Application Java pour terminal mobile utilisant le protocole Bluetooth

Travail de diplôme réalisé en vue de l'obtention du diplôme HES

par :

David Cautillo

Conseiller au travail de diplôme :

M. Peter Daehne, Professeur HES

Carouge, 22 août 2008

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Déclaration

Ce travail de diplôme est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre de bachelor en informatique de gestion. L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de diplôme, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de diplôme, du juré et de la HEG.

« J'atteste avoir réalisé seul(e) le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Carouge (GE), le 22 août 2008

David Cautillo

Remerciements

Mes remerciements vont tout d'abord à M. Peter Daehne qui m'a suivi durant la réalisation du présent mémoire. Sa disponibilité et ses précieux conseils m'ont permis de travailler dans les meilleures conditions.

Je remercie ensuite l'entreprise Motorola®, qui, par l'intermédiaire de son site de support aux développeurs, m'a donné accès à des rubriques réservées pour la rédaction de la partie Java.

Enfin, je remercie mes parents, Francis et Anita, ainsi que mon amie Aurélie, pour leur présence et leur soutien tout au long de ce travail.

Avant-propos

Depuis le lancement d'études pour la création d'un système de télécommunication universel en 1982, et la commercialisation des premiers téléphones portables utilisant la norme GSM® en 1991, ces concentrés de technologies n'ont cessé d'évoluer.

Actuellement, chaque appareil est capable de faire fonctionner des applications dans un environnement d'exécution Java intégré. Fort de ce constat, ce travail de diplôme a pour objectif de présenter différents points de la réalisation d'une application pour terminal mobile.

Le choix du protocole de communication Bluetooth™ pour cette étude, concerne la mise en relation du téléphone avec un module de localisation GPS portable.

Pour mener à bien la rédaction du présent travail, il a fallu effectuer une recherche préalable d'informations sur Internet auprès des principaux constructeurs de téléphones mobiles comme Motorola®, Ericsson® (devenu Sony Ericsson®) ou Nokia®, mais également auprès du fournisseur de module GPS RoyalTek Company®. De plus, nous nous sommes référés à différents sites web tel que celui de Sun Microsystems® pour la partie Java.

Après ces recherches, une première observation que nous avons effectuée est que le développement Java pour une plateforme mobile n'est pas aussi aisé qu'il pourrait l'être sur un ordinateur personnel. Plusieurs spécificités propres aux plateformes mobiles doivent être prises en compte et l'emploi d'un environnement de développement orienté mobilité est à préférer.

Table des matières

Déclaration i
Remerciements ii
Avant-propos iii
Table des matières iv
Liste des Tableaux vi
Liste des Figures vi
Introduction 1
1. Technologie Java et Java sur les terminaux mobiles 2
1.1 Principes généraux 2
1.1.1 Execution Engine ou moteur d'exécution 3
1.1.2 Class Loader ou chargeur de fichier class 3
1.1.3 Class file Verifier ou vérificateur de fichier class 4
1.1.4 Le système d'exceptions 4
1.2 Java 2 Micro Edition 6
1.2.1 KVM 6
1.2.2 Configuration CLDC 8
1.2.3 Profil MIDP et Midlet 10
1.2.4 Midlet : Exemple de création d'un menu 11
2. Technologie Bluetooth® 15
2.1 Principes d'utilisation et de fonctionnement 15
2.2 Les modes de communication 18
2.2.1 Liaisons synchrones SCO 19
2.2.2 Liaisons asynchrones ACL 19
2.3 Bluetooth et Java 20
2.3.1 Les threads 20
2.3.2 Exemple de méthodes de recherche Bluetooth 21
3. Le GPS 25
3.1 Principes d'utilisation et de fonctionnement 25
3.1.1 Trames NMEA 27
3.1.1.1 Exemple de parser de trames NMEA de type GLL 28
3.2 Critères de choix et récepteurs GPS 34
4. L'environnement de développement MotoDev Studio 37
4.1 Utilisation de MotoDev 37
4.1.1 Le Workspace 37
4.1.2 La perspective 38
4.1.3 Création d'un projet 39
4.1.4 Installation d'un plug-in externe 43
5. Java, Bluetooth, GPS et mobilité 45

5.1 Applications existantes 45
5.1.1 Beacon 45
5.1.2 Nav4All 46
5.1.3 Amaze 46
5.1.4 MGMaps 47
5.2 Ébauche de projet 47
5.2.1 Présentation 47
5.2.2 Mise en place 48
Conclusion 50
Glossaire 51
Bibliographie 55

Liste des Tableaux

[T_BLTH01] Tableau comparatif des technologies sans fil 16

[T_GPS01] Tableau comparatif de trois récepteurs GPS 36

Liste des Figures

[I_CLDC01] Configuration CLDC 7

[I_CDC01] Configuration CDC ……. 8

[I_PRCO01] Profil & Configuration 8

[I_MENU01] Aperçu du menu sur émulateur V3i 11

[I_BLTH01] Logo Bluetooth Special Interest Group 15

[I_BLTH02] Types de réseaux Bluetooth 18

[I_GPS01] Schéma d'exemple de triangulation 26

[I_GLL01] Console parser de trames GLL 34

[I_MOTO01] Fenêtre de sélection du WorkSpace 37

[I_MOTO02] Fenêtre principale de MotoDev Studio 38

[I_MOTO03] Fenêtre de création d'un nouveau projet 39

[I_MOTO04] Fenêtre de configuration du type de téléphone 40

[I_MOTO05] Fenêtre d'arborescence du projet 41

[I_MOTO06] Fenêtre de création d'une nouvelle Classe 41

[I_MOTO07] Nouveau fichier Classe et fenêtre de code 42

[I_MOTO08] Auto complétion et vérificateur de code 42

[I_MOTO09] Lancement du projet dans l'émulateur 43

[I_MOTO10] Menu d'installation de plug-in et mises à jour 43

[I_MOTO11] Champ de saisie d'un site de plug-in distant 44

[I_MOTO12] Détail du plug-in Subclipse 44

[I_MOTO13] Téléchargement du plug-in Subclipse avant installation 44

Introduction

Depuis les années 50 durant lesquelles le concept de téléphone mobile fit son apparition, celui-ci n'a eu de cesse de se moderniser passant ainsi d'une utilisation fastidieuse et coûteuse à une utilisation aisée et accessible à tous. Véritable compagnon des temps modernes, le mobile proprement dit vit le jour en 1985 grâce à Motorola® et son DynaTac™ [DYN01]. Cet appareil pesant près de 800 grammes, on ne peut pas dire que le mot « portabilité » avait le sens qu'on lui connaît aujourd'hui.

A l'heure actuelle, cette machine qui révolutionna la communication n'est plus un simple téléphone, mais embarque entre autres un système d'exécution Java qui permet le lancement d'applications diverses et variées, et ce au grand bonheur d'utilisateurs devenus « friants » de nouveautés. Ne se confinant plus à une utilisation sans autre interaction qu'avec le réseau GSM® lui-même, elle inclut aussi le support d'un protocole sans fil appelé Bluetooth™, lancé par le constructeur Suédois Ericsson® en 2001 avec le T39m® pour le transfert de données entre deux terminaux.

Ce travail de diplôme va donc exposer les différentes étapes pouvant entrer dans la réalisation d'applications Java pour mobile utilisant le protocole Bluetooth®.

Il va s'articuler en cinq parties principales traitant des points suivants :

- ce qu'est Java sur les terminaux mobiles
- le protocole Bluetooth®
- le système GPS® pour illustrer la communication via Bluetooth®
- les bibliothèques Java nécessaires
- l'environnement de développement MotoDev Studio

Enfin, nous présenterons l'existant et les futures applications possibles mettant en œuvre conjointement Java – Bluetooth – GPS dans le domaine de la mobilité des personnes.

Les différentes parties seront illustrées par un exemple de code Java présentant les principes abordés.

Dans le texte qui suit, les termes postfixés par un astérisque (*) sont définis dans le glossaire.

1. Technologie Java et Java sur les terminaux mobiles

Depuis ces dernières années, le marché juteux des télécommunications a connu une croissance exponentielle. Pour preuve, l'Institut Gartner affirme qu'il s'est vendu rien qu'en 2007 près de 1,15 milliard de téléphones cellulaires [GAR07], faisant ainsi la part belle aux trois principaux constructeurs mondiaux que sont Nokia, Motorola et Sony Ericsson.

Les utilisateurs de ces petits bijoux de technologie sont de plus en plus avides d'informations et de services. C'est pourquoi en juin 1999 Sun Microsystems - suite à l'impulsion de plusieurs fabricants dont Nokia et Motorola de voir un standard de développement pour téléphones mobiles - annonce la sortie de trois plateformes d'édition Java 2 : la SE (Standard Edition destinée aux postes de travail), la EE (Entreprise Edition, destinée au professionnels pour la mise en place d'applications client - serveur) et la ME pour Micro Edition, celle qui va être exposée plus loin dans cette partie.

1.1 Principes généraux

Avant de se concentrer sur Java Micro Edition il faut présenter le langage Java lui-même.

Baptisé Oak par ses créateurs travaillant pour Sun Microsystems en 1991, ce langage de programmation était au départ conçu pour contrôler plusieurs appareils électroniques en même temps. Le nom Java, quant à lui, apparut en 1995 juste avant le salon SunWorld qui eut lieu la même année.

