MASTER EN SCIENCES INFORMATIQUES

L'apprentissage par l'exemple et à distance de la programmation d'IHM sur appareils mobiles

Van Tongelen, Sophie

Award date:
2005

Développement d'interfaces utilisateurs sur appareils mobiles avec J2ME

Sophie VAN TONGELEN

31 août 2005

Table des matières

Chapitre 1

Introduction

Les appareils mobiles prennent de plus en plus de place dans notre société, que ce soit au niveau des téléphones mobiles ou des agendas électroniques. Ces appareils nécessitent une interface utilisateur appropriée. Leurs faibles performances mènent à éliminer tout gaspillage de ressources. La taille et la résolution des différents écrans constituent également de fortes contraintes.

J2ME est un outil spécifique, pour les appareils à faibles ressources tels que les téléphones mobiles et les PDA's. L'outil abordé ici est plus spécifiquement MIDP, un profil de J2ME. MIDP est destiné aux appareils à faibles ressources tels que les téléphones mobiles et les PDA's bas de gamme. Cependant, les PDA's deviennent de plus en plus performants. Ils sont à présent capables d'exécuter des applications sophistiquées. Pour créer des interfaces plus complexes que ne le permet MIDP, on utilisera un sous-ensemble de AWT. Il existe beaucoup de tutoriels sur le développement d'interfaces avec AWT.

L'approche pédagogique suivie est basée sur l'apprentissage par l'exemple qui permet une introduction très rapide à l'outil.

Ce tutoriel est découpé en deux parties principales. Il propose tout d'abord une vue générale de J2ME au niveau du contexte historique et de son architecture. La deuxième partie concerne le développement de l'interface en utilisant MIDP qui est elle-même découpée en trois modules. Le premier présente les composants de base et permet au développeur de se familiariser avec ce nouvel environnement. Le deuxième module présente les commandes et la gestion des événements. Le troisième module développe enfin la classe la plus importante de MIDP qui est la classe Form et ses différents Items. A ces modules, deux autres sections ont été ajoutées. La première propose le code d'un formulaire reprenant tous les éléments vus. La deuxième expose les nouveautés proposées par la version 2.0 de MIDP.

FIG. 2.1 – Correspondances entre les types d'appareils et les plate-formes adaptées

Chapitre 2

Technologies

2.1 J2ME : Généralités

En 1999, Sun Microsystems présente J2ME à la conférence des développeurs JavaOne. La plate-forme J2ME cible des appareils qui ont comme contraintes : une interface utilisateur et un affichage limités, une faible mémoire, une alimentation sur batterie et un encombrement et un poids faibles. La figure 2.1 montre les différentes plates-formes et le type d'appareil qui les supporte.

De nombreux appareils supportent J2ME, comme le montre la figure 2.2

L'architecture de J2ME, représentée sur le schéma de la figure 2.3 se base sur l'utilisation de configurations et de profils. Ceux-ci sont choisis en fonction des spécifications de l'appareil sur lequel on veut développer.

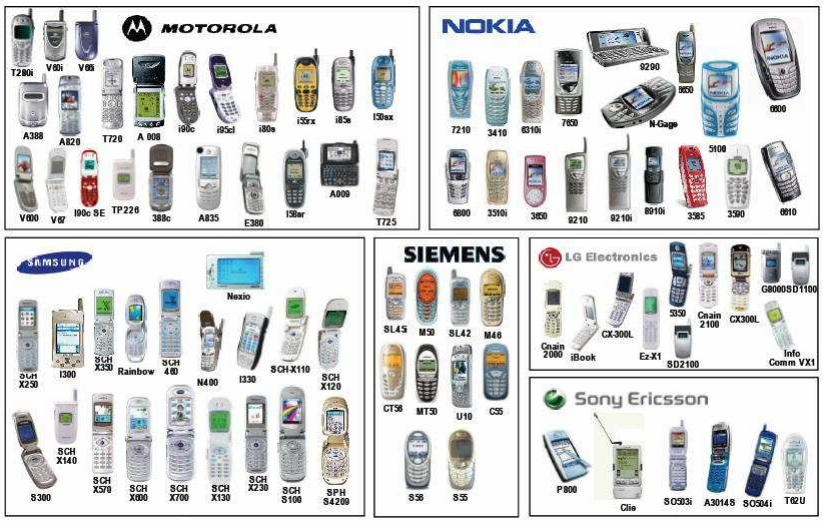


FIG. 2.2 – Echantillon d'appareils supportant la plate-forme J2ME

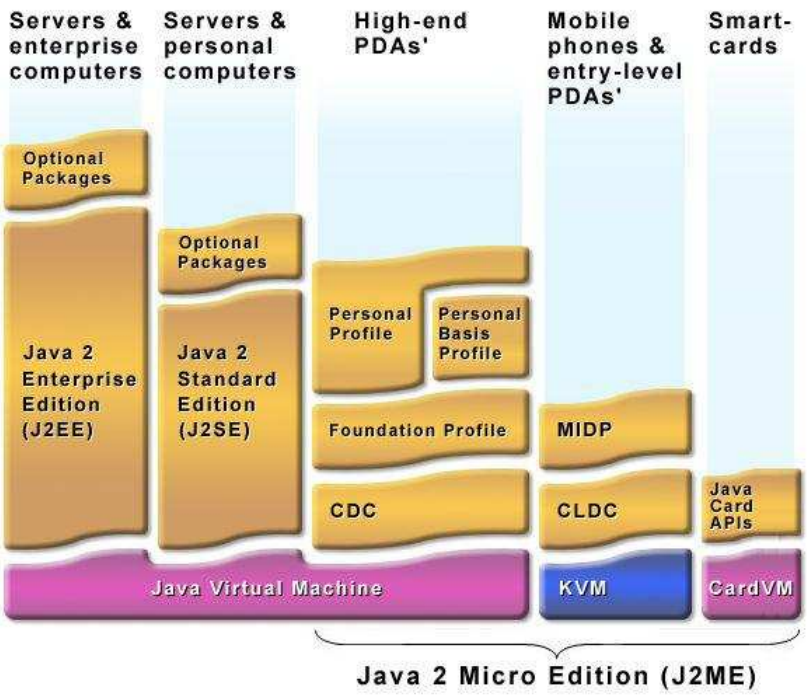


FIG. 2.3 – Architecture générale de J2ME

2.2 KVM - K Virtual Machine

La KVM est une machine virtuelle Java portable et compacte prévue pour des dispositifs avec des ressources limitées, tels que les téléphones portables, organizers personnels, appareils ménagers, distributeurs automatiques, etc... L'objectif principal d'une telle machine virtuelle est d'être la plus petite possible (40 à 80 ko) tout en préservant l'aspect du langage de programmation Java. La KVM n'est pas implémentée que par Sun Microsystems et a déjà été portée sur plusieurs dizaines de dispositifs par des constructeurs agréés par Sun. Pour les dispositifs ne possédant pas d'interface utilisateur capable de lancer des applications, il existe un gestionnaire d'applications Java qui joue le rôle d'interface entre le système d'exploitation hôte et la machine virtuelle.

2.3 Configurations

La configuration détermine une plate-forme minimale pour une catégorie d'appareils qui ont des besoins analogues :

- type et capacité de la mémoire totale
- type de connexion réseau disponible pour l'appareil
- type et vitesse de processeur.

Chaque configuration consiste en une machine virtuelle et un ensemble de classe qui fournit un environnement de développement pour les applications logicielles.

Il existe deux configurations définies : Connected Device Configuration (CDC) et Connected Limited Device Configuration (CLDC).

2.3.1 Connected Device Configuration (CDC)

Cette configuration s'adresse aux appareils qui se situent entre les appareils à faibles ressources et les postes de travail fixes. Ce sont les appareils grand public haut de gamme. Typiquement, ces appareils ont une mémoire supérieure à 2MB, des processeurs de capacité élevée et une connexion réseau permanente avec une large bande passante Exemples : téléviseurs et visiophones Internet, systèmes de navigation et de loisirs embarqués.

2.3.2 Connected Limited Device Configuration (CLDC)

Cette configuration s'adresse aux appareils à faibles ressources dont les caractéristiques matérielles sont les suivantes : une mémoire entre 128 Ko et 512 Ko disponible pour la plate-forme Java, un processeur de 16 bits ou 32 bits, une alimentation par batterie et une connexion intermittente avec une bande passante limitée. Exemples : téléphones portables, messagers de poches, organizers personnels, scanners de code barres.

2.4 Profils

Un profil complète une configuration en ajoutant des classes supplémentaires qui fournissent une plate-forme appropriée à une famille d'appareils. L'objectif est de donner plus de flexibilité pour maintenir la portabilité des applications entre les terminaux Les profils sont définis par le "Java Community Process".

MIDP et PDAP sont des profils définis pour la configuration CLDC. Foundation Profile, Personal Basis, Personal Profiles et RMI Profile sont des profils définis pour la configuration CDC.

