Mobiles Services and Midlets

Infrastructure de services sur terminaux mobiles

Auteurs : Mesud HASANOVIC, Lucas PRETRE
Classe : ETR5

Table des matières

1 Introduction **2**

2 J2ME - Java 2 Micro Edition **2**
2.1 Qu'est-ce que J2ME 2
2.2 Architecture 2
2.3 CLDC - Connected Limited Device Configuration 3
2.4 KVM - K Virtual Machine 5
2.5 CDC - Connected Device Configuration 5
2.6 Les profils J2ME 6
2.7 MIDP - Mobile Information Device Profile 7

3 MIDlets - Mobile Information Device Applets **8**
3.1 Généralités 8
3.2 Structure d'une application MIDP 9
3.3 Communiquer avec le gestionnaire d'applications 10
3.4 Le paquetage d'interface utilisateur (GUI) 11

4 J2ME Wireless Toolkit **12**
4.1 Obtenir le kit de développement 12
4.2 Utiliser le Wireless Toolkit 12
4.3 Fichiers JAR et JAD 13
4.4 Création d'une première application MIDP 14
4.5 Déploiement d'une MIDlet sur un terminal réel 15

5 Conclusion **16**

1 Introduction

Le développement d'une application dans un environnement mobile n'est pas aisé, notamment en raison de l'infrastructure relativement lourde requise. Une infrastructure mobile dépend d'une multitude de paramètres dont essentiellement la diversité des équipements impliqués.

Pour résoudre ce problème, diverses solutions ont été envisagées par les principaux constructeurs (Motorola, Ericsson et Nokia). Plusieurs systèmes d'exploitation ont été proposés dont Symbian et Linux. Ces solutions sont intéressantes mais doivent respecter une certaine stabilité, ce qui n'est pas encore le cas. Actuellement, la solution communément utilisée est le paquetage *J2ME* (Java 2 Micro Edition).

Cette infrastructure permet le téléchargement d'applicatifs dans le téléphone portable et dans un premier temps, des jeux. L'objectif de ce mini-projet consiste justement en la démonstration des fonctionnalités essentielles de cet environnement. Il faudra tout d'abord mettre sur pied un environnement de développement pour J2ME. Un tel environnement est mis à disposition par Sun Microsystems et il est autonome, c'est-à-dire qu'il n'a pas besoin d'un réseau mobile ou d'un terminal réel car il emule des téléphones portables. Il sera ensuite possible d'écrire quelques petits programmes afin de tester cet environnement.

2 J2ME - Java 2 Micro Edition

2.1 Qu'est-ce que J2ME

Java 2 Micro Edition (J2ME) est une plateforme Java pour les appareils embarqués comme les téléphones portables, agendas électroniques, ordinateurs de bord de véhicules et autres. A l'instar de J2EE, J2SE et Java Card, J2ME est constitué d'un ensemble d'API standards défini par un groupe d'experts incluant des constructeurs, développeurs de logiciels et fournisseurs de services.

La plateforme J2ME fournit la puissance et les avantages de la technologie Java pour des dispositifs embarqués ou mobiles ; celle-ci inclut une interface utilisateur flexible, un modèle de sécurité robuste, une palette de protocoles réseaux intégrée et une gestion des applications locales et réseau.

2.2 Architecture

L'architecture de J2ME définit des configurations, des profils et des paquetages facultatifs. Elle est optimisée en fonction de la mémoire, la capacité de traitement et les possibilités d'entrées-sorties et ceci pour chaque catégorie de dispositifs. Les configurations sont constituées d'une machine virtuelle et d'un ensemble minimal de classes. Elles fournissent les fonctionnalités de base pour des dispositifs aux caractéristiques semblables. Actuellement, il y a deux configurations de J2ME :

- Connected Limited Device Configuration (CLDC)
- Connected Device Configuration (CDC)

