Débuter en J2ME avec le profil MIDP

par Julien DEFAUT

Date de publication : 04/01/2005

Dernière mise à jour : 04/01/2005

Dans ce tutorial, nous allons découvrir le profil J2ME, MIDP, à travers deux exemples, un simple Hello World puis un début de mini-jeu. Cet article n'a donc pas la prétention d'être exhaustif mais juste de proposer une méthode pour débuter. Téléchargez la version pdf.

Introduction

Longtemps les téléphones portables ont été limités à des affichages simplistes, quelques lignes de texte avec un fond gris et 4 couleurs. L'évolution technologique fait qu'aujourd'hui des téléphones plus puissants, avec plus de mémoire et un meilleur affichage, apparaissent pour des coûts qui diminuent.

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

Avec Java, nous pouvons cependant entrevoir une solution. Sun nous propose une version allégée de J2SE, adaptée aux appareils de faible puissance appellée J2ME.

Dans ce tutorial, nous découvrirons le profil J2ME, MIDP, à travers deux exemples, un simple HelloWorld puis un début de mini-jeu. Cet article n'a donc pas la prétention d'être exhaustif mais juste de proposer une méthode pour débuter.

La partie suivante permet de replacer MIDP dans la technologie J2ME dans son ensemble.

1 - Les configurations et les profils

Les appareils mobiles possibles sont de nature trés différentes, J2ME définit alors deux types de spécifications fonctionnant conjointement, les configurations et les profils.

1.1 - Les configurations

Deux **configurations** sont essentiellement utilisées : Connected Device Configuration (CDC) et Connected Limited Device Configuration (CLDC) :

- La **CDC** est plus adaptée aux terminaux relativement puissant comme les PDA. En effet, elle nécessite une machine virtuelle java optimisée appelée CVM qui offre les même fonctionnalités que la JVM classique.

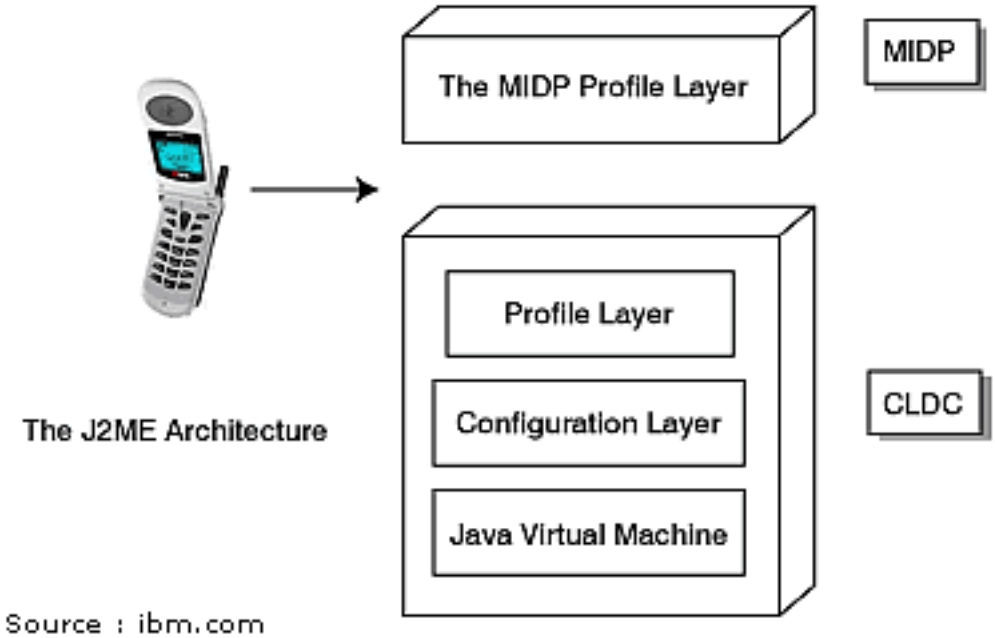
- La **CLDC** est, par contre, dédiée aux appareils avec de faibles capacités comme les téléphones portables. La machine virtuelle allégée correspondante est la KVM et ne possède pas certaines fonctions de la JVM classique.

1.2 - Les profils

Lorsqu'une configuration définit le fondement d'une application, un **profil** en fournit la structure. Les profils définissent l'ensemble des API à utiliser dans une application J2ME et sont conçus spécialement pour chaque configuration.