- *Mobile Information Device Profile*
  MIDP est le premier profil qui a été développé dont l'objectif principal est le développement d'applications sur des machines aux ressources et à l'interface limitées.
  MIDP est basé sur la configuration CLDC. Ses fonctionnalités principales sont l'interface client, la persistance de stockage, la mise en réseau, et l'application modèle. Il permet la gestion de l'interface utilisateur, la gestion de l'interface réseau, et la gestion d'une base de donnée sur le mobile.
- *PDA Profile*
  PDAP est proche de MIDP mais est plutôt destiné aux PDAs plus performant. Cependant ce profil a été abandonné en tant que tel, il se présente maintenant sous forme de 2 packages optionnels. Au niveau de l'interface, il utilise un sous-ensemble d'AWT.
- *Foundation Profile*
  FP ajoute des classes à CDC pour inclure les principales librairies de la version 1.3 de Core Java 2. Comme son nom l'indique, FP est un profil utilisé comme base pour la plupart des autres profils de CDC.
- *Personal Basis et Personal Profiles*
  Le Personal Basis Profile ajoute des fonctionnalités d'interfaces utilisateur basiques au Foundation Profile. Il ne permet pas à plus d'une fenêtre d'être active au même moment. Les plates-formes qui peuvent supporter une interface utilisateur plus complexe utilise le Personal Profile. Ces profils utilisent des sous-ensembles d'AWT pour la programmation d'interfaces.
- *RMI Profile*
  RMI Profile ajoute les librairies Remote Method Invocation de J2SE au Foundation Profile.

2.4.1 MIDP

MIDP fait partie du groupe de spécification JSR 037(MIDP pour les plateformes J2ME). Il existe deux versions : MIDP 1.0 et MIDP 2.0. Il est avant tout destiné aux téléphones mobiles et aux PDA bas de gammes. Il fournit des API pour le développement d'interfaces graphiques utilisateur, la connectivité réseau, le stockage local de données et la supervision d'applications. L'API du MIDP se compose des API du CDLC (Connected Limited Device Configuration) et de 3 packages :

- javax.microedition.midlet : cycle de vie de l'application. Socle technique destiné à gérer le cycle de vie des MIDlets.
- javax.microedition.lcdui : interface homme/machine. Fournit la gestion de l'interface utilisateur telle que les contrôles.
- javax.microedition.rms : base de données persistante légère.

Pour exécuter une application MIDP, un appareil doit avoir la configuration matérielle suivante :

- Écran : Taille de l'écran : 96x54 Profondeur d'affichage : 1-bit
- Dispositifs d'entrée : un ou plusieurs des mécanismes d'entrée suivants : clavier ou écran tactile.

– Mémoire : 256 kilobytes de mémoire non-volatile pour l'implémentation de MIDP, requis pour CLDC. 8 kilobytes de mémoire non-volatile pour les données persistantes et 128 kilobytes de mémoire vive pour l'exécution de Java.
– Réseau : bi-bande, sans fil, peut être intermittent, largeur de bande limitée.

Les MIDlets

Une MIDlet est une application java qui s'exécute sur des appareils MIDP.

Toutes les applications pour le profil MID doivent être dérivées de la classe MIDlet qui gère le cycle de vie de l'application. Cette classe appartient au paquet
javax.microedition.midlet. Elle permet le dialogue entre le système et l'application.

Pour gérer ce cycle de vie, la classe MIDlets défini les trois méthodes ci-dessous.

- startApp() ; : cette méthode est appelée à chaque démarrage ou redémarrage de l'application.
- pauseApp() ; : cette méthode est appelée lors de la mise en pause de l'application
- destroyApp(boolean unconditional) ; : cette méthode est appelée lors de la destruction de l'application. Si le boolean a comme valeur "false", la MIDlet peut ne pas se terminer et déclarer une MIDletStateException.

Ces trois méthodes doivent obligatoirement être redéfinies dans chaque MIDlet.

Le cycle de vie d'une MIDlet, représenté sur le schéma de la figure 2.4 comporte quatre états : loaded, active, paused, destroy. Quand une MIDlet est chargée sur un appareil et que le constructeur est appelé, cela se passe dans l'état "loaded". Ca peut être à n'importe quel moment avant l'appel la méthode starApp(). Après cet appel, la MIDlet passe à l'état "active". La méthode pauseApp() met la MIDlet en pause ce qui permet de libérer des ressources inutiles à la MIDlet dans cet état. Ca évite des conflits et une consommation inutile de la batterie. La méthode destroyApp() termine la MIDlet.

Le background et le foreground sont les deux états possibles d'une application. Une application en foreground consomme de l'énergie et c'est celle qui est exécutée. Les applications qui sont en attente sont en background et ne consomme pas d'énergie. La méthode pauseApp() met l'application en background.

Les API interface utilisateur

La classe `Display` est la base de toute interface utilisateur d'une MIDlet. Cependant, elle ne possède que peu de fonctionnalités et seule, elle n'est pas très utile. Il y a donc des sous-classes plus utiles. Les deux classes directement en dessous de `Displayable` sont `Screen` et `Canvas`. Elles forment les deux API's graphiques de MIDP et sont représentées à la figure 2.5.

L'API de haut niveau permet un niveau d'abstraction élevé et donc une plus grande portabilité. Elle est orientée écran et widget. Il y a peu de contrôle possible au niveau du look-and-feel. On ne peut donc pas agir sur l'apparence visuelle (couleurs, tailles, entrée) des composants de haut-niveau. Tout ça est géré par l'implémentation MIDP. Il est plus simple et plus puissant qu'AWT.

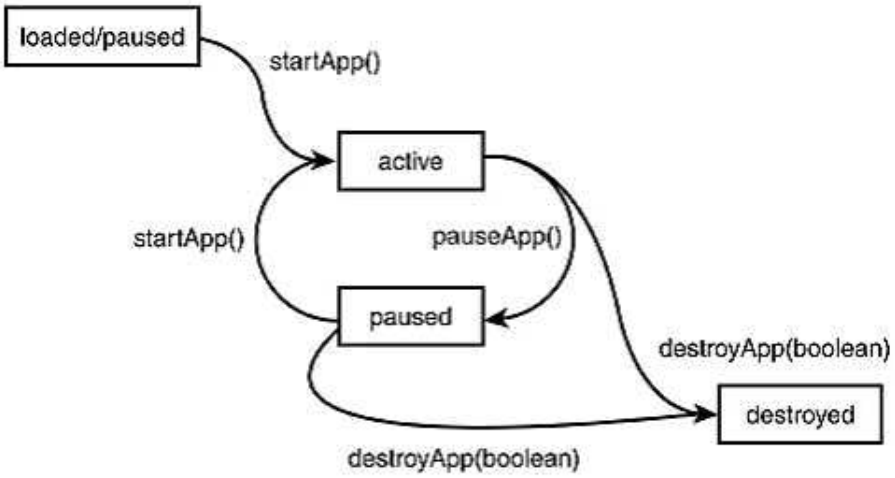


FIG. 2.4 – Cycle de vie d'une MIDlet

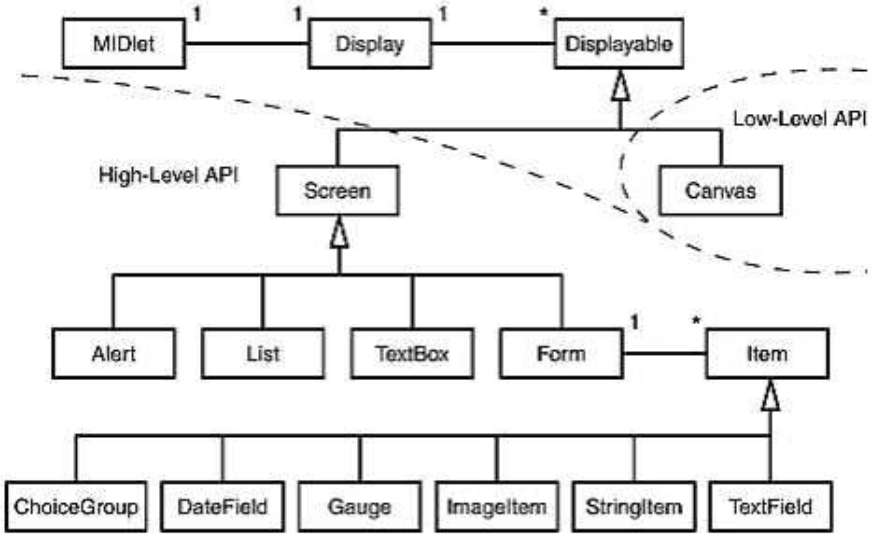


FIG. 2.5 – Représentation des différentes classes graphiques

Les classes qui implémentent l'API de haut niveau héritent toutes de la classe javax.microedition.lcdui.Screen.

L'API de bas niveau offre un contrôle beaucoup plus grand de l'apparence visuelle. Il offre des primitives de dessins et permet de dessiner pixel par pixel l'interface. Cette API est définie pour les applications, telles que les jeux, qui ont besoin d'un tel contrôle des éléments graphiques et d'un accès aux événements de bas-niveau. En contrepartie, la portabilité de ces applications est très faible puisque l'accès à des détails spécifiques au niveau hardware est permis. Les classes javax.microedition.lcdui.Canvas et javax.microedition.lcdui.Graphics implémentent l'API de bas niveau.

2.4.2 PDAP

Dans une MIDlet (ou PDAlet), les composants d'AWT et du paquet javax.microedition.lcdui peuvent être utilisés dans la même application mais seul un type de composant peut être visible à l'écran à un moment donné. Si un screen lcdui se trouve dans le focus de l'entrée, il cachera toutes les frames AWT, et inversement.

Le listing 1 montre une MIDlet vide et le listing 2 converti cette MIDlet vide en une PDAlet vide en appellant la méthode getDefaultToolkit() dans la méthode startApp(). L'outil par défaut est un sous-ensemble d'AWT et est utilisé pour créer une interface graphique utilisateur pour la PDAlet. La méthode getDefaultToolkit() rend disponible l'API PDAP à la PDAlet.

