application comportant de l'animation et du 3D,
- le développement des *MIDlets* intelligentes à l'aide des agents,
- le développement d'applications dans le domaine de la télé-médecine, télé-consultation bureautique, télé-direction d'entreprises, télé-conduite de projets,...

Le projet qui nous tient actuellement le plus à cœur est celui de la télé-consultation du dossier médical des personnes atteintes de maladie allant jusqu'à des pertes de connaissance.



Bibliographie

- [DEL 02] Bruno Delb, J2ME Application Java pour terminaux mobiles, EYROLLES, 2002.
- [ENR 01] Enrique Ortiz, Eric Giguere, Mobile Information Device Profile for Java 2 Micro Edition: Professional Developer's Guide, 2001.
- [GIG 00] Eric Giguere Java 2 Micro Edition: Professional Developer's Guide, 2000.
- [GIG 03] Eric Giguere, Over-the-Air Provisioning with the J2ME Wireless Toolkit, article, 2003.
- [HAS 03] Mesud Hasanovic, Lucas Pretre, Mobile Service and Midlet, 2003.
- [HEI 00] Guillaume Heilles, Core Sevelets and Java Server Page, CampusPress, 2000.
- [JAS 98] HUNTER Jason, CRAWFOERD William, Servlet Java, O'reilly, 1998.
- [KNU 01] Jonathan Knudsen, Wireless Java: Developing with Java 2 Micro Edition, 2001.
- [KNU 03] Jonathan Knudsen, Wireless Java: Developing with J2ME, Second Edition, 2003.
- [MAH 00] Qusay Mahmoud, MIDP Network Programming using HTTP and the Connection Framework, article, 2000.
- [MAH 01] Qusay Mahmoud, Learning Wireless Java, 2001.
- [MUC 02] John Muchow, Core J2ME Technology and MIDP, 2002.
- [MUL 97] Pierre-Alain Muller, modélisation objet avec UML, EYROLLES, 1997.
- [PIR 02] Vartan Piroumian, Wireless J2ME Platform Programming, 2002.
- [Qualcomm] Le site officiel du développement avec BREW.
<http://www.qualcomm.com/brew/>
- [RIG 01] Roger Riggs, Antero Taivalsaari, Mark VandenBrink, Programming Wireless Devices with the Java 2 Platform, Micro Edition, 2001.

- [SAM 02] Sami Djaber, Java 2 Micro Edition (J2ME) Versus Microsoft Embedded Architecture, DotNetGuru, 2002
- [SOM 88] Lan Sommerville, le génie logiciel et ses applications, paris, 1988.
- [WHI 02] James White, David Hemphill, Java 2 Micro Edition, 2002.
- [ZEN 03] Damien Zéni, Java 2 Micro Edition, Présentation personnelle, Février 2003.

Les sites web

- [www1] Le site officiel de *Sun* pour J2ME
<http://java.sun.com/j2me/>
- [www2] Le MIDP: émulateur, articles ,SDK
<http://java.sun.com/products/midp/>
- [www3] émulateur, articles, forum de discussion...
<http://www.forum.nokia.com>