Sun propose deux **profils** de référence J2ME : le profil Foundation et le profil Mobile Information Device Profile (MIDP).

- Le profil **Foundation** est destiné à la configuration CDC. Les développeurs qui utilisent ce profil ont accés à une implémentation complète des fonctionnalités de J2SE.
- Le profil **MIDP** est destiné à la configuration CLDC. Il prend en charge un nombre limité des classes de J2SE et définit des classes d'entrée/sortie et d'interface spécialisées pour une configuration CLDC.



The J2ME Architecture

Source : ibm.com

Pour le portage de J2SE vers J2ME, les modifications seront sûrement minimimes si la configuration de l'appareil est la CDC et le profil implémenté, le Foundation. Peut-être quelques optimisations du code dans la gestion des ressources pourraient s'imposer. Cependant vers MIDP et CLDC, le portage risque d'être plus fastidieux et l'adaptation du code va s'avérer inévitable. Ensuite, tout dépend de l'appareil cible et de la complexité de l'application à porter. L'interface graphique devra, le plus souvent, être complètement repensée.

2 - MIDP par l'exemple : un Hello World

2.1 - Ecrire, compiler, exécuter

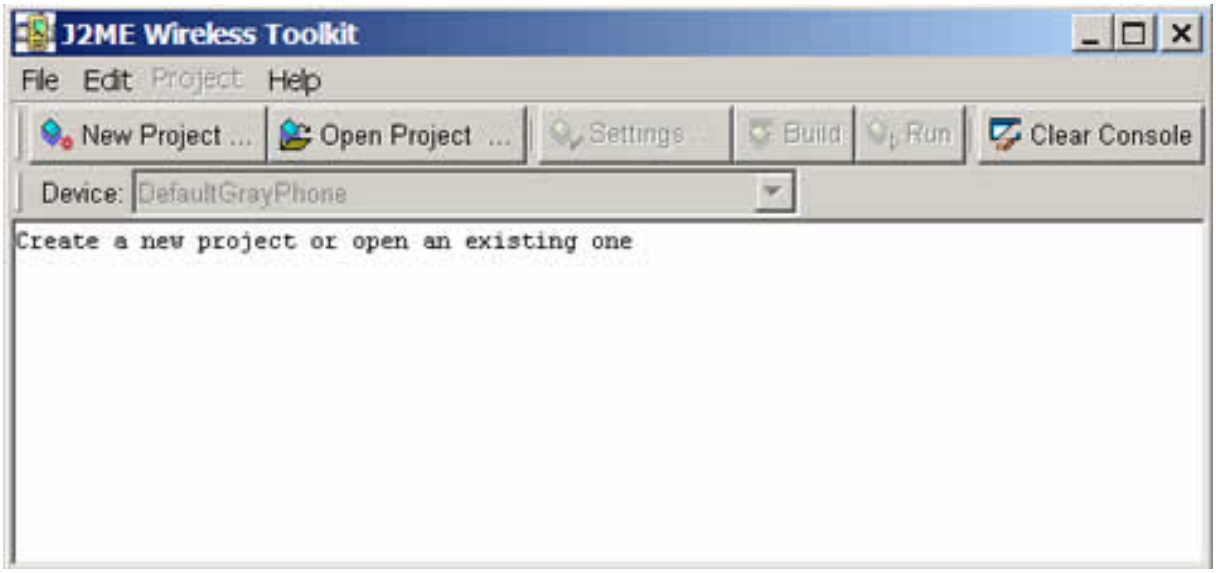
Un bon outil pour débuter est le J2ME Wireless Toolkit de Sun disponible gratuitement.

Ce kit offre une interface conviviale pour les outils de ligne de commande dont vous aurez besoin, ainsi que quelques outils rudimentaires de configuration et de gestion du code. Il comprend aussi des émulateurs de terminaux mobiles.

Téléchargez le et installez le. Veillez cependant à bien avoir installé le JDK standard de Sun avant.

Sachez qu'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, il doit être prévérifié à l'aide de l'outil Preverify.exe. Le Wireless Toolkit, est capable d'effectuer ces actions supplémentaires, si bien que vous n'avez pas à vous en soucier.

Lancez le programme KToolbar, la fenêtre ci-dessous apparaît.



Cliquez sur "New Project", et inscrivez dans les deux champs proposés "HelloWorld" qui sera le nom de notre projet.