Le profil PDA utilise les éléments d'interface communs d'AWT comme la solution pour les interfaces utilisateurs pour PDAlets.

AWT utilisé dans PDAP est un sous-ensemble de l'AWT de J2SE qui est "designé" pour utiliser l'espace limité de l'écran et les contraintes de mémoire des PDA. Ce sous-ensemble contient toutes les caractéristiques requises pour développer une application PDA.

Il est préférable de tester l'interface sur toutes les plates-formes sur lesquelles l'application est destinée à tourner pour s'assurer que les choix établis pour l'interface n'ont pas d'impacts négatifs sur l'application. Chaque plate-forme a des caractéristiques différentes.

Lorsque la plate-forme ne gère pas certains éléments, l'implémentation ne renvoie pas d'erreur. L'application continuera son exécution. Elle ignore silencieusement la ou les méthodes concernant le changement. Les méthodes principalement non supportées par certaines plates-formes sont surtout la gestion des fenêtres telles que les frames, fenêtres et boîtes de dialogue au niveau du redimensionnement et du positionnement.

Certaines plates-formes ne supportent que le noir et blanc tandis que d'autres supportent une palette de couleurs plus large. Cependant, ces dernières peuvent ne pas permettre le changement de couleurs des éléments GUI standards tels que les menus ou les barres de titres.

Certaines implémentations interdisent la personnalisation du pointeur que ce soit par le développeur ou par l'utilisateur.

Il est préférable d'éviter l'utilisation de fonts différents dans la GUI. Si on veut utiliser des fonts différents, il faut s'assurer que la plate-forme les supportent bien tous et que l'application ne tournera pas sur d'autres implémentations.

Concernant la gestion des fenêtres (les frames, fenêtres, et boîtes de dialogues), il existe également des restrictions ou interdictions sur certaines plates-formes. Elles concernent en général les points suivants.

- Certaines implémentations permettent l'affichage de plusieurs fenêtres actives au même moment, mais pas toutes.
- Le redimensionnement est limité ou interdit dans beaucoup d'implémentations. Le redimensionnement concernant les boîtes de dialogue peut amener un résultat différent de celui espéré : une taille différente ou aucun changement. Il faut donc éviter au maximum les besoins de redimensionnement des boîtes de dialogue.
- Le positionnement peut également faire l'objet de restrictions. Il n'est alors pas possible de déterminer exactement l'emplacement de la boîte de dialogue. Il peut y avoir différentes restrictions ou simplement une complète interdiction.

Les méthodes `setTitle()` et `getTitle()` n'ont pas toujours d'effet sur la boite de dialogue ou sur le frame.

Pour exécuter une application PDAP, un appareil doit avoir la configuration matérielle suivante :

- Affichage : résolution :128x128 pixels à 240x320 pixels ; profondeur : 1bit, couleur : 1-bit noir et blanc à 64K.
- Entrée : un appareil de pointage et une entrée pour les caractère.
- Réseau : bi-bande, sans fil, largeur de bande limitée, intermittence possible.
- Processeur ROM et RAM : un minimum de 1000KB de mémoire totale (ROM et RAM) disponible pour l'exécution java et les librairies.
- Capacités de l'interface utilisateur : texte, dialogue, boutons, champs d'entrée de texte et images.

KAWT

kAWT est utilisable avec MIDP4Palm.

kAWT est un sous-ensemble d'AWT qui tourne sur la KVM contrairement à AWT.

Le but du projet kAWT est de fournir une version simplifiée d'AWT pour la KVM : les classes originales com.sun.kjava incluses dans J2ME CLDC Beta1 et les versions EA plus anciennes de la KVM diffèrent des composants d'interfaces utilisateur Java standards par bien des aspects. Porter des applications sur PDAs devient un peu compliqué. Le problème le plus important est que seul le support limité pour la gestion d'événements est fourni par la KVM. Cette version simplifiée d'AWT ne laisse pas tous les programmes AWT s'exécuter sur le Palm sans adaptation.

Chapitre 3

Prédispositions pratiques

3.1 Téléchargements

Avant de commencer, il faut vous procurer les outils suivants :

Un EDI : L'EDI utilisé dans ce tutorial est netbeans4 pour les facilités qu'il offre grâce à son mobile pack pour la programmation avec J2ME. Il en existe cependant beaucoup d'autres. Netbeans est disponible à l'adresse suivante :

http ://www.netbeans.org/

Sur le site de sun, il est nécessaire de télécharger :

- un JDK supérieur à 1.3 :
  http ://java.sun.com/j2se/index.jsp
- le profil MIDP :
  http ://java.sun.com/products/midp/
- la configuration CLDC
  http ://www.sun.com/software/communitysource/j2me/cldc/
- J2ME wireless toolkit :
  http ://java.sun.com/products/j2mewtoolkit/index.html

Téléchargez l'émulateur palm OS et les ROM's à l'adresse suivante :

http ://www.palmos.com/dev/tools/emulator/

Pour l'installation et la configuration des paths, il existe un très bon article en français :

http ://www.clubj2me.org/article.php ?sid=1

3.2 Exécution sur l'émulateur Palm

Il faut télécharger MIDP pour PalmOS :

http ://java.sun.com/products/midp4palm/download.html

Vous devez décompresser dossier zip, dans le répertoire PRCfiles, il y a un fichier MIPD.prc qu'il faut installer sur l'émulateur Palm. Il suffit de glisser l'icône sur l'écran de l'émulateur.

Pour pouvoir exécuter une MIDlet sur le Palm, il vous faut convertir le fichier .jad ou .jar de votre MIDlet en un fichier .prc qui est directement exécutable sur Palm OS. Dans le répertoire Converter du dossier midp4palm1.0, vous devez exécuter le fichier converter.bat.

Si à l'exécution, le converter déclare l'erreur suivante :

```
Error: Java path is missing in your environment

       Please set JAVA\_PATH to point to your Java directory

       e.g. set JAVA\_PATH=c:\bin\jdk1.3\
```

Il faut configurer la variable d'environnement JAVA_PATH. Pour ce faire, dans Windows, il faut aller dans le panneau de configuration dans System. Dans les propriétés du système, cliquez sur l'onglet "Avancé" et aller dans Variables d'environnement. Ajoutez une nouvelle variable d'environnement JAVA_PATH avec comme valeur un chemin vers un jdk supérieur au jdk 1.3.

3.3 Créer une application MIDP

Créer un nouveau projet J2ME MIDP – Dans File, choisir New Project (Ctrl-Shift-N).
En dessous de Categories, sélectionner Mobile. Sous Projects, sélectionner Mobile Application et cliquer sur Next.

- Sous Project Name, entrer MyHello. Changer le Project Home pour une autre directory de votre système. A partir de maintenant, nous ferons référence au dossier par $PROJECTHOME.
- Vérifier la check box Create HelloMIDlet. Cliquer sur Next.
- Laisser le J2ME Wireless Toolkit avec la Target Platform sélectionnée.
- Cliquer sur Finish. L'IDE crée le $PROJECTHOME./MyHello project folder. Le dossier du projet contient toutes vos sources et meta-données du projet. Le projet MyHello s'ouvre dans la fenêtre des projets.

Éditer le code source Java – "Expand" le noeud du projet MyHello et double-cliquer sur le noeud du code source HelloMIDlet.java. Le code source s'affiche dans le Source Editor.

- Dans la méthode startApp(), remplacer "test string" avec le texte de votre choix, par exemple, "Hello World."

Compiler et exécuter un projet hoisir Run Main Project (F6) du menu Run. Double-cliquer sur la fenêtre d'Output pour la maximiser, vous pourrez voir ainsi l'entièreté de l'output. Noter que le fichier HelloMIDlet.java est construit et pré-vérifié avant d'être exécuté. Un émulateur s'ouvre pour afficher les résultats de l'exécution de la MIDlet. L'émulateur par défaut est le DefaultColorPhone.

Dans la fenêtre de l'émulateur, cliquer sur le bouton en dessous de la commande Launch. L'émulateur lance la MIDlet et affiche le texte entré précédemment.

Cliquer sur le bouton en dessous d'Exit pour fermer la MIDlet. Ensuite cliquer sur le bouton dans le coin supérieur droit de l'émulateur pour fermer la fenêtre de l'émulateur.

Pour de plus amples informations, vous pouvez aller sur le site de netbeans : http ://www.netbeans.org/kb/articles/quickstart-mobility-40.html

Chapitre 4

Développement d'IHM avec l'API de haut-niveau

4.1 Module 1 : Les composants simples

4.1.1 Introduction

Ce module a pour objectif de familiariser le développeur à l'environnement de MIDP. Il introduit les composants suivants :

- *Display et Displayable* : ce sont les composants de base dans l'API de haut-niveau. Ils permettent d'encapsuler les différents composants qui seront visibles à l'écran.
- *Ticker*
- *TextBox*
- *Alert*

Les trois derniers composants sont les composants les plus simples de l'API.

4.1.2 Display et Displayable

Définition

Le Display représente l'écran de l'appareil au niveau hardware. Il encapsule les éléments qui seront visibles à l'écran. Le Displayable contient les objets qui sont visibles à l'écran. Pour ajouter un élément au Display, il faut qu'il soit d'abord inclus dans un objet héritant de Displayable, un screen (TextBox, Alert, Form, List) ou un canvas (API de bas niveau).

