N° d'ordre : 01 / SII /TCO Année Universitaire : 2006/2007

UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEURE POLYTECHNIQUE

DEPARTEMENT TELECOMMUNICATION

MEMOIRE DE FIN D'ETUDES

en vu de l'obtention

du DIPLOME d'INGENIEUR

Spécialité : Télécommunication
Option : Signal, Image et Information

par : **RAFIDISON Maminiaina Alphonse**

DEVELOPPEMENT D'APPLICATIONS MOBILES SOUS JAVA MICRO EDITION : CAS DES TELEPHONES PORTABLES

Soutenu le lundi 10 décembre 2007 devant la Commission d'Examen composée de :

Président :

Monsieur RATSIHOARANA Constant

Examinateurs :

Madame RABEHERIMANANA Lyliane

Monsieur RAMORASATA Joseph Raphaël

Monsieur RAKOTOMALALA Mamy Alain

Directeur de Mémoire :

Monsieur RANDRIARIJAONA Lucien Elino

« Si l'Eternel ne bâtit la maison. Ceux qui la bâtissent travaillent en vain ;
Si l'Eternel ne garde la ville. Celui qui la garde veille en vain. »

Psaume 127 :1

C'est avec cette phrase de la bible que je voudrais remercier le Seigneur qui m'a donné la force, la santé, l'intelligence, le courage de m'avoir accompagné durant le chemin parcouru au cours de la réalisation de ce mémoire.

REMERCIEMENTS

Mes plus sincères remerciements vont à :

- Monsieur RAMANANTSIZEHENA Pascal, Directeur de l'Ecole Supérieure Polytechnique d'Antananarivo pour m'avoir permis de suivre les cinq années d'études au sein de son établissement.

- Monsieur RANDRIAMITANTSOA Paul Auguste, Chef de département Télécommunication à l'ESPA pour avoir accepté que je soutienne ce mémoire de fin d'études.

- Monsieur RANDRIARIJAONA Lucien Elino, Directeur de ce mémoire pour le thème très intéressant qu'il m'a donné à exploiter, ses conseils et ses critiques.

- Monsieur RATSIHOARANA Constant, Président de jury, pour avoir l'honneur de présider ce mémoire.

- Madame RABEHERIMANANA Lyliane, Membre de jury, pour avoir accepté d'examiner ce mémoire.

- Monsieur RAMORASATA Joseph Raphaël, Membre de jury, pour avoir accepté d'examiner ce mémoire.

- Monsieur RAKOTOMALALA Mamy Alain, Membre de jury, pour avoir accepté d'examiner ce mémoire.

- Tous les Enseignants qui m'ont formé durant les cinq années d'études.

- Mes très chers parents, mes frères et sœurs, mes amis, qui m'ont aidé dans tous les plans.

- Tous les personnels administratifs de l'école.

AVANT-PROPOS

Ce mémoire constitue le travail de fin du cursus d'ingéniorat à l'Ecole Supérieure Polytechnique d'Antananarivo, aboutissement de cinq années d'études. Il fusionne toutes les connaissances acquises durant le cursus afin de juger la qualité de l'étudiant en terme de recherche et de pratique. Ce présent mémoire se concentre sur la programmation d'un téléphone portable, pour mettre en valeur les connaissances que nous avons acquises en télécommunication surtout en matière de programmation.

TABLE DES MATIERES

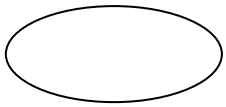
NOTATIONS

1. Abréviations

API	Application Programming Interface.
ASCII	American Standard Code for Information Interchange
AWT	Abstract Windows Toolkit
CDC	Connected Device Configuration.
CLDC	Connected Limited Device Configuration.
CVM	Compact Virtual Machine
GSM	Global System for Mobile communications
GUI	Graphic User Interface
HTTPS	Hypertext Transfer Protocol Secure
IHM	Interface Home Machine
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition.
J2SE	Java 2 Standard Edition
JAD	Java Application Descriptor
JAR	Java Archive
JCP	Java Community Process
JDK	Java Development Kit
JSR	Java Specification Request
JTWI	Java Technology for the Wireless Industry
JVM	Java Virtual Machine
KVM	Kilobyte Virtual Machine
MIDP	Mobile Information Device Profile
MMAPI	Mobile Media API.
N	Non
O	Oui
OS	Operating System
OTA	Over The Air
PDA	Personal Digital Assistant

PKI	Public Key Infrastructure
RAM	Random Acces Memory.
RMI	Remote Method Invocation
RGB	Red Green Bleu
RMS	Record Management System.
SMS	Short Message Service
Sup	Suppression
TCP	Transfer Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WMA	Wireless Messaging API.

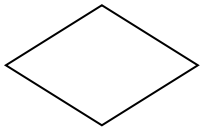
2. Notations spéciales



Début ou fin d'un programme ou d'un sous-programme

Opération

Toute opération qui crée, modifie, transfert des données et qui n'est pas définie par un symbole spécial



Test

Affichage des données

Acquisition ou initialisation des données

Connecteur

INTRODUCTION

La télécommunication a permis d'effacer les distances entre des personnes qui souhaitent se communiquer. En effet, Alexander Graham Bell, né en 1847 a inventé le téléphone en 1876 : c'est la première communication moderne reliant deux postes fixes par une paire de fil mise en place par Thomas Doolittle. Cependant la technologie ne cesse d'évoluer et le téléphone fixe domine le monde sauf dans les pays en voie de développement comme Madagascar. Chez nous, les principaux abonnés étaient les entreprises, mais depuis les années 90, des abonnés particuliers hébergent le téléphone chez eux pour les raisons familiales. Malgré l'insatisfaction des clients sur l'incapacité de portabilité, les opérateurs mobiles apparaissent progressivement sur le marché public au début de la vingt unième siècle. Le dernier modèle vient de perdre sa popularité et les utilisateurs font recours à l'usage des téléphones mobiles qui deviennent un outil nécessaire à la vie quotidienne des Malagasy. Ce produit attire presque la majorité des citadins pour certaines raisons particulières tel que sa portabilité, la facilité d'envoyer les donnés (parole, message).

A son apparition, les téléphones portables ont été limités à des affichages simplistes, quelques lignes de texte avec un fond gris et quatre couleurs. L'évolution technologique fait qu'aujourd'hui, des téléphones plus puissants, avec plus de mémoire, un meilleur affichage et une résolution élevée apparaissent avec un prix abordable.

On voit maintenant des appareils qui intègrent un appareil photo, d'autres un lecteur mp3 ou la radio, l'évolution de la téléphonie est loin de s'essouffler. Cependant, les téléphones évoluent mais sur des standards qui diffèrent selon les fabricants et les modèles. Le développement d'application passe en général par l'utilisation d'une API (Application Programming Interface) propriétaire souvent écrite en C ou C++. Le portage d'une application implique donc l'adaptation du code pour presque chaque modèle de téléphone, par conséquent, les coûts de production augmentent forcément.

Avec Java, nous pouvons cependant entrevoir une solution. *Sun* nous propose une version allégée de J2SE (Java 2 Standard Edition), adaptée aux appareils de faible puissance appelée J2ME (Java 2 Micro Edition).

Ce dernier langage domine le monde de la télécommunication, alors ce livre met en relief son importance sur la téléphonie mobile, le but est donc de concevoir un téléphone portable avec ce langage de programmation, dont la particularité est la capacité de lire des fichiers audio et vidéo.

Le démarche de présentation de notre ouvrage se divise en trois grandes parties et se présente comme suit: la première partie représente la J2ME, suivi du profil MIDP (Mobile Information Device Profile). Dans cette deuxième partie, on y trouve les méthodes pour la lecture de clips vidéo et de fichier vidéo. La conception et la réalisation de l'application sont entamées à la troisième partie de l'ouvrage.

CHAPITRE 1 J2ME : PRESENTATION GENERALE

J2ME est la plateforme Java utilisée pour le développement des applications sur des appareils mobiles tel que des PDA (Personal Digital Assistant), des téléphones cellulaires, des terminaux de points de vente, des systèmes de navigations pour voiture, etc. C'est une sorte de retour aux sources, puisque Java avait été initialement développé pour piloter des appareils électroniques. A vraie dire, J2ME signifie Java pour les petits appareils.

1.1. Introduction [1] [6] [7]

Historiquement, *Sun* a proposé plusieurs plateformes pour le développement d'applications sur des machines possédant des ressources réduites, typiquement les machines ne pouvant exécuter une JVM (Java Virtual Machine) répondant aux spécifications complètes de la plateforme J2SE telles que:

- **JavaCard** : pour le développement sur des cartes à puces.
- **EmbeddedJava** : pour le développement des systèmes embarqués.
- **PersonnalJava** : pour le développement sur des machines possédant au moins 2Mo de mémoire.

En 1999, *Sun* propose de mieux structurer ces différentes spécifications sous l'appellation J2ME. Seule la spécification JavaCard n'est pas incluse dans J2ME et reste une spécification à part.
Par rapport à J2SE, J2ME utilise des machines virtuelles différentes. Certaines classes de base de l'API sont communes, mais cependant, de nombreuses fonctions sont omises dans l'API de la dernière.
L'ensemble des appareils sur lequel peut s'exécuter une application écrite avec J2ME est tellement vaste et disparate que J2ME est composée de plusieurs parties : les configurations et les profils qui sont spécifiés par le JCP (Java Community Process). J2ME se base alors sur une architecture modulaire. Chaque configuration peut être utilisée avec un ensemble de packages optionnels qui permet d'utiliser des technologies particulières. Ces packages sont le plus souvent dépendant du matériel.

L'inconvénient de ce principe de packaging est qu'il déroge à la devise de Java "*Write Once, Run Anywhere*". Ceci reste cependant partiellement vrai pour des applications développées pour un profil particulier. Il ne faut cependant pas oublier que les types de machines cibles de J2ME sont tellement différents du téléphone mobile au set top box[1], qu'il est sûrement impossible de trouver un dénominateur commun. Ceci est associé à l'explosion du marché des machines mobiles expliquant les nombreuses évolutions en cours de la plateforme. Jusqu'ici c'est la J2ME qui est la plateforme objet de fréquente mise à jour.

1.2. Les configurations de J2ME [1] [2] [8] [9]

Les configurations définissent les caractéristiques de bases d'un environnement d'exécution pour un certain type de machine, possédant un ensemble de caractéristiques et de ressources similaires. Elles se composent d'une machine virtuelle et d'un ensemble d'API de base.
Deux configurations sont actuellement définies :

- **CLDC** (**C**onnected **L**imited **D**evice **C**onfiguration).
- **CDC** (**C**onnected **D**evice **C**onfiguration).

1.2.1. CLDC

La CLDC 1.0 est spécifiée dans la JSR 030 (Java Specification Request 030) : elle concerne des appareils possédant des ressources faibles (moins de 512 Kb de RAM, faible vitesse du processeur, connexion réseau limitée et intermittente) et une interface utilisateur réduite (par exemple un téléphone mobile ou un PDA "bas de gamme"). Elle s'utilise sur une machine virtuelle KVM (Kilobyte Virtual Machine). La version 1.1 est le résultat des spécifications de la JSR 139 : une des améliorations les plus importantes est le support des nombres flottants.

1.2.2. CDC

La CDC est spécifiée dans la JSR 036 : elle concerne des appareils possédant des ressources plus importantes (au moins 2Mb de RAM, un processeur 32 bits, une meilleure connexion au réseau),

[1] Constitue l'outil indispensable pour tous ceux qui veulent non seulement profiter de services vidéo en temps réel et à la demande mais aussi disposer d'une large variété de services multimédias interactifs via leur connexion Internet haut débit.

par exemple un set top box ou certains PDA "haut de gamme". Elle s'utilise sur une machine virtuelle CVM (Compact Virtual Machine).

1.2.3. Remarque

La CLDC et la CDC sont deux configurations différentes et indépendantes et ne peuvent pas être utilisées en même temps. La figure suivante montre cette propriété. La CLCD n'est ni un sous-ensemble de la plateforme J2SE (développement standard) ni un sous-ensemble de la configuration CDC.

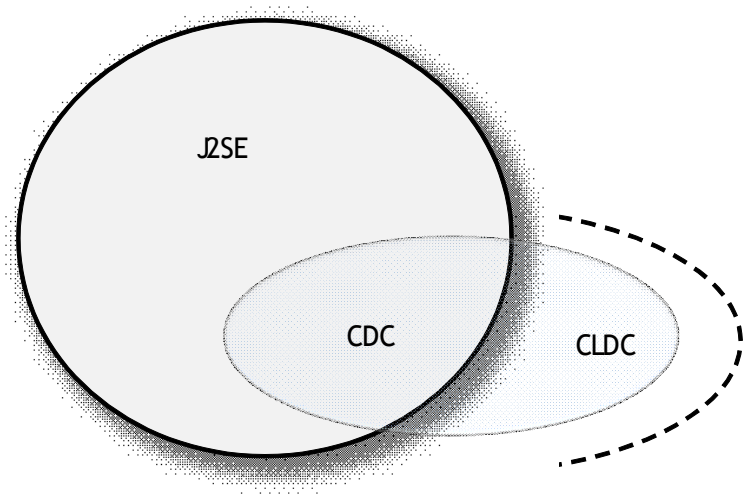


Figure 1.01 : représentation de CLDC et CDC par rapport au J2SE

1.3. Les profils [1] [13] [14] [15]

Les profils se composent d'un ensemble d'API particulier à un type de machines ou à une fonctionnalité spécifique. Ils permettent l'utilisation de fonctionnalités précises et doivent être associés à une configuration. Ils permettent donc d'assurer une certaine modularité à la plateforme J2ME.

Il existe plusieurs profils :

PROFIL	CONFIGURATION	JSR	NOTES
MIDP 1.0	CLDC	37	Package javax.microedition.*
Foundation Profile	CDC	46	
Personal Profile	CDC	62	
MIDP 2.0	CLDC	118	
Personal Basis Profile	CDC	129	
Mobile Media API (MMAPI) 1.1	CLDC	135	Permet la lecture de clips audio et vidéo
Wireless Messaging API (WMA)1.1	CLDC	120	Permet l'envoie et la réception de SMS

Tableau 1.01 : les différents profils de J2ME

Le tableau 1.02 ci dessous représente les utilisations possibles des profils:

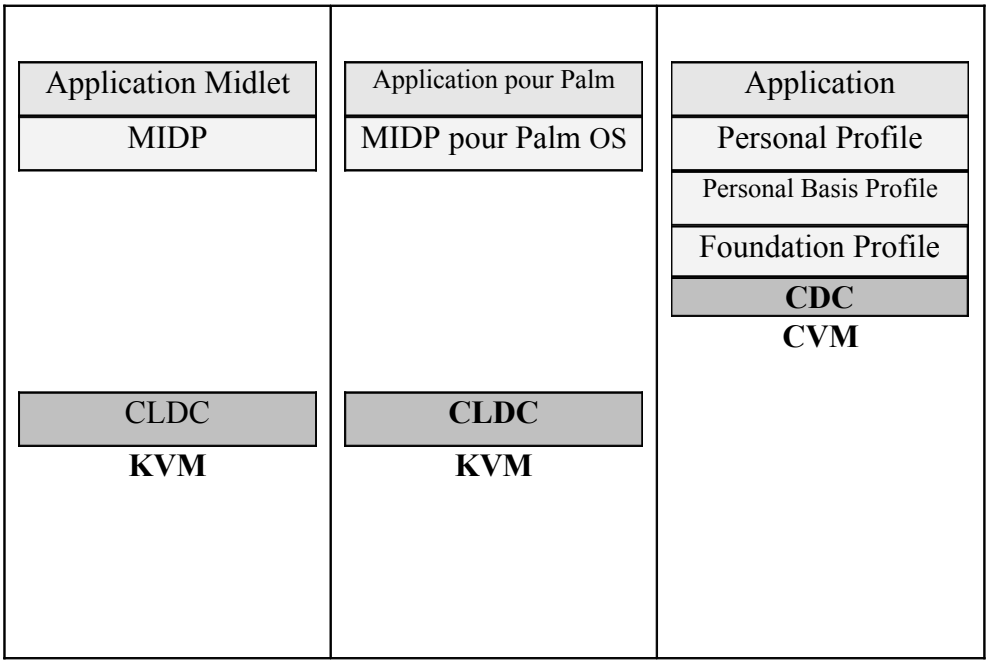


Tableau 1.02 : environnement de J2ME

MIDP est un profil standard qui n'est pas défini pour une machine particulière mais pour un ensemble de machines embarquées possédant des ressources et une interface graphique limitée.

Sun a développé un profil particulier nommé **KJava** pour le développement spécifique sur Palm. Ce profil a été remplacé par un nouveau profil nommé MIDP pour Palm OS.
Le **Foundation Profile** est un profil de base qui s'utilise avec CDC. Ce profil ne permet pas de développer des IHM (Interface Home Machine). Il faut lui associer un des deux profils supplémentaires:

- le **Personal Basis Profile** pour le développement d'application connectée avec le réseau.
- le **Personal Profile** pour le développement complet d'une IHM et d'applet grâce à AWT (Abstract Windows Toolkit).

Le **PersonalJava** est alors remplacé par le **Personal Profile**.
Le choix du ou des profils utilisés pour les développements est important car il conditionne l'exécution de l'application sur un type de machine supporté par le profil. Cette multitude de profils peut engendrer un certain nombre de problème lors de l'exécution d'une application sur différents périphériques car il n'y a pas la certitude d'avoir à disposition les profils nécessaires. Pour résoudre ce problème, une spécification particulière issue des travaux de la JSR 185 et nommée **JTWI (J**ava **T**echnology for the **W**ireless **I**ndustry**)** a été développée. Cette spécification impose aux périphériques qui la respectent de mettre en œuvres au minimum : **CLDC 1.0, MIDP 2.0, Wireless Messaging API 1.1 et Mobile Media API 1.1**. Son but est donc d'assumer une meilleure compatibilité entre les applications et les différents téléphones mobiles sur lesquelles elles s'exécutent.

1.4. J2ME en bref [10] [11] [12]

Java possède trois grandes plateformes dont J2ME est l'une des trois dédiée spécialement aux appareils de faibles capacités. La figure 1.02 illustre ses différentes plateformes.

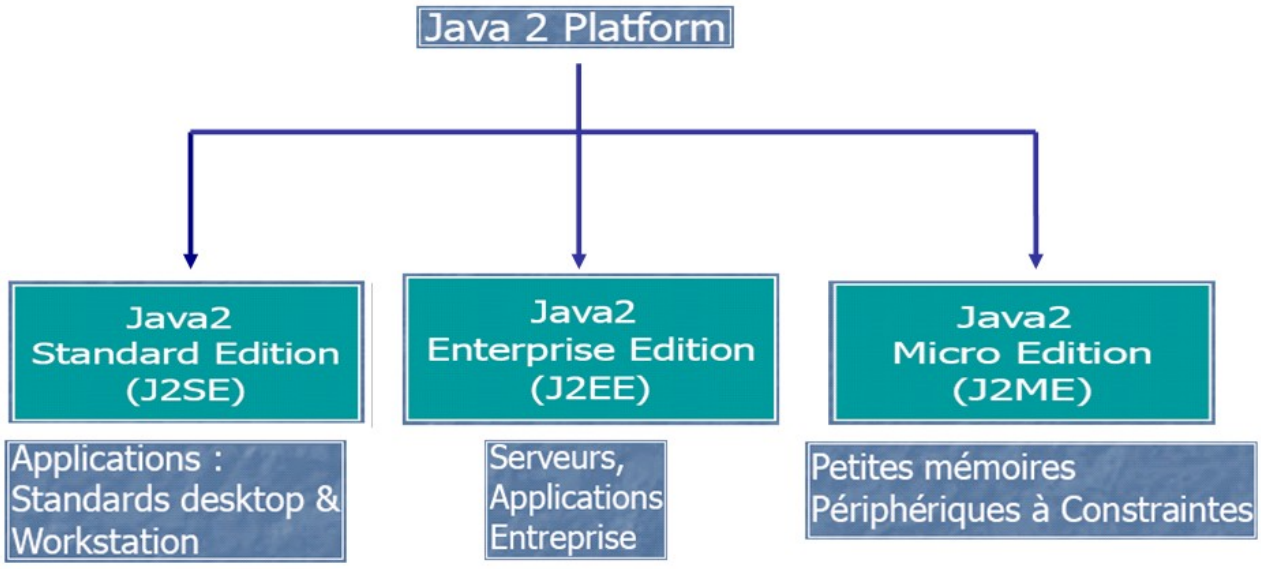


Figure 1.02 : les trois plateformes Java

Pour mieux comprendre l'environnement de J2ME la figure ci-dessous montre les différents profils et leurs configurations correspondantes.