Il faut savoir que Java applique le principe : « Write once run everywhere », et que, pour arriver à faire fonctionner des applications au sein d'environnements hétérogènes, le compilateur Java produit un code intermédiaire appelé byte-code. Le byte-code est interprété par une machine virtuelle dont il existe des implantations spécifiques à chaque système sur lequel elle est installée. C'est donc cette JVM - pour Java Virtual Machine - qui va assurer la portabilité du code Java. Cette portabilité est fondamentale, surtout dans un domaine aussi vaste que celui des téléphones portables.

La JVM est une couche logicielle spécifique à chaque appareil incluant un environnement d'exécution Java et dont la fonction est d'interpréter le byte-code. Cette interprétation se déroule en deux temps : l'analyse puis la traduction.

La JVM standard interprète donc ce byte-code instruction par instruction pour ensuite l'exécuter via un Execution Engine ou moteur d'exécution. Cependant, la JVM n'est pas uniquement constituée d'un moteur d'exécution, elle comprend aussi deux autres parties qui sont le Class Loader ou chargeur de fichier class et le Class Verifier ou vérificateur de fichier class [CLAS01]. Ces trois parties s'organisent et collaborent pour former le « runtime Java* ».

1.1.1 Execution Engine ou moteur d'exécution

Au centre de la machine virtuelle, le moteur d'exécution est le processeur virtuel Java, dont le rôle est de convertir les bytes-codes en instructions exécutables par le processeur de la plateforme sur laquelle la JVM est installée.

En effet, il fait le lien entre l'environnement de runtime Java et le système hôte. A l'intérieur du runtime Java, l'Execution Engine est en relation avec d'autres entités qui sont :

- le gestionnaire d'exceptions, de threads et de sécurité ;
- le gestionnaire de mémoire pour les zones de classes et méthodes, qu'elles soient natives ou non, ainsi que pour les zones réallouables.

1.1.2 Class Loader ou chargeur de fichier class

Comme son nom l'indique, le chargeur de fichier class a pour fonction de charger en mémoire les classes nécessaires à l'exécution d'un programme, et ce au moment de leur instanciation*.

Il a la capacité de charger un fichier class qui est fourni par l'environnement Java lui-même, mais également d'aller en charger d'autres qui peuvent se trouver ailleurs sur un lecteur, qu'il soit local ou mappé sur un réseau.

Afin de charger convenablement ses classes, un processus de vérification s'opère pour contrôler qu'aucune partie du code ne met en péril aussi bien la JVM que le système sur lequel elle s'exécute.

1.1.3 Class file Verifier ou vérificateur de fichier class

Chaque JVM possède un vérificateur de fichier class qui va s'assurer que le fichier de classe (ou class file) possède une structure interne correcte. Si ce vérificateur détecte une erreur dans un fichier, il va faire remonter l'erreur à travers le système d'exceptions. Ce système est abordé plus loin dans ce document.

Du fait qu'un fichier de classe (ou class file) n'est autre qu'une séquence de données binaires, la JVM n'est pas en mesure de déterminer si un class file particulier a été généré par un compilateur Java « correct » ou par un compilateur modifié, ce qui dans ce cas pourrait avoir pour conséquence de compromettre l'intégrité de la machine virtuelle.

Un des enjeux du Class Verifier est de concourir à la robustesse des programmes écrits en Java. Par exemple, dans le cas d'un fichier class compilé à l'aide d'un traducteur modifié (dans le mauvais sens du terme), celui-ci pourrait contenir une instruction byte-code dont le but serait de « jumper* » via un « goto » après la fin d'une méthode. L'exécution de cette instruction engendrerait le plantage de la JVM.

On distingue deux phases dans le fonctionnement du vérificateur de class :

- la première intervient juste après que le fichier class soit chargé en mémoire. Une vérification de la structure interne du fichier ainsi que du byte-code est effectuée ;
- la deuxième, qui a lieu pendant l'exécution du byte-code, où l'existence des classes, champs et méthodes référencées symboliquement est vérifiée.

1.1.4 Le système d'exceptions

En relation avec le moteur d'exécution, le système d'exceptions est géré par l'Exception Manager ou gestionnaire d'exceptions qui est une partie intégrante du runtime Java. Lorsqu'une exception survient - par exception on parle d'un événement

qui se produit lors de l'exécution d'un programme et qui perturbe son fonctionnement normal – la méthode qui en est à l'origine crée un objet d'exception. Cet objet va être transmis au runtime. Il contient bien évidemment des informations sur l'erreur elle-même, mais également sur son type et sur l'état du programme au moment où l'exception est survenue.

Le chemin que va emprunter l'exception depuis le point où elle est levée jusqu'à son traitement par le runtime peut se comparer à une pile. En effet, le runtime va chercher dans cette pile une méthode qui contient un bloc de code capable de traiter l'exception. Ce bloc s'appelle un Exception Handler.

La pile exposée plus haut se parcourt comme suit :

- recherche d'un Exception Handler dans la méthode qui a généré l'exception. Puis, si aucun n'est trouvé :
- recherche dans la méthode de plus haut niveau; à savoir celle qui a appelé la méthode qui provoque une exception ;
- ainsi de suite si aucun bloc n'est trouvé

Le parcours s'effectue donc dans l'ordre inverse de l'appel des méthodes lors de l'exécution du programme.

Pour rappel, une exception se traite la plupart du temps à l'aide des entêtes de bloc ci-dessus :

```
class Une_classe{

        public static void main(String[] args) {
        // début de la méthode main() ;
        try {
        // Traitement pouvant provoquer une erreur ;
        }
        catch (TypeException e) {
        // Traitement de l'erreur ;
        }
        // Suite de la méthode main si aucune erreur ;
        }

}
```

Un autre entête de bloc, le « `finally` » peut également être ajouté. Les instructions contenues dans ce bloc seront exécutées directement après le « `catch` ».

1.2 Java 2 Micro Edition

Une fois les principes généraux de Java exposés, nous pouvons nous intéresser à l'édition ME dédiée aux périphériques mobiles. Il faut savoir que J2ME ne définit pas une nouvelle version de Java, mais adapte Java pour les produits de type périphériques portables. C'est pourquoi une application écrite pour J2ME est compatible en exécution avec les autres éditions telles que SE ou EE, pour autant que les API* qu'elle utilise soient disponibles dans ces versions. Certes, il existe certaines contraintes, mais l'architecture de Java proprement dite ne change pas dans ME.

1.2.1 KVM

La KVM – KiloByte Virtual Machine – de J2ME est une nouvelle implantation de la machine virtuelle Java optimisée pour les périphériques portables. Elle accepte cependant le même byte-code ainsi que le même format de fichiers class que la JVM.

Comme vous pouvez l'imaginer, les principales contraintes à prendre en compte pour permettre à Java de fonctionner de manière fluide sur une plateforme mobile sont la quantité de mémoire disponible ainsi que les capacités du processeur. Cette mémoire est limitée et ne peut pas être étendue à volonté. La solution trouvée a été de réduire la taille de l'environnement runtime en supprimant les classes non indispensables pour former un nouveau groupe de classes appelé « core ». Ce cœur est en fait un sous-ensemble de l'environnement runtime de la version SE et permet à l'environnement Java mobile de s'exécuter plus rapidement et d'employer moins de ressources mémoire.

Dans J2ME, n'importe quelle classe fournie (en tant qu'élément de l'environnement d'exécution) peut être stockée en employant le format interne de la machine virtuelle. C'est-à-dire dans un format autre que celui du fichier class normal mais en comprenant toujours la possibilité de lire les classes définies par l'utilisateur dans ce format.

Malheureusement, un temps d'exécution réduit n'est pas particulièrement intéressant si en contrepartie il n'y a pas la possibilité d'interagir avec l'utilisateur ou avec d'autres périphériques externes. C'est pourquoi J2ME complète l'environnement d'exécution en définissant de nouvelles classes dédiées aux périphériques portables. Certaines

d'entre elles remplacent celles fournies dans l'édition standard et d'autres ajoutent de nouvelles fonctionnalités qui n'y sont pas implantées.

L'édition J2ME est, comme les autres éditions de Java, basée sur une architecture modulaire. Dans le cadre du développement d'applications pour terminaux mobiles, deux entités principales sont à retenir pour personnaliser l'environnement runtime :

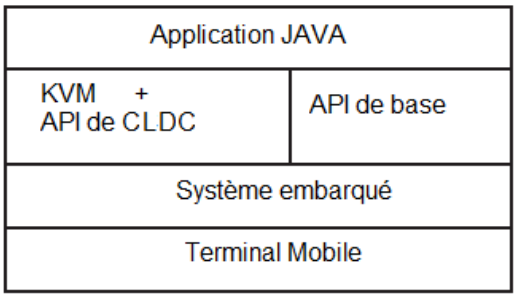
- les configurations
- les profils

Une configuration définit l'environnement d'exécution de base de J2ME comme une machine virtuelle avec un ensemble de classes qui peuvent fonctionner sur une famille de dispositifs portables ayant les mêmes possibilités; dans le cas présent, les terminaux mobiles. Deux configurations sont actuellement définies :

- CLDC pour Connected Limited Device Configuration
- CDC pour Connected Device Configuration

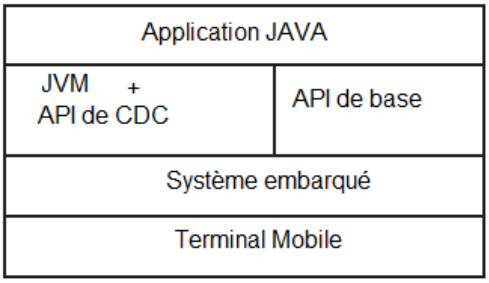
Chaque configuration met à disposition un nombre minimal de fonctions que tous les périphériques, dont la configuration matérielle le permet, peuvent supporter. La configuration CDC utilise la machine virtuelle Java classique tandis que CLDC utilise la KVM (voir l'illustration des couches ci-dessus). Elle est donc plus adaptée à des appareils de faible puissance comme les téléphones.