Code

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

public class HelloMid extends MIDlet {
    private Display display;

    public void startApp() {
```

```
        display = Display.getDisplay(this);
        }

        public void pauseApp() {
        }

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

Explications - Étapes

Les opérations réalisées quand la MIDlet est en background ne prennent effet que quand la MIDlet passe en foreground.

1. Déclaration de la variable :

   ```
   private Display display;
   ```

2. Appel de la méthode `getDisplay()` dans `startApp()` :

   ```
   display = Display.getDisplay(this);
   ```

   La méthode statique `getDisplay(MIDlet mid)` permet d'obtenir une référence vers la classe Display. Chaque MIDlet n'a accès qu'à une seule instance de cette classe qui lui est propre. Aussi, `getDisplay` renvoit toujours la même valeur à chaque appel.
   Une MIDlet invoque la méthode `getDisplay()` au premier appel de `startApp()`. Si la méthode `getDisplay()` doit effectivement se trouver dans startApp(), elle peut s'y trouver à n'importe quel moment.

La méthode `setCurrent()` permet d'associer un Displayable à un Display. Il ne devient visible qu'à ce moment là. La méthode `setCurrent` influence uniquement l'état interne de l'affichage Display et notifie au gestionnaire d'applications que la MIDlet voudrait voir le Displayable donné affiché à l'écran.

La méthode `getCurrent` permet de savoir ce qui est affiché à l'écran à un moment donné.

4.1.3 TextBox

Definition

Une TextBox est un Screen qui permet d'afficher et d'éditer du texte.

Explications - Étapes

1. Déclaration des variables

   ```
   private TextBox text;
   ```

2. Initialisation des variables

```
text = new TextBox("Write all you want",
            "You cannot write more character than 120",
            120, TextField.ANY);
```

Pour créer une TextBox, il faut spécifier les paramètres désirés dans le constructeur de la TextBox

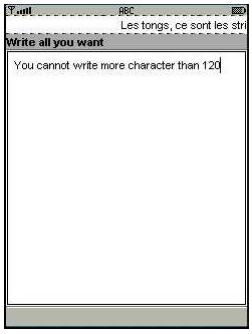
```
public TextBox(String title, String text, int maxSize,
        int constraints);
```

- *title* est utilisé comme titre de l'écran, de la TextBox.
- *text* détermine ce qui sera affiché à l'écran au début.
- *maxSize* détermine le nombre de caractères maximal qu'une TextBox peut contenir.
- *constraints* détermine le type de la TextBox. Elle peut être de type ANY, EMAILADDR, NUMERIC, PASSWORD, PHONENUMBER ou URL.

3. Ajout de la TextBox au Display

```
display.setCurrent(text);
```

Screenshots



4.1.4 Ticker

Definition

Le ticker implémente un texte qui défile continuellement à l'écran. Un ticker peut être attaché à un des quatres types de "screen" : "TextBox", "List", "Alert" ou "Form".

Explications - Étapes

1. Déclaration des variables

```
private String citation;
private Ticker tick;
```

2. Initialisation des variables

```
citation =
    "Les tongs, ce sont les strings des pieds.   Ph. Geluck    ";
tick = new Ticker(citation);
```

Remarquez les espaces ajoutés en fin de citation. Ils permettent une meilleure visibilité puisque une fois le texte terminé, il recommence depuis le début sans temps d'arrêt.
On crée un nouveau ticker grâce au constructeur suivant :

```
new Ticker(String str);
```

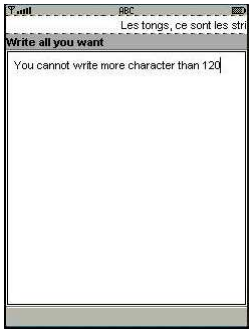
str : Le seul argument du constructeur est le texte qui défilera à l'écran. Aucun paramètres n'est fourni pour déterminer la direction ou la vitesse de défilement du texte. Ceux-ci sont déterminés par l'implémentation MIDP.

3. Ajout du Ticker à la TextBox

```
text.setTicker(tick);
```

Une fois initialisé, le ticker défile à l'écran, pour l'arrêter, il faut le supprimer au moyen de la méthode setTicker(null).

4.1.5 Screenshots



4.1.6 Alert

Definition

La classe Alert est une classe qui affiche à l'écran une boîte de dialogue. Il peut consister en un label, un texte ou une image. Il est possible de, soit définir le temps pendant lequel la boîte de dialogue est affichée avant de passer à un autre Screen, soit permettre à l'utilisateur de fermer lui-même la boîte.

Explications - Étapes

1. Déclaration des variables

```
private String title, content;
private AlertType alertTp;
```

2. Initialisation des variables

```
//Initialisation des variables nécessaires à alert
title = new String("Confirmation");
content = new String
    ("After this confirmation you have a TextBox");
alertTp = AlertType.CONFIRMATION;

Alert alert = new Alert (title,content,null,alertTp);
```

   Le constructeur utilisé ici est

```
public Alert(String title, String alertText, Image
alertImage, AlertType alertType);
```

   (a) *title* : String qui sera affiché comme titre de l'alerte.

   (b) *alertText* : Texte du corps de l'alerte.

   (c) *alertImage* : Image affichée à l'écran. Le seul format d'image accepté est le format png.

   (d) *alertType* : Les types d'alerte sont ALARM, CONFIRM, ERROR, INFO, WARNING.

3. Détermination du temps d'affichage de l'alerte

```
alert.setTimeout(Alert.FOREVER);
```

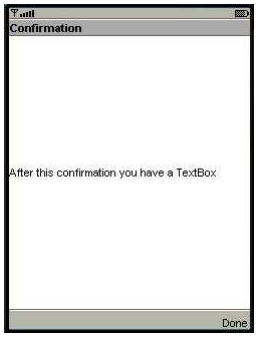
   Si on n'a pas une vision complète du message d'alerte et qu'il faut avoir recours au scroll, la limite de temps est automatiquement désactivée.

4. Ajout de l'alerte au display

```
display.setCurrent(alert, text);
```

   La méthode `setCurrent` possède ici deux paramètres, elle permet d'afficher l'alerte et de définir l'écran qui sera affiché à sa fermeture, dans ce cas-ci, c'est la TextBox text définie plus haut.

Screenshots



4.1.7 Code

```
// Il faut d'abord importer les packages
//contenant les classes midlet et lcdui

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

// Chaque application MID est dérivé de la classe MIDlet,
// il faut donc étendre la classe MIDlet

public class HelloMid extends MIDlet
    //implements CommandListener
{
    //déclaration des variables de base
    private Display display;
    private TextBox text;

    //déclaration des variables du ticker
    private String citation;
    private Ticker tick;

    //déclaration des variables pour l'alerte
    private String title, content;
    private AlertType alertTp;

    public void startApp() {
        display = Display.getDisplay(this);
        text = new TextBox("Write all you want",
                "You cannot write more character than 120",
                    120, TextField.ANY);

        //Initialisation et Ajout du ticker à la TextBox
        citation =
            "Les tongs, ce sont les strings des pieds.
                    Ph. Geluck    ";
        tick = new Ticker(citation);
        text.setTicker(tick);

        //Initialisation des variables nécessaires à alert
        title = new String("Confirmation");
        content = new String
                ("After this confirmation you have a TextBox");
        alertTp = AlertType.CONFIRMATION;

        //Initialisation de l'alerte
        Alert alert = new Alert (title,content,null,alertTp);
        alert.setTimeout(Alert.FOREVER);
```

```
            //Ajout de l'alerte au display, une fois fermée,
            //la TextBox sera affichée.
            display.setCurrent(alert, text);
        }

        public void pauseApp() {
        }

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

4.2 Module 2 : Les commandes

4.2.1 Définition

Pour créer des menus et ou des boutons, on utilise la classe "Commands". Elle permet d'implémenter des commandes qui, en fonction de l'environnement, seront affichées comme boutons et/ou dans un scroll menu.

4.2.2 Objectif

Le premier module de ce tutoriel a introduit l'environnement grâce à l'apprentissage des composants les plus simples. Vous êtes capable d'afficher des éléments à l'écran mais pas d'agir sur ces éléments, de voyager entre les screens ou de quitter l'application. Ce module va vous permettre d'avancer en ce sens.

4.2.3 Étapes

1. Déclaration et initialisation des variables

```
Command clear = new Command("Clear", Command.SCREEN, 0);
Command exit = new Command("Exit", Command.EXIT, 0);
```

   Pour créer une commande, on utilise le constructeur et on spécifie les arguments :

```
public Command(String label, int type, int prioriy);
```

   Une fois la commande créée, il est impossible de changer les arguments (label,type,priorité). Les arguments type et priorité sont utilisés lors du positionnement de la commande à l'écran.
   - *label* : Le label de la commande consiste en un String qui sera affiché à l'écran pour dénommer la commande.
   - *type* : Les différents types de commandes sont
     - Command.BACK permet à l'utilisateur de retourner à l'écran précédent.
     - Command.CANCEL quand on a besoin de notifier à l'écran une réponse négative.

  - Command.EXIT utilisée pour spécifier une commande de sortie de l'application
  - Command.HELP passé quand l'application requiert un écran d'aide.
  - Command.ITEM permet de dire à l'application qu'un item "explicite" a été ajouté au screen.
  - Command.OK permet de notifier au screen une réponse positive.
  - Command.SCREEN type qui spécifie une screen-specific Command de l'application. Pas de signification "generic".
  - Command.STOP interrompt une procédure en cours.
- *priority* : En fonction du niveau de priorité, les commandes seront affichées de manière plus ou moins accessible à l'utilisateur. Le niveau de priorité le plus élevé correspond à la plus petit valeur, c'est-à-dire 0.

2. Ajout des commandes au screen