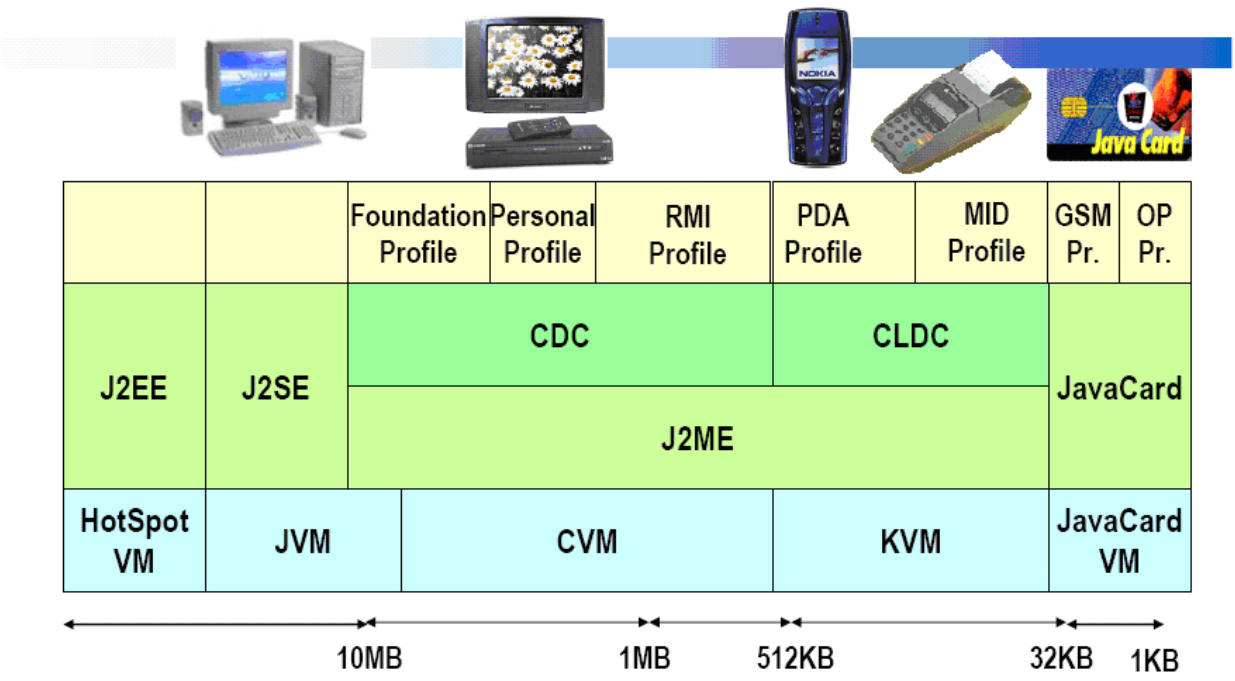


Figure 1.03 : mise en évidence de J2ME avec sa configuration et son profil

Le profil est implémenté au dessus de la configuration. Une application est construite au dessus de la configuration et du profil. Elle ne peut utiliser que les classes des librairies que la configuration et le profil fournissent.
Le profil inclut des librairies qui sont plus spécifiques pour la catégorie d'appareil.
L'ensemble formé par la configuration et le profil sur un appareil donné s'appelle un **stack**.
La figure suivante montre le concept de couches de la plateforme J2ME.

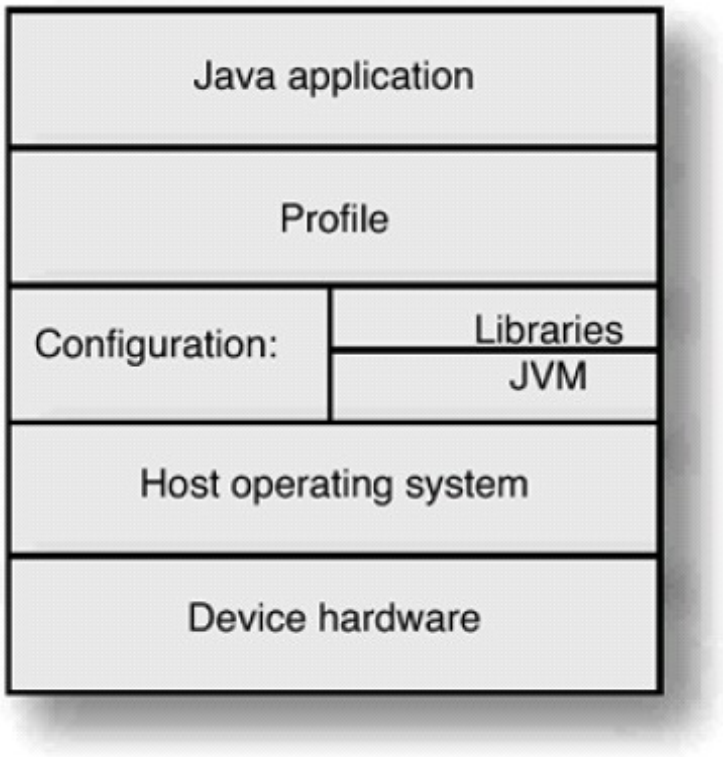


Figure 1.04 : la plateforme J2ME

Jusqu'ici, on est capable de choisir la configuration et le profil à utiliser dans une application J2ME. Mais dans toute la suite, on se consacrera sur la configuration CLDC et le profil MIDP puisqu'on s'intéresse à des téléphones mobiles. Le problème qui se pose donc, c'est de créer le code source à compiler. Pour le résoudre, le chapitre 2 parlera la technique de programmation du terminal mobile ainsi que les évolutions du profil MIDP.

CHAPITRE 2 MIDP : DEVELOPPEMENT D'UNE APPLICATION EN J2ME

C'est le premier profil qui a été développé dont l'objectif principal est le développement d'application sur des machines aux ressources et à l'interface limitées tel qu'un téléphone cellulaire. Ce profil peut aussi être utilisé pour développer des applications sur des PDA de type Palm.
L'API du MIDP 1.0 se compose des API du CDLC et de trois packages :

- **javax.microedition.midlet** : cycle de vie de l'application
- **javax.microedition.lcdui** : interface homme machine
- **javax.microedition.rms** : persistance des données

2.1. Les Midlets [1] [15]

Les applications créées avec MIDP sont des Midlets : ce sont des classes qui héritent de la classe abstraite **javax.microedition.midlet.Midlet**. Cette classe permet le dialogue entre le système et l'application.
Elle possède trois méthodes qui permettent de gérer le cycle de vie de l'application en fonction des états possibles: **startApp()**, **pauseApp()**, **destroyApp()**.

Ces trois méthodes doivent obligatoirement être redéfinies pour cause d'abstraction de la classe Midlet.

Exemple :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Test extends MIDlet
{
  public Test()
 {
 }
 public void startApp()
 {
```

```
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}
}
```

Le cycle de vie d'une midlet est semblable à celui d'une applet. Elle possède trois états:

- **paused**
- **active**
- **destroyed**

Le changement de l'état de la midlet peut être provoqué par l'environnement d'exécution ou la midlet.

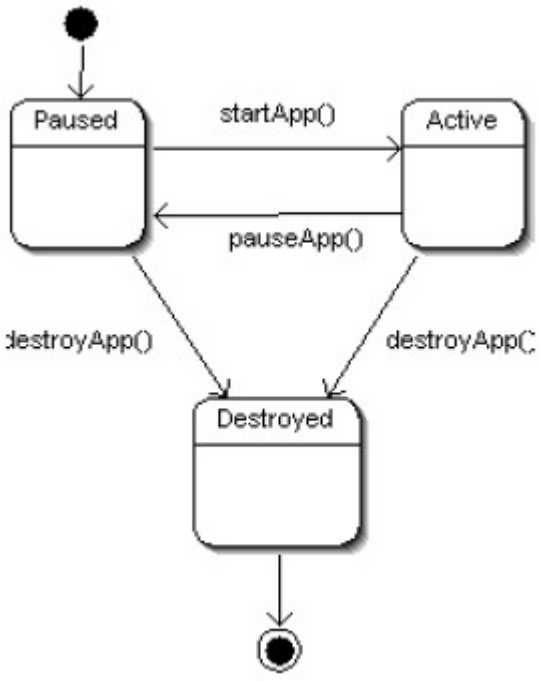


Figure 2.01 : schéma d'un cycle de vie d'une application

La méthode **startApp()** est appelée lors du démarrage ou redémarrage de la midlet. Il est important de comprendre que cette méthode est aussi appelée lors du redémarrage de la midlet : elle peut donc être appelée plusieurs fois au cours de l'exécution de la midlet.
La méthode **pauseApp()** est appelée lors de mise en pause de la midlet.
La méthode **destroyApp()** est appelée juste avant la destruction de la midlet.

2.2. Les interfaces utilisateurs [1] [5]

Les possibilités concernant l'IHM de MIDP sont très réduites pour permettre une exécution sur un maximum de machines allant du téléphone portable au PDA. Ces machines sont naturellement et physiquement pourvues de contraintes fortes concernant l'interface qu'ils proposent à leurs utilisateurs.

Avec le J2SE, deux API standards permettent le développement d'IHM : AWT et Swing. Ces deux API proposent des composants pour développer des interfaces graphiques riches de fonctionnalités avec un modèle de gestion des événements complets.
Ils prennent en compte un système de pointage par souris, avec un écran couleur possédant de nombreuses couleurs et une résolution importante.
Avec MIDP, le nombre de composants et le modèle de gestion des événements sont spartiates. Il ne prend en compte qu'un écran tactile souvent monochrome ayant une résolution très faible. Avec un clavier limité en nombres de touches et dépourvu de système de pointage, la saisie de données sur de tels appareils est particulièrement limitée.
L'API pour les interfaces utilisateurs du MIDP est regroupée dans le package **javax.microedition.lcdui**. Elle se compose des éléments de haut niveau (GUI : Graphic User Interface) et des éléments de bas niveau (jeu).

2.2.1. La classe Display

Pour pouvoir utiliser les éléments graphiques, il faut obligatoirement obtenir un objet qui encapsule l'écran. Un tel objet est du type de la classe **Display**. Cette classe possède des méthodes pour afficher les éléments graphiques.
La méthode statique **getDisplay()** renvoie une instance de la classe **Display** qui encapsule l'écran associé à la midlet fournie en paramètre de la méthode.

Exemple :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Hello extends MIDlet
{
    private Display display;
public Hello()
{
     display = Display.getDisplay(this);
}
public void startApp()
{
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}
}
```

Les éléments de l'interface graphique appartiennent à une hiérarchie d'objets : tous les éléments affichables héritent de la classe abstraite **Displayable**.

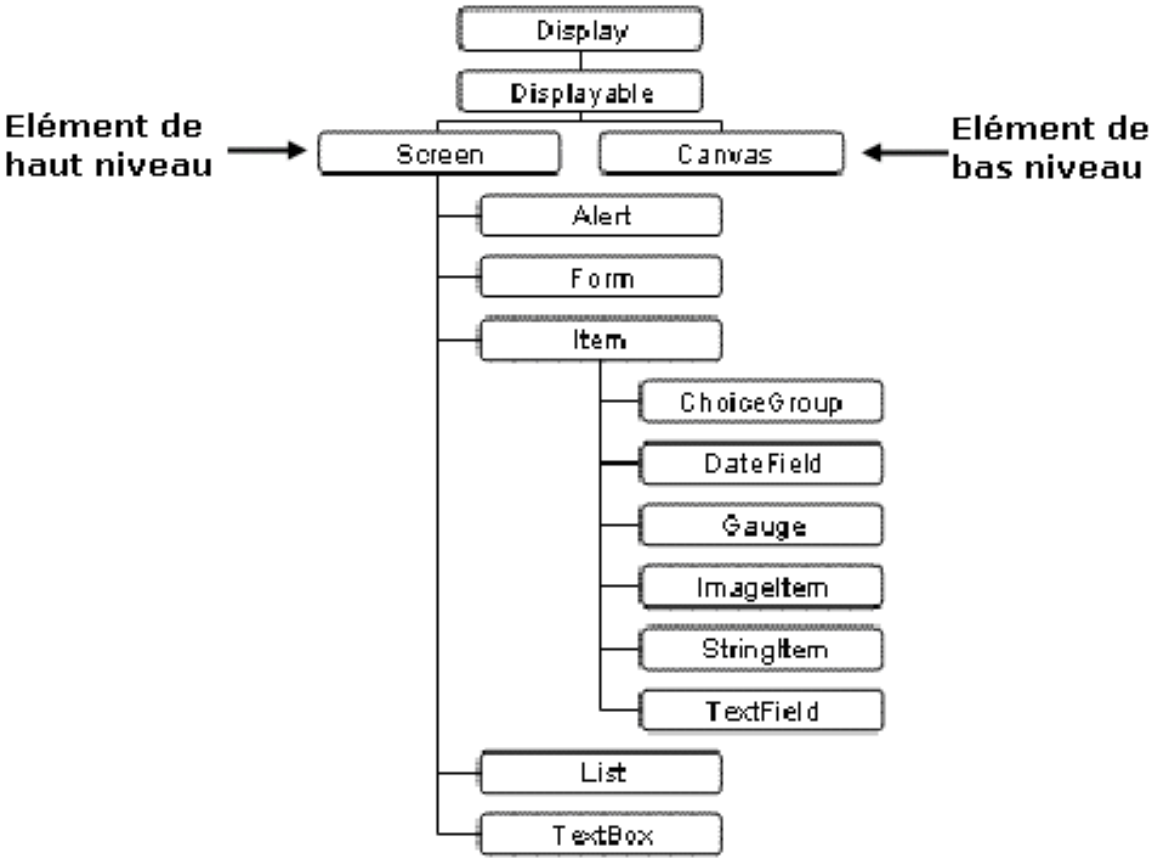


Figure 2.02 : les éléments de l'interface graphique

La classe **Screen** est la classe mère des éléments graphiques de haut niveau. La classe **Canvas** est la classe mère des éléments graphiques de bas niveau.

Il n'est pas possible d'ajouter directement un élément graphique dans un **Display** sans qu'il soit inclus dans un objet héritant de **Displayable**.

Un seul objet de type **Displayable** peut être affiché à la fois. La classe Display possède la méthode **getCurrent()** pour connaître l'objet courant affiché et la méthode **setCurrent()** pour afficher l'objet fourni en paramètre.

2.2.2. La classe TextBox

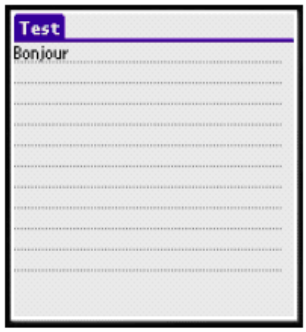
Ce composant permet de saisir du texte

Exemple :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Hello extends MIDlet
{
        private Display display;
        private TextBox textbox;
public Hello()
{
        display = Display.getDisplay(this);
        textbox = new TextBox("", "Bonjour", 20, 0);
}
public void startApp()
{
        display.setCurrent(textbox);
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}
}
```

Résultat :

sur l'émulateur Palm



sur l'émulateur de téléphone mobile

Figure 2.03 : mise en évidence d'un affichage d'un texte

2.2.3. La classe List

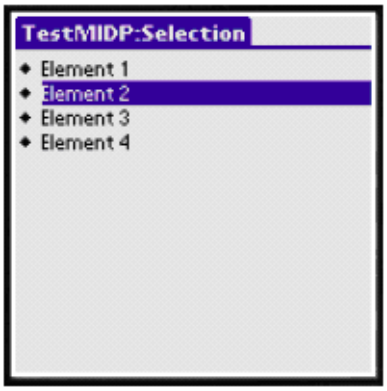
Ce composant permet la sélection d'un ou plusieurs éléments dans une liste d'éléments.

Exemple :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Test extends MIDlet
{
        private Display display;
        private List liste;
        protected static final String[] elements = {"Element 1","Element 2","Element 3","Element
        4"};
public Test()
{
        display = Display.getDisplay(this);
        liste = new List("Selection", List.EXCLUSIVE, elements, null);
}
public void startApp()
{
        display.setCurrent(liste);
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}
}
```

Résultat :

sur l'émulateur Palm



sur l'émulateur de téléphone mobile

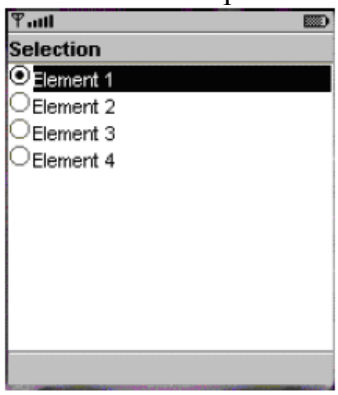


Figure 2.04 : mise en évidence d'une liste de sélection

2.2.4. La classe Form

La classe **Form** permet d'insérer dans l'élément graphique qu'elle représente d'autres éléments graphiques : cette classe sert de conteneurs. Les éléments insérés sont des objets qui héritent de la **classe abstraite Item**.

Exemple :

```
import javax.microedition.midlet.* ;
import javax.microedition.lcdui.*;
public class Hello extends MIDlet
{
        private Display display;
        private Form mainScreen;
public Hello()
{
        display = Display.getDisplay(this);
}
public void startApp()
 {
        mainScreen = new Form("Hello");
        mainScreen.append("Bonjour");
        display.setCurrent(mainScreen);
```

```
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}
}
```

2.2.5. La classe Item

La classe **javax.microedition.lcdui.Item** est la classe mère de tous les composants graphiques qui peuvent être insérés dans un objet de type Form.

Cette classe définit seulement deux méthodes: **getLabel()** et **setLabel()** qui sont le getter et le setter pour la propriété label.

Il existe plusieurs composants qui héritent de la classe Item

CLASSE	ROLE
ChoiceGroup	sélection d'un ou plusieurs éléments
DataField	affichage et saisie d'une date
Gauge	affichage d'une barre de progression
ImageItem	affichage d'une image
StringItem	affichage d'un texte
TextField	saisie d'un texte

Tableau 2.01 : les héritiers de la classe Item

Exemple :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Hello extends MIDlet
{
        private Display display;
        private Form form;
        private ChoiceGroup choiceGroup;
        private DateField dateField;
```

```
        private DateField timeField;
        private Gauge gauge;
        private StringItem stringItem;
        private TextField textField;
public Hello()
 {
        display = Display.getDisplay(this);
        form = new Form("Ma form");
        String choix[] = {"Choix 1", "Choix 2"};
        stringItem = new StringItem(null,"Mon texte");
        choiceGroup = new ChoiceGroup("Sélectionner",Choice.EXCLUSIVE,choix,null);
        dateField = new DateField("Heure",DateField.TIME);
        timeField = new DateField("Date",DateField.DATE);
        gauge = new Gauge("Avancement",true,10,1);
        textField = new TextField("Nom","Votre nom",20,0);
        form.append(stringItem);
        form.append(choiceGroup);
        form.append(timeField);
        form.append(dateField);
        form.append(gauge);
        form.append(textField);
}
public void startApp()
{
        display.setCurrent(form);
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional){}
}
```

Résultat :

sur l'émulateur Palm OS sur l'émulateur de téléphone mobile

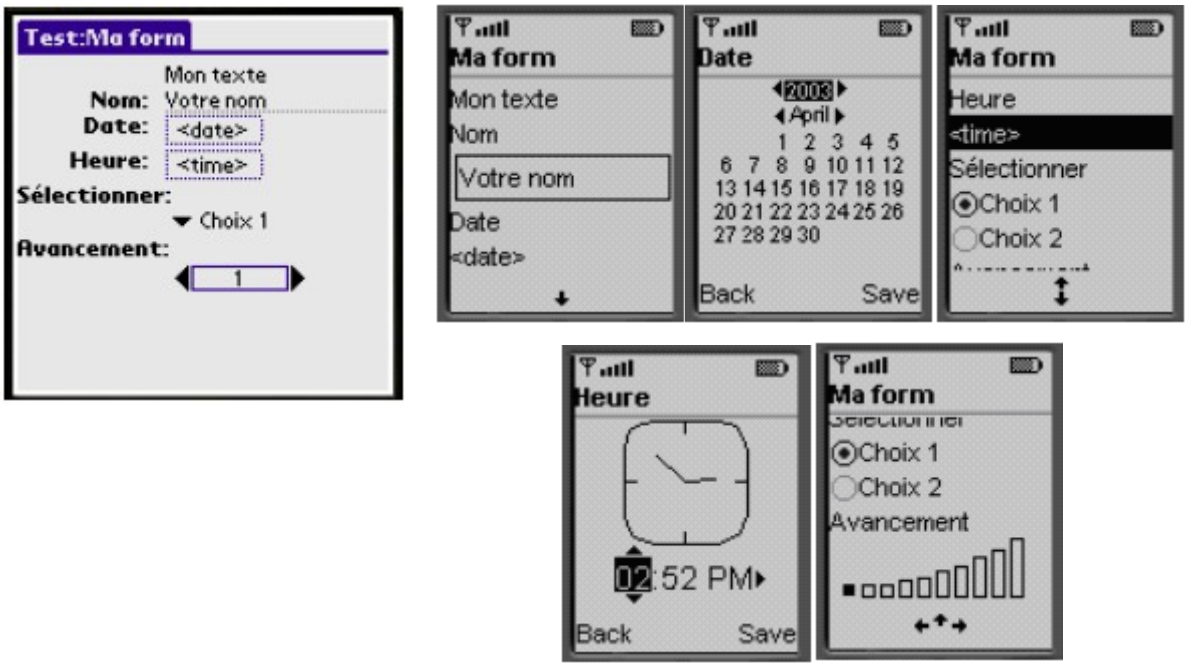


Figure 2.05 : les contenus de la classe Item

2.2.6. La classe Alert

Cette classe permet d'afficher une boîte de dialogue pendant un temps déterminé.
Elle possède deux constructeurs :

- un demandant le titre de l'objet
- un demandant le titre, le texte, l'image et le type de l'alerte utilisé.

Elle possède des getters et des setters sur chacun de ces éléments.
Pour préciser le type de la boîte de dialogue, il faut utiliser une des constantes définies dans la classe **AlertType** dans le constructeur ou dans la méthode **setType()** :

CONSTANTE	TYPE DE LA BOITE DE DIALOGUE
ALARM	Informer l'utilisateur d'un événement programmé
CONFIRMATION	Demande la confirmation à l'utilisateur
ERROR	Informer l'utilisateur d'une erreur
INFO	Informer l'utilisateur
WARNING	Informer l'utilisateur d'un avertissement

Tableau 2.02 : les différents types de constante d'alerte

Pour afficher un objet de type **Alert**, il faut utiliser une version surchargée de la méthode **setCurrent()** de l'instance de la classe Display. Cette version nécessite deux paramètres : l'objet Alert à afficher et l'objet de type Displayable qui sera affiché lorsque l'objet Alert sera fermé.
La méthode **setTimeout()** qui attend un entier en paramètre permet de préciser la durée d'affichage en milliseconde de la boîte de dialogue. Pour la rendre modale, il faut lui passer le paramètre **Alert.FOREVER**.

Exemple:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Test extends MIDlet
{
        private Display display;
        private Alert alert;
        private Form form;
        public Test() {
        display = Display.getDisplay(this);
        form = new Form("Hello");
        form.append("Bonjour");
        alert = new Alert("Erreur", "Une erreur est survenue", null, AlertType.ERROR);
        alert.setTimeout(Alert.FOREVER);
}
public void startApp()
{
        display.setCurrent(alert, form);
}
public void pauseApp() {}
public void destroyApp(boolean unconditional) {}
}
```

Résultat :

sur l'émulateur Palm OS

sur l'émulateur de téléphone mobile

Figure 2.06 : affichage d'une erreur

2.3. La gestion des événements [1]

Les interactions entre l'utilisateur et l'application se concrétisent par le traitement d'événements particuliers pour chaque action.