Pour CLDC :



[I_CLDC01]

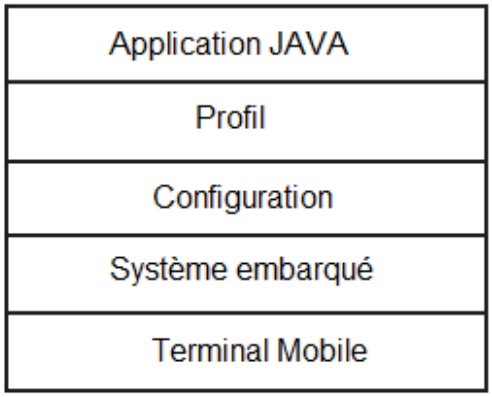
Pour CDC :



[I_CDC01]

Un profil, quant à lui, ajoute des classes dites « spécifiques au domaine » à une configuration particulière de J2ME. À la différence des configurations qui sont orientées matériel, un profil est orienté application. En effet, il fournit des classes qui sont adaptées à des utilisations spécifiques dans une application comme les interfaces utilisateur. Cet aspect sera développé plus loin lors de la présentation d'un exemple de création de menu.

La figure ci-dessous montre comment profils et configurations cohabitent pour former un environnement d'exécution Java.



[I_PRCO01]

Voyons maintenant la configuration CLDC et les profils plus en détail.

1.2.2 Configuration CLDC

Si dans cette partie on ne traite que de la configuration CLDC, cela est dû au fait qu'elle est plus adaptée aux petits terminaux dont la puissance reste modeste. A contrario, la configuration CDC est destinée à des appareils possédant une

configuration matérielle (processeur et mémoire) plus robuste. La CLDC concerne donc les appareils possédant de 128Ko à 512Ko de RAM*, un processeur lent ainsi qu'une connectivité réseau limitée et intermittente. [CLDC01]

Définie par la spécification JSR* 030 [JSR30] dans sa version CLDC 1.0, cette configuration n'a pour but que de définir les bases communes à l'ensemble des appareils mobiles sur les points suivants :

- entrée-sorties (I/O)
- réseau
- sécurité
- internationalisation

Cependant, la configuration CLDC souffre de certaines limitations qui ne sont pas détaillées dans ce dossier. De ces limitations nous n'en retiendrons qu'une : le non-support de la virgule flottante. Il est donc impossible d'utiliser les types qui en découlent, à savoir « `float` » ou « `double` »; mais le problème peut être contourné en utilisant un package nommé MathFP. [MAT01]

De sa présentation dans la partie 1.2.2, CLDC hérite donc d'une part de plusieurs classes de la Java Standard Edition :

- java.lang
- java.util
- java.io

et implante d'autre part des classes spécifiques qui sont regroupées dans le GCF – Generic Connection Framework – disponibles dans le package java.microedition.io. Pour plus de détails concernant ce framework, vous pouvez vous référer au lien de la bibliographie [GCF01].

Pour information voici la liste des classes spécifiques à CLDC :

- Connection
- HttpConnection
- ConnectionNotFoundException
- Connector
- ContentConnection
- Datagram

- DatagramConnection
- InputConnection
- OutputConnection

1.2.3 Profil MIDP et Midlet

Par MIDP il faut comprendre Mobile Information Device Profil. Il s'agit d'un profil J2ME développé par Sun Microsystems et appliqué par certains fabricants de téléphones mobiles comme Sony Ericsson, Nokia, Motorola etc. Il est constitué d'un ensemble d'API définissant la manière dont les applications interagissent avec l'interface des téléphones mobiles.

De cette définition, on peut déduire qu'une Midlet est en fait une application répondant aux normes définies par un profil MIDP.

Le MIDP utilise dans sa version 1.0 la spécification JSR 37 [JSR37], ce qui lui confère les possibilités suivantes :

- gestion de l'interface utilisateur
- gestion de l'interface de connexion (réseau)
- gestion de stockage de données sur le terminal

En plus d'inclure les API de la configuration CLDC, MIDP ajoute trois autres packages correspondant aux possibilités énoncées plus haut :

- java.microedition.midlet : cycle de vie d'une application
- java.microedition.lcdui : interface et contrôle pour l'utilisateur
- java.microedition.rms : stockage de données persistantes

Afin de pouvoir gérer le cycle de vie d'une Midlet, elle doit obligatoirement hériter de la classe abstraite java.microedition.midlet. On pourra donc intervenir sur son état de trois façons :

- via la méthode startApp() : démarrage ou redémarrage de l'application
- via pauseApp() : pour sa mise en pause
- et via destroyApp() : pour sa destruction

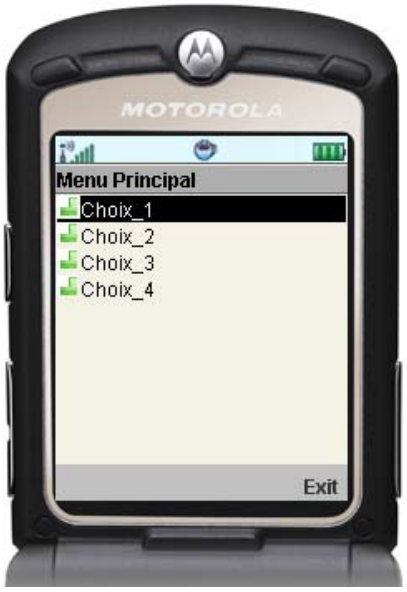
La version 2.0 de MIDP quant à elle, étend encore les possibilités de la version 1.0 en s'appuyant sur la spécification JSR 118 [JSR118]. [MIDP01]

1.2.4 Midlet : Exemple de création d'un menu

Afin d'illustrer les propos du point précédent concernant les Midlets, voici un exemple détaillé de réalisation d'un menu principal d'application destinée à un téléphone portable. Ceci pourrait constituer la première étape pratique de codage sous J2ME.

Ce menu est constitué d'une liste de quatre choix couplés chacun à une image ainsi que d'une commande « Exit » donnant la possibilité de mettre fin à l'application. Cette Midlet a été réalisée et testée pour un Motorola V3i™. Chaque annotation du type « Point.. » renvoie à un commentaire explicatif en fin de code.

Aperçu du rendu via un émulateur correspondant au modèle de téléphone précité :



[I_MENU01]

```
// Point A
import java.io.IOException;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
// Point B
public class Menu extends MIDlet implements CommandListener {
```

```
public Menu(){
	// Point C
	display = Display.getDisplay(this);
	buildMainMenu();
	started = true;
}
// Point D
public void startApp(){
	display.setCurrent(mainMenu);
}
public void pauseApp(){
	display.setCurrent(display.getCurrent());
}

public void destroyApp(boolean flag){
	notifyDestroyed();
}

private void buildMainMenu(){
	try{// Point E
		Image image =
		Image.createImage("/image.png");
		Image image1 =
		Image.createImage("/image.png");
		Image image2 =
		Image.createImage("/image.png");
		Image image3 =
		Image.createImage("/image.png");
		// Point F
		mainImageArray = (new Image[] {
		image, image1, image2, image3
		});
	}
	catch(IOException ioexception){
// Point G
	System.err.println("Impossible de charger
	l'image");
	}
// Point H
	String choix[] = {
	"Choix_1", "Choix_2", "Choix_3", "Choix_4"
	};
	mainMenu = new List("Menu Principal", 3, choix,
	mainImageArray);
// Point I
	mainMenu.addCommand(CMD_EXIT);
	mainMenu.setCommandListener(this);
}

// Point J
public void commandAction(Command command, Displayable
displayable){
	String s = displayable.getTitle();
	if(s.equals("Menu Principal")){
		if(command == CMD_EXIT)
			destroyApp(true);
```

```
                }
        }

        private Display display;
        private boolean started;
        private Command CMD_EXIT = new Command("Exit", 7,  0);
        private List mainMenu = null;
        Image mainImageArray[];
        String mainStringArray[];
}
```

Correspondance entre les points relevés et leur explication :

Point A : Importation des packages nécessaires à l'application.

Point B : Afin de pouvoir être exécutée sur un téléphone portable, la classe nouvellement créée doit étendre la classe de base MIDlet fournie par le package java.microedition.midlet. Il faut également implémenter l'interface CommandListener fournie pour permettre l'utilisation des touches de l'appareil. De ce fait, il est nécessaire d'implanter la méthode commandAction() définie au point J.