A la fenêtre suivante, "Settings for project HelloWorld" appuyez sur "OK". Un nouveau répertoire a été créé et la console en indique le chemin.

Rendez-vous dans le répertoire du projet, il contient plusieurs dossiers dont un dossier "src", créez dans ce dossier "src" un fichier appelé HelloWorld.java, éditez le avec un éditeur de texte comme le Bloc-Notes ou WordPad et placez-y le code suivant :

HelloWorld.java

```
// contient les éléments de base
import javax.microedition.midlet.*;
// contient les éléments permettant de gérer l'interface
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener
{
  private Display _display;
  private TextField _textField1;
  private Command _commandExit;
  private Form _form1;

  public HelloWorld()
  {
    // fait un lien avec l'affichage
    _display = Display.getDisplay(this);

    // creation d'un objet formulaire sur lequel on peut placer des composants
    _form1 = new Form("Test de HelloWorld");

    // creation d'un bouton pour sortir du programme
    _commandExit = new Command("Exit", Command.SCREEN,1);

    // creation d'un champ de texte contenant notre Hello World
    _textField1 = new TextField("","Hello World !",15,TextField.ANY);

    // ajout des composants au formulaire
    _form1.addCommand(_commandExit);
    _form1.append(_textField1);
    _form1.setCommandListener(this);
  }

  // évènement exécuté au démarrage de l'application
  public void startApp()
  {
    // affichage du formulaire
    _display.setCurrent(_form1);
  }

  // évènement exécuté lors de la mise en pause de l'application
  public void pauseApp()
  {
  }

  // évènement exécuté lorsque l'application se termine
  public void destroyApp(boolean unconditional)
  {
  }

  public void commandAction(Command c, Displayable s)
  {
    // lors du clic sur le bouton Exit
```

HelloWorld.java

```
        if (c == _commandExit)
        {
          // appel manuel à la fonction de fermeture
          destroyApp(false);
          // on demande au manager de fermer l'application
          notifyDestroyed();
        }
      }
    }
```

Revenez dans la KToolbar et cliquez Project>Package>Create Package.

Deux fichiers importants (HelloWorld.jar et HelloWorld.jad) sont générés dans le répertoire "bin".

Cliquez sur Run pour tester. Un émulateur apparaît et aprés quelques clics, vous pouvez voir apparaître notre "Hello World !".

Le .jar et le .jad générés peuvent ainsi être transmis sur votre téléphone mobile, je l'ai fais par infrarouge avec mon Nokia 7210 et Nokia PC Suite 5 et tout fonctionne parfaitement.

2.2 - Le fonctionnement

Le code du HelloWorld semble assez clair et commenté pour en saisir le principe, cependant certains éléments utilisés sont importants à connaître.

Tout programme doit hériter de la classe MIDlet. En effet, cette classe impose l'implémentation de trois méthodes abstraites qui sont startApp(), pauseApp() et destroyApp(). Ces méthodes sont appelés par le gestionnaire d'application de l'appareil lorsque c'est nécessaire. La communication avec celui-ci est bidirectionnelle, la classe MIDlet fournit donc quelques méthodes :

NotifyPaused() : demande au gestionnaire de mettre en pause l'application.

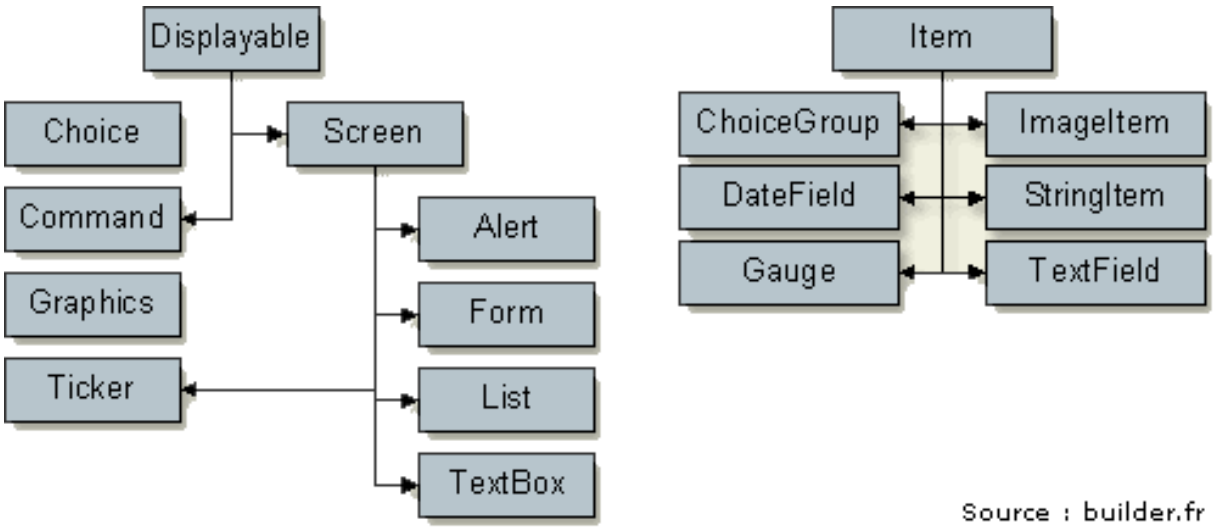
ResumeRequest() : demande au gestionnaire de relancer une application qui a été mise en pause.