MIDP définit interface de type **Listener** pour la gestion des événements :

INTERFACE	ROLE
CommandListener	Ecoute pour une activation d'une commande
ItemStateListener	Ecoute pour un changement d'état d'un composant (modification du texte d'une zone de texte, …)

Tableau 2.03 : interface de type Listener

Pour la gestion des événements, la classe établie doit implémenter **CommandListener** qui exige l'existence d'une fonction **commandAction** pour l'écoute d'un événement. Son argument est la commande encours ainsi qu'un display.

Exemple :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Ecoute extends MIDlet implements CommandListener
{
        Form form;
        Command cmExit;
        ……..
public Ecoute()
{
        form=new Form("Test");
        cmExit=new Command("Exit",Command.EXIT,1);
        form.addCommand(cmExit);
        form.setCommandListener(this);
        ……
}
public void startApp()
{
        ……
        display.setCurrent(form);
}
public void pauseApp()
{
        …….
}
public void destroyApp(boolean unconditional)
{
        …….
}
public void commandAction(Command c,Displayable s)
{
      if(c==cmExit)
              {
                      destroyApp(false);
                      notifyDestroyed();
              }
```

```
        ……..
}
}
```

2.4. Le Stockage et la gestion des données [1]

Avec MIDP, le mécanisme pour la persistance des données est appelé **RMS** (**R**ecord **M**anagement **S**ystem). Il permet le stockage de données et leur accès ultérieur.

RMS propose un accès standardisé au système de stockage de la machine dans lequel s'exécute le programme. Il n'impose pas aux constructeurs la façon dont les données doivent être stockées physiquement.

Du fait de la simplicité des mécanismes utilisés, RMS ne définit qu'une seule classe : **RecordStore.** Cette classe ainsi que les interfaces et les exceptions qui composent RMS sont regroupées dans le package **javax.microedition.rms**.

Les données sont stockées dans un ensemble d'enregistrements (records). Un enregistrement est un tableau d'octets.

Chaque enregistrement possède un identifiant unique nommé **recordId** qui permet de retrouver un enregistrement particulier.

A chaque fois qu'un ensemble de données est modifié (ajout, modification ou suppression d'un enregistrement), son numéro de version est incrémenté.

Un ensemble de données est associé à un unique ensemble composé d'une ou plusieurs Midlets (Midlet Suite).

Un ensemble de données possède un nom composé de 32 caractères maximum.

L'accès aux données se fait obligatoirement en utilisant un objet de type RecordStore.

Les principales méthodes sont :

METHODE	ROLE
int addRecord(byte[], int, int)	Ajout d'un nouvel enregistrement
void closeRecordStore()	Fermer l'ensemble d'enregistrement
void deleteRecord(int)	Supprimer l'enregistrement dont l'identifiant est fourni en paramètre

static void deleteRecordStore(String)	Supprimer l'ensemble d'enregistrements dont le nom est fourni en paramètre
Enumeration enumerateRecords(RecordFilter ,RecordComparator, boolean)	Renvoyer une énumération pour parcourir tout ou partie de l'ensemble
String getName()	Renvoyer le nom de l'ensemble d'enregistrements.
int getNextRecordId()	Renvoyer l'identifiant du prochain enregistrement créé
int getNumRecords()	Renvoyer le nombre d'enregistrement contenu dans l'ensemble
byte[] getRecord(int)	Renvoyer l'enregistrement dont l'identifiant est fourni en paramètre
int getRecord(int, byte[], int)	Obtenir les données contenus dans un enregistrement dont l'identifiant est fourni en paramètre. Envoie le nombre d'octets de l'enregistrement
int getRecordSize(int)	Renvoyer la taille en octets de l'enregistrement dont l'identifiant est fourni en paramètre
int getSize()	Renvoie la taille en octets occupée par l'ensemble
static String[] listRecordStores()	Renvoyer un tableau de chaîne de caractères contenant le nom des ensembles de données associées à l'ensemble de Midlet courant
static RecordStore openrecordStore(String, boolean)	Ouvrir un ensemble de données dont le nom est fourni en paramètre. Celui ci est créé s'il n'existe pas et que le boolean est à true
void setRecord(int, byte[], int, int)	Mettre à jour l'enregistrement précisé avec les données fournies en paramètre

Tableau 2.04 : les fonctions de la classe RecordStore

Pour pouvoir utiliser un ensemble d'enregistrements, il faut utiliser la méthode statique **openRecordStore()** en fournissant le nom de l'ensemble et un booléen qui précise si l'ensemble doit être créé au cas ou celui ci n'existe pas. Elle renvoie un objet RecordStore qui encapsule l'ensemble d'enregistrements.

L'appel de cette méthode peut lever l'exception **RecordStoreNotFoundException** si l'ensemble n'est pas trouvé, **RecordStoreFullException** si l'ensemble de données est plein ou **RecordStoreException** dans les autres cas problématiques.

La méthode **closeRecordStore()** permet de fermer un ensemble précédemment ouvert. Elle peut lever les exceptions **RecordStoreNotOpenException** et **RecordStoreException**.

2.5. Packager une midlet [1]

Une application constituée d'une suite de midlets est packagée sous la forme d'une archive **.jar**. Cette archive doit contenir un fichier **manifest** et tous les éléments nécessaire à l'exécution de l'application (fichiers .class et les ressources telles que les images, ...).

2.5.1. Le fichier manifest

Ce fichier contient des informations sur l'application. Il contient aussi une définition de propriétés utilisées par l'application. Ces propriétés sont sous la forme clé/valeur.
Plusieurs propriétés sont définies par les spécifications des midlets : celles−ci commencent par MIDLet−.

Il est possible de définir ces propres attributs

PROPRIETES	ROLE
MIDlet−Name	Nom de l'application
MIDlet−Version	Numéro de version de l'application
MIDlet−Vendor	Nom du fournisseur de l'application
MIDlet−Icon	Nom du fichier .png contenant l'icône de l'application
MIDlet−Description	Description de l'application
MIDlet−Info−URL	URL des informations concernant la Midlet
MIDlet−Jar−URL	URL de téléchargement de fichier jar
MIDlet−Jar−Size	taille en octets du fichier .jar
MIDlet−Data−Profile	information sur le profil
MicroEdition−Configuration	information sur la configuration

Tableau 2.05 : définition des attributs

2.6. Evolution de MIDP et ses nouvelles options [3] [4]

2.6.1. Introduction [3]

Une nouveauté (MIDP 2.0) a été créée en 2003, une amélioration de l'ancienne version (MIDP 1.0) qui concerne les éléments suivants :

- les interfaces utilisateurs comportant un multimédia (MMAPI), une amélioration des formulaires et des images RGB
- les interfaces spéciales jeux (Game API).

MIDP 2.0 offre aussi des outils pour la sécurité tels que

- HTTPS (Hypertext Transfer Protocol Secure)
- PKI (Certificats)
- JAR (Java Archive) signé : la signature et la clé publique sont ajoutées au JAD (Java Application Descriptor)
- Permission : nouvelles entrées du JAD : **MIDlet-Permissions**, **MIDlet-Permissions-Op**

Il fait partie de l'amélioration, le lancement Push de MIDlet sur des demandes entrantes de connexions réseaux

L'entrée MIDlet-Push-<n> du JAD précise :

- L'URL local de connexion URL
- La classe de la MIDlet
- L'adresse IP de la machine autorisée à lancer un push

Exemple : MIDlet-Push-1 : socket:// :76, exemple.PushLet.192.161.60

Cependant, d'autres nouveautés apparaissent qu'on va voir dans les paragraphes qui se succèdent.

2.6.2. Service de messagerie (SMS) [4]

Ce service de messagerie aide le développeur à concevoir une application ciblant le transfert de texte dans les appareils mobiles. Une bibliothèque de base a été créée nommée : **WMA** (**W**ireless **M**essaging **A**PI).

2.6.2.1. Généralité

SMS signifie Short Message Service, que l'on pourrait traduire par service de messages courts. Ce service est disponible sur réseau GSM et permet à un utilisateur de téléphone mobile de composer un message textuel et de l'envoyer à un autre téléphone mobile.

Généralement quand on parle de messages courts on les appelle des SMS, mais ceci est un abus de langage car SMS désigne le service. Un message devrait être désigné par l'acronyme SM mais pour ne pas perturber la majorité des lecteurs, on les appellera tout de même SMS.

Ces messages peuvent contenir au plus 160 caractères et leur format est défini par une recommandation de GSM

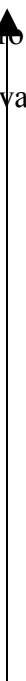
2.6.2.2. Utilisation dans J2ME

L'envoi et la réception de SMS ne sont pas directement une fonctionnalité du profil MIDP. Cela se comprend, car certains appareils compatibles MIDP, n'ont pas la possibilité d'envoyer de tels messages.

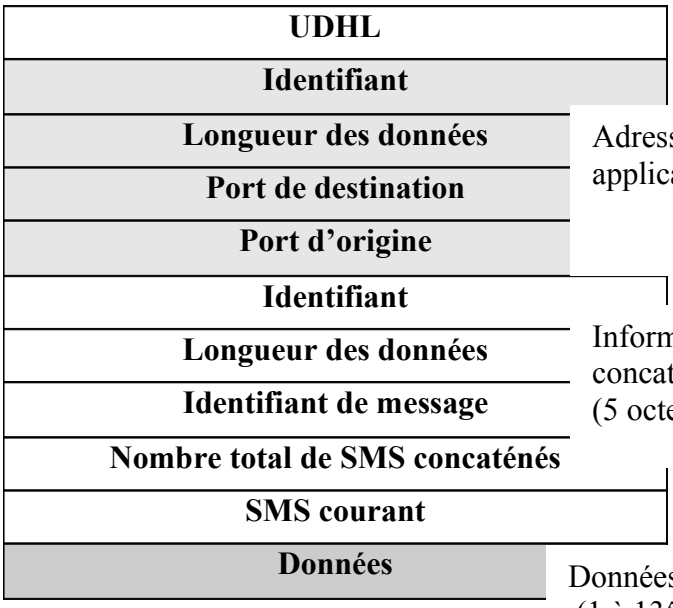
Pour pouvoir utiliser le service de message court, il faut que le terminal supporte le paquetage optionnel « Wireless Messaging Api 1.0 JSR 120». Grâce à la librairie fournie par *Sun*, la compatibilité avec passablement de téléphones portables modernes est garantie.

2.6.2.3. Numéro de port

Le schéma suivant montre la structure d'un message SMS.



En-tête (1 octet)



de port. Ce numéro de port a exactement le même rôle qu'un num

sert à différencier l'application destinatrice sur un téléphone. Ainsi plusieurs applications peuvent utiliser le service SMS sur un même téléphone et cela sans risque de voir leurs messages se mélanger.

2.6.2.4. Réception de message

Lors de tests, il est vite apparu, qu'il n'est pas possible de réceptionner les SMS « standards » qui sont envoyés sur le téléphone. De même qu'une application ne peut pas capter les messages TCP qui ne lui sont pas destinés, il est impossible de réceptionner les SMS qui sont destinés à une autre application.

Si la possibilité d'intercepter les SMS « standards » avait été laissée, on aurait eu des problèmes de concurrence. Imaginer deux applications qui attendent un SMS, comment le système aurait pu savoir à qui délivrer les messages entrants.

La solution consiste à écouter l'arrivée de SMS sur un autre port que le port standard. Comme pour TCP/UDP, il existe aussi 2^16 ports, on n'a donc le choix pour choisir un port pour son application.

Mais bien que l'utilisation de n'importe quel numéro de ports puisse fonctionner, il faut tout de même se référer à la spécification GSM qui précise que seuls les numéros de ports de 16000 à 16999 sont réservés aux applications.