Point C : Création du menu principal et association de celui-ci à l'appareil.

Point D : La classe que l'on vient de créer étendant la classe Midlet, il est nécessaire d'implémenter les trois méthodes abstraites héritées : startApp(), pauseApp(), destroyApp() qui définissent les états que peut prendre l'application. notifyDestroyed() demande au gestionnaire de fermer d'application.

Point E : Pour rendre le menu plus agréable, on déclare quatre images provenant de fichiers de type *.png; ce type étant le seul supporté pour une application J2ME sur terminal mobile.

Point F : Les quatre images ainsi créées sont ajoutées au tableau d'images associées aux options du menu.

Point G : Le bloc précédent qui contient les points E et F est encadré par une exception de type IO qui, lorsqu'elle est levée, nous informe que un ou plusieurs chemins fournis pour les images ne sont pas corrects.

Point H : Ici on déclare un tableau de String portant les noms des différents choix de notre menu. Ce tableau va ainsi être utilisé pour créer une List nommée « mainMenu » déclarée avec les paramètres suivants : Titre du menu – Nombre d'index – Liste textuelle du menu – Tableau des images associées.

Point I : Une commande est ajoutée à la liste afin de permettre de quitter l'application lorsque cette action sera effectuée. Pour traiter les commandes, il faut associer un CommandListener au menu. La variable de commande CMD_EXIT est déclarée en fin de code avec les paramètres suivants : Nom affiché – Valeur correspondante à un type d'action (7 pour EXIT) – Une valeur de priorité.

Point J : Cette méthode traite les événements de type Command qui ont été reçus par l'objet de type Displayable actuellement affiché. On teste si l'objet en cours d'affichage correspond bien au menu principal, et si la commande Exit est pressée pendant l'affichage, on quitte l'application en appelant la méthode destroyApp().

Il est bien entendu possible de déclarer d'autres actions comme SELECT BACK, CANCEL, qui correspondent respectivement aux numéros d'actions 1, 2 et 3.

2. Technologie Bluetooth®

Du point de vue historique, le mot « Bluetooth™ » est tiré d'un surnom associé au roi Danois Harald Blåtand (en français « Dent Bleue ») [BLTH01]. Développé à l'origine par le constructeur Suédois Ericsson® en 1994, ce n'est qu'en 1998 qu'un groupe, composé au départ de cinq sociétés dont Ericsson, décide de poursuivre le projet et de produire les spécifications Bluetooth 1. Ce groupe, nommé SIG – pour Special Interest Group – joue un rôle important dans le développement de cette technologie en l'intégrant à leurs différents produits et en la commercialisant.

[I_BLTH01] [BLTH02]

Dans cette partie seront exposés successivement : les principes d'utilisation et de fonctionnement, les modes de communications suivis d'un exemple pratique de mise en œuvre du Bluetooth dans une application Java J2ME.

2.1 Principes d'utilisation et de fonctionnement

Dédié au transport de données sans fil, le protocole Bluetooth trouve son utilité dans plusieurs domaines, que ce soit pour coupler un téléphone à un ordinateur personnel, connecter deux terminaux entre eux ou encore les relier à divers accessoires comme des casques, des micros, ou des combinés clavier / souris. Cette technologie a certains avantages vis-à-vis de ses concurrents que sont le vieillissant IrDA* et le récent Wifi*. Simple de mise en œuvre, car intégré en standard dans quasiment tous les mobiles récents, ce protocole est utilisable dans une zone d'environ une dizaine de mètres entre les émetteurs-récepteurs et ne nécessite pas que les appareils soient parfaitement alignés pour fonctionner; deux avantages parmi d'autres qui lui confèrent une place de choix dans des domaines d'applications très variés.

Pour information, en 2005 l'institut Gartner annonçait qu'il se vendrait en 2009 plus de 580 millions de cellulaires équipés en standard du Bluetooth [GAR05].

Ci-dessous un tableau comparatif du Bluetooth face à ses principaux concurrents. On y retrouve la puissance de transmission, le taux de transfert, la zone de couverture etc.

	IrDA	Wireless	Bluetooth™
Type de connexion	Infrared, narrow beam	Spread spectrum	Spread spectrum
Spectrum	Optical 850 nm	RF 2.4 GHz	RF 2.4 GHz
Puissance de transmission	100mW	100mW	1mW
Taux de transfert	16Mbps	54Mbps et plus	1Mbps
Zone de couverture	1 mètre	100 mètres	10 mètres
Nb de périphériques supportés	2		8
Adresse matérielle	32 bit ID physique	48 bit de type MAC	48 bit de type MAC

[T_BLTH01]

La bande passante mise à disposition par le Bluetooth 1 n'est certes pas très importante puisque d'environ 1Mbits/sec théorique (54Ko/sec en pratique), mais permet tout de même une utilisation fluide dans le cas des exemples d'emplois énoncés plus haut. De plus, moyennant une configuration adaptée, il est tout à fait envisageable de rendre la navigation sur Internet possible depuis un mobile via le couplage avec un PC bénéficiant d'un accès web; ceci évitant les coûts aujourd'hui encore élevés d'une connexion GPRS* proposée par les opérateurs mobiles et facturée suivant le forfait choisi à la quantité de données envoyées et reçues.

Les caractéristiques principales du Bluetooth sont donc :

- technologie embarquée standard
- utilisation aisée et domaines variés
- faible coût
- basée sur des ondes radio
- faible consommation électrique

Le concept de PAN - pour Personal Area Network - qui symbolise une sorte de bulle dans laquelle un utilisateur bénéficie à tout moment et sans effort d'une connectivité réseau pour ses activités personnelles (voir exemples d'utilisations ci-dessus) englobe

donc parfaitement le protocole Bluetooth. Un réseau utilisant ce protocole est appelé un PicoNetwork; nous y reviendrons ultérieurement. Un PAN fait également référence à l'IrDA, à l'USB* et au FireWire*, même si les deux dernières ne sont pas sans-fil [PAN01].

Utilisant les ondes radio, les communications Bluetooth opèrent dans une gamme de fréquences allant de 2,4 GHz à 2,483 GHz et répondent à la norme IEEE 802.15.1* dans sa version 1. Les données sont transmises sous forme de paquets comme dans n'importe quel réseau IP wireless.

Chaque paquet ainsi transmis est muni de blocs de contrôle composés entre autres de l'identifiant de l'appareil auquel il est destiné. Afin d'assurer la confidentialité des communications, un algorithme de chiffrement est intégré au protocole : il s'agit du chiffrement à flot E0. Il se compose d'un générateur pseudo-aléatoire par combinaison de quatre LFSR – pour Linear Feedback Shift Register – pouvant se ramener au Summation Generator de Rueppel [RUEP01] [RUEP02]. Deux paramètres sont utilisés :

- longueur de clé
- vecteur d'initialisation

La longueur de clé peut varier mais est généralement de 128 bits. Elle est négociée entre les deux appareils. Le vecteur d'initialisation correspond quant à lui à l'adresse logique du terminal maître.

Pour coupler deux terminaux mobiles, une clé secrète dite « code » est à fournir au début du processus de couplage. Pour sécuriser la connexion, E0 est donc utilisé et se met en place en trois étapes : préparation de la clé, puis génération du flux et enfin chiffrement de ce dernier.

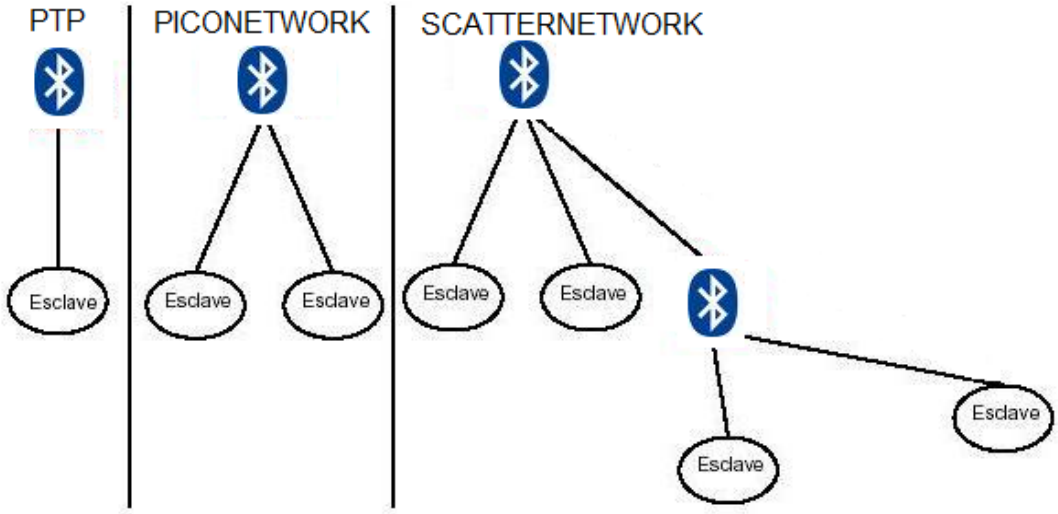
Bluetooth répond à un schéma de communication de type maître-esclave; le maître étant celui qui initie la connexion. En effet, lors de la communication entre périphériques, les données de contrôles énoncées plus haut permettent la mise en place d'un réseau appelé PicoNetwork. Ce type de réseau peut accueillir au maximum sept appareils esclaves et un maître (généralement un ordinateur). Dans ce cas, le PC agit comme un routeur* / switch* en aiguillant le trafic entre les différents appareils. De la même manière, il existe deux autres types de réseaux Bluetooth :

- ScatterNetwork
- PTP : point à point

Le ScatterNetwork donne la possibilité de créer un réseau en rendant le maître d'un PicoNetwork esclave dans un autre. Ainsi, le nombre de périphériques interconnectés peut aller jusqu'à 72, soit 10 PicoNetworks de 8 appareils auxquels il faut en retirer 8 ayant le rôle de passerelle.