NotifyDestroyed() : demande au gestionnaire de fermer l'application, l'appel à destroyApp() juste avant est important puisque le gestionnaire ne le fera pas automatiquement si NotifyDestroyed() est appelée.

GetAppProperty() : demande de retourner des informations concernant l'application.

Le package javax.microedition.lcdui contient les éléments de l'interface utilisateur. Si vous avez déjà réalisé une application Swing avec J2SE, vous observerez des similitudes dans ce fonctionnement à la manière d'un arbre où chaque objet est encapsulé dans d'autres (en bref, le principe de la programmation objet).

Voici un schéma donnant un aperçu de la hiérarchie des composants :



Vous trouverez, dans le répertoire "apps" du Wireless Toolkit, un projet nommé "UIDemo" qui illustre, avec des exemples, l'utilité de chacun de ces composants.

Téléchargez les sources et packages du projet HelloWorld.

3 - MIDP par l'exemple : un petit jeu

Les jeux pour la téléphonie mobile se créent aujourd'hui un nouveau marché, les écrans gagnent en résolution, en nombre de couleurs, les mémoires des téléphones augmentent ainsi que la puissance des processeurs. De plus en plus, nos appareils se transforment en une petite console de jeux portable. Les constructeurs fournissent des outils permettant de transférer des applications simplement et rapidement à partir de son ordinateur, les coûts des jeux sont pour le moment dérisoires ou gratuit ... et il est simple, avec quelques connaissances en développement, d'entrer dans ce marché grandissant.

Pour illustrer ce propos, nous allons créer ici le début d'un jeu de type pong/arkanoïd : avec deux touches le joueur pourra déplacer une barre horizontalement pour tenter de faire éviter à une balle qui rebondit de tomber en bas de l'écran. On pourra ainsi appréhender quelques élements nécessaire à la réalisation d'un jeu : dessiner à l'écran, implémenter un timer et utiliser des commandes utilisateur.

Créez un nouveau projet avec le programme KToolBar du Wireless Toolkit (cf chapitre précédent), appelé le "Bing", placez dans le répertoire "src" un fichier "Bing.java". A l'intérieur, mettez le code ci-dessous. Compilez et éxécutez (cf chapitre précédent), le jeu devrait fonctionner avec les touches 4 et 6 de l'émulateur.

Bing.java

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


// classe principale
public class Bing extends MIDlet
{
  private Display _display; // Display est indispensable à l'affichage
  private Moteur _moteur; // objet qui va gérer le déroulement du jeu
  private TimerBalle _timerBalle; // objet qui exécute une méthode à intervalle régulier
  private Timer _timer; // gestion du temps

  // constructeur, appelé au démarrage
  public Bing()
  {
    // initialisations
    _moteur = new Moteur();
    _timerBalle = new TimerBalle(_moteur);
    _timer = new Timer();
    _display = Display.getDisplay(this);
  }

  // appelé à la fermeture de l'application
  protected void destroyApp(boolean unconditional)
  {
    _timer.cancel(); // arret du timer
  }

  // appelé au démarrage de l'application
  protected void startApp()
  {
    _display.setCurrent(_moteur); // _moteur hérite de Canvas
    _timer.schedule(_timerBalle, 1, 20); // toutes les 20 ms, _timerBalle est appelé
  }

  // appelé lors de la mise en pause de l'application
  protected void pauseApp(){}
}