Le gros inconvénient d'écouter sur un autre port, c'est qu'il faut aussi envoyer le message de commande sur ce même port. Et puisqu'on sait qu'aucun téléphone ne permet de sélectionner le

port lors de l'envoi d'un SMS. Il va falloir créer une deuxième application pour l'envoi des messages de commande.
Ce point est particulièrement ennuyeux, pas du fait qu'il faille développer une deuxième application, mais parce que celle-ci devra être installée sur un téléphone compatible avec la librairie WMA. Ce qui réduit quand même passablement la liste des appareils utilisables.
Ainsi on passe d'une compatibilité complète (tous les téléphones portables) à une compatibilité réduite aux téléphones modernes.

2.6.3. Multimédia [4]

Le multimédia propose la mise en œuvre de son, image, vidéo. C'est dans cette partie qu'on essaie de développer comment lire un fichier audio ou vidéo avec les mobiles. Mobile Media API (MMAPI) rend un grand service à partir duquel on peut lire les fichiers ayant les extensions .midi, .mpeg1, .mp3, etc.

2.6.3.1. Mobile Media API

MMAPI autorise la génération de tonalité, playback, enregistrement et capture. Il tient compte aussi des contraintes de mémoire sur des plateformes CLDC. Concernant son extensibilité, des nouvelles fonctionnalités peuvent être rajoutées ; d'autres formats de contenu audio, vidéo peuvent être supportés.

2.6.3.2. Gestion et traitement du contenu

La gestion consiste à distinguer deux grandes familles :

- **Gestion du protocole** (Protocol Handling) qui lit les données d'une source (exemple : fichier, serveur (réseau)).
- **Gestion du contenu** (content Handling) : une fois lu, le contenu est décodé et restitué via une sortie du périphérique (haut parleur pour le son, écran pour la vidéo).

Trois classes essentielles travaillent dépendamment les unes des autres pour faire marcher un multimédia :

- **Player** : lit les données d'une source, les traite et les restitue.
- **Manager** : pour créer des players, jouer des tonalités, contient aussi les protocoles et les types de contenus que le mobile supporte.
- **DataSource** : définit comment des données d'une source (fichier) sont lues. On peut créer un nouveau DataSource (pour un nouveau format par exemple)

L'arborescence de la figure 2.08 présente les interactions entre elles :

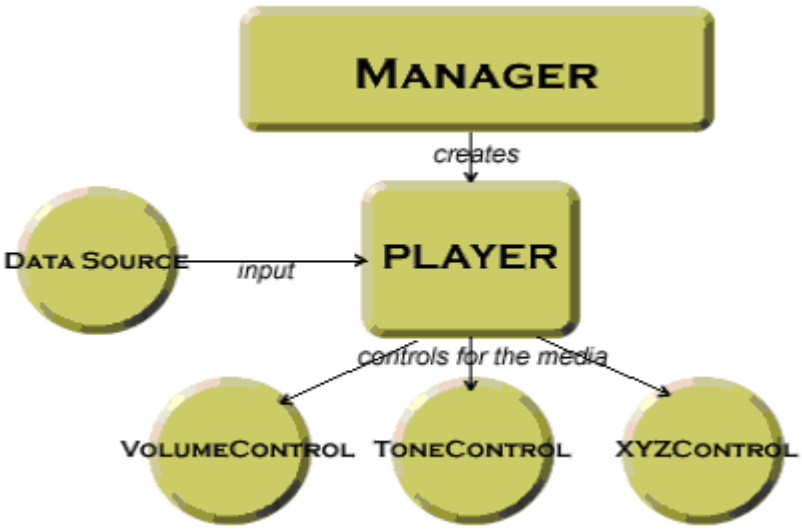


Figure 2.08 : liaison entre Manager, Player, DataSource

2.6.3.3. La classe Manager

C'est la classe centrale pour créer des «players » et offre trois méthodes pour indiquer la source de données :

- **createPlayer (DataSource source)**
- **createPlayer (InputStream stream, String type)**
- **createPlayer(String locator)**

Exemples:

Player p= Manager.createPlayer (“http://nokia.com/chanson.mp3”);

Player p= Manager.createPlayer (“http://movies.com/film.mpg”);

Player p= Manager.createPlayer (“capture: //vidéo”);

2.6.3.4. La classe Player

La classe Player est créée à partir de celle de Manager. Elle permet de restituer un contenu multimédia et possède différents états tel que : UNREALIZED, REALIZED, PREFETCHED, STARTED, et CLOSED.

- unrealized: état initial, juste après création
- realized : initialise les informations sur les medias (type, périphérique de capture, ect).
- prefetched : établit la connexion avec le média et la charge, établit les connexions réseau et la charge.
- started : mode lecture

Le passage d’un état à un autre se fait par intermédiaire d’une fonction comme :

- **realized()**
- **prefetch()**
- **start()**
- **stop()**
- **deallocate()**
- **close()**

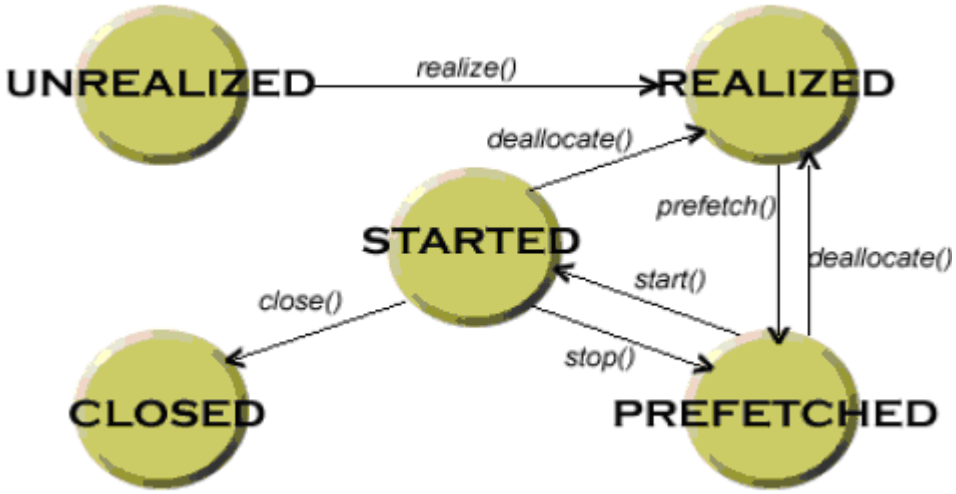


Figure 2.09 : interaction entre les états

2.6.3.5. Snapshot

A partir d'un clip sur l'écran, il est possible de capturer des images et les afficher à l'aide d'un **ImageItem**. La méthode **getSnaphot** sur un objet **VideoControl** effectue cette opération.

2.6.3.5.1. Capture d'une image

La capture d'image n'étant bien entendu possible qu'avec des appareils munis d'un multimédia, il est normal que cette option ne fasse pas partie des fonctions de base de MIDP. En fait, les fonctions se rapportant à la capture d'image, sont disponibles au travers d'un package optionnel: Mobile Média API (MMAPI).

2.6.3.5.2. Fonctionnement

Pour prendre une photo, il faut premièrement crée un objet du type « **Player** », que l'on trouve dans la librairie MMAPI. Ensuite il faut l'initialiser selon la figure 2.09.

Quand on a crée l'objet, il faut successivement appeler ces méthodes realize(), prefetch(), et start().

Après avoir initialisé les éléments de pilotage, il suffit de faire appel à la méthode **getSnapshot()** pour obtenir une photo.

2.6.3.5.3. Problème rencontré

Image floue et bleue

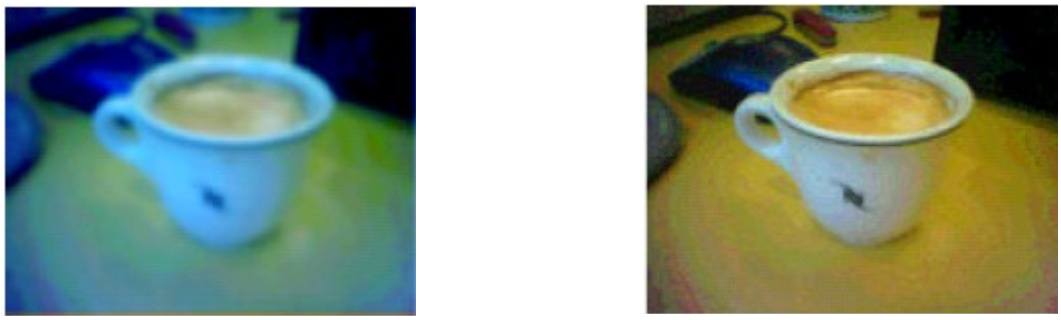


Figure 2.10 : différence entre la qualité de l'image capturée et celle de l'originale

A chaque capture, l'image était de très mauvaise qualité et de couleur bleutée (photo de gauche). Ce problème venait du fait qu'une fois démarrée la méthode start(), la capture a besoin d'un petit moment pour se préparer. Si à ce moment là on la fait, la qualité de l'image est dégradée. Alors il suffisait donc de patienter une seconde entre le moment d'ouverture et la prise de la photo.

2.7. Outils de développement de MIDP [3]

Plusieurs outils sont disponibles pour la mise en œuvre, mais les plus connus sont les suivants :

OUTILS	DESCRIPTION
J2ME Wireless Toolkit	Simple logiciel spécial mobile, sans éditeur de texte
Sun ONE Studio Mobile Edition	Outil plus conforme, les mots réservés sont colorés. Il existe des codes par défaut lors de création d'un nouveau fichier.
Nokia Developer's Suite Beta J2ME	Spéciale J2ME, avec un éditeur de texte. Tous les émulateurs portent la marque Nokia.
BlackBerry Java Development Environment	Logiciel Java capable de compiler les applications J2ME.

Tableau 2.06 : outils de développement de MIDP

Les figures qui se succèdent représentent quelques outils de développement du MIDP

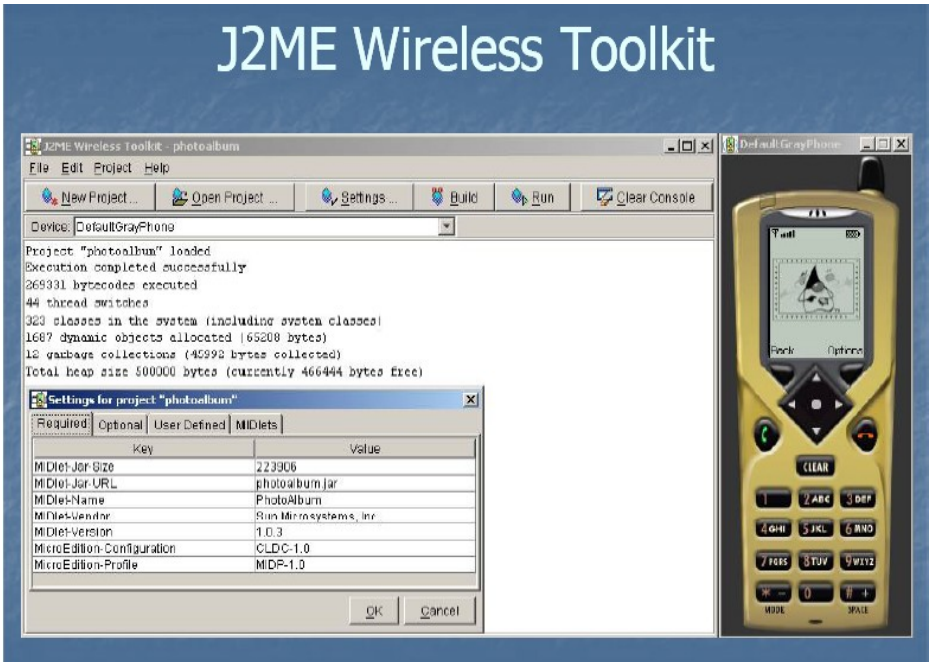


Figure 2.11: J2ME Wireless Toolkit

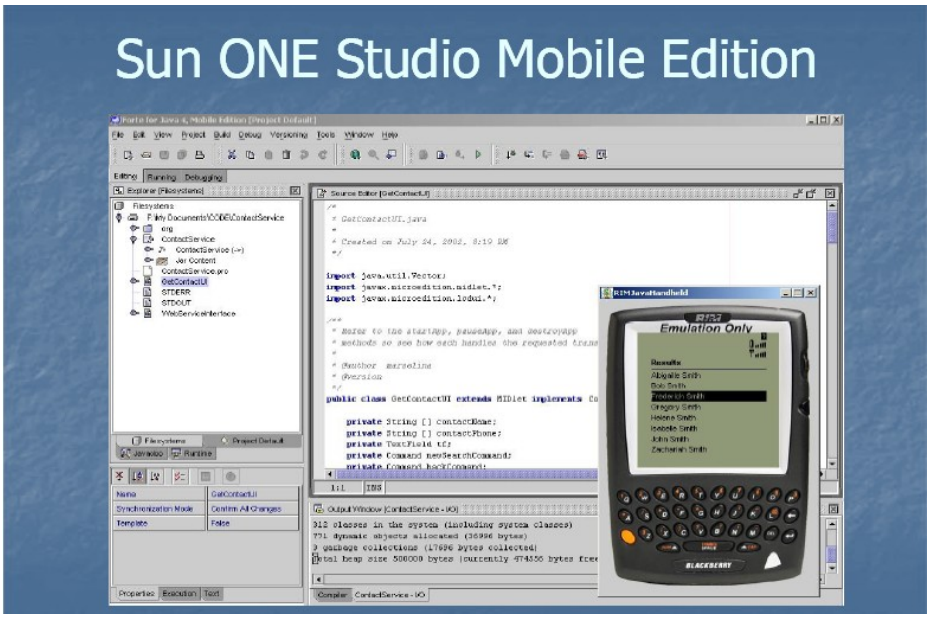


Figure 2.12: Sun One Studio Mobile Edition

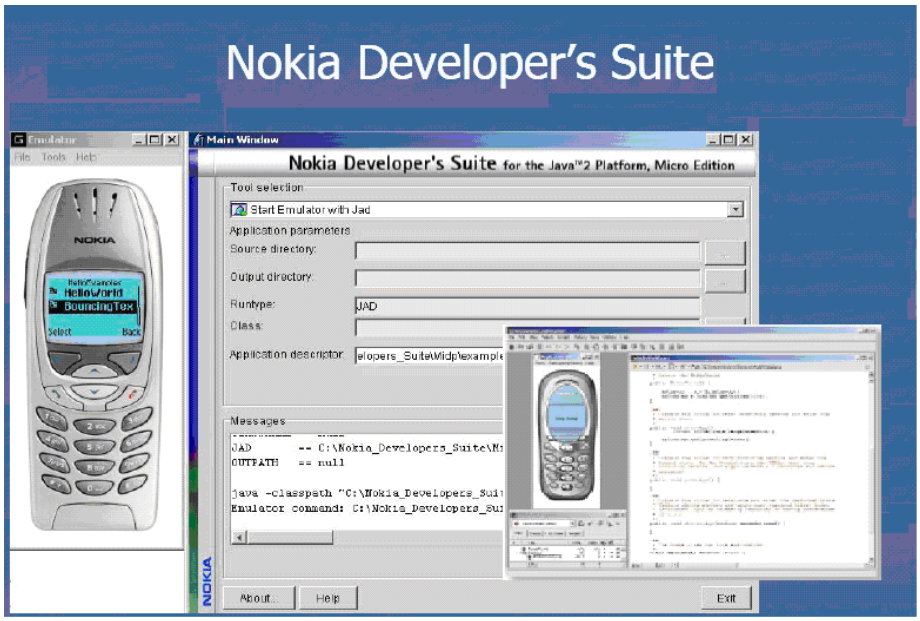


Figure 2.13: Nokia Developer's Suite Beta J2ME

Figure 2.14 : BlackBerry Java Development Environment

Chaque logiciel offre quatre sortes d'émulateurs. Parmi eux, J2ME Wireless Toolkit ne dispose pas d'un éditeur de texte. Il est possible de choisir le profil convenable à l'application à développer.

Sun One Studio Mobile Edition est le plus facile à utiliser puisqu'il existe déjà des codes gérés automatiquement lors de création d'une application.

Maintenant, tous les problèmes sont résolus concernant la J2ME (choix du profil et la configuration) et le MIDP, il nous reste que la réalisation (chapitre 3).

CHAPITRE 3 REALISATION

3.1. Objectif de la conception

Auparavant, presque les systèmes embarqués sont programmés avec le langage C++, mais cela n'empêche plus d'autre langage de s'y intégrer. Ici le but est de programmer un téléphone portable qui sera facile à manipuler par n'importe quel utilisateur. Par rapport aux téléphones mobiles standards, celui-ci sera capable de lire des fichiers audio et vidéo de diverses extensions. En outre, la langue malagasy sera aussi présente comme l'un des langages d'instructions.

3.2. Langage de programmation

Java possède trois grandes plateformes dont l'une entre elles est la J2ME, il est donc évident qu'il soit notre langage de programmation.

3.2.1. Présentation du langage Java

Le langage Java a été conçu en 1995 par James Gosling de *Sun Microsystems*. C'est un langage moderne orienté objet ressemblant fortement au C++. Etant donné que le langage C++ comportait trop de difficultés, James Gosling décida de créer un langage orienté objet reprenant les caractéristiques principales du C++, mais éliminant ses points difficiles, et en le rendant moins encombrant et plus portable (il devait pouvoir être intégré dans n'importe quel appareil …).

De plus, l'Internet rassemblant sur une même structure des machines différentes, il fallait un langage capable de fonctionner sur chacune d'elles : Java était conçu pour être portable ; le web était limité en bande passante et Java était conçu pour être petit.

Le fichier source d'un programme écrit en Java est un simple fichier texte dont l'extension est par convention .java. Ce fichier source doit être un fichier texte sans mise en forme particulière ou caractères spéciaux, c'est à dire qu'il contient uniquement les ASCII de base.

Lorsque le programme est prêt à être essayé, il s'agit de le traduire en langage machine (le compiler) à l'aide d'un compilateur. Toutefois, contrairement aux langages compilés traditionnels, pour lesquels le compilateur crée un fichier binaire directement exécutable par un processeur donné (c'est à dire un fichier binaire contenant des instructions spécifiques à un processeur), le

code source Java est compilé en un langage intermédiaire (appelé pseudo code ou byte code) dans un fichier portant le même nom que le fichier source à l'exception de son extension (. class).
Cette caractéristique est majeur, car c'est elle qui fait qu'un programme écrit en Java est portable, c'est à dire qu'il ne dépend pas d'une plateforme donnée. En réalité, le code intermédiaire n'est exécutable sur aucune plateforme sans la présence d'une machine virtuelle, l'interpréteur tournant sur une plateforme donnée, et capable d'interpréter le code intermédiaire.
La figure suivante met en évidence ce principe :

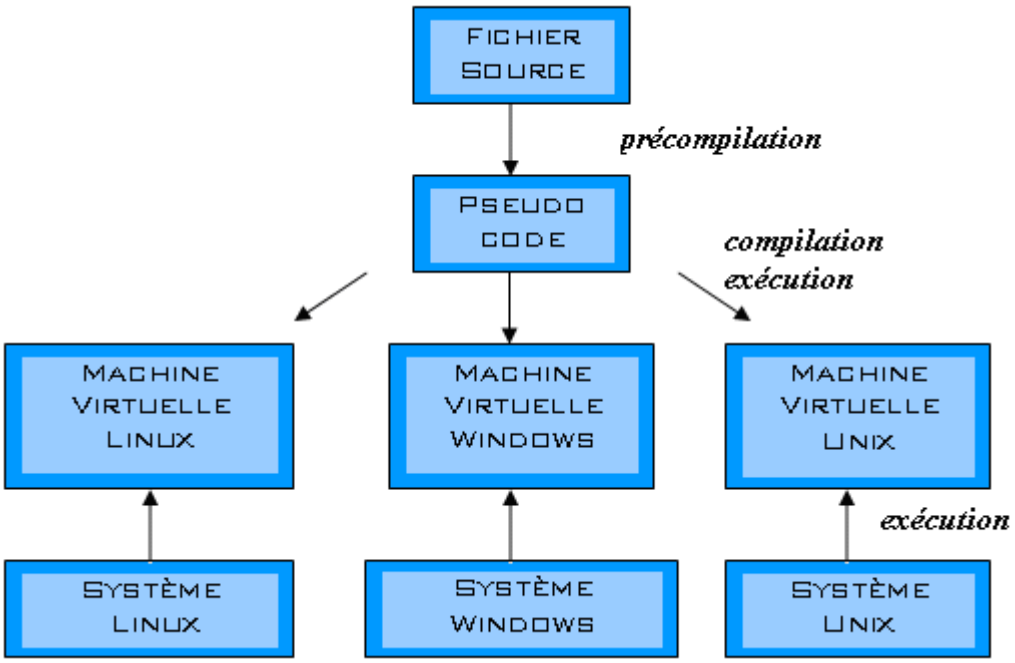


Figure 3.01 : compilation d'un programme Java

3.2.2. Justification du choix du langage

Outre les qualités mentionnées ci-dessus, le choix de ce langage pour la conception repose sur le fait que le langage a été étudié depuis la troisième année. Pour de raison économique, les outils servant à développer Java sont téléchargés gratuitement sur Internet avec des documents d'apprentissages.

3.3. Outil de simulation du terminal mobile

Le logiciel de simulation est le **Sun Java(TM) Wireless Toolkit 2.5 for CLDC** conçu par la société *Sun Microsystems*. Il est téléchargeable gratuitement sur Internet avec une taille de 40 Mo.

3.3.1. Installation

L'installation du logiciel est détaillée dans la documentation fournie avec l'émulateur pour chaque type de système d'exploitation. Dans notre cas, on se contente du système d'exploitation Windows.