```
text.addCommand(clear);
text.addCommand(exit);
```

   L'ajout de la commande à n'importe quelle sous-classe de Displayable se fait facilement grâce à la méthode `addCommand(Command Cmd)`.

   Une même commande peut être ajoutée à plusieurs screen. Il n'est donc pas nécessaire d'en créer plusieurs instances.

   L'ordre dans lequel les commandes sont ajoutées au screen n'a aucun impact sur leur positionnement. Il n'y a aucun layout ou autre moyen de positionnement fournis par MIDP. Tout est géré par ce dernier en fonction des caractéristiques matérielles de l'appareil et est déterminé uniquement par les paramètres de type et de priorité. La commande qui aura la plus petite valeur de priorité sera la plus accessible.

   Le type de la commande détermine le positionnement de la commande dans un menu. Il permet aussi de garder une certaine consistance avec les autres applications. Si le bouton EXIT est toujours en bas à gauche et que cela a été codé dans les propriétés de l'appareil, le bouton EXIT de la nouvelle application sera placé à cet endroit.

   Chaque appareil a un nombre de "soft buttons" déterminé. Ce type de boutons n'a pas de fonctionnalité définie. Une fonctionnalité peut leur être assignée dynamiquement grâce aux commandes. Si le nombre de commandes dépasse le nombre de "soft buttons" disponibles, les commandes affichées à l'écran sous forme de boutons seront celles qui auront la priorité la plus importante, les autres auront leur place dans un menu.

   (MIDP 2.0 Les commandes peuvent désormais être ajoutées à un item et plus seulement à un screen. Cela se fait de façon très simple. D'abord, il faut créer l'item puis y ajouter la commande et enfin enregistrer le command listener.

3. Gestion des événements

   Il existe deux types d'événements : l'événement `Screen` et l'événement `ItemStateChanged`. Les listeners correspondants sont respectivement CommandListener et ItemStateListener.

   Le traitement associé à une action effectuée sur une commande est effectué dans une interface CommandListener. Cette interface définit une méthode,

commandAction, qui est appelée si une commande est déclenchée. Le listener correspondant est mis en place en implémentant l'interface CommandListener. Vous devez alors enregistrer cette dernière avec la méthode `setCommandListener (CommandListener myListener)`.

Toute modification interactive de l'état d'un élément de formulaire déclenche un événement itemStateChanged (exemple : modification d'un texte, sélection d'un élément d'une liste, ...). Le listener correspondant est mis en place en implémentant l'interface ItemStateListener. Vous devez alors enregistrer l'objet ItemStateListener auprès d'un formulaire Form avec la méthode setItemStateListener (ItemStateListener myListener).

La classe `CommandsMidlet` doit implémenter l'interface `CommandListener` avec comme unique méthode `commandAction` implémentée plus bas.

```
public class CommandsMidlet extends HelloMid
       implements CommandListener
```

Pour être notifié quand un utilisateur active une Command, vous devez enregistrer un CommandListener avec le Displayable auquel la commande a été ajoutée. Vous pouvez faire cela en invoquant la méthode suivante :

```
public void
       setCommandListener(CommandListener cl);
```

Contrairement à Swing et à AWT, MIDP ne permet l'enregistrement que d'un seul CommandListener à un moment donné. Lorsque la méthode setCommandListener(CommandListener cl) est appelée, tout listener précédent est remplacé par le nouveau. Si la méthode est appelée avec l'argument `null`, le listener précédent est supprimé. Cela semble un peu limité, mais permet une meilleure gestion des ressources en évitant la prolifération de classes d'événements qui sont très chères en terme de mémoire. Cependant, même si cela semble quelque peu limité, les MIDlets peuvent faire énormément de choses avec seulement un listener par screen.

`CommandListener` est une interface dotée d'une seule méthode :

```
public void commandAction (Command c, Displayable d) {
    if(c==exit)
    notifyDestroyed();
    else if (c==clear)
        clear();
}
```

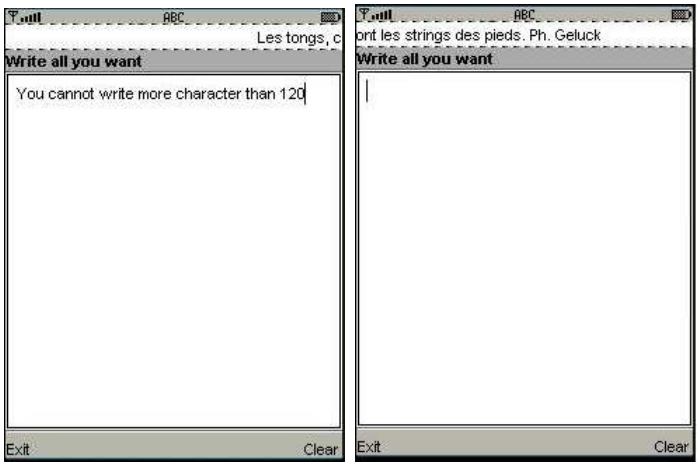
La méthode commandAction() est appelée quand n'importe quelle `Command` du Displayable est activée. C'est à l'intérieur de cette méthode que l'on notifie au gestionnaire quelles sont les actions à effectuer lors de l'activation d'une commande. La méthode `notifyDestroyed` notifie au gestionnaire que la MIDlet se termine volontairement.

– *Command c* : Cet argument est le plus utile, il permet de savoir quelle est la commande que l'utilisateur a activée. A ce moment, on appelle la méthode correspondant à l'action que l'utilisateur veut réaliser.

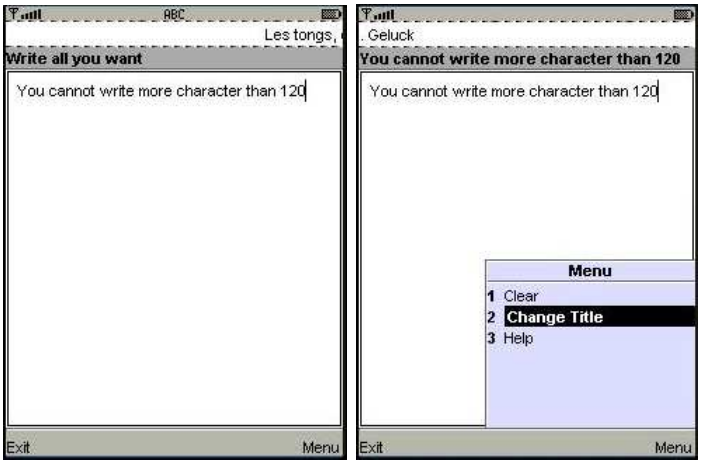
– *Displayable d* : Cet argument est utile lorsqu'une même commande est assignée à plusieurs screens et que l'action à réaliser dépend du screen courant ou lorsque l'action nécessite une référence au screen pour la réalisation de sa tâche.

4. Positionnement des commandes en fonction des places disponibles

   Si l'application propose deux commandes. Les deux commandes seront affichées à l'écran.



   Une nouvelle application propose quatre commandes. Ce nombre dépasse le nombre de "soft buttons" disponible sur la plupart des appareils. Les captures d'écran montrent comment les commandes sont agencées en fonction de l'appareil et du type de la commande.



5. code avec deux commandes

   Le code ci-dessous étend la précédente classe `HelloMid`. Cela évite la reprogrammation de tout le code et permet de voir uniquement les lignes qui implémentent les commandes, objet de ce module.

```
/*
* CommandsMidlet.java
*
```

```
* Created on 8 novembre 2004, 11:56
*/

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

/**
*
* @author  Sophie Van Tongelen
* @version
*/
public class CommandsMidlet extends HelloMid implements
CommandListener{
    //déclaration des diverses commandes

    static final Command clear = new Command("Clear", Command.SCREEN, 0);
    static final Command exit = new Command("Exit", Command.EXIT, 0);
    Display display;

    //implémentation des commandes
    public void commandAction (Command c, Displayable d) {
        if(c==exit)
        notifyDestroyed();
        else if (c==clear)
            clear();
    }

    //implémentation de l'action correspondante au bouton clear.
    public void clear(){
        text.setString("");
    }

    public void startApp() {
        boolean first = !started;
        super.startApp();