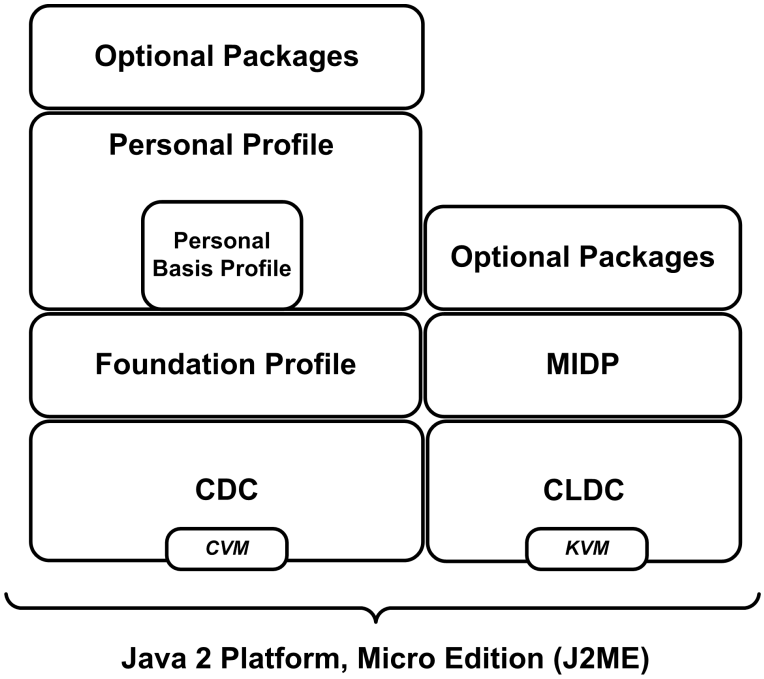

FIG. 1 – Java 2 Micro Edition

Il est important de remarquer sur la figure 1 qu'au dessus des profils se trouvent des bibliothèques optionnelles que chaque constructeur est libre d'ajouter à son produit. Le paquetage optionnel qu'on risque de trouver le plus souvent actuellement est celui gérant la technologie Bluetooth. Parfois, les fonctionnalités ajoutées sont propriétaires et ne s'appliquent que pour les produits d'un fabriquant. L'exemple type concerne les bibliothèques pour les jeux. Afin d'exploiter au mieux les particularités matérielles des appareils, les constructeurs proposent leurs propres API dédiées aux jeux. Il faut donc être conscient que l'utilisation de telles API limite la portabilité.

2.3 CLDC - Connected Limited Device Configuration

La CLDC est la plus petite des deux configurations, conçue pour des dispositifs avec des processeurs lents, une capacité mémoire limitée ainsi que des connexions réseau intermittantes. Ces dispositifs ont typiquement des processeurs 16 ou même 32 bits et de 128 à 512 ko de mémoire disponible pour la plateforme Java et ses applications associées.

La CLDC concerne les domaines suivants :

- le langage Java et les fonctionnalités d'une machine virtuelle
- les bibliothèques "noyau" de Java (`java.lang.*, java.util.*`)
- les entrées-sorties
- le réseau
- la sécurité
- l'internationalisation

Par contre, la CLDC **ne** concerne **pas** :

- la gestion de la durée de vie de l'application
- l'interface utilisateur
- la gestion des événements
- l'interaction entre l'utilisateur et l'application

La spécification de la CLDC concernant la sécurité impose que :

- une étape de vérification des classes téléchargées soit exigée pour assurer la sécurité bas-niveau de la machine virtuelle ;
- les applications soient protégées les unes des autres en les exécutant dans des contextes fermés (*sandbox*) ;
- les classes des paquetages systèmes protégés ne peuvent pas être redéfinies par une application.

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

- pas de types de données à virgule flottante (`float` et `double`)
- pas de JNI™(Java Native Interface™)
- les chargeurs de classe ne peuvent pas être définis par le développeur
- pas de paquetage `java.lang.reflect`
- pas de groupes de `Threads` ni de `Threads` démons
- pas de finalisation des instances de classes ; la méthode `Object.finalize()` n'existe pas
- pas d'optimisation de la mémoire avec les références faibles (*weak references*)
- la gestion des erreurs est limitée

Le premier point ci-dessus provient du fait que la majorité des dispositifs-cibles de la CLDC n'ont pas le matériel nécessaire pour l'arithmétique à virgule flottante. Les autres points ont été omis soit en raison de problèmes de sécurité, soit en raison de limitation de mémoire.

La CLDC requiert que la Java VM puisse identifier et rejeter les classes invalides. Cependant, la vérification standard définie par le J2SE demande trop de ressources (mémoire et consommation). C'est pourquoi la CLDC définit un autre méchanisme. Il s'agit en fait de faire une pré-vérification (figure 2) de la classe avant chargement sur les terminaux mobiles. Ce procédé ajoute à la classe un attribut "stackmap" qui augmente sa taille d'environ 5%. Ainsi la vérification sur le terminal destinataire est plus rapide et utilise moins de mémoire ; ceci en préservant le même niveau de sécurité.

Seules les classes appopriées pour les dispositifs mobiles sont présentes dans l'API de la CLDC. Celles-ci sont des sous-ensembles des plus grandes éditions de Java comme J2SE et J2EE. Ces dernières fournissent une vaste palette de fonctionnalités d'entrée-sorties pour des systèmes de stockage et de communication réseau. Cependant, vu les contraintes des terminaux mobiles, il est difficile d'y inclure toutes ces fonctionnalités. Ainsi, l'approche des connexions est généralisée, c'est-à-dire qu'elles sont créées à partir d'une seule classe système `javax.microedition.Connector`.