Il faut remarquer que l'installation du Toolkit exige au minimum la présence de JDK 1.5 pour que l'émulateur fonctionne correctement.

L'installation de l'émulateur se fait par l'exécution du programme **sun_java_wireless_toolkit-2_5**. Après, on suit tout simplement les instructions d'installation du produit. Les instructions de l'installation consistent à accepter les termes de licence du produit et à indiquer l'emplacement du JVM installé sur la machine (JDK 1.5 au minimum).

3.3.2. Développement d'une Midlet avec le logiciel

3.3.2.1. Les projets

La construction d'une MIDlet s'articule autour d'un projet du Toolkit. Le résultat final du projet est une MIDlet qui est portable sur les appareils mobiles sous condition du choix du profil et de la configuration. Un projet contient tous les fichiers nécessaires à la construction d'une MIDlet tels que les fichiers source Java, les fichiers de ressources (image, vidéo, audio) et même les descripteurs de MIDlet.

Pour créer un projet avec l'émulateur Sun Java(TM) Wireless Toolkit 2.5 for CLDC, on lance l'émulateur par l'élément appelé **ktoolbar** qui se situe dans le menu démarrer > tous les programmes> Sun Java(TM) Wireless Toolkit 2.5 > ktoolbar. La fenêtre présentée sur la figure 3.02 apparaît sur l'écran.

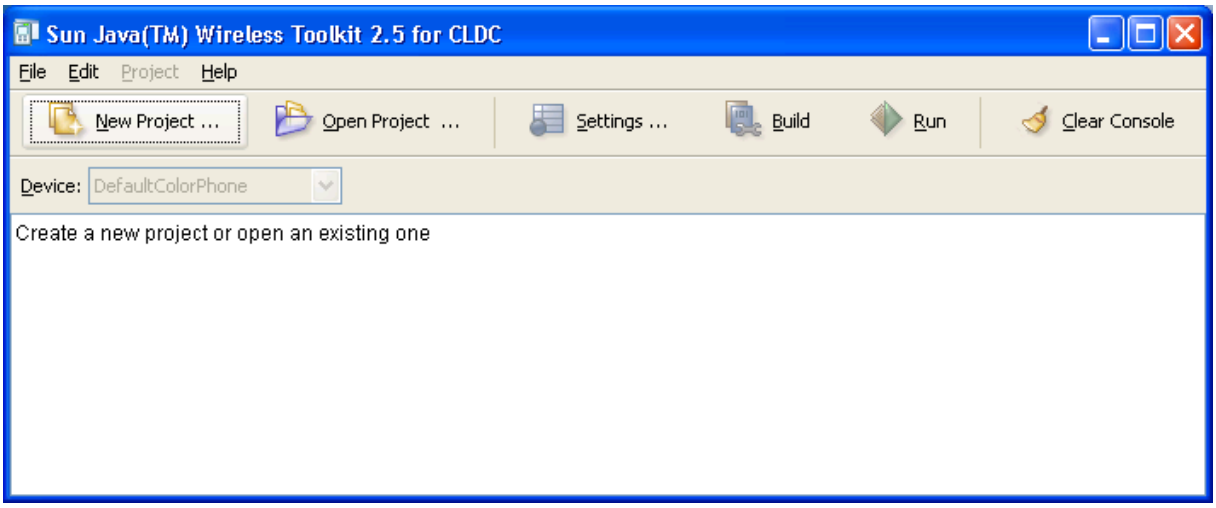


Figure 3.02 : la console d'administration du logiciel

Ensuite pour créer un projet, appuyer sur **new project**, une nouvelle fenêtre apparaît pour saisir le nom du projet et le nom de la classe principale.

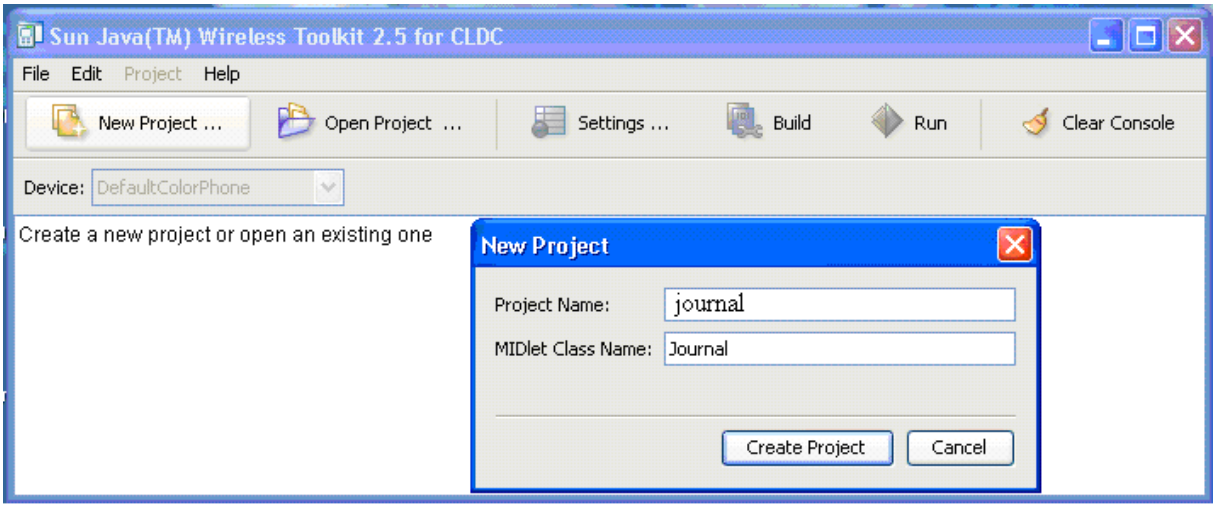


Figure 3.03 : création d'un projet

Le nom du projet est journal et le nom de la classe principale est Journal. Après l'appui sur le bouton **Create Project**, le projet est crée.

3.3.2.2. L'emplacement des fichiers

Après la création d'un projet, des répertoires sont crées automatiquement pour stocker les ressources de l'application, les codes sources, les .class, les .jar et .jad, ainsi que les fichiers contenant la base de données. La figure ci-dessous explique cet emplacement :

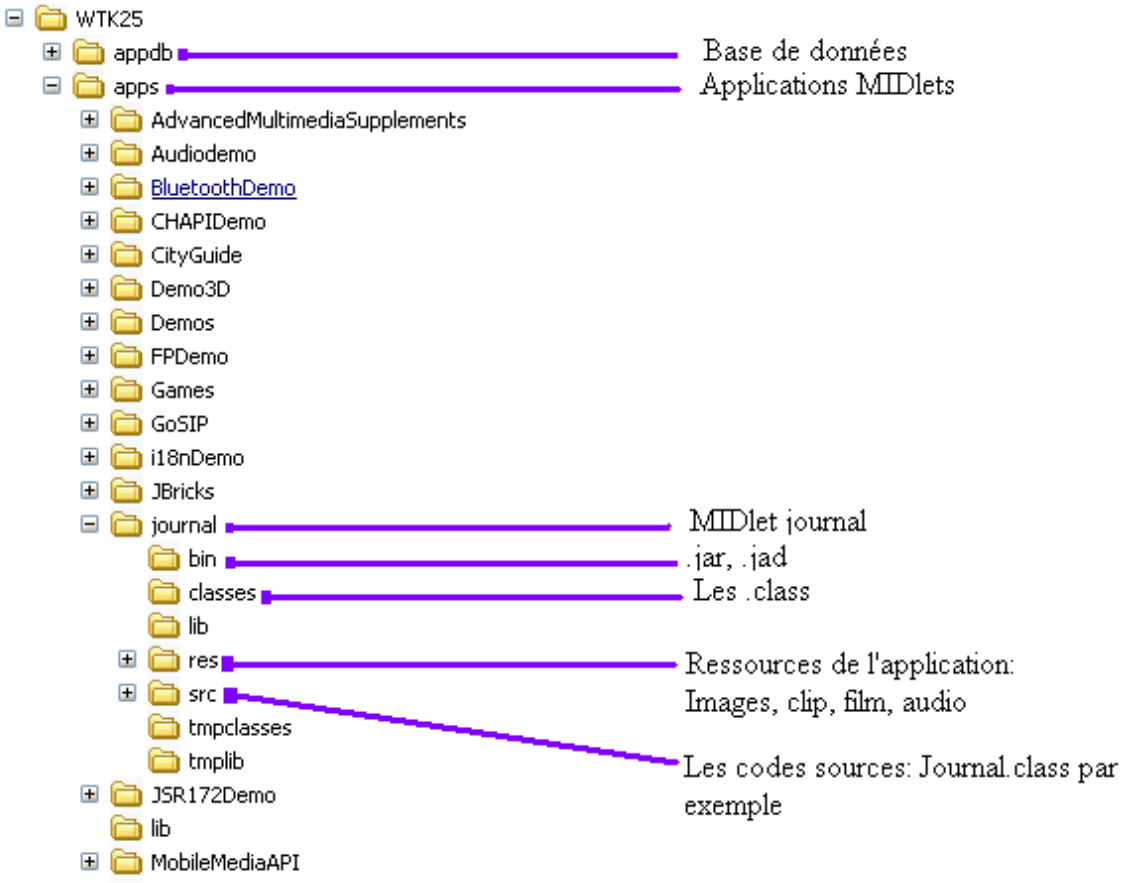


Figure 3.04 : hiérarchie des répertoires

3.3.2.3. Cycle simple de développement d'une MIDlet

Le cycle simple correspond à l'édition du code source, compilation de ce code source par l'émulateur, et s'il n'y a pas d'erreur de syntaxe, le programme s'exécute sur l'émulateur. Au cas où le code source contient un ou plusieurs erreurs, la compilation ne se fait pas et des messages d'erreurs apparaît sur la console, il faut alors corriger les erreurs sur le code source avant de lancer une nouvelle compilation.

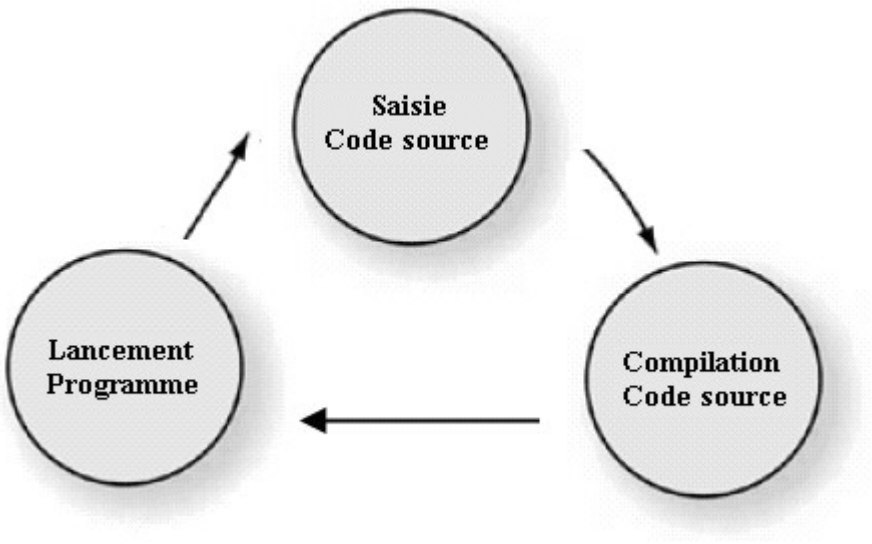


Figure 3.05 : cycle simple de développement d'une MIDlet

Notons que le logiciel ne dispose pas d'éditeur de code source, alors nous faisons appel à d'autres éditeurs de texte comme **bloc notes** ou **JCreator LE** et par la suite, enregistrer le code source avec l'extension .java

3.3.2.4. Cycle de développement complet d'une MIDlet

Le cycle simple de développement d'une MIDlet s'arrête jusqu'à la création des classes correspondant à chaque code source. En effet, l'émulateur n'exécute que ces classes compilées et non pas des produits MIDlets finis.

Le cycle de développement complet d'une MIDlet se fait en suivant les étapes suivantes : lancement de l'émulateur, création d'un projet, construction du projet, création des fichiers avec des extensions **.jar et .jad**. Ces derniers fichiers construits sont les MIDlets qu'on peut introduire dans les terminales mobiles réelles.

3.4. Déploiement d'une MIDlet sur un terminal mobile

Une MIDlet Java peut être déployée sur un périphérique mobile :

- soit directement par liaison USB entre un ordinateur et le mobile en utilisant le logiciel de téléchargement (Oxygène phone manager, NOKIA phone manager,…).

- soit par l'OTA (Over The Air) : téléchargement à partir d'un serveur en utilisant l'URL de la MIDlet.

3.5. Les menus offerts à l'utilisateur

La cardinalité du menu offert est au nombre de dix: aide, bonus, clip, configuration, jeux, journal, message, musique, répertoire, sonnerie.

3.5.1. Aide

Comme son nom l'indique, ce menu viendra en aide aux usagers en tant que manuelle d'utilisation. Son contenu représentera différentes suites d'instructions.

3.5.2. Bonus

Il nous ferra voir les besoins usuels de l'utilisateur. Dans notre cas, on va offrir : l'heure, calendrier, et une calculatrice.

3.5.3. Clip

Le lecteur sera capable de lire des fichiers d'extensions diverses. La durée du chargement dépendra de la taille du fichier. Un clip apparaîtra tout de suite sur l'écran ; cependant il sera possible de capturer une image puis y afficher directement.
Le passage vers la lecture du fichier suivante ou précédente ou stopper un clip sera à la disposition des utilisateurs, la visualisation en plein écran sera un atout.

3.5.4. Configuration

Deux langues étrangères sont reconnues internationales : l'anglais et le français, alors pour que l'application puisse utiliser par divers pays, on doit employer ses deux langages.
L'objectif de la configuration est de changer le langage selon votre choix. En plus, notre langue maternelle sera présente afin de surmonter la domination des autres.

3.5.5. Jeux

Notre jeu sera un vrai divertissement puisqu'il exige une concentration, une réflexion totale du joueur. Ici, ce n'est pas l'habitude qui compte mais l'intelligence. Ce jeu sera destiné à toutes les catégories d'âge et proposera beaucoup d'option comme facile et difficile. On pourra configurer aussi les touches de direction suivant votre préférence.

Le jeu consistera à former quatre mots repartis dans 15 cases et une case vacante pour le déplacement. Les mots à trouver seront les suivants :

- JAVA : notre langage de programmation
- J2ME : Java 2 Micro Edition
- MIDP : Mobile Information Device Profile
- TCO : Abréviation de la télécommunication

J	A	V	A
J	2	M	E
M	I	D	P
T	C	O	

Figure 3.06: présentation du jeu

3.5.6. Journal

Ce sera dans ce menu qu'on trouve les historiques des appels émis et reçus avec leurs coûts respectifs ainsi qu'un bloc-notes pour mémoriser certain événement.

3.5.7. Message

Il sera possible d'envoyer et de recevoir un message texte (sms) à partir de la station mobile. Lors de la réception (respectivement l'envoie), le message sera stocké dans la boîte de réception (respectivement dans la boîte d'envoie). En plus, l'utilisateur ne sera pas obligé de retenir les numéros de son correspondant avant d'envoyer un message puisqu'on pourra parcourir directement dans le menu répertoire.

3.5.8. Musique

Dans ce menu musique, on se contentera seulement d'écouter un fichier audio. Puisqu'il n'y a plus des animations sur l'écran, alors on les remplacera par une image animée (.gif), juste pour la bonne présentation de l'interface utilisateur.

3.5.9. Répertoire

Ce sera inutile de mémoriser tous les numéros de votre correspondant et de composer son numéro avant appel. La conception de ce menu est basée sur la manipulation de base de données. Cette manipulation consistera à supprimer, modifier ou ajouter un correspondant.

3.5.10. Sonnerie

Lors d'une arrivée d'un appel, d'un message, une alerte sonore préviendra l'utilisateur. C'est à ce moment là où on aura besoin de la sonnerie. En plus, ce menu sera responsable du réglage du volume sauf pour la musique et le clip.

3.6. Organigrammes des menus

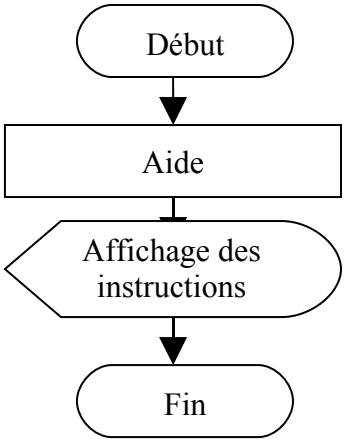


Figure 3.07 : organigramme du menu aide

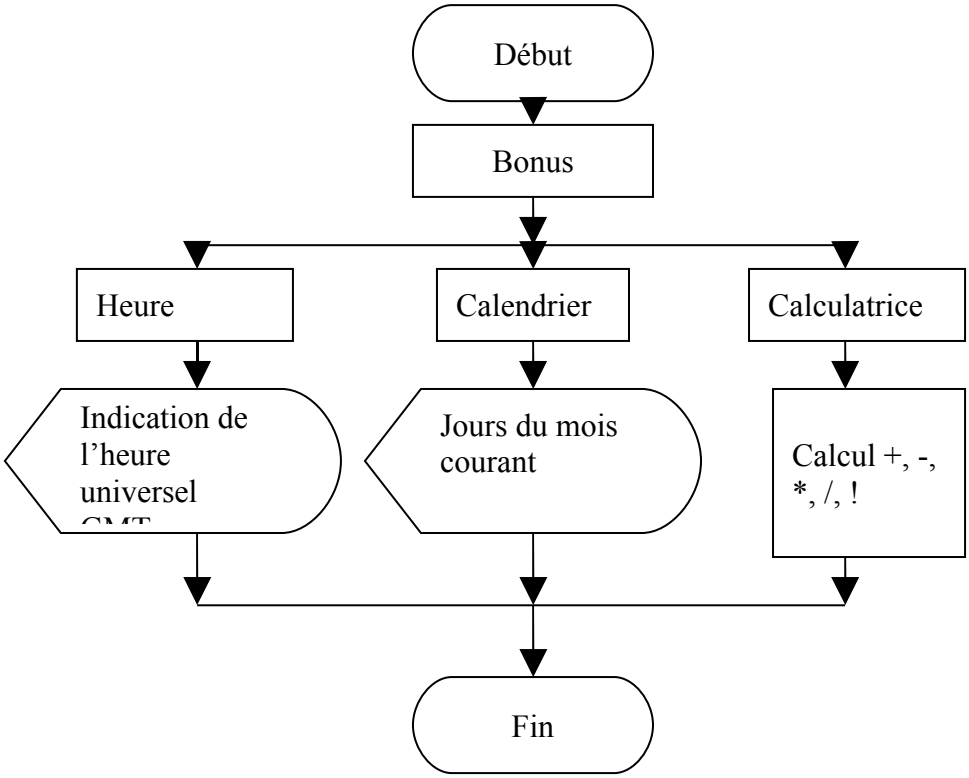


Figure 3.08 : organigramme du menu bonus

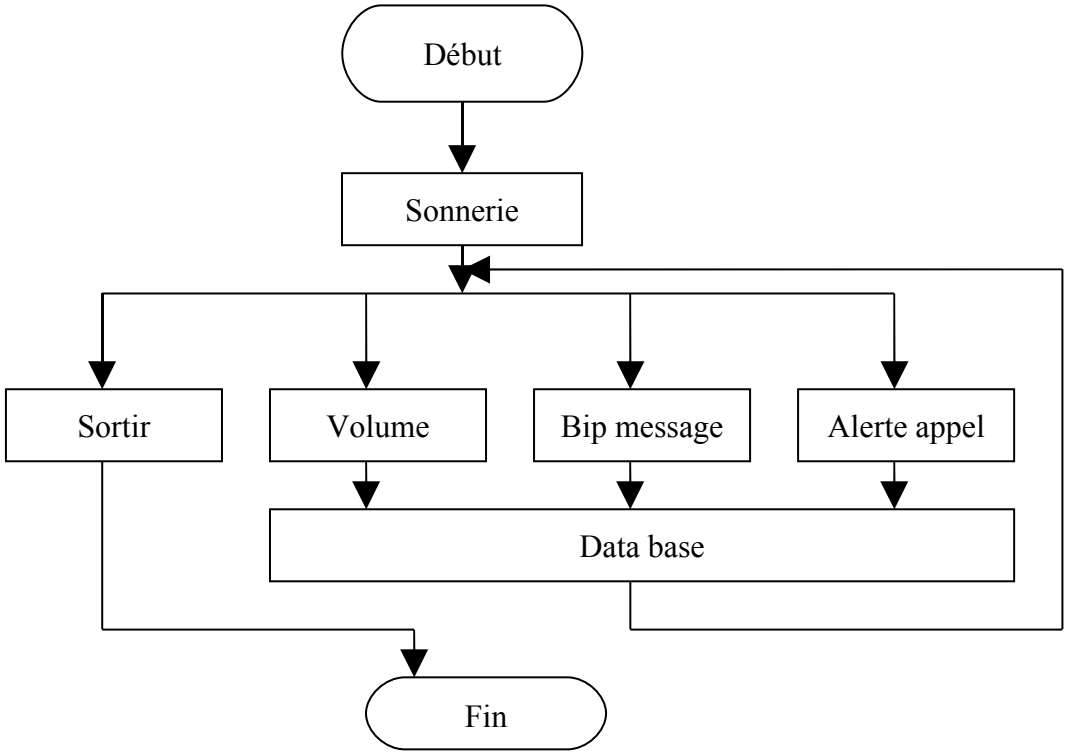


Figure 3.09 : organigramme du menu sonnerie

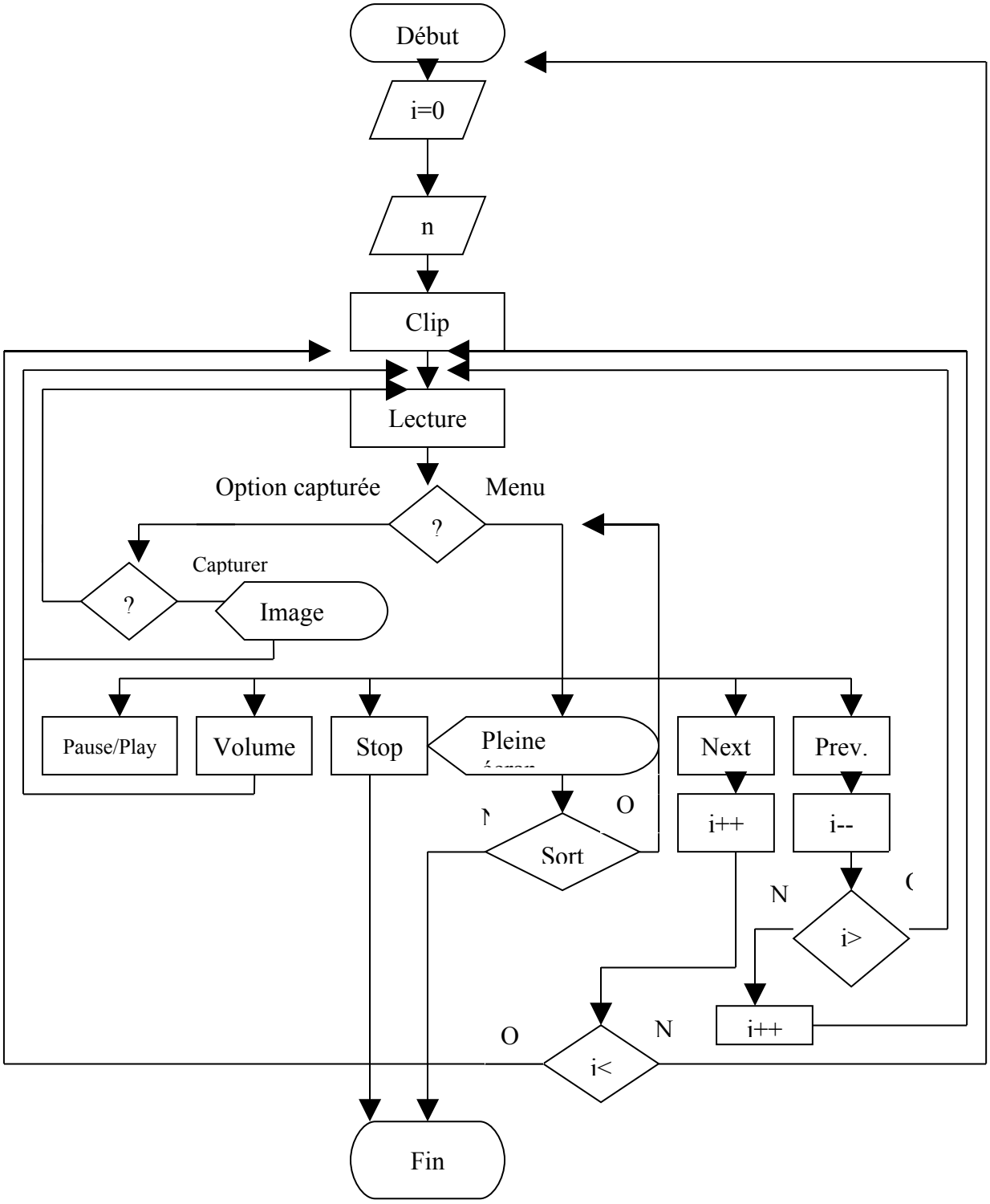


Figure 3.10 : organigramme du menu clip

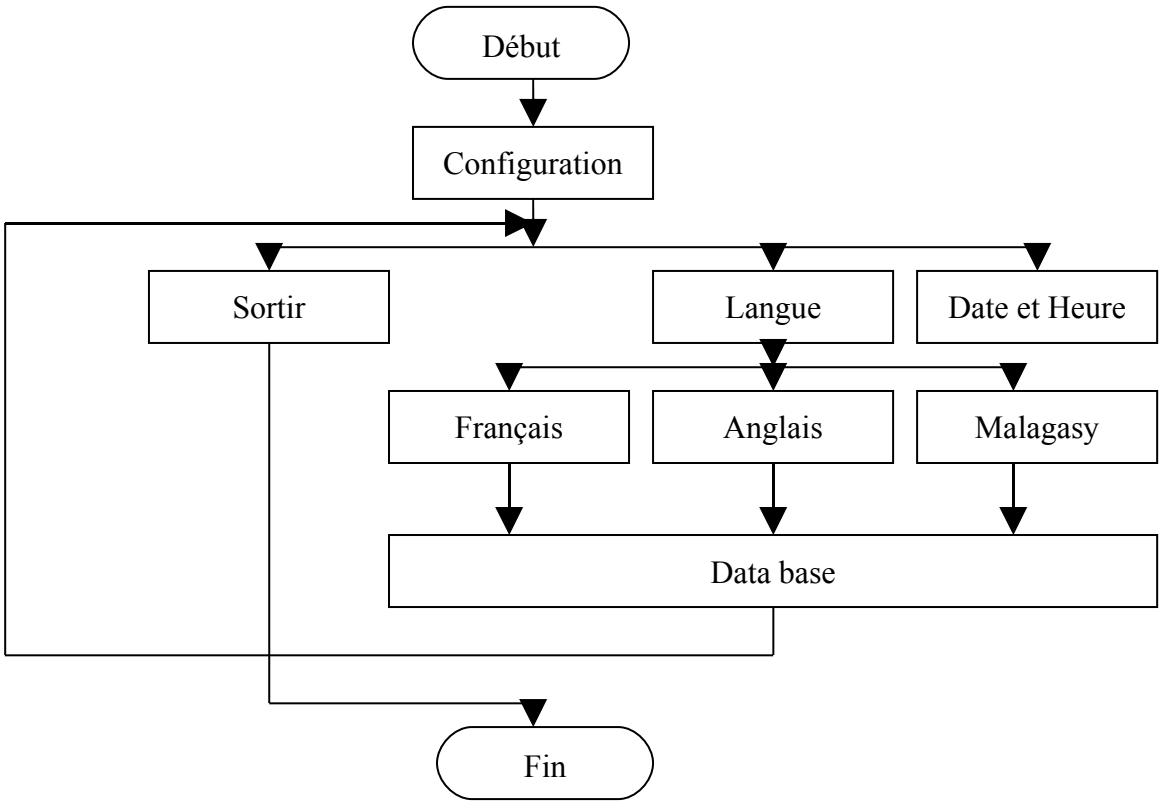


Figure 3.11 : organigramme du menu configuration

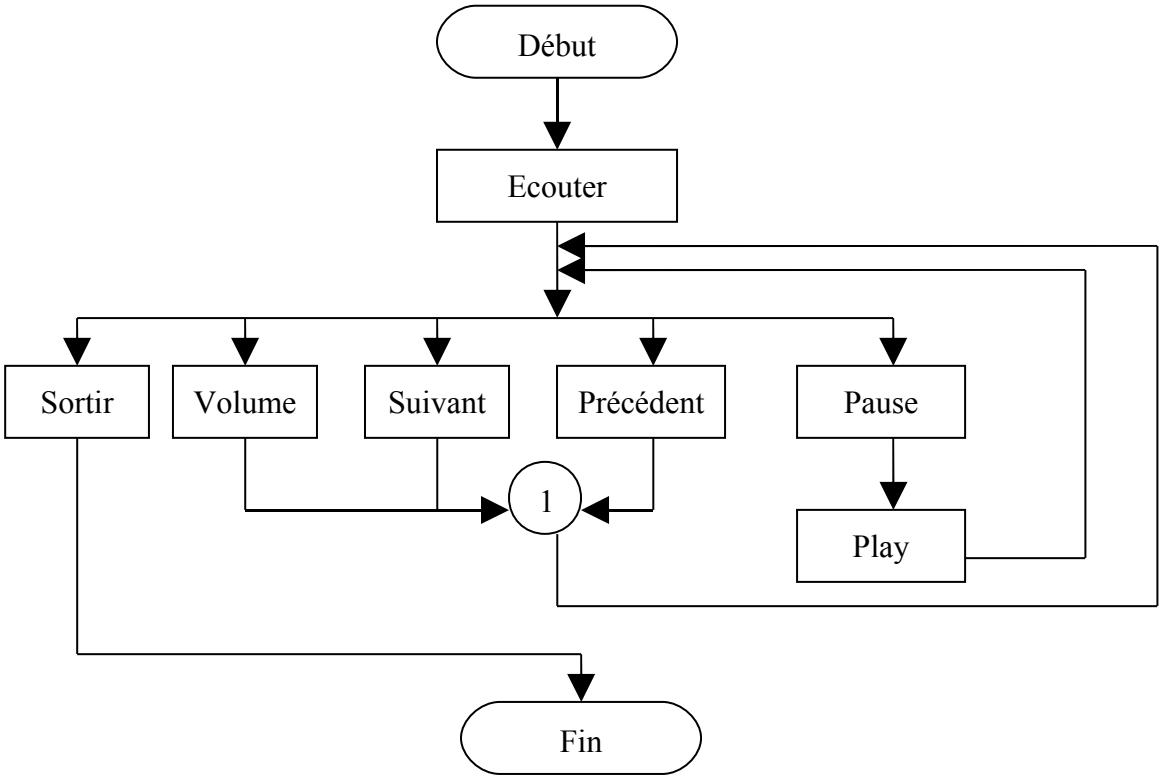


Figure 3.12 : organigramme du menu musique

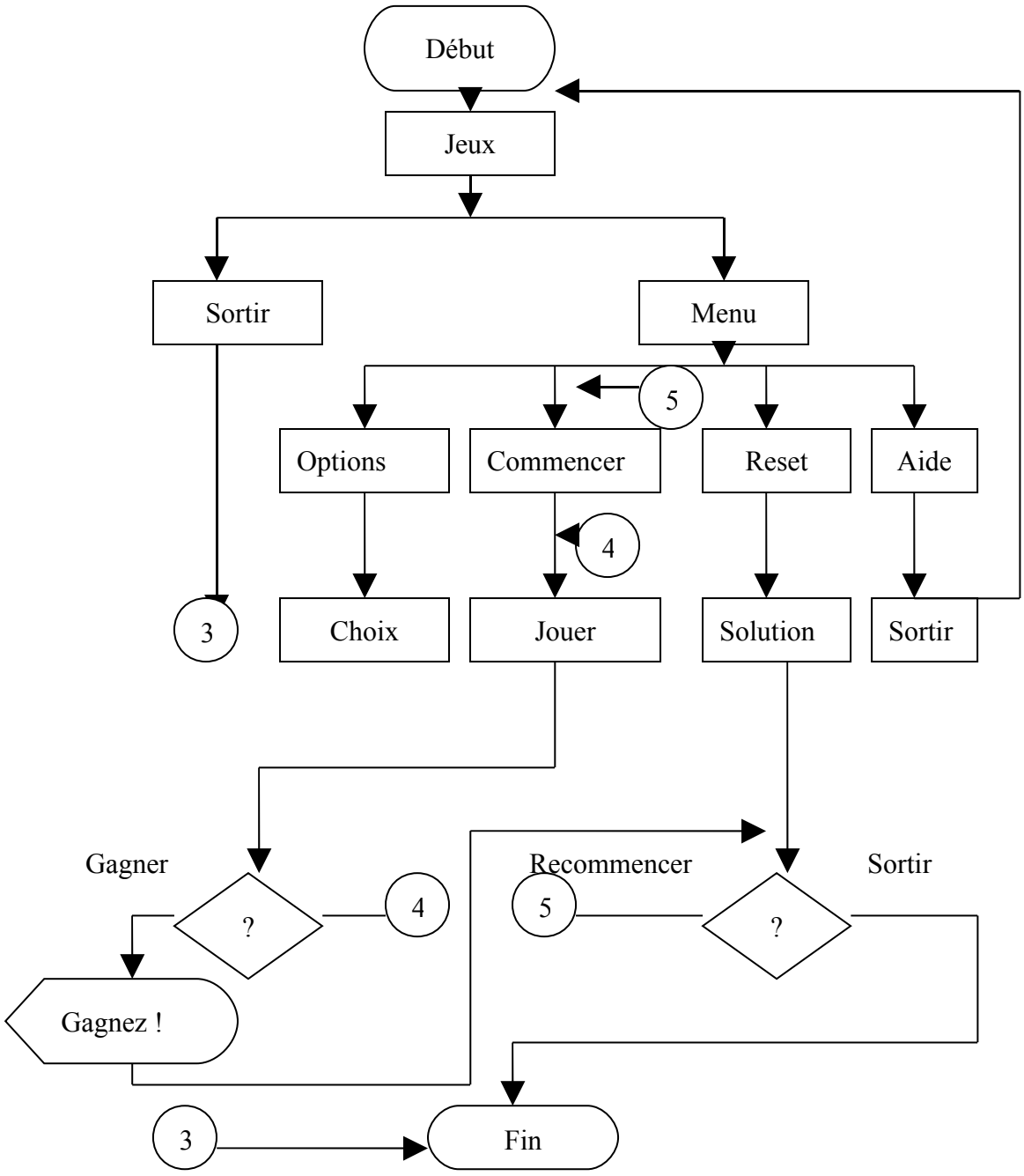


Figure 3.13 : organigramme du menu jeu

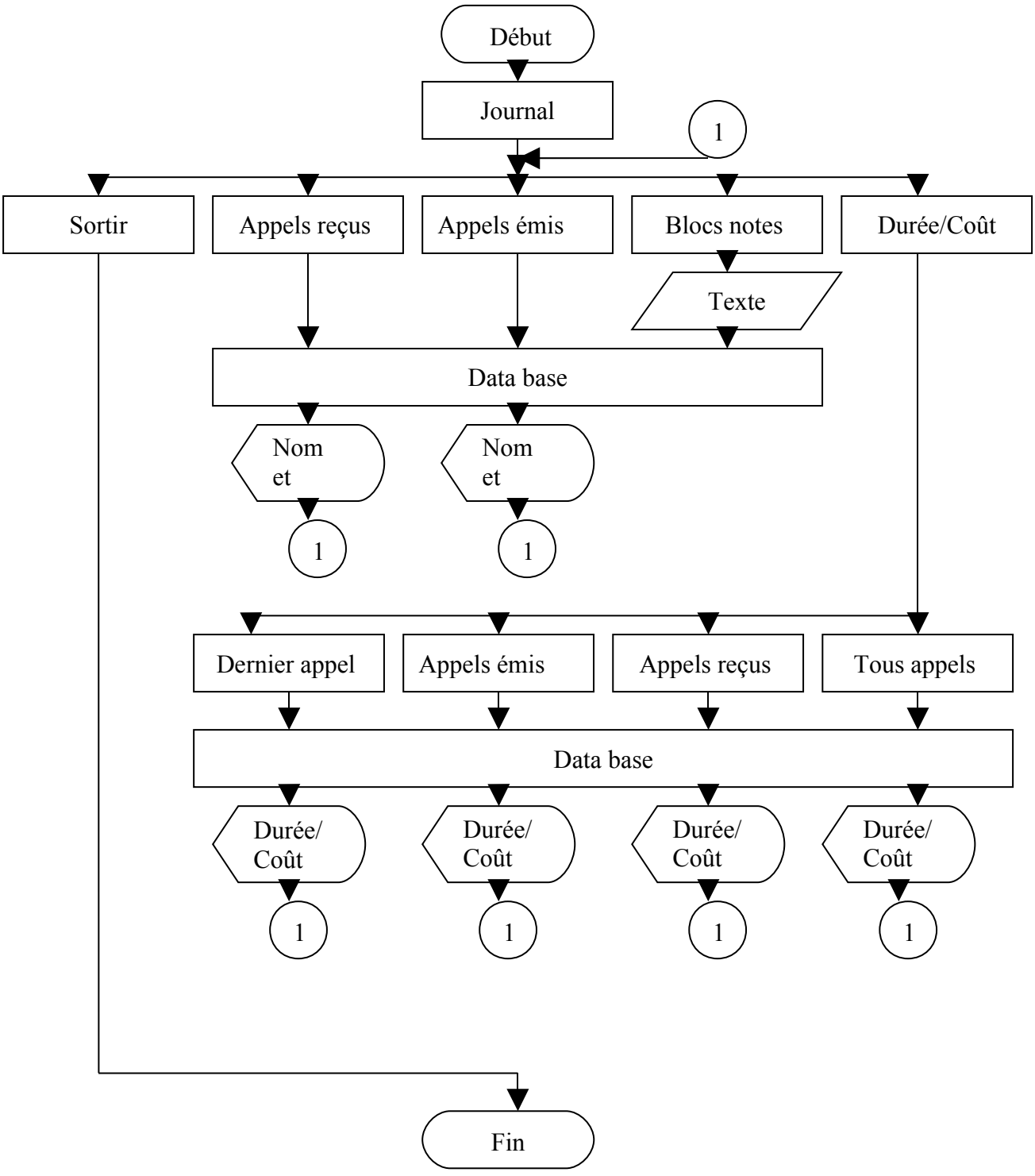


Figure 3.14 : organigramme du menu journal

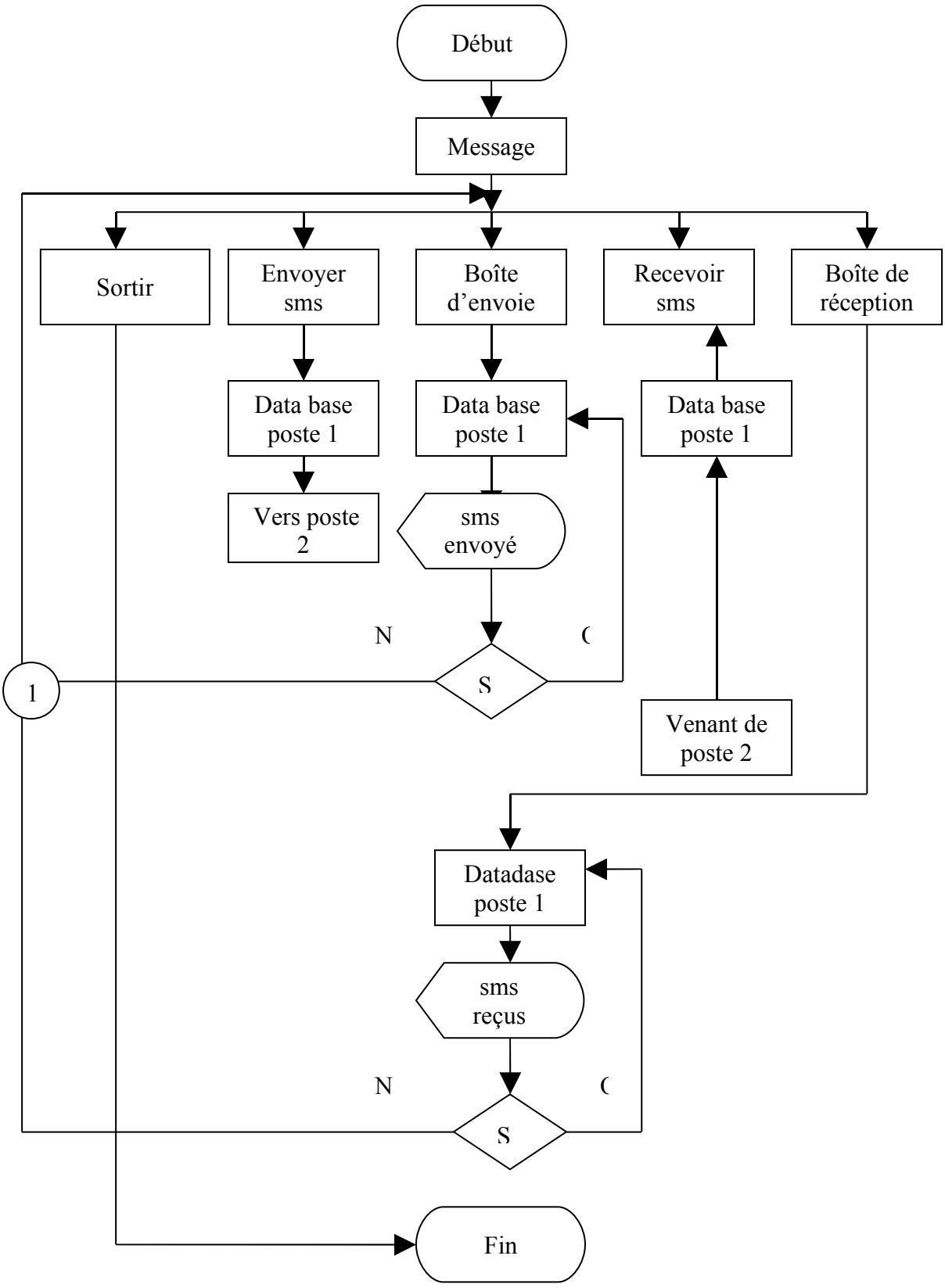


Figure 3.15 : organigramme du menu message

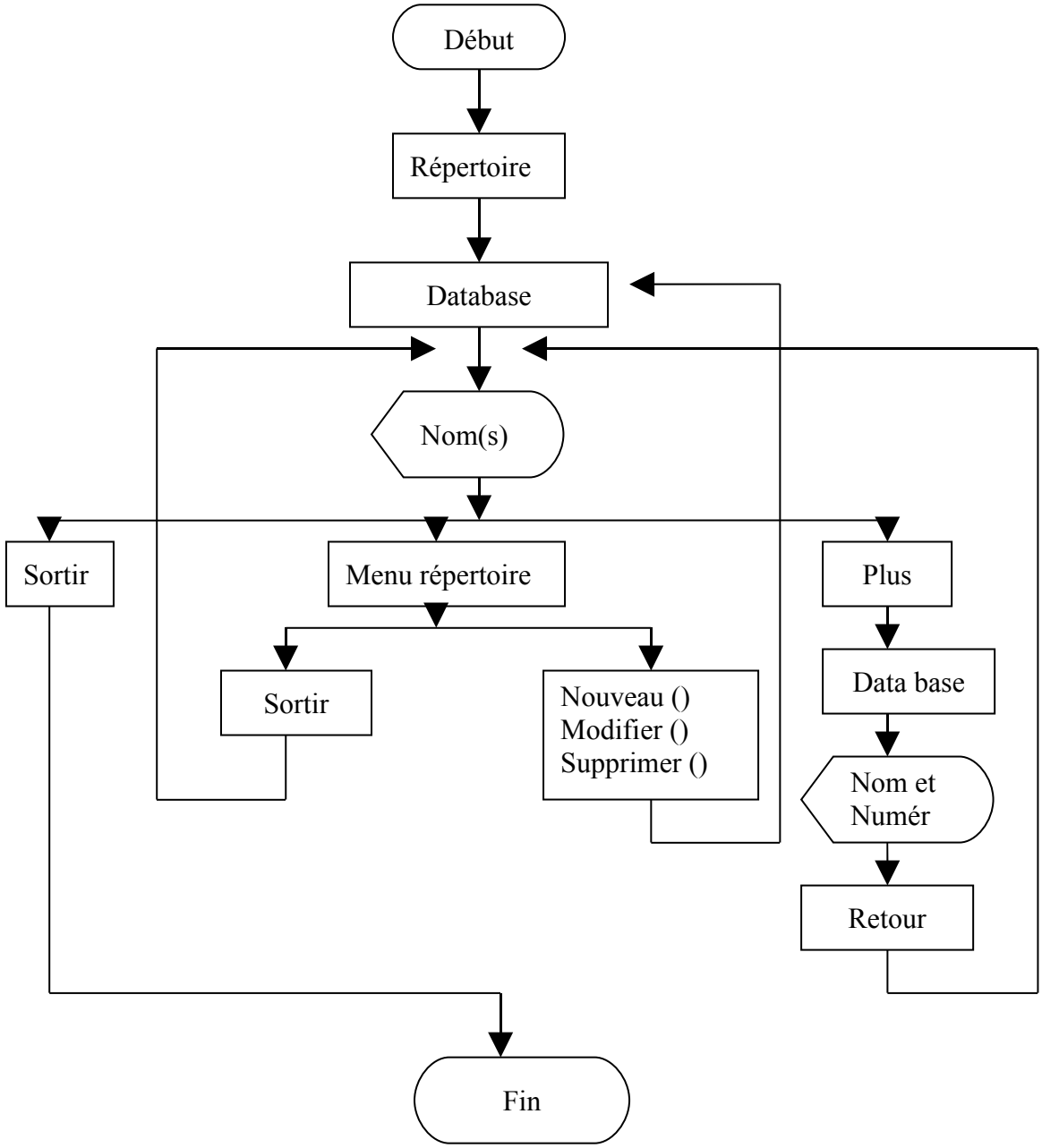


Figure 3.16 : organigramme du menu répertoire

3.7. Manipulation de l'application (Résumé)

L'objectif est atteint, ce paragraphe représente alors les interfaces graphiques offertes aux utilisateurs avec les manuels d'utilisations.

Le lancement de l'application nous fait voir les menus disponibles.

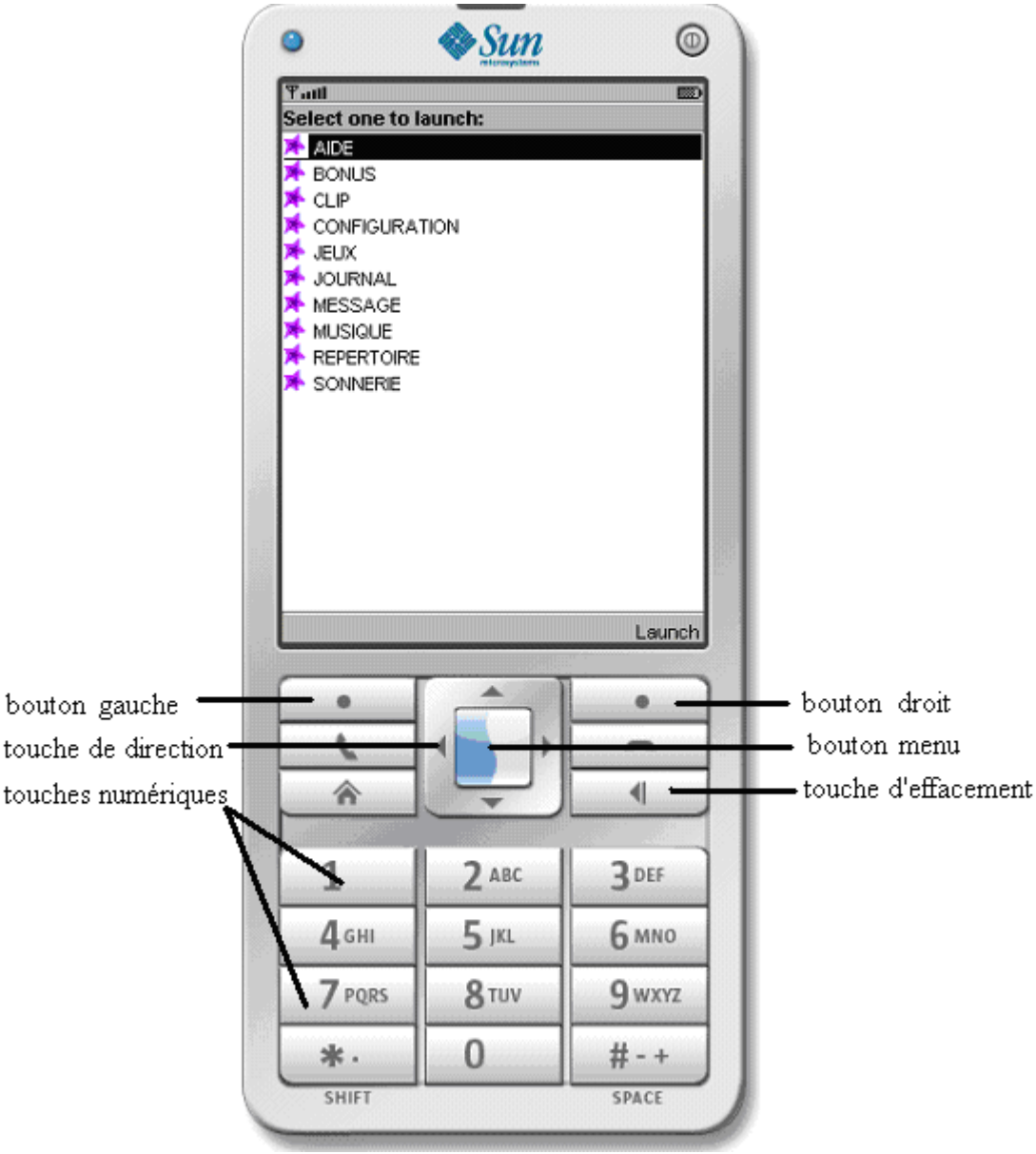


Figure 3.17 : description de l'émulateur avec les menus

Sélectionner un menu, puis appuyer sur le bouton programmable « **launch** » pour lancer une application.

3.7.1. Aide

Une fois entrée dans ce menu, un manuel d'utilisation défile sur l'écran. Pour quitter l'aide, il suffit d'appuyer sur le bouton *sortir*.

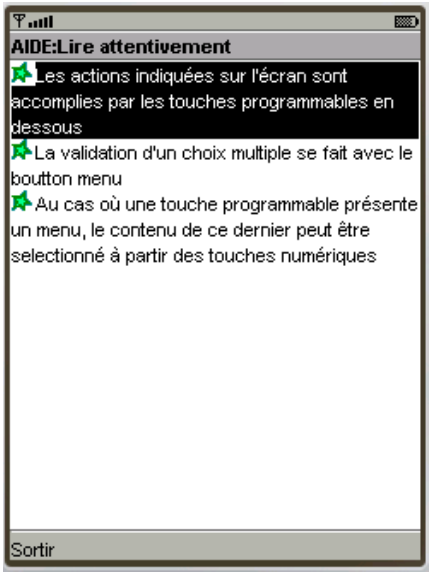


Figure 3.18: aide

3.7.2 Bonus

En fait, deux options sont disponibles : l'une calculatrice et l'autre calendrier.

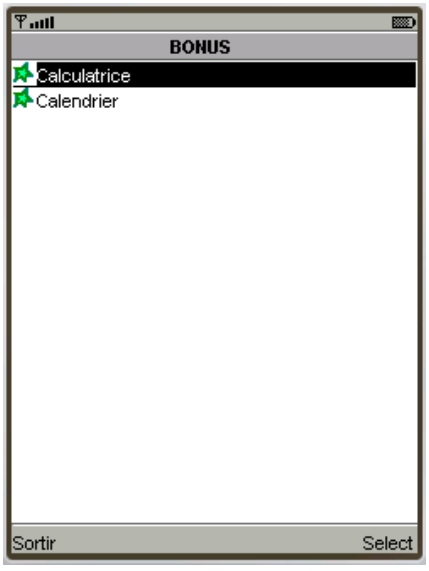


Figure 3.19 : les contenus du menu bonus

3.7.2.1. Calculatrice

Notre calculatrice effectue cinq opérations élémentaires (+, -, *, /, !), le bouton *menu* est responsable de la validation de cette opération. On vous demande d'introduire deux chiffres à partir des touches numériques. Un clic sur le bouton gauche fait apparaître le résultat de votre calcul. Pour calculer un factoriel, il suffit qu'on remplisse seulement le premier chiffre avant de valider le bouton *calculer*.

L'appuie sur le bouton *sortir* vous fait revenir vers le menu principale de bonus.

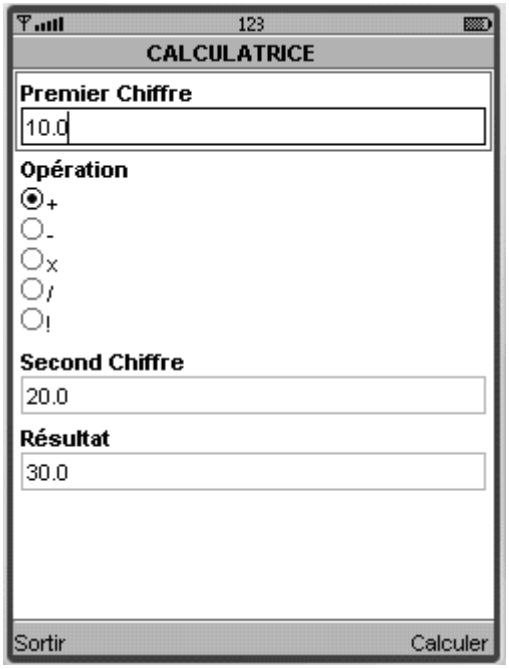


Figure 3.20: calculatrice

3.7.2.2. Calendrier

Lorsqu'on choisit cette option à partir du bouton *select*, l'heure et la date actuelle apparaissent sur l'écran. Quand on appuie sur le bouton *menu* et si vous choisissez l'heure, une horloge vous indique l'heure en temps universel. Il est possible de régler cette montre en utilisant la touche de direction, puis valider avec le bouton *save*, *back* si vous souhaite abandonner l'action.

De même pour la date puisqu'il est possible de la modifier, en outre, on peut visualiser la date des années à venir et des années passées en manipulant toujours la touche de direction. Le bouton *sortir* vous fait revenir au menu principal.