        //si c la 1ere exec de startapp, les commandes sont installées
        if(first){
        text.addCommand(clear);
        text.addCommand(exit);
        text.setCommandListener(this);
        }
    }

    public void pauseApp() {
    }

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

6. code avec quatre commandes

```
/*
* CommandsMidlet.java
*
* Created on 8 novembre 2004, 11:56
*/

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

/**
*
* @author  Domotech
* @version
*/
public class Commands2Midlet extends HelloMid implements
CommandListener{
    static final Command clear = new Command("Clear", Command.SCREEN, 1);
    static final Command exit = new Command("Exit", Command.EXIT, 0);
    static final Command help = new Command("Help", Command.HELP, 2);
    static final Command chgTitle = new Command("Change Title", Command.SCREEN, 2);

    Display display;

    //implem des commandes
    public void commandAction (Command c, Displayable d) {
        if(c==exit){
            destroyApp(true);
            notifyDestroyed();
        }
        else if (c==clear)
            clear();
        else if(c==help)
            help();
        else if(c==chgTitle)
            chgTitle();
    }

    public void help(){
    }

    public void chgTitle(){
        String newTitle = text.getString();
        text.setTitle(newTitle);
    }

    public void clear(){
```

```
            text.setString("");
        }

        public void startApp() {
            boolean first = !started;
            super.startApp();

            //si c la 1ere exec de startapp, les commandes sont installées
            if(first){
            text.addCommand(clear);
            text.addCommand(exit);
            text.addCommand(help);
            text.addCommand(chgTitle);
            text.setCommandListener(this);
            }
        }

        public void pauseApp() {
        }

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

4.3 Module 3 : Form et Items

4.3.1 Objectif

Ce module a pour but d'introduire la classe la plus importante de l'API de haut-niveau, la classe `Form`.

4.3.2 Définitions des classes

La classe `Form` est la sous-classe la plus importante de MIDP. Elle peut contenir autant d'objets `Item` que désirés.

La classe `Item` est une classe abstraite qui ne peut être instanciée. De nouveau, on ne peut agir sur la taille et le positionnement des items. Tout ça est géré par MIDP. Les sous-classes de `Item` sont : `ChoiceGroup, DateField, Gauge, ImageItem, StringItem, TextField`. Chacune sera développée en son temps plus bas.

La classe `List` fournit diverses listes dont l'utilisateur peut sélectionner un ou plusieurs items. Cette classe ne sera pas présentée ici car son utilisation, excepté le fait que ce soit un `Screen` à part entière, est très proche de l'item ChoiceGroup.

Form

1. Création d'un *Form*. La classe Form s'utilise comme tout autre classe `Screen`. Pour en créer une instance, on utilise un des deux constructeurs suivants dont on spécifie les paramètres :

```
public Form(String title);
public Form(String title, Item[] items);
```

   – *title* constitue le titre du screen `Form`.
   – *items* fournit les différents items que l'on veut ajouter au screen sous forme de tableau d'items.

   Le premier constructeur permet une meilleure structuration du code aussi ce sera celui utilisé ici.

```
private Form form;

//Initialisation
form = new Form("Items");
```

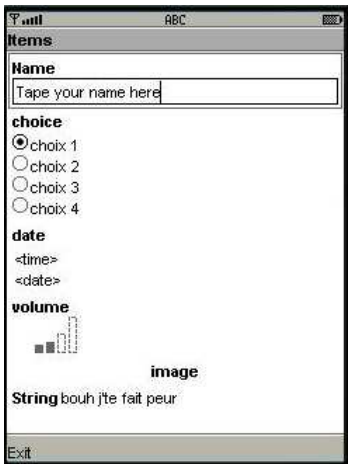
2. Ajout d'un item au screen form. Pour ajouter un item au screen form, la classe form fournit la méthode suivante :

```
public void append(Item it);
```

   Une même instance de `Command` ou de `Ticker` peut être ajoutée à plusieurs screens différents. Ce n'est pas le cas pour les items. Une instance d'item ne peut être ajoutée qu'à un screen `Form` à un moment donné.

3. Ajout du screen *form* au display par la méthode `setCurrent()`.

```
d.setCurrent(form);
```



4.3.3 Création d'items

ChoiceGroup

1. Définition
   L'item ChoiceGroup permet de fournir une liste d'éléments que l'utilisateur peut sélectionner. Il y a deux types de ChoiceGroup : EXCLUSIVE, MULTIPLE. La classe liste en fournit un troisième qui est le type IMPLICIT.
2. Explications - Étapes
   (a) Initialisation des variables nécessaires à choiceIt

```
labChoice = "choice";
choicetype = ChoiceGroup.EXCLUSIVE;
imgChc = null;
String[] elements =
    {"choix 1","choix 2","choix 3","choix 4"};
choiceIt =
    new ChoiceGroup(labChoice,choicetype,elements,imgChc);
```

   Pour créer un item ChoiceGroup, il faut passer quatre paramètres au constructeur :

```
public ChoiceGroup(String label, int choiceType,
String[] stringElements, Image[] imageElements);
```

   - *label*
   - *choiceType* :
     – EXCLUSIVE : ce type de ChoiceGroup fournit une liste d'éléments dont un seul peut être sélectionné à la fois.
     – MULTIPLE : dans ce type de ChoiceGroup, l'utilisateur peut sélectionner plusieurs éléments à la fois.
     Pour la liste, le type IMPLICIT permet d'envoyer une commande pour signaler le changement d'état suite à la sélection.
   - *stringElements* : ce paramètre est un tableau de string où chaque string désigne un élément de la liste et en est son label. Cependant, les éléments peuvent également être ajoutés grâce à la méthode `append(String str, Image img);`. Le string fait alors office de label et on peut y associer une image. On utilise alors le constructeur suivant :

```
public ChoiceGroup(String label, int choiceType);
```

   - *imageElements* : ce tableau d'images permet d'associer une image à chaque élément.

   (b) Ajout du ChoiceGroup au screen form

```
form.append(choiceIt);
```

DateField

1. Définition

L'item DateField permet d'afficher et de modifier la date et l'heure d'un objet de type `Date`.

2. Explications - Étapes
   (a) Initialisation des variables nécessaires à *dateIt*

```
labDF = "date";
mode = DateField.DATE_TIME;
dateIt = new DateField(labDF,mode);
```

   Pour créer un item `DateField`, il faut passer deux paramètres au constructeur :

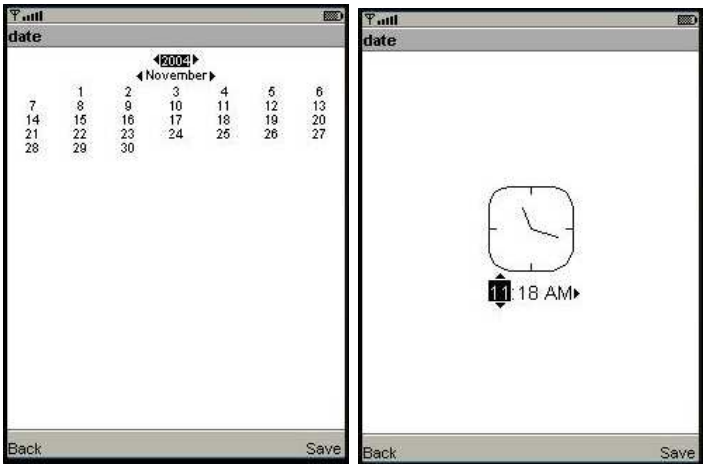
```
public DateField(String label,int mode,TimeZone timeZone);
```

   – *label* : c'est le label du DateField
   – *mode* : il y a trois mode :
     • DATE : Affiche uniquement la date.
     • TIME : Affiche uniquement l'heure.
     • DATE_TIME : Affiche la date et l'heure.
   – *timeZone* : permet de définir un fuseau horaire.

   (b) Ajout du `DateField` au screen form

```
form.append(dateIt);
```

3. Screenshots



Gauge

1. Définition
   L'item `Gauge` permet la visualisation de l'avancement et/ou d'un état à un moment donné. Elle se présente sous la forme de plusieurs barres verticales de même hauteur pour une jauge non interactive, d'hauteur croissante pour une jauge interactive.
2. Explications - Étapes
   (a) Initialisation des variables nécessaires à *gaugeIt*

```
labGauge = "volume";
interactive = true;
maxValue = 8;
initialVal = 4;
gaugeIt =
    new Gauge(labGauge,interactive,maxValue,initialVal);
```

Pour créer un item `Gauge`, il faut passer quatre paramètres au constructeur :

```
public Gauge(String label,boolean interactive,int
                maxValue,int initialValue);
```

- *label* : c'est le label de la jauge.
- *interactive* : une jauge interactive sera par exemple une représentation du volume que l'utilisateur pourra à son gré augmenter ou diminuer. Une jauge non interactive sera par exemple la visualisation du temps restant de chargement.
- *maxValue* : c'est la valeur maximal de la jauge.
- *initialValue* : c'est la valeur de la jauge affichée à l'écran à l'état initial

(b) Ajout de l'item `Gauge` au screen form

```
form.append(gaugeIt);
```

ImageItem

1. Définition
   Cet item permet d'afficher une image non interactive. L'image ne peut être que du format png.
2. Explications - Étapes
   (a) Initialisation des variables nécessaires à *imgIt*

```
labImg = "image";
Image img = null;
layout = ImageItem.LAYOUT_CENTER;
altTxt = "altText???"; ACHTUNG
imgIt= new ImageItem(labImg,img,layout,altTxt);
```

Pour créer un item ImageItem, il faut passer quatre paramètres au constructeur :

```
public ImageItem(String label,Image img,int
                                    layout,String altText);
```

- *label* : c'est le label de l'image.
- *img* : image qui sera affichée à l'écran.
- *layout* : Il y a 6 types de layout :
  - LAYOUT_CENTER : centrée horizontalement.
  - LAYOUT_DEFAULT : dépend de l'appareil

- LAYOUT_LEFT : alignée à gauche
- LAYOUT_NEWLINE_AFTER : une ligne est dessinée sous l'image.
- LAYOUT_ : une ligne est dessinée au-dessus de l'image.
- LAYOUT_RIGHT : alignée à droite

– *altText* : ACHTUNG

(b) Ajout de l'ImageItem au screen form

```
form.append(imgIt);
```

StringItem

1. Définition
   L'item StringItem permet d'afficher une chaîne de caractère à l'écran qui soit non interactive.
2. Explications - Étapes
   (a) Initialisation des variables nécessaires à strIt