Le PTP représente la plus petite cellule où seuls deux appareils communiquent entre eux.

Afin de mieux comprendre comment les différents réseaux s'organisent, voici une illustration schématique :



[I_BLTH02]

2.2 Les modes de communication

Du point de vue technique, il faut savoir que Bluetooth (comme IP énoncé à la partie précédente) utilise un système en couches pour la transmission de données. Le niveau le plus bas de la connexion est constitué de deux couches : physique et logique. La couche physique, constituée elle-même des couches Radio Fréquence et Bande de Base, prend en charge la partie matérielle pour le transfert des bits. La fonction de la couche logique (lien et transport) est de transmettre les données entre deux périphériques dans le cas de transports indépendants.

Ainsi donc, c'est au niveau de la couche Radio Fréquence que les flux de données à transmettre sont transformés afin d'être émis via le matériel physique de transmission. Ce matériel, et afin d'améliorer la qualité des communications, va opérer des sauts de

fréquence (79) sur la plage 2,4GHz—2,48GHz. La technique employée pour ces sauts est dite FHSS pour Frequency Hopping Spread Spectrum [FHSS01]. La gamme de fréquence disponible est découpée en multiples canaux et les communications se font en sautant successivement d'un canal à l'autre à un rythme fixé au moment de la mise en relation entre l'émetteur et le récepteur.

Pour gérer le type de communication il existe cinq transports logiques possibles, cependant nous n'en retiendrons que deux principaux :

- SCO : Synchronous Connection-Oriented
- ACL : Asynchronous Connection-less Link

C'est la couche Bande de Base qui se charge de les gérer. C'est également à son niveau que sont définies les adresses matérielles physiques des appareils. Ces adresses sont équivalentes aux adresses MAC d'une carte réseau. Elles sont nommées BD_ADDR.

2.2.1 Liaisons synchrones SCO

Ce type de liaison peut atteindre un débit de plus ou moins 400Kbits/sec en envoi comme en réception et ce de manière continue sans interruption de débit. Il sera utilisé de préférence pour le partage de fichiers entre plusieurs périphériques dans le cadre d'un PicoNetwork lorsque chacun d'entre eux veut accéder aux fichiers de l'autre.

2.2.2 Liaisons asynchrones ACL

Contrairement au type de liaison ci-dessus, ACL permet d'obtenir un débit élevé dans un sens (émission) et faible dans l'autre (réception). Le taux de transfert peut atteindre environ 700Kbits/sec - 50 Kbits/sec. On retrouve ce type de connexion dans le cadre d'une utilisation précitée à savoir : accès au web depuis un téléphone mobile via un ordinateur. En effet, il n'est pas forcément nécessaire d'avoir une vitesse d'upload* importante lorsque l'on ne fait que visiter des pages internet ou lire ses mails.

2.3 Bluetooth et Java

Dans le cadre de la programmation Java appliquée au Bluetooth, le package javax.bluetooth provenant de la spécification JSR 82 [JSR82] est nécessaire [BLTH03]. Dans cette partie, nous allons voir pas à pas comment réaliser une classe permettant la détection de périphériques Bluetooth. Cette détection est utile pour la suite du dossier, car elle permet de mettre en relation un mobile avec par exemple un module GPS. Un chapitre sera dédié à cette technologie et aux méthodes de récupération des données qu'elle peut fournir.

2.3.1 Les threads

Avant de commencer, il est nécessaire d'introduire le concept de Threads. En effet, comme énoncé dans la partie 1.2 du présent dossier, les terminaux mobiles étant de faible puissance, il faut utiliser la mémoire disponible et les temps processeur* de manière optimale. Par threads, on entend que « lorsqu'un programme s'exécute, le système crée un processus dédié auquel il alloue une certaine quantité de mémoire et un temps processeur. De ce fait, un programme peut être amené à créer un ou plusieurs sous-processus pour permettre l'exécution en parallèle de plusieurs opérations » [THRD_01].

Pour la recherche de périphériques, la classe à coder peut être lancée et interrompue via les touches du clavier; elle peut donc tout à fait être considérée comme un Thread. Cependant, notons simplement que pour gérer ces actions possibles, le système des threads utilisé en étendant la classe de base « Threads » permet de la mettre à tout moment dans quatre états différents :

- création via la méthode : run()
- exécutable via la méthode : start()
- en exécution testée via la méthode isAlive()
- bloquée via la méthode wait (temps en millisecondes)

Information : une page web très complète en français est disponible à la référence [THRD_02] de la bibliographie. De nombreux détails y sont fournis avec des exemples concrets de codes Java montrant leur mise en œuvre; raison pour laquelle ce point ne fera pas l'objet d'un développement plus poussé.

2.3.2 Exemple de méthodes de recherche Bluetooth

Vous trouverez ci-dessous un exemple regroupant plusieurs méthodes de classe capables de rechercher des appareils dont la fonction Bluetooth est activée et se trouvant en mode détectable. Toutes les instructions nécessitant une explication seront détaillées pas à pas.

Point A : Afin de pouvoir utiliser les classes fournies nécessaires au Bluetooth, aux exceptions et au stockage des équipements Bluetooth détectés, on importe :

```
import javax.bluetooth.*;
import java.io.IOException;
import java.util.Hashtable;
```

Point B : On déclare une nouvelle classe qui étend la classe de base Thread (voir point 2.3.1 ci-dessus) et implémente l'interface DiscoveryListener provenenant du package javax.bluetooth. Cette interface étant naturellement abstraite, il est nécessaire d'implémenter les méthodes qu'elle définit : deviceDiscovered(), servicesDiscovered(), serviceSearchCompleted() et inquiryCompleted().

```
public class BT extends Thread implements DiscoveryListener{
    public BT(){
        prhs_bluetooth = new Hashtable();
        serviceURL = "";
        nom_prh_bluetooth = "";
        recherche_prh = false;
        recherche_service = false;
        fermerBT = false;
    }
```

Le constructeur de la classe inclut six variables utiles : une variable de stockage des périphériques trouvés, des strings mémorisant l'url d'un périphérique et son nom, et trois booléens nécessaires pour connaître l'état de la recherche. Ces variables sont déclarées comme suit :

```
private Hashtable prhs_bluetooth;
private String serviceURL;
private String nom_prh_bluetooth;
private boolean fermerBT;
private boolean recherche_prh;
private boolean recherche_service;
private DiscoveryAgent agent_recherche;
```

Point C : La première méthode à coder sera nommée setupBT(). Dans un souci du détail, il est nécessaire de savoir si le mobile sur lequel on exécute l'application embarque une puce Bluetooth. Dans le cas où il n'en posséderait pas, une exception

est levée. Si au contraire le Bluetooth est supporté, il faut mettre en œuvre l'agent de recherche d'équipement Bluetooth qui existe dans la JVM et on lance la recherche via la méthode startDeviceSearch() :

```
private void setupBT(){
    try{
       LocalDevice notre_mobile =
       LocalDevice.getLocalDevice();
        agent_recherche = notre_mobile.getDiscoveryAgent();
    }
    catch(BluetoothStateException bluetoothstateexception){
       System.err.println("Bluetooth non supporté." +
       bluetoothstateexception);
    }
}
```

Point D : Détail de la méthode startDeviceSearch(). On passe la valeur de recherche_prh à true pour signaler que l'action est en cours et on avertit les autres processus.

```
public void startDeviceSearch()
 {
     recherche_prh = true;
     synchronized(this)
     {
         notify();
     }
 }
```

Voici la méthode principale run() qui orchestre le fonctionnement de la classe BT; elle se comporte comme suit : initialisation du Bluetooth et, si la connexion n'est pas interrompue, on teste les variables booléennes énoncées plus haut pour connaître dans quel état se trouve la recherche.

```
public void run(){
  setupBT();
  while(!fermerBT)
     if(recherche_prh)
     discoverDevices();
         else if(recherche_service)
                 discoverService();
         else synchronized(this) {
                 try{ wait();}
catch(InterruptedException interruptedexception){
System.err.println("Un autre thread a relancé run" +
interruptedexception);}
             }
     }
```

L'appel à la méthode discoverDevices() lance la recherche par l'intermédiaire de l'agent en invoquant startInquiry. Ses paramètres sont : un IAC (Inquiry Access Code) de type GIAC (Generic IAC : 0x9e8b33) utilisé par défaut pour rechercher tous les périphériques se trouvant à portée, ainsi que le listener de découverte (DiscoveryListener).

```
private synchronized void discoverDevices()
  {
      recherche_prh = false;
      prhs_bluetooth.clear();
      try
      {
      agent_recherche.startInquiry(0x9e8b33, this);
      }
    catch(BluetoothStateException
    bluetoothstateexception)
      {
     System.err.println("Erreur") ;
      }
  }
```

Une fois un périphérique trouvé, la méthode deviceDiscovered() est appelée, elle récupère son nom et son adresse. Ces données sont ensuite ajoutées au Hashtable prhs_bluetooth si elles n'y sont pas déjà.

```
public void deviceDiscovered(RemoteDevice phr_decouvert,
DeviceClass type_phr){
    String s;
    try {s = phr_decouvert.getFriendlyName(true);}
    catch(IOException ioexception){
           s = phr_decouvert.getBluetoothAddress();}
    RemoteDevice remotedevice1 =
    (RemoteDevice)prhs_bluetooth.get(s);
    if(remotedevice1 == null){
       prhs_bluetooth.put(s, phr_decouvert)
       }
}
```

Point E : Il est possible d'afficher un message lorsque la recherche est terminée via la méthode héritée : inquiryCompleted().

```
public void inquiryCompleted(int arg0) {
   fermerBT = true;
   System.out.println("fin de la recherche");
}
```

Les autres méthodes annoncées au point A sont utiles pour la recherche de services sur les périphériques Bluetooth qui ont été trouvés. servicesDiscovered() permet de récupérer l'URL* du premier service offert par l'appareil sélectionné. serviceSearchCompleted() est enfin appelée lorsque l'URL a été récupérée.