Figure 3.21: date et heure

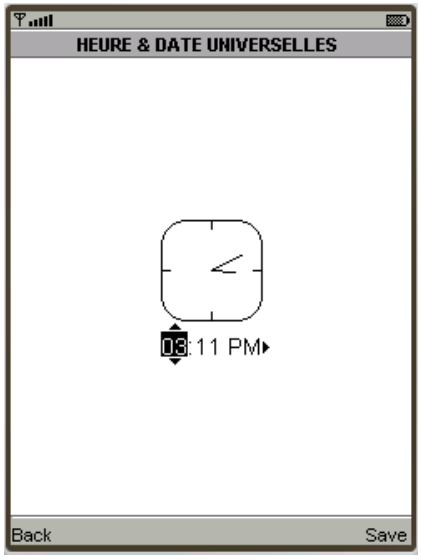


Figure 3.22: horloge

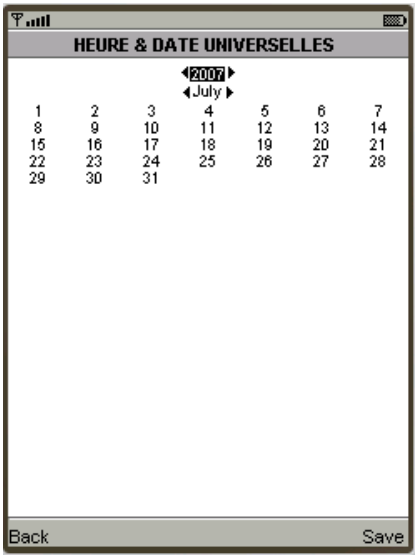


Figure 3.23: jour / mois/ année

3.7.3. Clip

Dès que vous lancez le menu clip, c'est le temps de détente parce que des clips vidéo se succèdent l'un après les autres. La première option offerte est la capture d'une image sur l'écran, puis l'afficher directement. Cette action est assurée par le bouton gauche *capture*.

Figure 3.24 : vidéo

Si vous cliquez le bouton droit *menu*, six fonctions sont à votre disposition :

- Suspendre/ Continuer (touche numérique 1) : suspendre/continuer la lecture
- Volume (touche numérique 2) : c'est une autre interface qu'on y contrôle le volume avec les touches de direction. Ce volume varie dans une plage de 0 à 100, et le bouton *Ok* valide sa valeur, *Annuler* pour abandonner l'action.

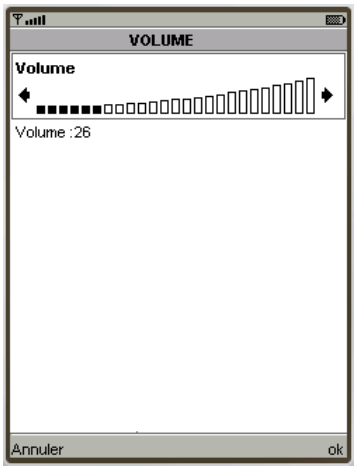


Figure 3.25 : volume

- Suivant (touche numérique 3) : lecture du clip suivant
- Précédent (touche numérique 4) : retour vers le clip précédent.
- Plein écran (touche numérique 5) : L'option plein écran bascule vers le mode plein écran et l'appui sur n'importe quelle touche revient au mode fonctionnement normale.

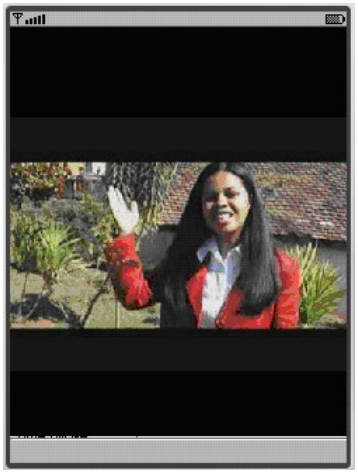


Figure 3.26 : clip en plein écran

- Quitter (touche numérique 6) : cette option fait stopper le clip en cours et quitter immédiatement le menu clip.

3.7.4. Configuration

Le choix du menu configuration fait voir deux éléments de la liste : la langue et date & heure. Le bouton gauche conduit vers le menu principal, l'autre bouton valide la sélection d'un élément de la liste.

Figure 3.27 : menu configuration

- Sélection de la langue : vous avez trois choix, valider un avec le bouton menu suivi du bouton droit *Ok*, et appuyer *Retour* pour revenir au menu configuration.

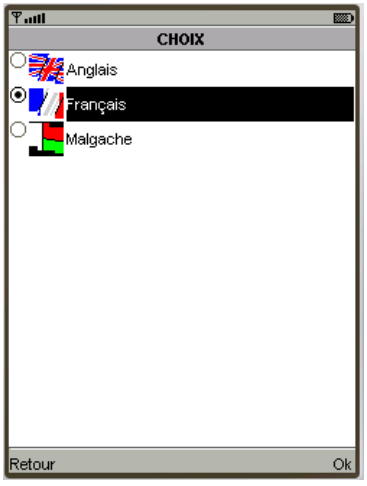


Figure 3.28 : choix de langue

- Sélection de la date et heure : une instruction vous guide jusqu'à la fin du réglage.

3.7.5. Jeux

Le gagnant du jeu sera la personne qui arrive à former dans l'ordre les quatre mots. Cet objectif est atteint en manipulant les touches de directions, c'est à dire à partir de la case vacante qu'on déplace les lettres. Si vous avez gagné ou triché, un message vous informe. Pour quitter le jeu, on appuie simplement sur le bouton gauche *quitter* tandis que le bouton droit présentant un menu qui offre quatre fonctions telles que :

- Commencer (touche numérique 1) : pour commencer le jeu.
- Options (touche numérique 2) : pour choisir le niveau du jeu et la configuration des touches.
- Ouvrir (touche numérique 3) : pour voir la solution du jeu.
- Aide (touche numérique 4) : si on a besoin d'aide.

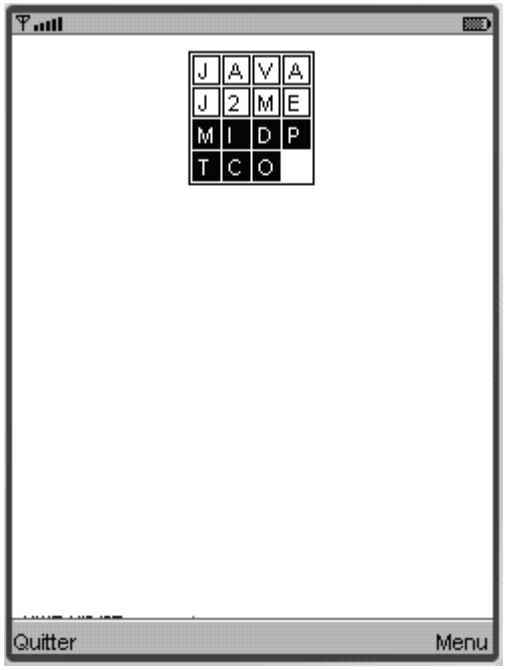


Figure 3.29 : jeu

3.7.6. Journal

L'historique des appels est disponible dans le journal : appels reçus, appels émis, blocs-notes, et durée/coût. La figure 3.30 éclaircit cela.

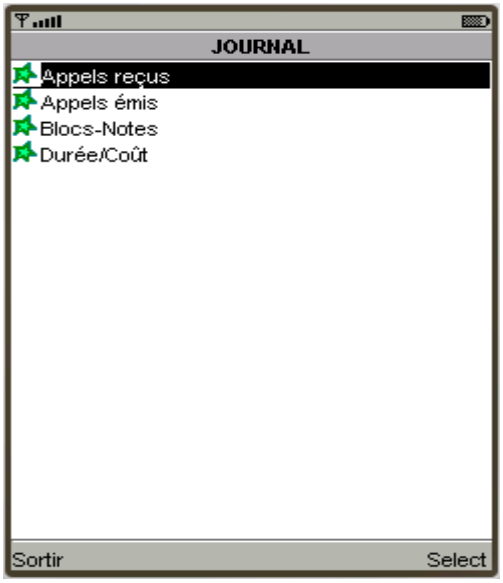


Figure 3.30 : les contenus du menu journal

Le bouton *Sortir* nous amène au menu principal, tandis que l'autre lance l'option choisie.

- Appels reçus : les appels reçus sont enregistrés dans cette option avec la date et l'heure précise. Il est possible de voir plus d'information sur les appels en appuyant sur le bouton *Plus*, en outre on peut les supprimer également avec le bouton menu et l'interface *Menu Derniers Appels* apparaît. Lors de la suppression, une confirmation vous interroge, il suffit de choisir entre le bouton *Oui* et *Non*.

Figure 3.31 : appels reçus

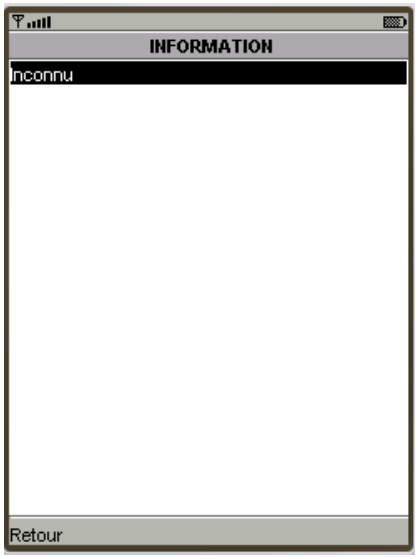


Figure 3.32 : information sur appel reçus

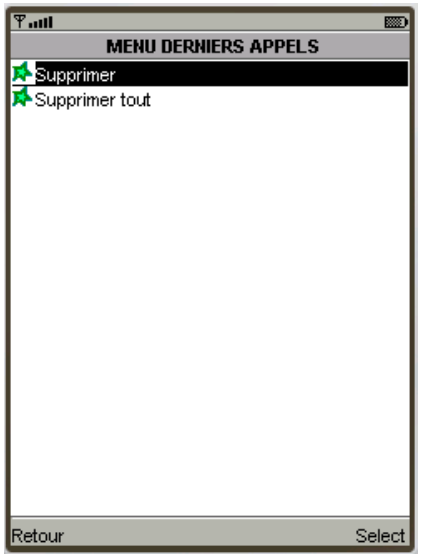


Figure 3.33 : suppression d'un (des) appel(s) reçu(s)

- Appels émis : cette option fonctionne exactement comme la première.

- Blocs-Notes : Ici, vous pouvez écrire des notes en faisant l'enregistrement avec le bouton *Enreg*.

Figure 3.34 : blocs-notes

- Durée/Coût : elle dispose quatre éléments tels que le dernier appel, appels émis, appels reçus et tous appels. Il suffit de sélectionner un élément, une même interface apparaît mais la durée et le coût sont différents.

Figure 3.35 : durée et coût

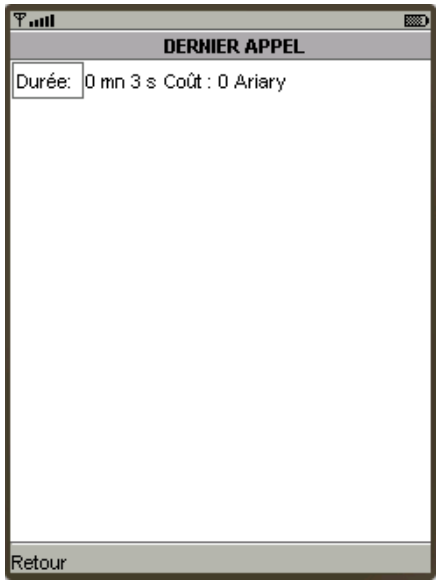


Figure 3.36 : dernier appel

3.7.7. Message

La réception et l'envoie d'un message nécessitent la présence d'un opérateur mobile. Jusqu'ici, cette application n'est pas encore intégrée dans un téléphone mobile, donc pas de numéro d'identification.

Pour surmonter ce problème, on va attribuer un numéro à chaque émulateur. La simulation consiste donc à envoyer un message à partir de la station mobile +261354139600 vers un destinataire ayant le numéro +261354139601.

Lors du lancement de ce menu, une liste constituée de quatre éléments apparaît, la sélection d'un élément se fait avec le bouton *select*.

Figure 3.37 : menu message

- Envoyer message : on vous demande de saisir le numéro de votre correspondant ou parcourir au menu répertoire à partir du bouton gauche. Le bouton droit a deux options : *Ok* pour valider le numéro et *Sortir* pour quitter. Quand vous n'abandonnez pas l'action, c'est le moment de saisir le message, puis envoyer avec le bouton droit.

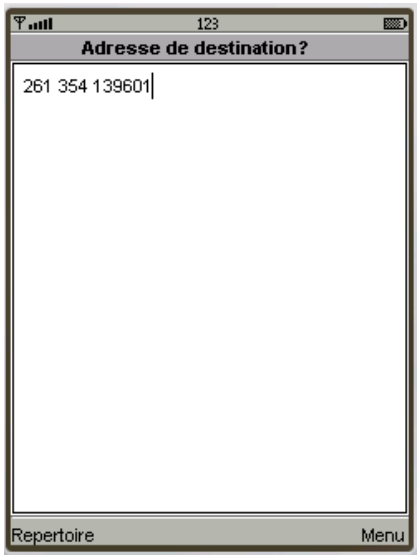


Figure 3.38 : saisie de numéro

Figure 3.39 : saisie du message et son envoie vers +261354139601

Ce message sera enregistré dans la boîte d'envoie de la poste émettrice et dans la boîte de réception de la poste réceptrice.

- Recevoir message : un bip sonore vous avertit l'arrivée d'un nouveau message sans cliquer sur cette option. Avec le bouton droit *Répondre*, vous pouvez répondre le message.

Figure 3.40 : réception du message venant de +261354139600

Figure 3.41 : réception du message venant de +261354139601

- Boîte d'envoie : les messages envoyés sont stockés dedans. Le premier appuie sur le bouton droit sera la lecture du message et le second appuie sera sa suppression en intervenant l'interface de confirmation. Le bouton gauche est toujours disponible pour sortir ou abandonner.

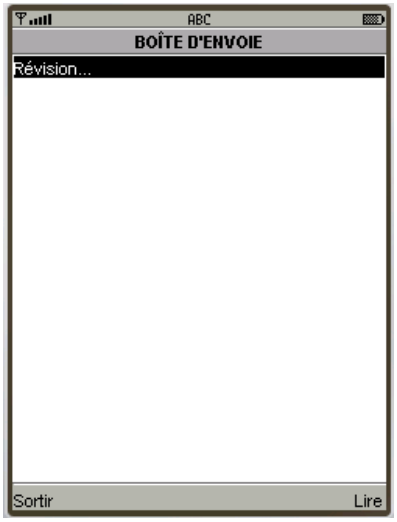


Figure 3.42 : liste des messages dans la boîte d'envoie du+261354139601

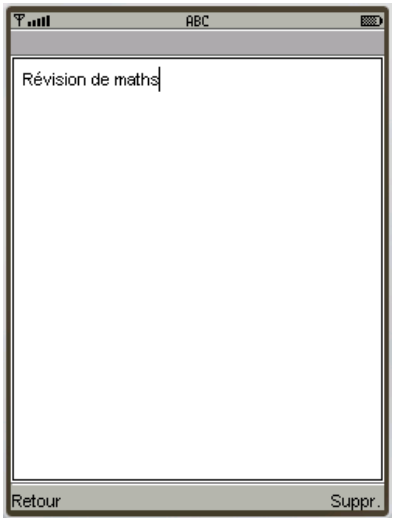


Figure 3.43 : lecture d'un message envoyé

Figure 3.44 : confirmation lors de la suppression d'un message

- Boîte de réception : on y trouve les messages reçus avec l'heure et la date de réception.

Figure 3.45 : liste des messages dans la boîte de réception

Figure 3.46 : lecture d'un message reçu

3.7.8. Musique

L'exécution de la musique débute la lecture d'un fichier audio, avec une image animée sur l'écran. Le bouton *Sortir* vous conduit vers le menu principal tandis que le bouton menu droit encapsule quatre fonctions :

- Volume : toujours la même interface et la même utilisation.
- Pause : suspendre la lecture, ce bouton devient *play* et alternativement.
- Suivant : lecture du fichier audio suivant
- Précédent : lecture du fichier joué précédemment

Figure 3.47 : aperçu lors de la lecture d'un fichier audio

3.7.9. Répertoire

Une liste vide est apparue lors de l'exécution de ce menu. Deux boutons sont disponibles l'un *Sortir* pour retourner vers le menu principal, et l'autre *Plus* pour saisir le nom et le numéro de votre correspondant.

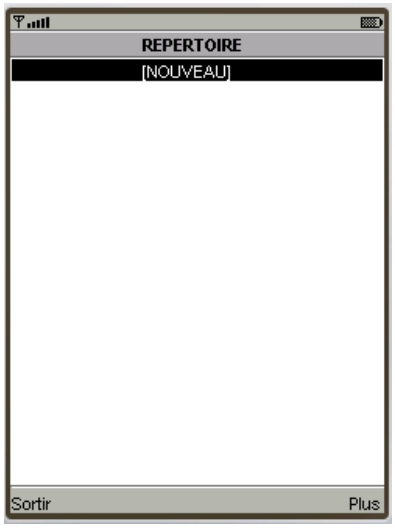


Figure 3.48 : répertoire

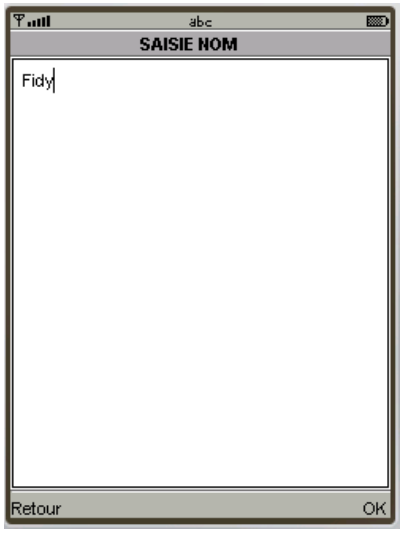


Figure 3.49 : saisie du nom

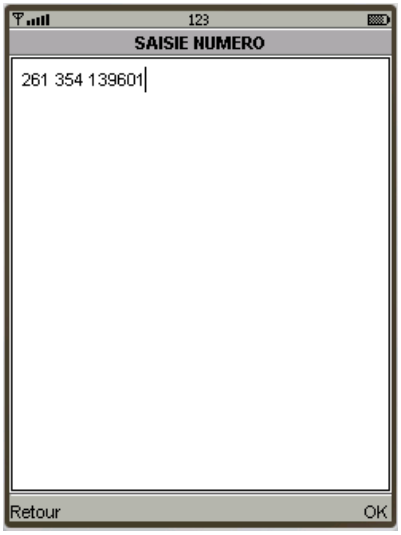


Figure 3.50 : saisie du numéro

Quand vous avez fini la saisie, le bouton gauche présente une étiquette *Terminer*, il suffit de la valider et l'enregistrement est fait.

A partir de ce moment là, la liste n'est plus vide, alors le bouton *Plus* vous informe sur le nom sélectionné.

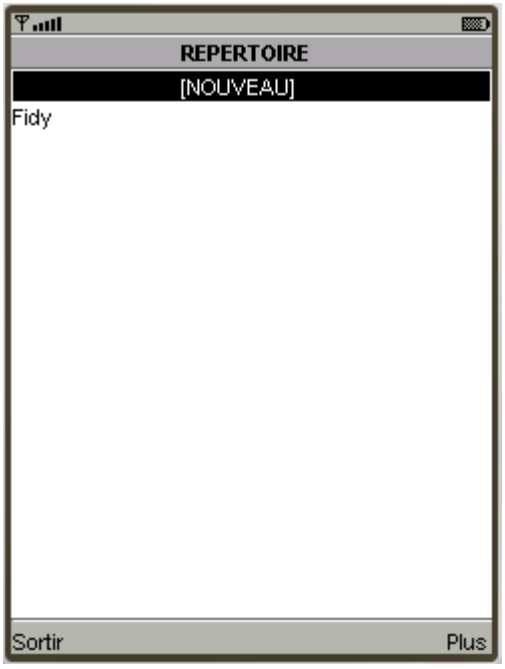


Figure 3.51 : résultat de l'enregistrement

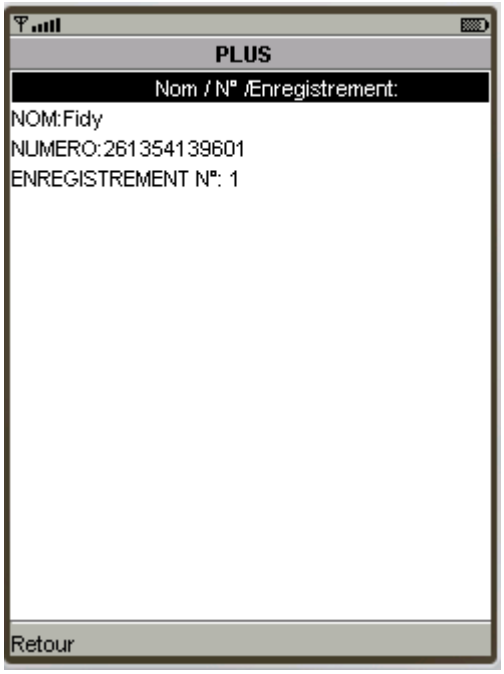


Figure 3.52 : information sur le nom enregistré

A part d'un ajout, il est possible de supprimer et de modifier un enregistrement en appuyant sur le bouton menu. Lors de la suppression, une autre interface vous confirme si vous souhaitez vraiment supprimer ce nom.

Figure 3.53 : menu répertoire

Figure 3.54 : confirmation de suppression d'un correspondant

La modification d'un enregistrement s'effectue exactement comme l'action de saisir. Quand vous appuyez sur le bouton menu et si la sélection est sur *Nouveau*, on peut voir le nombre des espaces libres et utilisés.

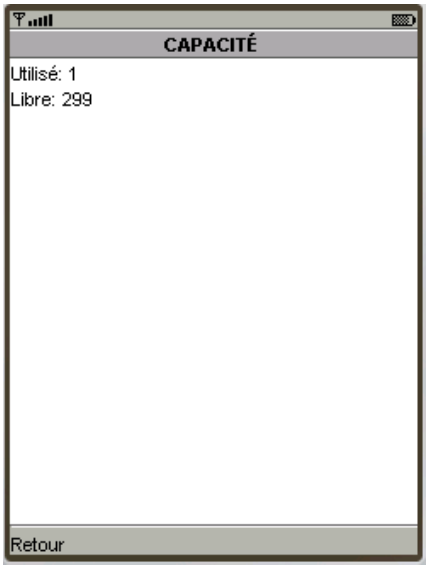


Figure 3.55 : capacité du répertoire

3.7.10. Sonnerie

Le menu sonnerie expose trois configurations concernant le volume et l'alerte lors d'une arrivée d'un événement, l'appuie sur le bouton *select* valide votre choix.

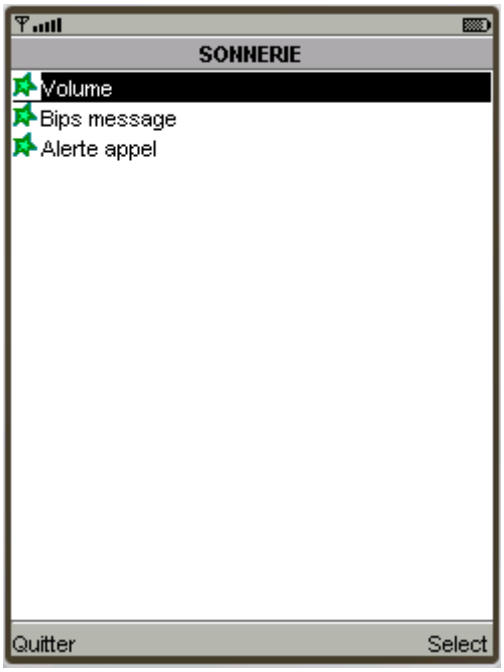


Figure 3.56 : menu sonnerie

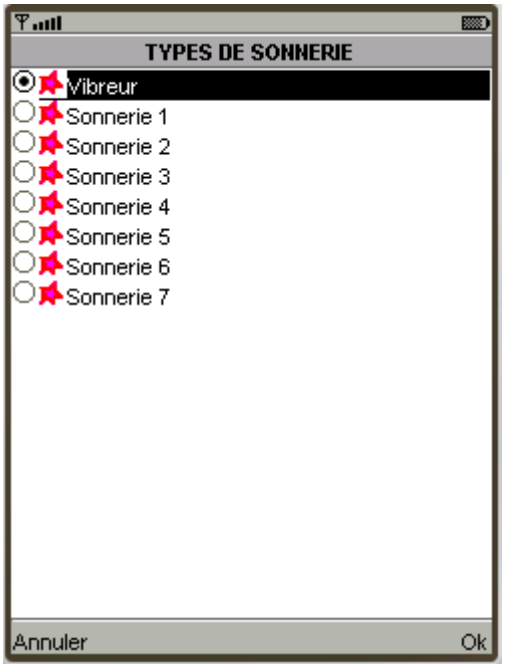


Figure 3.57 : type de sonnerie

- Volume : une interface semblable à celui du volume dans le menu clip apparaît, et même fonctionnement.
- Bips message : il suffit de faire votre choix entre le type de sonnerie disponible en appuyant sur le bouton *Ok*, et *Annuler* pour abandonner l'action.
- Alerte appel : même que le précédent, mais spécialement lors d'une arrivée d'un appel.

CONCLUSION

J2ME est la plateforme Java utilisée pour le développement des applications mobiles. Elle offre deux types de configuration : CLDC et CDC qui correspondent chacune à un profil. Pour le développement des applications sur les téléphones portables, on doit choisir le profil MIDP qui utilise la configuration CLDC. Le travail effectué est donc la conception d'un système d'exploitation d'un téléphone portable.

Avant de commencer ce travail, je pensais qu'il serait facile d'atteindre les objectifs fixés à l'introduction. Mais au fur et à mesure de l'avancement de l'étude, je me suis rendu compte que de nombreux problèmes surgissaient me rendant la tâche difficile.

C'était la première fois que je réalisais une application destinée à fonctionner ailleurs que sur un ordinateur de bureau. Ainsi de nombreuses difficultés que je n'avais pas prévues me sont apparues.

C'était un travail plein d'inquiétude car les documents disponibles pour la programmation J2ME sont très rares, cela s'explique par sa nouveauté dans le monde de programmation.