```
labStr = "String";
txt = "bouh j'te fait peur";
strIt= new StringItem(labStr,txt);
```

Pour créer un item StringItem, il faut passer deux paramètres au constructeur :

```
public StringItem(String label,String text);
```

– *label* : c'est le label de la chaîne.
– *text* : constitue le corps du texte.

(b) Ajout du StringItem au screen form

```
form.append(strIt);
```

TextField

1. Définition
   L'item `TextField` permet d'afficher et d'éditer du texte. C'est le pendant de `TextBox`, dans la classe `Item`
2. Explications - Étapes
   Pour créer un `TextField`, on utilise le constructeur suivant :

```
public TextField(String label,String text,int
                                    maxSize,int constraints);
```

Les paramètres sont les mêmes que pour la `TextBox` introduite dans le premier module. Les contraintes sont également les mêmes.

4.3.4 Code complet

```
/*
* FormMidlet.java
*
* Created on 4 novembre 2004, 12:46
*/

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

/**
*
* @author  Domotech
* @version
*/
public class FormMidlet extends MIDlet
    implements CommandListener{

    private Form form;
    private Display d;
    static final Command exit =
            new Command("Exit", Command.EXIT, 0);

    // Déclaration des variables pour le ChoiceGroup
    private String labChoice;
    private int choicetype;
    private Image[] imgChc;
    private ChoiceGroup choiceIt;

    // Déclaration des variables pour le DateField
    private String labDF;
    private int mode;
    private DateField dateIt;

    // Déclaration des variables pour le Gauge
    private String labGauge;
    private boolean interactive;
    private int maxValue;
    private int initialVal;
    private Gauge gaugeIt;

    // Déclaration des variables pour l'ImageItem
    private String labImg;
    private Image img;
    private int layout;
    private String altTxt;
    private ImageItem imgIt;
```

```
// Déclaration des variables pour le StringItem
private String labStr;
private String txt;
private StringItem strIt;

// Déclaration des variables pour le TextField
private String labTF;
private String txtFd;
private int maxSize;
private int constraints;
private TextField txtFdIt;

public void commandAction (Command c, Displayable d) {
    if(c==exit){
        destroyApp(true);
        notifyDestroyed();
    }
}

public void startApp() {
    d =  Display.getDisplay(this);

    //Initialisation de form
    form = new Form("Items");

    //Ajout de la commande exit au screen form
    form.addCommand(exit);
    form.setCommandListener(this);

    //Initialisation des variables nécessaires à choiceIt
    labChoice = "choice";
    choicetype = ChoiceGroup.EXCLUSIVE;
    imgChc = null;
    String[] elements =
        {"choix 1","choix 2","choix 3","choix 4"};
    choiceIt =
        new ChoiceGroup(labChoice,choicetype,elements,imgChc);

    //Initialisation des variables nécessaires à dateIt
    labDF = "date";
    %mode = DateField.DATE_TIME;
    dateIt = new DateField(labDF,mode);

    //Initialisation des variables nécessaires à gaugeIt
    labGauge = "volume";
    interactive = true;
    maxValue = 8;
    initialVal = 4;
    gaugeIt =
        new Gauge(labGauge,interactive,maxValue,initialVal);
```

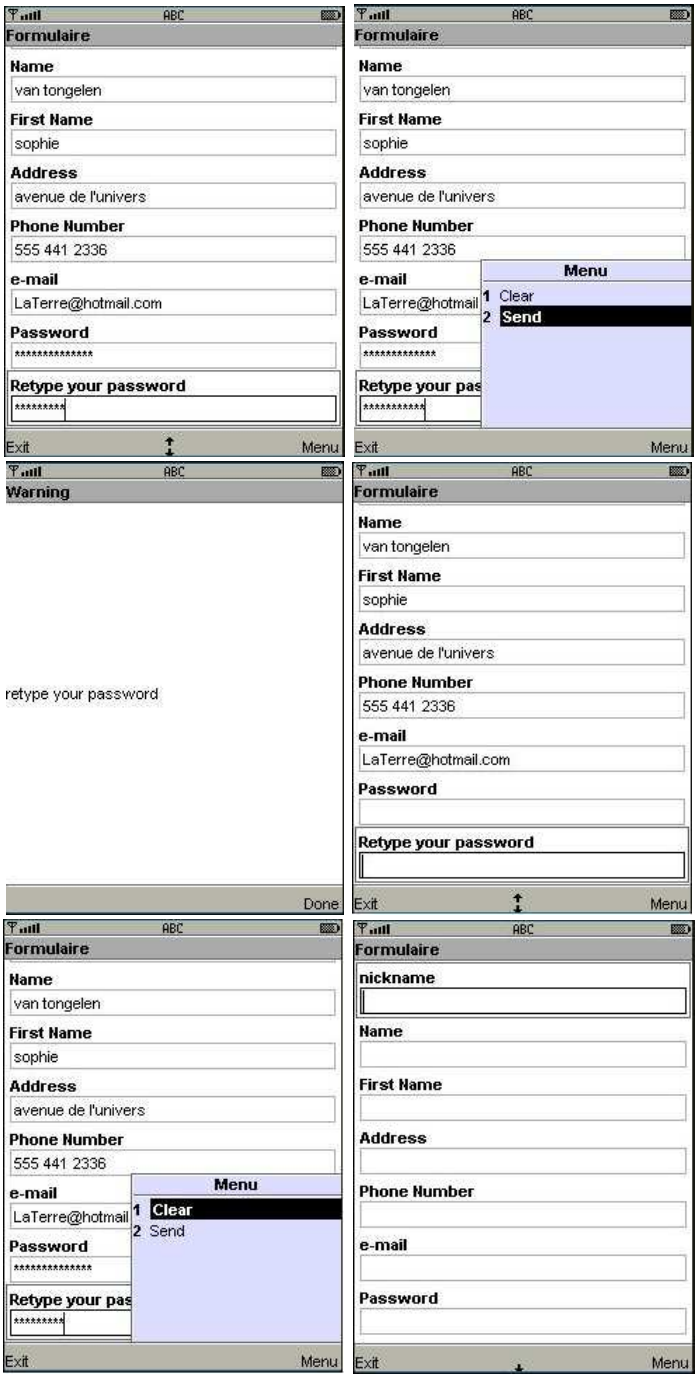
```
            //Initialisation des variables nécessaires à imgIt
            labImg = "image";
            Image img = null;
            layout = ImageItem.LAYOUT\_CENTER;
            altTxt = "altText???";
            imgIt= new ImageItem(labImg, img,layout,altTxt);

            //Initialisation des variables nécessaires à strIt
            labStr = "String";
            txt = "bouh j'te fait peur";
            strIt= new StringItem(labStr,txt);

            //Initialisation des variables nécessaires à txtFdIt
            labTF = "Name";
            txtFd = "Tape your name here";
            maxSize = 20;
            constraints = TextField.ANY;
            txtFdIt= new TextField(labTF,txtFd,maxSize,constraints);

            //Ajout des différents items au screen form
            form.append(txtFdIt);
            form.append(choiceIt);
            form.append(dateIt);
            form.append(gaugeIt);
            form.append(imgIt);
            form.append(strIt);
            d.setCurrent(form);
        }

        public void pauseApp() {
        }

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

4.4 Formulaire

Les captures d'écran ci-dessous montre le fonctionnement du formulaire.

La première figure représente le formulaire rempli. Les champs *Password* et *retype your password* sont remplis différemment de manière à produire une erreur lors de l'envoi du formulaire. Pour envoyer le formulaire, l'utilisateur va dans le menu et sélectionne *send* (figure 2). Lors de l'envoi, l'application détecte la non correspondance des mots de passe et demande à l'utilisateur de retaper un mot de passe via une alerte (figure 3). Lorsque l'utilisateur ferme l'alerte, l'application retourne au formulaire dont les champs concernant le mot de passe sont vides (figure 4). Si l'utilisateur veut remettre les champs du formulaire

à zéro, il utilise la commande *clear* dans le menu (figure 5). Les champs du formulaire sont alors vides (figure 6).