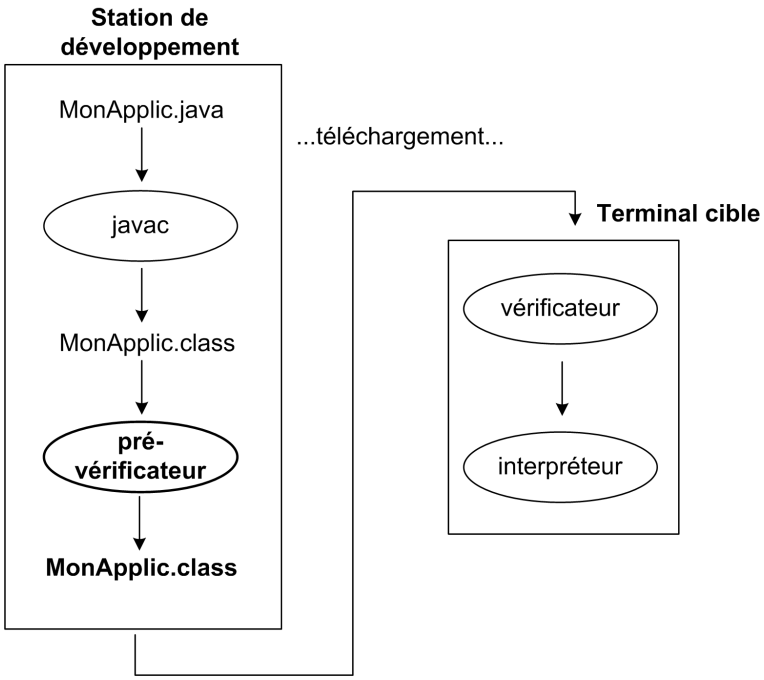

FIG. 2 – Pré-vérification de classe

2.4 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, organiseurs 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 (JAM™- Java Application Manager™) qui joue le rôle d'interface entre le système d'exploitation hôte et la machine virtuelle.

2.5 CDC - Connected Device Configuration

La CDC fournit une machine virtuelle et des bibliothèques de classes de base aux applications Java adressées aux appareils tels que les PDA avancés, les TV numériques interactives, les tablettes d'accès à Internet, etc... Typiquement, ces dispositifs sont basés sur un processeur de 32 bits et ont plus de 2 Mo de mémoire totale pour le stockage de la machine et des bibliothèques virtuelles. La CDC contient la machine CVM (Compact Virtual Machine). CVM est une machine virtuelle complète conçue pour des dispositifs ayant besoin de l'ensemble des fonctionnalités et services de Java 2, mais avec une taille réduite. Cette configuration s'inscrit donc dans le cadre d'une architecture Java presque complète.
En principe, les dispositifs basés sur CDC doivent obligatoirement disposer d'une connexion réseau.

Pour un environnement J2ME complet, la CDC emploie le profil de base (Foundation), qui est un ensemble d'API Java fournissant les fonctionnalités communes à tous les dispositifs cibles.

2.6 Les profils J2ME

Les configurations fournissent le fondement d'une application, alors qu'un profil en fournit la structure. Les profils définissent l'ensemble complet de classes API qui seront mises à disposition d'une application J2ME et sont conçues spécifiquement pour une configuration donnée. On y trouve des méthodes permettant de gérer l'affichage, la saisie utilisateur et la gestion de la persistance des données. Sun propose deux profils de référence J2ME :

- le profil Foundation
- le profil Mobile Information Device Profile (MIDP)

Le profil *Foundation* est destiné à la configuration CDC et offre par conséquent toute la richesse d'une machine virtuelle quasiment identique à la machine virtuelle standard. Cela signifie que les développeurs utilisant le profil *Foundation* auront accès à une implémentation complète des fonctionnalités de J2SE.

Le profil MIDP, conjointement à la configuration CLDC, fournit la plateforme Java destinée aux appareils mobiles actuels. Il prend en charge un sous-ensemble limité de la bibliothèque de classes J2SE et définit des classes d'entrées-sorties et d'interface utilisateur personnalisée pour une utilisation sur une configuration CLDC.

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

- PDA
- PersonalJava Basic
- PersonalJava
- RMI (Remote Method Invocation)
- Game
- ...

A part le profil PDA qui cible les agendas électroniques du type *Palm Pilot* et qui est basé sur CLDC, tous les autres s'appuient sur la configuration CDC. Le profil *Personal* s'adresse spécialement aux décodeurs TV numériques intégrant les fonctionnalités de courrier électronique et de navigation Web. Quant au *Personal Basic*, il cible les applications dans les domaines de l'automobile et de la télématique. Le profil *Game* est destiné aux consoles de jeu fixes ou mobiles. Finalement RMI est un profil dédié pour les applications distribuées.

Il existe également des profils plus spécialisés dont voici un extrait :

- J2ME/CLDC pour JStamp (Droid et Robot MindStorm)
- STIP (Small Terminal Interoperability Platform) pour les terminaux de paiement
- iMODE
- TINI (Tiny InterNet Interface)
- ...

On se focalisera plus partculièrement sur MIDP car il est destiné au gigantesque marché des téléphones portables qui a besoin de nouveaux services pour continuer à se développer.

2.7 MIDP - Mobile Information Device Profile

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

Aujourd'hui, on est confronté à deux versions de MIDP. En effet la version 2.0 est d'ores et déjà disponible. Elle répond mieux aux appareils actuels et permet de mieux les exploiter. Bon nombre de constructeurs leaders de la téléphonie mobile se sont associés pour développer ensemble ce standard, qui permet d'étoffer les technologies Java sur les téléphones mobiles. Le standard MIDP 2.0 supporte les nouvelles et toutes dernières applications de jeux, graphiques, applications vidéo, vocales et sécuritaires, de même que bon nombre d'autres fonctions liées aux téléphones mobiles et aux organiseurs PDA. Dans un premier temps, on va examiner les fonctionnalités de la première édition puis on mentionnera les nouveautés apportées par la seconde.