Un bénéfice intellectuel a été reçu durant ce mémoire, puisque la réalisation m'a apporté des connaissances supplémentaires sur les systèmes embarqués.

« L'insatisfaction est humaine ». Ce travail n'est qu'une fondation de départ car le temps de réalisation est limité. Une amélioration est toujours possible à tous ceux qui s'intéressent à ma recherche, par exemple on peut y intégrer un appareil photo, radio et une option permettant d'enregistrer la voix.

Quant au langage Java, il offre une très grande possibilité en matière de programmation des téléphones portables avec la plateforme J2ME. Cette dernière ne cesse d'évoluer grâce à l'apparition successive d'une nouvelle version de CLDC, MIDP, MMAPI, WMA. Par conséquent, elle est la plateforme Java la plus récente et ayant un avenir éclatant sur le domaine de programmation des systèmes embarqués.

Pour moi, les critiques et les suggestions sont les encouragements les plus appréciées, donc avant de terminer, j'accepterai avec joie tous les critiques que vous pourrez me signaler et je serai prêt à vous satisfaire.

ANNEXE : Code source du menu musique

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import javax.microedition.rms.*;

public class Mp3 extends MIDlet implements CommandListener
{
        private Display display=null;
        private Form fnMain,form;
        private Player p,p1;
        Command sortir,pause,play,volume,ok,back,next,prev;
        VideoControl vc;
        Gauge gauge;
        int volume1=15,num=0;
        VolumeControl volc;
        String tra[]=new String[16];
        StockDatabase dataMp3,dataRecus,dataSon1,dataVol;
        List liste;
        String file[]={"audio/Vavaka.wav","audio/Akaiky mpamonjy.wav"};
        String title[]={"VAVAKA: Ny Fanilo","AKAIKY MPAMONJY: Mamy"};
        String gif[]={"image/gif.gif","image/vorona.gif"};

        public Mp3()
        {
                dataMp3=new StockDatabase();
                try
```

```
        {
        dataMp3.open("mp3");
                for(int i=2;i<=dataMp3.database.getNumRecords();i++)
                {
                        tra[i]=new String(dataMp3.database.getRecord(i));
                }
        }
        catch (Exception e){}
        sortir = new Command(tra[2],Command.EXIT,1);
        pause = new Command(tra[3],Command.ITEM,1);
        volume = new Command(tra[4],Command.ITEM,1);
        play = new Command(tra[5],Command.EXIT,1);
        ok = new Command(tra[6],Command.ITEM,1);
        back = new Command(tra[7],Command.EXIT,1);
        next=new Command(tra[12],Command.OK,1);
        prev= new Command(tra[13],Command.OK,1);
        gauge=new Gauge(tra[8],true,100,15);
        gauge.setPreferredSize(300, 20);
        display=Display.getDisplay(this);
        fnMain=new Form(tra[9]);
        form=new Form(tra[10]);

        fnMain.addCommand(sortir);
        fnMain.addCommand(volume);
        fnMain.addCommand(pause);
        fnMain.addCommand(next);
        fnMain.addCommand(prev);
        fnMain.setCommandListener(this);
        form.addCommand(back);
        form.addCommand(ok);
        form.setCommandListener(this);
}
```

```
public void startApp()
{
        try
        {
        Ticker ticker = new Ticker("");
        ticker.setString(title[num]);
        fnMain.setTicker(ticker);
        InputStream ins = getClass().getResourceAsStream(file[num]);
        p1=Manager.createPlayer(ins,"audio/x-wav");
        InputStream is = getClass().getResourceAsStream(gif[num]);
        p = Manager.createPlayer(is, "image/gif");
        p.realize();
        p1.realize();
                vc=(VideoControl)p.getControl("VideoControl");
                volc=(VolumeControl)p1.getControl("VolumeControl");
                if(vc!=null)
                {
                        fnMain.append("                    ");

fnMain.append((Item)vc.initDisplayMode(vc.USE_GUI_PRIMITIVE,null));
                        volc.setLevel(volume1);
                        Display.getDisplay(this).setCurrent(fnMain);
                }
        p.setLoopCount(465);
        p.start();
        p1.start();
        }
        catch(IOException ioe)
        {
                System.out.println(ioe.toString());
        }
        catch(MediaException mo)
```

```
            {
                System.out.println(mo.toString());
            }
        }
        public void pauseApp()
        {
            try
            {
                p1.stop();
                p.stop();
                fnMain.deleteAll();
                dataSon1=new StockDatabase();
                dataVol=new StockDatabase();
                dataSon1.open("sonnerie1");
                dataVol.open("volume");
                InputStream ins = getClass().getResourceAsStream(new
String(dataSon1.database.getRecord(2)));
                Player p2=Manager.createPlayer(ins,"audio/x-wav");
                p2.realize();
                p2.setLoopCount(2);
                p2.start();
                VolumeControl volc=(VolumeControl)p2.getControl("VolumeControl");
                int feo=Integer.parseInt((new String(dataVol.database.getRecord(2))));
                volc.setLevel(feo);
            }
            catch(IOException ioe)
            {
                System.out.println(ioe.toString());
            }
            catch(MediaException mo)
                    {
                            System.out.println(mo.toString());
```

```
                    }catch (Exception e2) {}
            try
            {
                dataRecus=new StockDatabase();
                Calendar date = Calendar.getInstance();
                String stringEmis="Privé"+":"+date.getTime();
                dataRecus.open("reçus");
                dataRecus.add(stringEmis);
            }
    catch (Exception e)
     {
       try
       {
       }
       catch (Exception e2) {}
     }
}
        public void destroyApp(boolean unconditional)
        {
                try
                    {
                        p1.stop();
                        p.stop();
                    }
                    catch(MediaException mo)
                    {
                            System.out.println(mo.toString());
                    }
        }
        public void commandAction(Command c,Displayable s)
        {
            if(c==sortir)
```

```
{
        destroyApp(false);
        notifyDestroyed();
}
else
        if(c==pause)
        {
                try
                {
                        p.stop();
                        p1.stop();
                        fnMain.removeCommand(pause);
                        fnMain.addCommand(play);
                }
                catch(MediaException mo)
                {
                                System.out.println(mo.toString());
                }
        }
else
        if(c==play)
        {
                try
                {
                        p.start();
                        p1.start();
                        fnMain.removeCommand(play);
                        fnMain.addCommand(pause);
                }
                catch(MediaException mo)
                {
                                System.out.println(mo.toString());
```

```
                }
        }
else
        if(c==volume)
        {
                form.deleteAll();
                form.append(gauge);
                volume1=gauge.getValue();
                form.append(tra[11]+gauge.getValue());
                display.setCurrent(form);
        }
else
        if(c==ok)
        {
                        volume1=gauge.getValue();
                        volc.setLevel(volume1);
                        display.setCurrent(fnMain);
        }
else
        if(c==back)
        {
                display.setCurrent(fnMain);
        }
else
        if(c==next)
        {
                        if(num<title.length-1)
                        {
                                        try
                                        {
                                                p.stop();
                                                p1.stop();
```

```
                                        fnMain.deleteAll();
                                    }
                                    catch(MediaException mo)
                                    {
System.out.println(mo.toString());
                                    }
                                num=num+1;
                                startApp();
                        }
                        else
                        {
                                try
                                    {
                                        p.stop();
                                        p1.stop();
                                        fnMain.deleteAll();
                                    }
                                    catch(MediaException mo)
                                    {
System.out.println(mo.toString());
                                    }
                                num=0;
                                startApp();
                        }
                }
        else
                if(c==prev)
                {
                        if(num>0)
                        {
                                try
                                {
```

```
                                        p.stop();
                                        p1.stop();
                                        fnMain.deleteAll();
                                }
                                catch(MediaException mo)
                                {
        System.out.println(mo.toString());
                                }
                        num=num-1;
                        startApp();
                }
                else
                {
                        try
                                {
                                        p.stop();
                                        p1.stop();
                                        fnMain.deleteAll();
                                }
                                catch(MediaException mo)
                                {
        System.out.println(mo.toString());
                                }
                        num=0;
                        startApp();
                }
        }
    }
}
```