// classe exécutant _moteur.TestAndMove() à intervalle régulier
class TimerBalle extends TimerTask
{
  private Moteur _moteur;

  TimerBalle(Moteur moteur)
  {
    _moteur = moteur;
  }

  public void run()
  {
    _moteur.TestAndMove();
  }
}

// classe gérant le déroulement du jeu
```

Bing.java

```
class Moteur extends Canvas
{
  private int _height, _recH, _x, _speedX, _zoneH;
  private int _width, _recW, _y, _speedY, _zoneW;
  private int _barrePosX, _barrePosY, _barreH, _barreW;
  private String _dirBarre = ""; // direction LEFT ou RIGHT de la barre
  private boolean _painting = false;
  private boolean _perdu = false; // indique si la balle est sortie ou pas

  public Moteur()
  {
    _height = getHeight(); // hauteur totale de l'écran
    _width = getWidth(); // largeur totale de l'écran
    _recW = 2; // taille de la boule : largeur
    _recH = 2; // taille de la boule : hauteur
    _speedX = -1; // vitesse de la boule selon X
    _speedY = -1; // vitesse de la boule selon Y
    _x = 50; // position X initiale de la boule
    _y = 60; // position Y initiale de la boule
    _zoneW = _width; // largeur de la zone de jeu
    _zoneH = _height - 11; // hauteur de la zone de jeu (vide de 11 pixel en bas)
    _barrePosX = 60; // position X initiale de la barre
    _barrePosY = _zoneH - 4; // position Y initiale de la barre
    _barreH = 3; // hauteur de la barre
    _barreW = 10; // largeur de la barre
  }

  // appelé par le timer toutes les 20 ms
  public void TestAndMove()
  {
    // les tests ne se font que lorsque l'application ne rafaichit pas l'ecran
    if (_painting) return;

    // détection des collisions contre les murs et modification du sens du mouvement
    if (_x >= _zoneW - 1 || _x <= 1) _speedX = -_speedX;
    if (_y <= 1) _speedY = -_speedY;

    // si la boule touche le bas de la zone de jeu, c'est perdu
    if (_y >= _zoneH - 1) _perdu = true;

    // détection des collisions contre la barre et modification du sens du mouvement
    if (_y >= _barrePosY && _x >= _barrePosX && _x <= _barrePosX + _barreW)
      _speedY = -_speedY;

    // affectation des modification de sens sur la position de la boule
    _x += _speedX;
    _y += _speedY;

    // appel à la methode paint() pour afficher les changements
    repaint();
  }

  // en fonction de la dernière touche appuyée _dirBarre change de valeur
  // et sa position est modifiée
  private void scrollBarre()
  {
    if (_dirBarre == "LEFT") _barrePosX -= 2;
    else if (_dirBarre == "RIGHT") _barrePosX += 2;
  }

  protected void paint(Graphics g)
  {
    _painting = true;
```

Bing.java

```
      // on regarde l'état des touches et on fait bouger la barre
      scrollBarre();

      // dessin du fond
      g.setColor(255, 255, 255);
      g.fillRect(0,0,_width,_height);

      // dessin d'un cadre noir pour délimiter la zone de jeu
      g.setColor(0,0,0);
      g.drawRect(0,0,_zoneW-1,_zoneH-1);

      if (!_perdu) // si le jeu continue
      {
        // dessin de la balle
        g.fillRect(_x, _y, _recW, _recH);

        // dessin de la barre
        g.fillRect(_barrePosX, _barrePosY, _barreW, _barreH);
      }
      else // si on a perdu, un message apparaît
      {
        g.drawString("Perdu ! Gros Nul !", 7, 35, 0);
      }

      _painting = false;
    }

    // lorsque une touche est appuyée _dirBarre prend une valeur LEFT ou RIGHT
    protected void keyPressed(int keyCode)
    {
      if (keyCode==Canvas.KEY_NUM4) // gauche
      {
        _dirBarre="LEFT";
      }
      if (keyCode==Canvas.KEY_NUM6) // droite
      {
        _dirBarre="RIGHT";
      }
    }