Le MIDP v1.0 contient les trois bibliothèques suivantes :

- `javax.microedition.midlet`
- `javax.microedition.lcdui`
- `javax.microedition.rms`

La première s'occupe de la gestion du cycle de vie des applications appelées MIDlets (Mobile Information Device Applet). La seconde gère l'affichage sur écran LCD des différents composants. Il faut préciser que l'interface graphique n'est pas très complexe et qu'elle est plutôt orientée formulaire. La troisième est chargée de l'enregistrement persistant des données (base de données) et offre des objets tel que le `RecordStore`.

Le MIDP v2.0, comme on l'a précisé ci-dessus, offre la même chose que l'édition précédente avec néanmoins quelques améliorations qui concernent principalement l'affichage. A part quelques "retouches", MIDP v.2.0 apporte de nouveaux paquetages que voici :

- `javax.microedition.lcdui.game`
- `javax.microedition.media`
- `javax.microedition.media.control`
- `javax.microedition.pqi`

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

Support jeux : Le MIDP 2.0 ajoute une API jeux standard pour les jeux complexes, utilisant ainsi le potentiel graphique des appareils pour simplifier le développement et mieux contrôler les fonctions graphiques et la performance.

Support média : Avec le MIDP 2.0, les développeurs peuvent exploiter toutes les fonctionnalités audio des appareils modernes. Les applications MIDP utilisant une plateforme standard sont capables d'intégrer les séquences de tonalités et les fichiers WAV.

Connectivité élargie : Le standard MIDP 2.0 est compatible avec les principaux standards de connectivité, élargis au-delà de l'HTTP à l'HTTPS, au datagrammes, aux sockets et sockets serveurs, ainsi qu'à la communication par interface série, proposant ainsi différents moyens d'échanger des informations.

Sécurité de bout en bout : Pour protéger le réseau, les applications et les mobiles, le standard MIDP 2.0 est assorti d'une sécurité de bout en bout, robuste et basée sur des standards ouverts. Ce modèle de sécurité supporte l'HTTPS et s'appuie sur des standards généraux de type SSL et WTLS pour transmettre des informations cryptées.

Méthode OTA (Over The Air) : La méthode OTA fait désormais partie du standard MIDP. Elle permet de reconnaître, d'installer, d'actualiser et d'éliminer des suites MIDlet sur les mobiles. Un fournisseur de services peut donc identifier le type de suites MIDlet qui fonctionnent sur un appareil spécifique, et vérifier ensuite l'état de l'installation, de l'actualisation ou de l'élimination des suites MIDlet sur ce même appareil.

Architecture push : Le MIDP 2.0 intègre un modèle d'architecture serveur *push*, qui permet d'enregistrer les MIDlets, et de les activer dès qu'un appareil reçoit une information du serveur. Ainsi, on pourra consulter en temps réel sur un mobile les actualités, la bourse et les enchères en ligne.

3 MIDlets - Mobile Information Device Applets

3.1 Généralités

Les MIDlets (ou applications MIDP) sont les éléments principaux d'une application Java embarquée. Pour mieux comprendre leur fonctionnement, on peut faire l'analogie avec les *applets* ou les *servlets*. Pour rappel, le cycle de vie d'une applet est géré par un conteneur, le navigateur web en l'occurance, dont le rôle est d'interagir avec celle-ci sous la forme de méthodes de notification prédéfinies (`init(), paint(), destroyed(), ...`).
Une servlet possède les mêmes caractéristiques qu'une applet excepté le fait que le conteneur est un moteur de servlet (Tomcat, WebSphere, WebLogic, ...).

Quant aux MIDlets, elles représentent le pendant des applets et des servlets pour J2ME avec comme conteneur un téléphone mobile ou un organiseur personnel. Ainsi, en cas de mise à jour d'une

application embarquée, un simple téléchargement du code de la MIDlet via un serveur quelconque est possible. Un programme développé pour un profil particulier est en mesure de s'exécuter sur tous les périphériques appartenant à la même famille.