BIBLIOGRAPHIE

[1] J. M. Doudoux, *Développons en Java,* Version 0.90 du 11/09/2006

[2] V. Piroumian, *Wireless J2ME Platform Programming*, publié le 25 Mars 2002 par Prentice Hall PTR

[3] T. Riadh, *Développement d'applications mobiles sous JAVA Micro Edition J2ME*, publié en 2005

[4] J. David, *MMSCam, Pilotage à distance d'un téléphone MMS*, publié en 2003 - Ecole d'ingénieurs du Canton de Vaud, département d'électricité et d'informatique

[5] L. E. Randriarijaona, *Programmation Orienté Objet,* cours 3me année (A.U. : 2004-2005), et 4me année (A.U. : 2005-2006), Dép. Télécommunication ESPA

[6] J. Defaut, *Débuter en J2ME avec le profil MIDP*, publié le 04 Janvier 2005

[7] http://defaut.developpez.com/tutoriel/java/j2me/fichiers/j2me.pdf

[8] http://forte.sun.com/ffj/documentation

[9] http://web.developpez.com/

[10] http://www-adele.imag.fr/users/Didier.Donsez/cours.pdf

[11] http://www.ducrot.org/java/PolycopieJAVA.pdf

[12] http://www.sun.com/developers/evangcentral/

[13] Diedi, *Java embarqué – XML*, Université Paris 8- Saint Denis publié le 18 Octobre 2005

[14] P. Paradinas, *J2ME, Java Card Architecture et Sécurité*, publié en Mars 2003

[15] Sun, *Mobile Edition Tutorial*, publié en Septembre 2002

RENSEIGNEMENTS

Nom :	RAFIDISON
Prénoms :	Maminiaina Alphonse
Titre du mémoire :	Développement d'applications mobiles sous Java Micro Edition : Cas des téléphones portables.
Nombre de page :	83
Nombre de tableau :	08
Nombre de figures :	75
Mots clés :	CDC, CLDC, Java, J2ME, MIDlet, MIDP, téléphone mobile, système embarqué.
Directeur de mémoire :	M. RANDRIARIJAONA Lucien Elino
Adresse de l'auteur :	Lot E 382 Tsarahonenana Moramanga 514 Région Alaotra Mangoro MADAGASCAR *Tél* : + (261) 324139600 ou + (261) 340177285 *e-mail* : aina_mamy_fidy@yahoo.fr

RESUME

De nos jours, on converge petit à petit à l'usage des systèmes embarqués ; en même temps *Sun* ne cesse de conquérir la réalité, d'où la naissance de J2ME. J2ME est une nouvelle plateforme Java pour développer les applications sur les appareils mobiles. Un premier profil était lui associé nommé MIDP dont l'objectif principal est de développer une application sur des machines aux ressources et à interface limitées. Cet ouvrage exploite alors la programmation d'un téléphone portable avec quelques particularités telles que la lecture des fichiers audio et vidéo, ainsi que la capture des images, sans oublier les fonctions de bases.

ABSTRACT

Nowadays, everyone is interesting gradually for the use of embedded systems; at the same time *Sun* tries conquering the reality, so J2ME was born. J2ME is a new punt forms Java for developing applications on the mobile apparatuses. A first profile in which is associated was named MIDP, whose principal objective is to develop an application on machines to the limited resources and interface. This work exploits mobile phone's programming with regular characteristics such as reading audio and video files, as well as image's capture, without forgetting the bases functions.