Monsieur Oscar Vivall, travaillant chez le constructeur SonyEricsson, met à disposition sur le site web de support aux développeurs, un exemple concret et fonctionnel de la recherche Bluetooth; voir le lien [BLTH04] dans la bibliographie.

3. Le GPS

Requis pour la géolocalisation par satellite, le projet de GPS – pour Global Postioning System – date des années 1970 et fut développé par l'armée américaine. Hormis à des fins militaires, il est par exemple utilisé par les professionnels pour gérer une flotte de véhicules ou par les particuliers lors de longs déplacements. Ce système est actuellement d'une précision de moins d'une dizaine de mètres et permet de se repérer sur Terre au moyen de 31 satellites gravitant à 20 000 Km d'altitude.

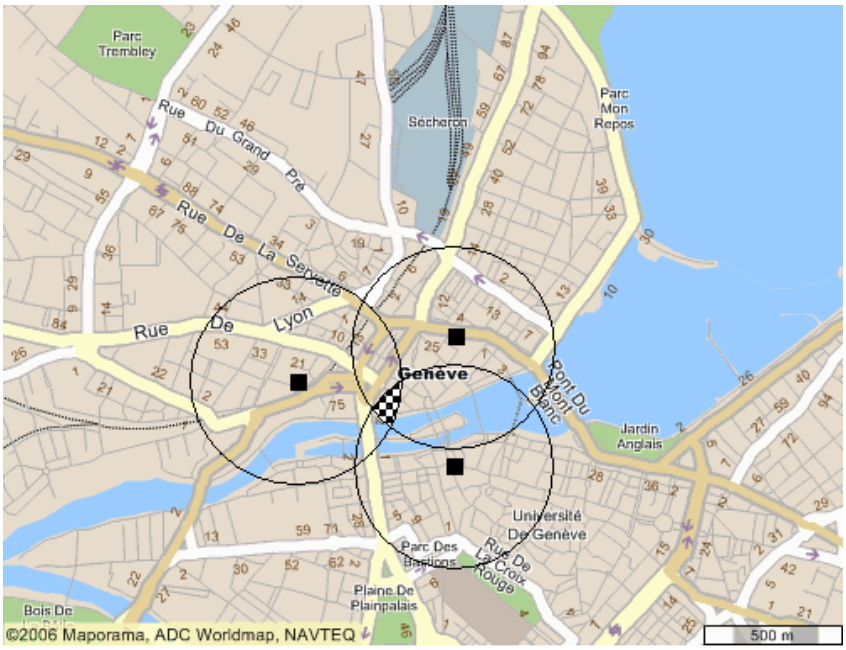
Dans un premier temps, nous présenterons comment le GPS fonctionne et quelles indications il permet d'obtenir. Nous verrons ensuite comment les traiter via un exemple de « parser de trames envoyées par un récepteur Bluetooth » écrit en Java. Enfin, une liste de plusieurs récepteurs GPS sera présentée afin de définir certains critères de choix.

3.1 Principes d'utilisation et de fonctionnement

Le GPS civil peut fonctionner grâce aux satellites mis en orbite autour de la Terre ; ils émettent sur la fréquence précise de 1575,42 Mhz. Les ondes ainsi transmises traversent les différentes couches de l'atmosphère et sont captées par des récepteurs de différents types; nous y reviendrons. La question qui semble être la plus intéressante est : comment la localisation fonctionne-t-elle ? Pour y répondre il faut connaître succinctement la notion de triangulation.

Prenons comme exemple la Cibi*. Lorsque des utilisateurs de ce système de communication veulent situer une personne qui émet des propos plus ou moins choquants sur les ondes, elles utilisent la triangulation. Sachant qu'une Cibi, grâce à son antenne, émet et reçoit dans un certain rayon, c'est avec une carte et un compas que l'on peut découvrir où se trouve la personne. Pour ce faire, il faut qu'un minimum de trois individus assez éloignés les uns des autres soient en mesure de capter le message en question. Grâce au compas, chaque personne trace un cercle sur la carte dont le centre est la position exacte où elle se trouvait au moment de l'écoute. Le diamètre de ce cercle correspond évidemment à la distance maximale de réception que peut atteindre l'appareil. Ainsi, les trois cercles vont se croiser, délimitant une zone depuis laquelle le message a été émis. Voici un schéma explicatif sur lequel on

retrouve les trois cercles. Imaginons que dans le cas présent une Cibi ne peut émettre et recevoir que sur quelques centaines de mètres. Dans la réalité cela est de l'ordre de plusieurs kilomètres.



[I_GPS01]

Les carrés noirs correspondent aux différentes Cibis et la zone de croisement des trois cercles recouverte d'un damier indique où se trouve la personne recherchée. De plus, il est possible d'augmenter la précision de la localisation en ajoutant un quatrième cercle symbolisant un auditeur de plus.

En ce qui concerne le GPS, les récepteurs font plus ou moins le même travail. Le nombre de satellites nécessaires à une location précise (regroupant toutes les informations que peut offrir le GPS) est de quatre. Ainsi donc, chaque satellite embarque une horloge atomique* et envoie à intervalles réguliers un message par ondes radio. Ce message comporte l'identifiant du satellite, sa position, et le moment précis auquel il a été émis. Des satellites, le récepteur va capter les messages et, par déduction du rayon d'émission de chacun d'eux, sera en mesure de calculer sa position par triangulation.

Les données du système GPS exploitables via un récepteur sont entre autres :

- l'heure du message
- la latitude
- la longitude

- le nombre de satellites actuellement acquis pour la mesure
- l'altitude par rapport au niveau moyen des océans
- la vitesse de déplacement

Elles sont donc calculées et envoyées par le récepteur GPS sous forme de trames*. Ces trames peuvent être de différents types suivant les informations désirées, mais sont toutes formées de caractères ASCII*.

Un des deux protocoles qui définit l'envoi de ces trames est le NMEA – pour National Marine & Electronics Association. NMEA est, comme son nom l'indique, une association fondée par un groupement de professionnels de l'industrie de l'électronique, en collaboration avec des fabricants, des distributeurs, et des revendeurs. Un des ses buts a en fait été de standardiser et d'harmoniser les équipements de la marine.

Pour information, il en existe un second, le SiRF, dont la structure est différente du premier car représentée en base hexadécimale* et non en base 10. Sa complexité est donc d'un niveau supérieur. [SIRF01]

3.1.1 Trames NMEA

Le standard NMEA-0183 définit non seulement une interface matérielle (port série RS-232), mais également un protocole utilisé entre un récepteur GPS et une machine capable d'interpréter les données. Le protocole met à disposition plusieurs types de trames (chaque type est identifié par trois lettres) suivant les données que l'on souhaite exploiter. Voici les principaux types :

- GGA : l'heure, latitude & longitude, nombre de satellites et altitude
- GLL : l'heure, latitude & longitude
- GSA : satellites utilisés
- GSV : satellites vus par le récepteur
- RMC : latitude & longitude, direction, vitesse

Chaque trame débute par un « $ » suivi de deux caractères identifiant l'émetteur de la trame (GP pour le GPS), accompagnés ensuite de l'identifiant du type de trame représenté par les trois lettres énoncées ci-dessus. Ces premières informations sont terminées par une « , » servant à séparer chaque information contenue dans la trame.

Pour terminer la trame, un checksum* de contrôle d'intégrité est ajouté. On obtient donc pour une trame simple comme GLL :

*$GPGLL,461035.10,N,60824.47,E,120000,A*2C*

Dans l'ordre : GPS utilisé, trame GLL, coordonnées de la HEG, mesure obtenue à midi pile, A = données valides et checksum.