4.4.1 Code

```
/*
```

```
 * FormulaireMidlet.java
 *
 * Created on 10 novembre 2004, 17:10
 */

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

/**
 *
 * @author  Domotech
 * @version
 */
public class FormulaireMidlet extends MIDlet implements
CommandListener{
    Display display;
    Form form;
    TextField nick, name, first, address, phone, mail, pw, retypePw;
    Command exit, clear, send;

    public void clear(){
        nick.setString("");
        name.setString("");
        first.setString("");
        address.setString("");
        phone.setString("");
        mail.setString("");
        pw.setString("");
        retypePw.setString("");
    }

    public void send(){
        String verifpw1 = pw.getString();
        String verifpw2 = retypePw.getString();
        verifyPW(verifpw1,verifpw2);
    }

    public void commandAction(Command c, Displayable s) {
        if(c==exit){
            notifyDestroyed();
        }
        else if(c==clear)
            clear();
        else if(c==send)
            send();
    }

    public void verifyPW(String pw1, String pw2){
        if (!(pw1.equals(pw2))){
        Alert alert = new Alert ("Warning","retype your password",null,AlertType.WARNING);
```

```
            alert.setTimeout (Alert.FOREVER);
            display.setCurrent (alert,form);
            pw.setString("");
            retypePw.setString("");
            }
        }

        public void startApp() {
            display = Display.getDisplay (this);

            form = new Form("Formulaire");

            exit = new Command("Exit", Command.EXIT, 0);
            clear = new Command("Clear", Command.SCREEN, 0);
            send = new Command("Send", Command.SCREEN, 0);

            nick = new TextField("nickname","",20,TextField.ANY);
            first = new TextField("First Name","",20,TextField.ANY);
            name = new TextField("Name","",20,0);
            address = new TextField("Address","",20,0);
            phone = new TextField("Phone Number","",20,TextField.PHONENUMBER);
            mail = new TextField("e-mail","",20, TextField.EMAILADDR);
            pw = new TextField("Password","",20,TextField.PASSWORD);
            retypePw = new TextField("Retype your password","",20,TextField.PASSWORD);

            form.append(nick);
            form.append(name);
            form.append(first);
            form.append(address);
            form.append(phone);
            form.append(mail);
            form.append(pw);
            form.append(retypePw);

            form.addCommand(exit);
            form.addCommand(clear);
            form.addCommand(send);

            form.setCommandListener(this);

            display.setCurrent(form);
        }

        public void pauseApp() {
        }

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

4.5 MIDP 2.0

4.5.1 Objectifs

MIDP 2.0 amène plusieurs nouveaux éléments dans le développement des interfaces graphiques. Il permet plus de créativité. Ce module vous introduit les concepts de "CustomItem", qui permet de créer complètement un nouvel item, et offre deux nouveaux items : "Spacer" et "StringItem". MIDP 2.0 introduit également le concept de taille aux classes Item et CustomItem.

4.5.2 Étapes

La taille des items

Pour chaque item, standard ou personnalisé, vous pouvez éditer les tailles minimum et préférée de l'écran. La taille minimum d'un item est la plus petite taille à partir de laquelle un item peut être affiché et fonctionner correctement. La taille préférée fournit la taille optimale par rapport à l'écran. Tandis que les composants développés avec la classe Item peuvent prendre les caractéristiques standards de l'écran, les caractéristiques minimales et préférées doivent être établies manuellement pour les CustomItems.

De nouvelles méthodes ont été ajoutées pour permettre de définir la taille des items :

- setPreferredSize(int width, int height)
- int getPreferredHeight()
- int getPreferredWidth()
- int getMinimumWidth()
- int getMinimumHeight()

Spacer et StringItem

L'item "Spacer" permet d'espacer les éléments graphiques à l'écran. C'est un élément graphiques invisible dont on peut déterminer facilement la taille. L'item "StringItem" permet d'afficher un texte non-éditable par l'utilisateur.

```
import javax.microedition.lcdui.Spacer;

private Spacer spacer;

    // put some space between the items to segregate
    spacer = new Spacer(20, 20);
    form.append(spacer);
```

CustomItem

Les CustomItems peuvent prendre tous les deux des entrées au clavier numérique et au pointeur. Les modes d'interaction sont
KEY_PRESS, KEY_RELEASE, KEY_REPEAT, POINTER_DRAG,
POINTER_PRESS, et POINTER_RELEASE.

Une sous-classe de CustomItem peut rendre aisée la création interactive de composants tels que les tableurs ou les calendriers. La classe permet aux utilisateurs de mettre à jour le contenu directement dans le composant existant ou dans un composant secondaire tel qu'un TextField. Les mises à jour qui ont été entrées dans le composant secondaire sont automatiquement copiées dans le composant original lorsque la session est complète.

Pour construire un nouvel item, il faut définir une nouvelle classe qui étend la classe CustomItem. Celui-ci pourra ensuite être ajouté au "Form" comme n'importe quel autre "Item".

```
public class CustIt extends CustomItem {
    public CustIt( String exemple ){
        super( exemple );
    }
...
 }
```

```
        CustIt ci = new CustIt( "label" );
        mi.setLayout( Item.LAYOUT_CENTER |
                      Item.LAYOUT_NEWLINE_BEFORE |
                      Item.LAYOUT_NEWLINE_AFTER );

        f.append( ci );
```

Vous pouvez trouver l'ensemble des modifications dans le document de Nokia que vous trouverez en annexe ou à l'adresse suivante :
http ://ncsp.forum.nokia.com/download/ ?asset$_i d = 327; ref = devx$

Chapitre 5

Guidelines

Vous trouverez ci-dessous quelques principes de design d'une interface graphique pour PDA. Si vous voulez approfondir le sujet, vous pouvez allez sur les sites suivants.

Le site de PalmOS :

http ://www.palmos.com/dev/support/docs/ui/UIGuidelinesTOC.html

Le site de Symbian :

http ://www.symbian.com/

Pour plus d'informations sur l'ergonomie, vous trouverez divers liens intéressants à l'adresse suivante :

http ://interface.free.fr/Interface/ergonomie.html

La taille de l'écran impose une interface simple où le nombre de données n'est pas trop important. Il est préférable, par exemple, d'organiser en fonction des tâches, de manières à ce qu'ils ne possèdent pas trop d'items. De longs menus sont parfois difficiles à manipuler dans un environnement mobile.

Une interface utilisateur sur PDA doit être simple et facile d'utilisation à cause de la difficulté de navigation. Il faut donc avoir un bon équilibre dans la division des tâches pour ne pas en avoir trop et, en même temps, il ne faut pas exécuter trop d'actions pour arriver à une information simple.

Quand c'est possible, il est préférable de fournir une boîte de sélection plutôt que d'imposer à l'utilisateur d'entrer des données lui-même.

Avec J2ME, il est préférable d'utiliser l'API de haut-niveau pour permettre une plus grande portabilité des applications. Si on utilise l'API de bas-niveau, il vaut mieux garder la partie plate-forme indépendante. Cependant, il faudra parfois deux ou trois versions d'une même application pour qu'elle puisse tourner sur des environnements très différents.

Une interface ne devraient jamais être spécifique à une seule taille d'écran. Elle devrait s'y adapter dynamiquement en fonction des caractéristiques physiques. L'API de haut-niveau de MIDP permet déjà une excellente adaptation.

Bibliographie

[ABE01] David ABELE. Programmation de dispositifs électroniques à faibles ressources de calcul et j2me, 2001.

[ADA02] Lamont ADAMS. J2me : les bases de la programmation gui, 2002. http ://www.zdnet.fr/builder/programmation/java_c_cplusplus.

[And01a] Dean W. Andreakis. Java 2 micro edition, Avril 2001. http ://venus.cs.depaul.edu/MS-seminar/symposium2001/andreakis1.ppt1.

[And01b] Dean W. Andreakis. J2me cldc/midp overview ui proposal, février 2001.

[Del02] Bruno Delb. Présentation de j2me : Première partie, janvier 2002. http ://developpeur.journaldunet.com/tutoriel/jav/020129jav_j2me1_1.shtml.

[DOU04] Jean-Michel DOUDOUX. Développons en java, 2004. http ://perso.wanadoo.fr/jm.doudoux/java/tutorial/index.htm.

[Gig02] Eric Giguere. J2me optional packages, 2002. http ://developers.sun.com/techtopics/mobility/midp/articles/optional/.

[GIG05] Eric GIGUERE. Using custom items in midp 2.0, February 05. http ://developers.sun.com/techtopics/mobility/midp/ttips/customitem/.

[J2Ma] Importing existing j2me midp source code into netbeans ide 4.0. http ://www.netbeans.org/kb/articles/import-mobility-40.html.

[J2Mb] J2me devices. http ://developers.sun.com/techtopics/mobility/device/device.

[J2Mc] J2me midp development for netbeans ide 4.0 quick start guide. http ://www.netbeans.org/kb/articles/quickstart-mobility-40.html.

[J2Md] J2me : Step by step.

[J2Me] Java 2 micro edition (j2me). http ://krugazor.free.fr/J2ME.pdf.

[KEO03] Jim KEOGH. The new pda profile, jdj, janvier 2003. http ://www.sys-con.com/author/?id=1202.

[KN03] Jonathan Knudsen and Dana Nourie. Wireless development tutorial part i, September 2003. http ://developers.sun.com/techtopics/mobility/midp/articles/wtoolkit.

[Kon03] Mikko Kontio. Custom gui development with midp 2.0, 2003. http ://www-128.ibm.com/developerworks/library-combined/wi-developui/tmp0000.html.

[Kon04] Mikko Kontio. Architectural manifesto : Designing mobile user interfaces, 2004. http ://www-128.ibm.com/developerworks/wireless/library/wi-arch4/index.html.

[MAH01a] Qusay MAHMOUD. chapter 5 : Midp gui programming, learning wireless java, 2001. http ://developers.sun.com/techtopics/mobility/midp/chapters/wjqusay/ch5.pdf.

[MAH01b] Qusay MAHMOUD. Midp gui programming, learning wireless java, 2001. http ://www.onjava.com.

[Rit] Simon Ritter. Javatm 2 micro edition, connected limited device configuration, mobile information device profile, wireless messaging api (wma) mobile media api,. htp ://www.sun.com/developers /evangcentral.

[RIV01] Michel RIVEILL. J2me, Janvier 2001. http ://rangiroa.essi.fr/cours/info-mobile/02-slides-j2me.pdf.

[SP] Gillian Hayes Shwetak Patel, Heather Mahaney. Ui environments. http ://www.cc.gatech.edu/classes/AY2005/cs4470_fall/lectures/ lecture12-uienvironments.ppt.

[SUN] What's new in midp 2.0.

[SUN02] Midp style guide mobile information device profile (midp) 1.0a, Août, 2002. http ://java.sun.com/j2me/docs/alt-html/midp-style-guide7/.

[TOP02] K. TOPLEY. *J2ME in a Nutshell*. 2002.