3.2 Structure d'une application MIDP

La procédure de création, d'installation et d'exécution d'une application MIDP est différente de celle utilisée pour une application Java standard. On va donc examiner la structure d'une application MIDP à travers un exemple, `HelloMIDlet`, dont voici le listing :

```
/**
 * <p>Title: HelloMIDlet</p>
 * <p>Description: Première MIDlet semblable à un "Hello World"</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Mesud HASANOVIC et Lucas PRETRE
 * @version 1.0
 */

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

public class HelloMIDlet extends MIDlet implements CommandListener {
  private Display affichage;
  private TextField champTexteHello;
  private Command boutonQuitter;
  private Form formPrincipal;

  public HelloMIDlet() {
    // Retourne l'objet Display unique à cette MIDlet
    affichage = Display.getDisplay(this);

    // Crée le form principal
    formPrincipal = new Form("HelloMIDlet");

    // Crée le bouton de commande pour quitter
    boutonQuitter = new Command("Quitter", Command.SCREEN, 1);

    // Crée un champ texte d'une ligne et de 15 caractères de long
    // avec un étiquette (label) "Texte"
    champTexteHello = new TextField("Texte", "Hello World!", 15, TextField.ANY);

    // Ajoute les composants à un form et met en place
    // l'écouteur d'événements
    formPrincipal.addCommand(boutonQuitter);
    formPrincipal.append(champTexteHello);
    formPrincipal.setCommandListener(this);
  }

  public void startApp() {
    // Sélectionne affichage comme objet visible
    affichage.setCurrent(formPrincipal);
  }

  public void pauseApp(){/*L'application va être mise en pause*/}

  public void destroyApp(boolean unconditional) {/*L'application se termine*/}

  public void commandAction(Command c, Displayable s) {
    // Si on "clique" sur le bouton quitter
    if (c == boutonQuitter) {
      // La méthode destroyApp() doit être appelée manuellement
```

```
      destroyApp(false);
      // Demande au gestionnaire d'application de terminer l'application
      notifyDestroyed();
    }
  }
}
```

Cet exemple est le traditionnel *Hello World !* adapté pour les MIDlets. Comme on peut le constater, deux paquetages sont importés : `javax.microedition.midlet` et `javax.microedition.lcdui`. Le premier paquetage comporte la classe de base `MIDlet` de l'application MIDP et le second fournit un ensemble d'éléments "interface utilisateur" de type `Swing`.

`HelloMIDlet` hérite de la classe de base `MIDlet`. Cette dernière propose trois méthodes abstraites dont le gestionnaire d'applications de l'appareil se sert pour communiquer avec les applications qu'il exécute. La méthode `startApp()` est appelée immédiatement après le constructeur et à chaque fois qu'une application est activée (et non pas seulement lors de son lancement initial !). Une application peut passer de l'état actif à l'état inactif plusieurs fois au cours d'une même exécution ; par conséquent il n'est pas judicieux d'y placer des instructions d'initialisation (ne devant être exécutée qu'une seule fois). Ce genre d'instructions est bien évidemment à placer dans le constructeur.

La méthode `destroyApp()` est appelée par le gestionnaire pour indiquer qu'une application est sur le point de se fermer. Contrairement à `startApp()`, cette méthode ne sera appelée qu'une seule fois au cours de l'exécution d'une application ; on peut donc y inclure du code de nettoyage. Vu que la spécification de la CLDC n'inclut pas la méthode `finalize()` pour les objets, il faudra effectuer le nettoyage ici. Cependant, un appareil mobile est généralement moins stable que la plateforme standard et sera donc régulièrement éteint ou réinitialisé par l'utilisateur. En conséquence, on ne peut pas compter sur le fait que la méthode `destroyApp()` soit exécutée.

La troisième et dernière méthode abstraite, `pauseApp()` est à première vue étrange. Son objectif est de signaler que l'application va se mettre en pause. Cet événement se produit lorsque l'utilisateur passe à une autre application ou se sert d'une fonction de l'appareil qui empêche la poursuite de l'exécution de l'application. Comme la plupart des appareils n'a pas la puissance nécessaire pour être réellement multitâche, une application sera dans ce cas de figure. Il faut donc veiller à libérer un maximum de ressources au sein de cette méthode. A la reprise de l'application, le gestionnaire appelera la méthode `startApp()`.

3.3 Communiquer avec le gestionnaire d'applications

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

`notifyDestroyed()` indique au gestionnaire que l'application demande sa fermeture. L'appel de cette méthode n'entraînera pas l'exécution de la méthode `destroyApp()`, il faudra donc l'appeler manuellement.

`notifyPaused()` notifie le gestionnaire que l'application demande sa mise en pause.

`resumeRequest()` demande au gestionnaire d'effectuer la reprise de l'application si celle-ci a été mise en pause.

`getAppProperty()` extrait des informations de configuration concernant une application, à partir d'un fichier "manifest" ou d'un fichier descripteur de l'application. Pour simplifier, on peut dire que cette méthode accède à un fichier d'initialisation privé.

3.4 Le paquetage d'interface utilisateur (GUI)

Le paquetage `javax.microedition.lcdui` contient les éléments de l'interface utilisateur exploités avec les applications MIDP. La plupart de ces éléments a des équivalents `Swing`, bien que les noms soient différents. Le système d'événements sous-jacent fonctionne en grande partie de la même manière pour les deux API. On pourra donc enregistrer de la même façon des objets d'écoute d'événements, comme cela est fait dans le constructeur de l'exemple "fil rouge".
Dans ce cas, `HelloMIDlet` implémente l'interface `CommandListener` et sert d'objet d'écoute pour l'unique objet `Command` qu'`HelloMIDlet` contient. Cependant, d'autres techniques sont envisageables.