Certains récepteurs offrent la possibilité de paramétrer le type de trames que l'on souhaite recevoir. Ceci est très utile suivant le programme qui les utilisent et la manière dont il les traite. Le paramétrage se fait via l'envoi à l'appareil d'une trame de configuration. Pour que le récepteur nous transmette les trames du type GLL de l'exemple précédent il faut lui envoyer la commande suivante :

*$PSRF100,01,9600,10,1,0,*0C*

$ débute la trame de configuration, comme toutes les trames qui seront envoyées par le récepteur. Ensuite vient l'entête du protocole suivi du type de trame désirée (correspondances ci-dessous), le taux de transfert en bauds*, le nombre de secondes entre chaque envoi, l'activation du checksum (0 = non, 1 = oui), la parité* et enfin le checksum de la trame de commande [NMEA01].

Trames et valeurs de configuration associées :

GGA : 00 GLL : 01 GSA : 02 GSV : 03 RMC : 04

3.1.1.1 Exemple de parser de trames NMEA de type GLL

Dans cette partie, un exemple fonctionnel de parser de trames de type GLL est présenté. Il se compose de deux classes, l'une fournissant un objet dont les attributs regroupent les données fournies par une trame GLL (GllData), et l'autre s'occupant de la découpe de la trame envoyée par le récepteur GPS (GllParser). Pour plus de clarté, on convient que la trame est ici incluse dans le code et que des instructions de type System.out.println affichent le résultat en interrogeant l'objet de type GllData créé. La trame est celle correspondant à l'exemple du point précédent. Enfin, une copie d'écran de la console après exécution est fournie.

```
// Point A
public class GllData {
       private String gpsTime;
       private char latitudeDirection;
       private char longitudeDirection;
       private float longitude;
       private float latitude;
  // Point B
    public GllData(){
              longitudeDirection = 0;
              latitudeDirection = 0;
              longitude = 0;
              latitude = 0;
              gpsTime = "";
         }
  // Point C
    public void setTime(String time){this.gpsTime = time;}

    public void setLatitude(float latitude, char direction){
               this.latitude=latitude;
               this.latitudeDirection = direction;
     }

    public void setLongitude(float longitude, char directionLo){
               this.longitude = longitude;
               this.latitudeDirection = directionLo;
     }
  // Point D
     public String getGPSTime(){return gpsTime ;}

     public float getLatitudeSeule() { return latitude ;}

     public String getLatitudeComplete(){
           return Float.toString(latitude)+latitudeDirection;
          }

     public float getLongitudeSeule(){ return longitude ;}

     public String getLongitudeComplete(){
           return Float.toString(longitude)+longitudeDirection;
          }
}
```

Cette classe ne présentant pas de difficultés particulières et ne faisant pas appel à des classes de packages spéciaux voici, donc la correspondance entre les points relevés et leur explication succincte:

Point A : Déclaration de la classe et des variables nécessaires.

Point B : Constructeur de la classe.

Point C : Méthodes d'accès aux attributs de l'objet; elles permettent à la classe appelante de stocker les données dans l'objet GllData. Par convention, tous les noms de ces méthodes commencent par le mot « set » suivi du nom de l'attribut qu'elles permettent de changer.

Point D : Méthodes d'accès aux attributs de l'objet; elles permettent à la classe appelante de connaître la valeur des attributs de l'objet GllData. Par convention, tous les noms de ces méthodes commencent par le mot « get » suivi du nom de l'attribut dont elles permettent de récupérer la valeur.

Voici maintenant la classe GllParser qui, comme son nom l'indique, va permettre de découper la trame pour n'en retenir que les données utiles. Le format de la trame obéissant au standard énoncé dans la partie 3.1.1, il est nécessaire de procéder à la récupération des données dans le bon ordre en ce qui concerne la longitude et la latitude.

```
public class GllParser
{
// Point A
    private GllParser() {}
// Point B
    private static GllData parseGll(String trameGll, GllData
data){
   // Point C
       if(trameGll.charAt(0) == '$'){
        String typeTrame = trameGll.substring(1, 6);
   // Point D
        if(typeTrame.equals("GPGLL")) {
   // Point E
         int tabVirgule[] = trouverVirgules(trameGll);
   // Point F
         data.setTime(extraireDonnees(tabVirgule, trameGll, 4));
   // Point G
        data.setLongitude(parseFloat(extraireDonnees(tabVirgule,
     trameGll, 2)),
     extraireDonnees(tabVirgule, trameGll, 3).charAt(0));
   // Point H
       data.setLatitude(parseFloat(extraireDonnees(tabVirgule,
     trameGll, 0)),
     extraireDonnees(tabVirgule, trameGll, 1).charAt(0));
        }
       } else {System.out.println("Mauvais format de trame");}
        return data;}

  // Point I
    private static String extraireDonnees(int tabVirgule[],
String trameGll, int i){
       String substring;
```

```
       substring =  trameGll.substring(tabVirgule[i] + 1,
tabVirgule[i + 1]);
        return substring;
    }

// Point J
    private static int[] trouverVirgules(String trameGll){
        int i = 0;
        int tabVirgule[] = new int[trameGll.length()];
        char tabTrame[] = trameGll.toCharArray();

// Point K
        for(int j = 0; j < tabTrame.length; j++){
            if(tabTrame[j] == ','){
            tabVirgule[i] = j;
            i++;}
        }
        return tabVirgule;
    }

// Point L
    private static float parseFloat(String partieTrameGll){
        float f;
        f = Float.parseFloat(partieTrameGll);
        return f;
    }

// Point M
    public static void main (String args[]) {
       GllData data = new GllData();
       parseGll("$GPGLL,461035.10,N,60824.47,E,120000,A", data);

       System.out.println("Temps GPS : " +
       data.getGPSTime().substring(0, 2)+ " Heures " +
       data.getGPSTime().substring(2, 4)+" Minute(s)
       "+data.getGPSTime().substring(4, 6)+" Seconde(s)");

       System.out.println("Latitude Complète : " +
       data.getLatitudeComplete());

       System.out.println("Longitude Complète : " +
       data.getLongitudeComplete());

    }
}
```

Correspondance entre les points relevés et leur explication :

Point A : Déclaration de la classe et de son constructeur; l'exemple présenté devant s'exécuter seul grâce à la méthode main() (point K), le constructeur reste vide.

Point B : Méthode principale de la découpe de trame; elle reçoit en paramètre la trame à découper ainsi qu'un nouvel objet de type GllData exposé plus haut dont elle va, via les accesseurs, définir la valeur des attributs. Cet objet sera nommé data.

Point C : La première étape consiste à vérifier si la trame reçue est effectivement une trame GPS. Pour ce faire, on teste si elle commence par le symbole « $ » grâce à la méthode charAt() pour l'indice 0. Si cela n'est pas le cas, un message d'erreur est affiché.

Point D : Une fois le premier test réalisé, il convient de s'assurer également que la trame est au format GLL. On vérifie donc que les cinq caractères « GPGLL » sont bien présents après le « $ ». La méthode substring() permet de réaliser cette opération en ne retournant que les caractères présents dans un intervalle défini en paramètre (de l'indice 1 à l'indice 5). L'indice 0 correspondant bien évidement au « $ ».

Point E : Ici, on crée un tableau qui aura pour rôle de stocker la position des virgules qui délimitent les différentes informations de la trame. Ainsi, on fait appel à la méthode trouverVirgule() à qui l'on fait passer en paramètre la trame à découper (voir point J).

Point F : A ce point, on va faire appel à la méthode extraireDonnees() (point I) pour connaître le temps GPS et ainsi définir la valeur de l'attribut « time »de l'objet de type GllData (nommé data). L'accesseur utilisé est setTime() avec comme paramètre le temps GPS sous forme de String.

Point G : La logique est la même que pour le point précédent. Cependant, on traite ici la longitude qui sera stockée dans l'objet sous forme de float. La méthode parseFloat() est ainsi appelée dans ce but (voir point L). L'accesseur nécessaire est setLongitude().

Point H : Idem que le point G pour la latitude insérée dans l'objet via la méthode d'accès setLatitude().

Point I : extraireDonnees() constitue la méthode de récupération de données proprement dite sous forme de String. Elle reçoit trois paramètres que sont le tableau d'indices des virgules (voir point J), la trame, ainsi que l'indice du tableau de positions des virgules à partir duquel il faut extraire les données. C'est-à-dire pour le temps GPS qui se situe après la cinquième virgule de la trame, l'indice 4 est donc fourni (le premier

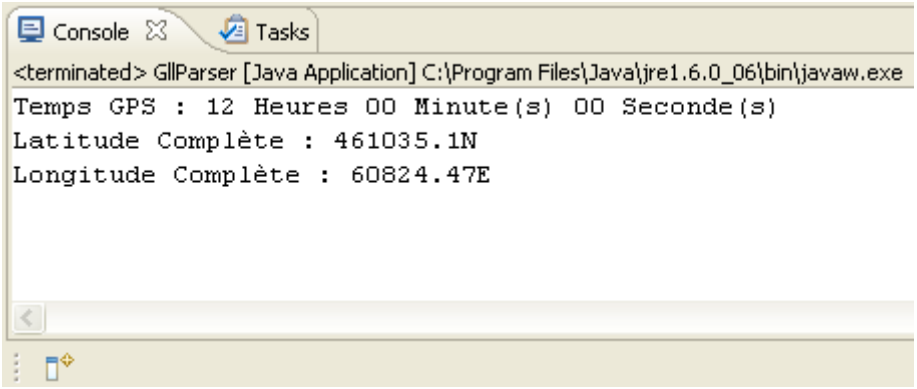
indice d'un tableau étant l'indice 0). Le découpage s'effectue ainsi jusqu'à la prochaine virgule de la trame.