    // si une touche est relachée _dirBarre indique
    // qu'aucun mouvement de barre ne doit être fait
    protected void keyReleased(int keyCode)
    {
      if (keyCode==Canvas.KEY_NUM4 || keyCode==Canvas.KEY_NUM6)
      {
        _dirBarre = "";
      }
    }
  }
```

Cet exemple permet de comprendre certains mécanismes. Les commentaires assez fournis vous permettent sûrement déjà d'en saisir des éléments.

Le principe est le suivant :

- La classe principale Bing est exécutée au démarrage du programme et hérite de la classe MIDlet indispensable au fonctionnement d'un programme MIDP. Ses fonctions utiles sont d'initialiser l'affichage, d'instancier la classe Moteur qui sera le cerveau du jeu et d'initialiser le timer qui va réguler la vitesse du jeu (sans lui, la vitesse dépendrait de la puissance du processeur).

- Une petite classe TimerBalle hérite de la classe TimerTask qui impose l'implémentation d'une méthode run(). Cette méthode sera appelée par le timer à intervalle régulier. run() va intervenir sur le mouvement de la balle en appellant une autre méthode de Moteur.

- La classe moteur, à sa construction, va initialiser l'ensemble des variables nécessaires au déroulement du jeu : les vitesses, les tailles d'écran et de zone de jeu, les tailles des sprites, ... Une méthode essentielle est TestAndMove(). C'est elle qui est appelée par le timer : elle vérifie, à intervalle régulier, la position de la balle et la fait rebondir si besoin.

Le mouvement de la barre va dépendre des touches pressées (4 et 6). On ne pouvait pas intervenir sur sa position dés que l'appui sur 4 ou 6 était détecté, en effet la méthode keyPressed() ne réagit que à l'instant ou la touche est appuyée et n'est pas appelé tant que la touche n'est pas relachée. Donc, on crée une "mémoire de touche", la variable _dirBarre retient la dernière touche appuyée et dés que le 4 ou le 6 est relaché, _dirBarre sera vidée pour indiquer qu'aucune touche n'est restée appuyée. Ceci évite au joueur de devoir "tappoter" sur le clavier pour réussir à faire avancer la barre.

Nous avons évoqué TestAndMove(), en plus de modifier les positions de la boule, elle va rendre effectives toutes les modifications visuelles en appelant la méthode paint(). Cette méthode propre à la classe Canvas dont hérite Moteur permet de dessiner sur l'écran. On peut ainsi afficher du texte, mettre des couleurs, des cercles, des carrés, des images, ... Ici nous allons à chaque fois (à chaque appel du timer) tout effacer et tout redessiner, c'est ce qu'on appelle aussi le rafraichissement. C'est assez gourmand en ressources mais pour la simplicité de notre jeu, ça ne pose pas de problème.

Bien que le choix d'un jeu ait été fait, il est bien sûr possible de créer avec le profil MIDP des applications plus conventionnelles. D'ailleurs, le Wireless Toolkit de Sun renferme des exemples trés intéressant sur la plupart des applications possibles de MIDP, essayez les ...

Téléchargez les sources et packages du projet Bing.

Conclusion

La fusion PDA, téléphone, console de jeux, lecteur mp3 commence à se faire. Bientôt les débits de communications augmenteront et les applications logicielles deviendront vraiment communiquantes (avouons qu'aujourd'hui c'est assez laborieux) et les accés aux bases de données, à divers serveurs ouvriront encore plus les possibilités des développeurs. Avec MIDP, aujourd'hui dans sa version 2, les développeurs peuvent déjà toucher un bon nombre d'usagers. Au fur et à mesure, les avancées technologiques importantes toucheront le grand public et on y a déjà remarqué l'importance de la téléphonie.

En plus des SDK propriétaires à chaque constructeur, d'autres technologies essayent de s'insérer dans ce marché. On trouve, par exemple, un framework .NET dépouillé même si peu des téléphones actuels l'utilisent (plutôt reservé aux PDA de type PocketPC) ... Macromedia propose dans son nouveau Flash MX 2004 de créer des applications pour les appareils mobiles, un FlashPlayer allégé existe aussi. Si Macromedia réussit à bien adapter sa technologie dans la mobilité, cette société a des chances de succés. En effet, Flash, contrairement aux idées reçues, devient un environnement de plus en plus puissant pour sa cible : les applications légères pour un temps de développement relativement court.

Attendons de voir l'avenir. Regardons comment l'industrie et le public vont réagir ...

Téléchargements

Téléchargez la version pdf de cet article.

Téléchargez les sources et packages du projet HelloWorld.

Téléchargez les sources et packages du projet Bing.