Ce même paquetage définit principalement trois familles de composants GUI, regroupées ici en fonction de leur classe de base :

Les composants `Screen` (écran) proviennent de la classe abstraite `Screen` et fournissent les gadgets logiciels GUI de type fenêtre traditionnelle. L'objet `Form` utilisé dans `HelloMIDlet` est un descendant de `Screen` qui peut contenir et afficher des contrôles GUI. Parmi les autres composants `Screen` figurent `Alert` (une boîte de dialogue), `List` (affiche des groupes d'actions) et `TextBox` (écran de saisie de texte multiligne).

Les composants `Item` (rubrique) correspondent aux contrôles traditionnels, comme le composant `TextField` (dans notre "fil rouge" : "Hello World !") qui provient de la classe abstraite `Item`. Celle-ci fournit une API uniforme pour l'attribution de label, la gestion d'événements et l'affichage de contrôles. `ChoiceGroup`, `DateField`, `Gauge`, `ImageItem` et `StringItem` sont d'autres composants `Item`.

Les composants `Miscellaneous Displayable` (affichages divers) proviennent tous de la classe abstraite `Displayable` de niveau supérieur ou bien y sont liés d'une manière ou d'une autre. Cette famille inclut des composants tels que `Command` (un bouton de commande), `Ticker` (texte défilant), `Graphics` (graphiques) et `Choice` (interface pour manipuler des sélections prédéfinies).

Cet ensemble graphique est géré par un objet `Display` dont chaque application a accès à une instance privée unique. Cette instance peut être extraite via la méthode statique `Display.getDisplay (MIDlet m)`. Il est courant de conserver une référence unique à cette instance dans une variable d'instance (comme fait dans l'exemple "fil rouge"). En plus des méthodes pour définir la cible sur un élément `Screen` particulier (`setCurrent()`) et extraire cet élément avec la cible (`getCurrent()`), `Display` propose également diverses méthodes servant à obtenir des informations sur les capacités d'affichage de l'appareil, notamment pour savoir si la couleur est supportée(`isColor()`) et combien de couleurs sont affichables (`numColors()`).

4 J2ME Wireless Toolkit

4.1 Obtenir le kit de développement

Pour ce faire, il suffit de se rendre sur le site de Sun Microsystems et de télécharger une des 2 versions disponibles du *Wireless Toolkit*. En effet, la première version, c'est-à-dire la 1.04, exploite le MIDP 1.0 et la seconde le MIDP 2.0. C'est donc principalement sur la version du profil J2ME fourni que se différencient ces deux éditions de l'utilitaire. Cette plateforme de développement existe aussi bien pour Linux que pour Windows ou Solaris SPARC. Il faudra néanmoins des droits d'administrateur pour l'installer.
Une fois cette boîte à outils installée, tout ce dont on a besoin pour compiler, pré-vérifier et simuler des applications J2ME se trouve désormais sur le disque.
A part la plateforme *Sun*, il en existe une panoplie d'autres. Presque chaque constructeur de téléphones portables en offre une avec les modèles de simulation identiques à leur gamme de produits réels. L'éditeur *Borland* fournit également une extension (*MobileSet*) à son célèbre logiciel *JBuilder* afin de s'adapter à la technologie J2ME.
Nous nous concentrerons malgré tout sur le *Wireless Toolkit* de *Sun* car il est largement suffisant, simple à installer et à utiliser, neutre de toute implémentation et indépendant de l'éditeur du code Java.

4.2 Utiliser le Wireless Toolkit

Les méthodes de création et de déploiement d'applications J2ME sont différentes par rapport aux applications standards. En effet, il n'existe pas de vérificateur de classe d'exécution pour l'implémentation MIDP. Une fois que le code est compilé et avant qu'il puisse être exécuté sur un appareil MIDP, il doit être prévérifié à l'aide de l'outil `Preverify.exe`. De plus, un ensemble de bibliothèques de classes et de fichiers de démarrage doit être inclus via un commutateur de ligne de commande avec le compilateur Java. Le Wireless Toolkit est capable d'effectuer ces actions supplémentaires et permet donc de ne pas s'en soucier. La création d'une application avec le Wireless Toolkit implique cinq étapes de base :

1. Lancer le toolkit (*KToolbar*)
2. Créer un projet à l'aide du bouton *Create Project*. Saisir le nom du projet ainsi que le nom de la classe MIDlet principale (sans le .class) qui devra être exécutée pour démarrer l'application. Le toolkit crée alors un répertoire nommé avec le nom du projet. Ce répertoire se situe dans `<répertoire d'installation>\apps\<le nom du projet>`
3. Vérifier que les propriétés du projet qui s'affichent sont correctes. Elles vont être décrites plus loin dans ce document.
4. Rédiger le code source Java dans l'éditeur de votre choix, puis copier le fichier correspondant dans le sous-répertoire `src` du projet.

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

Project - Package - Create Obfuscated Package. Malheureusement *Sun* n'a pas intégré d'office cet utilitaire. Un message expliquant la marche à suivre pour l'installer apparaîtra lors de la première tentative de création d'un paquetage protégé.

4.3 Fichiers JAR et JAD

Les applications MIDP sont organisées en suites pouvant contenir une ou plusieurs applications MIDP. Chaque suite MIDP est distribuée sous la forme d'une archive Java ou JAR. Le Wireless Toolkit peut la créer par le biais du menu *Project - Package*, comme expliqué précédemment. Chaque fichier JAR comporte un fichier *manifest* qui fournit des informations sur les classes qu'il contient. La boîte de dialogue *Project Settings* qui apparaît immédiatement après la création d'un projet Wireless Toolkit permet de définir le contenu de ce fichier *manifest*. Les manifestes supportent neuf attributs dont six doivent être définis pour chaque application MIDP. Ces attribus sont illustrés dans le tableau 1.

MIDlet-Name	Nom de la suite
MIDlet-Version	Numéro de version de la suite
MIDlet-Vendor	Développeur de la suite
MIDlet-<x>	Une entrée pour chaque application MIDP de la suite, contenant le nom de l'application, une icône optionnelle et le nom de la classe qui doit être démarrée pour exécuter la MIDlet
MicroEdition-Profile	Le profile requis par la suite
MicroEdition-Configuration	La configuration requise par la suite

TAB. 1 – Paramètres pour le fichier JAR (manifeste)

Un fichier JAD (Java Application Descriptor) peut également être fourni pour une suite MIDP. Ce fichier comporte des informations complémentaires sur la configuration d'une application. Cette dernière peut y accéder au moment de son exécution à l'aide de la méthode `getAppProperty()` de la classe `MIDlet`. Les fichiers JAD peuvent être considérés comme des fichiers de configuration textuels (à l'instar d'un fichier INI sous Windows). Tout comme les manifestes JAR, ils requièrent eux aussi un ensemble de six attributs, listés dans le tableau 2.

MIDlet-Name	Nom de la suite
MIDlet-Version	Numéro de version de la suite
MIDlet-Vendor	Développeur de la suite
MIDlet-<x>	Une entrée pour chaque application MIDP de la suite, contenant le nom de l'application, une icône optionnelle et le nom de la classe qui doit être démarrée pour exécuter la MIDlet
MIDlet-Jar-URL	URL du fichier JAR associé
MIDlet-Jar-Size	Taille du fichier JAR associé

TAB. 2 – Paramètres pour le fichier JAD

On remarque que certains attributs doivent être présents dans les deux types de fichiers. Les valeurs fournies pour ces attributs communs doivent correspondre exactement les unes aux autres ; sans quoi le gestionnaire d'applications ne chargera pas la suite. Bien que le Wireless Toolkit se charge de ces

tâches à la place de l'utilisateur, il est important de garder ce point à l'esprit dans le cas où il faudrait modifier manuellement un manifeste JAR ou un fichier JAD.

4.4 Création d'une première application MIDP

On va examiner maintenant le processus de création et d'exécution d'une application à l'aide du Wireless Toolkit. Premièrement, il faut créer un projet et lui donner le nom *HelloMIDlet*. Bien évidemment, le nom de la classe MIDlet doit être aussi *HelloMIDlet*. Apparaît ensuite la boîte de dialogue permettant de spécifier les attributs des fichiers JAR et JAD. Les paramètres par défaut illustrés à la figure 3 doivent être corrects et suffisants pour créer une application qui fonctionne.

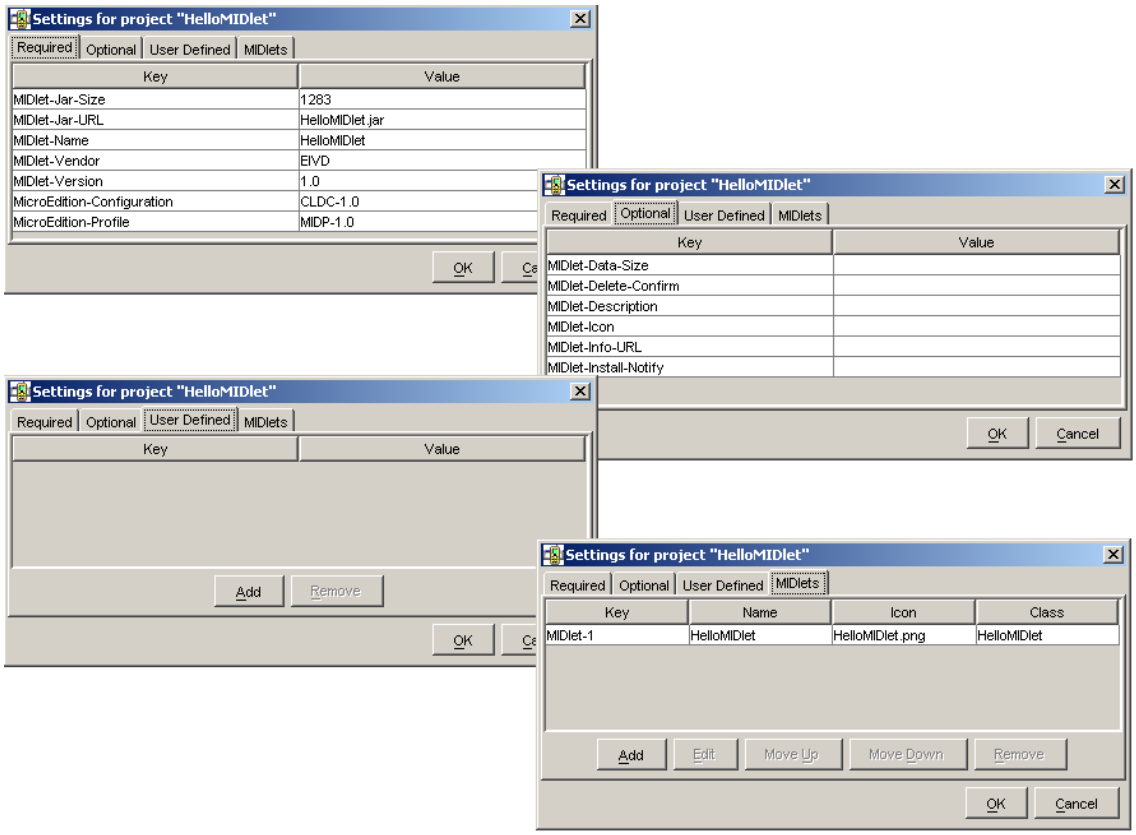

FIG. 3 – Paramètres lors de la création d'un projet

Une fois ceci fait, le toolkit crée un dossier du projet et demande alors de placer le code source (de l'exemple "'fil rouge") dans le sous-répertoire `src`. Cliquer ensuite sur *Build*. La compilation devrait se passer sans problème si le code est correct. Il suffit ensuite de cliquer sur le bouton *Run* et sélectionner *HelloMIDlet* dans le menu d'application qui s'affiche dans l'émulateur. On voit sur la figure 4 le résultat obtenu.

FIG. 4 – Résultat de l'exécution

4.5 Déploiement d'une MIDlet sur un terminal réel

Lorsque la première application est écrite puis simulée grâce à un utilitaire adapté, il est temps de l'exécuter sur un téléphone portable, par exemple. Plusieurs manières de charger le code dans le dispositif sont envisageables. La première qui ne coûte rien est d'utiliser les possibilités de connexion tels que l'infrarouge ou le bluetooth pour autant de disposer d'un utilitaire adéquat. Cette méthode peut être pratique mais elle est très restreinte par le fait qu'elle nécessite la proximité du serveur d'applications.

L'autre manière, qui est la plus utilisée et qui repousse toutes les limites géographiques, est de s'appuyer sur une connexion sans fil du type GPRS. Couplée à un protocole de communication, cette méthode permet de télécharger des applications Java aussi bien à la maison, qu'au travail ou en vacances de l'autre côté du globe. Afin que les fichiers téléchargés puissent être correctement transférés et interprétés par le terminal, il faut faire quelques manipulations du côté serveur. En effet, il faut définir le type MIME (Multipurpose Internet Mail Extensions) pour les fichiers `.jad` et `.jar` sur le répertoire dédié à ces fichiers. A titre de rappel, le type MIME permet au destinataire de savoir de quelle manière interpréter l'information reçue. Par exemple pour un serveur *Apache* il faut ajouter ou modifier le fichier `.htaccess` dans le répertoire où sont les midlets et y ajouter ceci :

```
#Ajout du type de fichier pour pouvoir télécharger des MIDlet
AddType text/vnd.sun.j2me.app-descriptor jad
AddType application/java-archive jar
```

Si on en a l'opportunité, on peut en lieu et place directement modifier le fichier de configuration globale MIME `mime.types` et y ajouter ceci :

```
# This is a comment. I love comments.

# This file controls what Internet media types are sent to the client for
# given file extension(s).  Sending the correct media type to the client
# is important so they know how to handle the content of the file.
# Extra types can either be added here or by using an AddType directive
# in your config files. For more information about Internet media types,
# please read RFC 2045, 2046, 2047, 2048, and 2077.  The Internet media type
# registry is at <ftp://ftp.iana.org/assignments/media-types/>.

# MIME type Extension
application/java-archive jar
text/vnd.sun.j2me.app-descriptor jad
```

Tout est désormais prêt au déploiement de l'application.

5 Conclusion

La mise en place d'une plateforme technique commune à tous les périphériques est une opération très délicate. Chaque appareil possède des spécificités qui impliquent aux applications de s'adapter à différentes caractéristiques d'affichage ou de pointage. Toute la difficulté de concevoir des technologies pour l'embarqué réside dans cette complexité inhérente à la diversité de l'offre.

Java 2 Micro Edition est une architecture technique dont le but est de fournir un socle de développement aux applications embarquées. L'intérêt étant de proposer toute la puissance d'un langage tel que Java associé aux services proposés par une version bridée de J2SE. Ce qu'il faut retenir de l'architecture J2ME c'est qu'elle se compose :

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

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

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