Point J : La méthode trouverVirgule() crée un tableau tabVirgule de positions des virgules dans la trame stockées sous forme d'entiers (voir point K). Pour ce faire, elle reçoit en paramètre la trame GLL sous forme de String qu'elle va tout d'abord stocker dans un tableau de Char nommé tabTrame. Pour remplir ce tableau, la méthode toCharArray() de trameGll est invoquée. Cette méthode découle du type String.

Point K : Une boucle de type « for » va servir à analyser la trame et rechercher la position des virgules qui s'y trouvent. Si une virgule est trouvée, on stocke la valeur de j dans tabVirgule, puis on incrémente la valeur de l'indice courant afin de pouvoir enregistrer la position d'une prochaine virgule « i++ ».

Point L : parseFloat() a pour fonction de retourner la latitude ou la longitude sous forme de float. En effet, elle reçoit en paramètre une partie de la trame découpée par la méthode extraireDonnees(). L'appel à la méthode parseFloat() de la classe Float permet cette opération.

Point M : main() étant la première méthode appelée lors de l'exécution de la classe GllParser, il convient de créer tout d'abord le nouvel objet de type GllData, nommé data, avec lequel on souhaite travailler (modifier ses attributs). On invoque ensuite la méthode parseGll() de la classe en lui fournissant la trame à découper, l'objet data, et pour cet exemple on affiche les informations que l'objet contient. Voir ci-dessous pour l'affichage des données dans la console.



[I_GLL01]

3.2 Critères de choix et récepteurs GPS

Dans un marché en pleine expansion depuis les années 1990, les récepteurs GPS, tout d'abord destinés à une utilisation militaire, se retrouvent aujourd'hui dans bien des domaines d'applications civils du quotidien. De ces domaines on peut citer : le guidage routier, la navigation maritime et aérienne, les sports de nature. Chacun de ces usages nécessite normalement un appareil possédant des fonctionnalités dédiées. Il peut s'agir d'appareils soit intégrés par les constructeurs (notamment pour l'automobile ou l'aéronautique), soit portatifs mais avec écran, soit mobiles mais dont l'usage ne peut se faire qu'en étant couplés à une machine comme des notebooks, des téléphones ou des PDA. Dans ce dernier cas, on parle de récepteur GPS de type « antenne », se contentant uniquement de recevoir les signaux des satellites, de les interpréter et les codifier pour les transmettre ensuite à d'autres appareils capables de les utiliser. Cependant, les applications restent la plupart du temps les mêmes que celles des autres types de récepteurs.

Il faut donc, avant de choisir son GPS, savoir de quelle manière on souhaite s'en servir. Cette section, en rapport direct avec les différentes parties de ce dossier déjà présentées, traitera des récepteurs de type antenne.

Voici tout d'abord une liste des principaux critères à prendre en compte lors de l'achat d'un récepteur GPS. Elle sera suivie d'une description de trois modèles représentant au mieux ces critères.

Puce GPS : C'est le premier critère décisif dans le choix du module. En effet, les plus répandues sont fabriquées par le constructeur californien Sirf. Elles se modernisent au fil des années et en sont actuellement à la version Sirf Star III. D'autres constructeurs, comme Garmin ou Nemerix, fournissent également des puces GPS. Sachez cependant qu'elles permettent toutes de transmettre les trames dans les différents formats exposés dans la partie 3.1; il n'y a donc aucun souci de compatibilité. Les seules différences notables sont les temps d'acquisition des satellites à froid ou à chaud [GPS01] et la consommation électrique.

Connectivité : Même si un récepteur GPS antenne se doit d'être mobile, sa connectivité vers d'autres appareils auxquels il peut être couplé a son importance. Privilégiez la connectique Bluetooth lors des déplacements et la connectique USB dans les autres cas. Vérifiez la présence d'un connecteur permettant d'adjoindre une antenne extérieure au récepteur, ce qui lui permettra d'obtenir une réception bien

meilleure notamment dans le cas d'une utilisation en milieu couvert ou encaissé. Notez que ce genre d'antenne est le plus souvent orientable.

Poids : Pour une utilisation exclusivement mobile, comprenez transport à pied ou à vélo, le poids est essentiel. Il varie la plupart du temps entre 70 et 150 grammes. Les plus légers sont donc logiquement dépourvus d'une connectivité étendue et d'un panel d'accessoires conséquent.

Autonomie : Rien de plus pénible que de devoir changer les piles d'un appareil. Actuellement, la quasi-totalité des récepteurs fonctionnent sur batterie rechargeable, dont l'autonomie en fonctionnement varie entre 6 et plus de 30 heures. Certains appareils haut de gamme comprennent un capteur solaire qui permet d'atteindre les 100 heures avec une luminosité suffisante.

Accessoires : Suivant le constructeur, une variété importante d'accessoires existe, allant d'un étui de transport, d'une dragonne, d'un chargeur secteur ou allume-cigare, à l'antenne extérieure et au support pour pare-brise à ventouse dans le cas d'une utilisation en voiture, ou aimanté pour le vélo par exemple. Rajoutons qu'il existe dans certains cas la possibilité de changer la batterie d'origine par une autre fournissant une meilleure autonomie, mais avec un temps de charge et un encombrement supérieurs.

Support : Aspect parfois négligé, il dépend du constructeur. En effet, la mise à disposition de programmes de tests, de cartes, de mises à jour firmware*, de cartographie, d'aide en ligne ou par téléphone sont le plus souvent offerts uniquement par les fabricants renommés.

Prix : La somme allouée à l'achat étant variable suivant les moyens de chacun, sachez que pour ce type de récepteur, les prix débutent à environ 55.- euros – soit environ 90.- francs suisse.

Le tableau ci-dessous compare les trois modules GPS Bluetooth que nous avons retenus : le RoyalTek RBT-2200, le Just-Mobile GP-01 et le Garmin GPS-10.

Produit / Spécifications	RoyalTek Cie Ltd RBT-2200	Just-Mobile GP-01	Garmin GPS-10
Image			
Puce GPS **Temps acquisition**	SirfStar III 37 sec max	SirfStar III 42 sec max	Garmin WAAS enabled 45 sec max
Connectivité PC **Antenne externe**	Bluetooth Oui suivant version	Bluetooth, USB Non	Bluetooth Non
Poids	60 grammes	55 grammes	80 grammes
Batterie **Autonomie**	Li-ion 6 à 7 heures	Li-ion 8 à 9 heures	Li-ion 12 heures
Accessoire fourni	Chargeur voiture	Chargeur voiture Chargeur secteur Câble USB	Chargeur voiture Logiciel de cartographie
Support	Drivers & Démo	Drivers & Démo	Drivers, Cartes
Prix	~ 55 euros	~ 80 euros	~ 110 euros

[T_GPS01]

4. L'environnement de développement MotoDev Studio

Ce chapitre présente l'IDE – pour Integrated Development Environment – fourni par le constructeur Motorola.

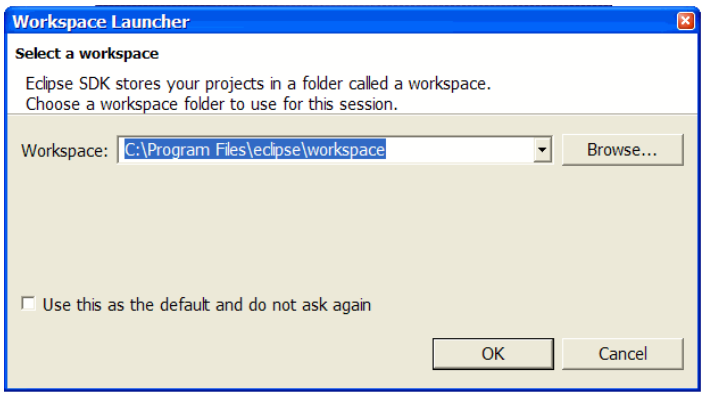
Basé sur l'IDE open-source Eclipse [ELCI01], il met à disposition des émulateurs propres aux différents modèles de téléphones de la marque. Durant la dernière année d'étude HES à la HEG, nous avons pu prendre en main cet environnement, que ce soit tant pour les cours de génie logiciel que pour la réalisation de notre applicatif dans le cadre du GREP. C'est pourquoi MotoDev Studio ME nous est apparu comme le meilleur choix face à son rival NetBeans développé lui par Sun Microsystems. Sachez cependant que d'autres fabricants comme Ericsson ou Nokia mettent eux aussi à disposition soit des IDE dédiés à leur marque, soit des modules à intégrer à des environnements de développement existants.

Nous ferons tout d'abord un tour d'horizon de l'IDE, puis nous accentuerons notre présentation sur la procédure de création et le test du code de la partie 1.2.4 : création d'un menu.

4.1 Utilisation de MotoDev

4.1.1 Le Workspace

Motodev Studio est disponible au lien de la bibliographie [MOTO01]; le téléchargement nécessite préalablement la création d'un compte utilisateur gratuit sur le site developer.motorola.com. Une fois l'application installée, vous obtenez au premier lancement la fenêtre suivante dont voici une copie d'écran :



[I_MOTO01]

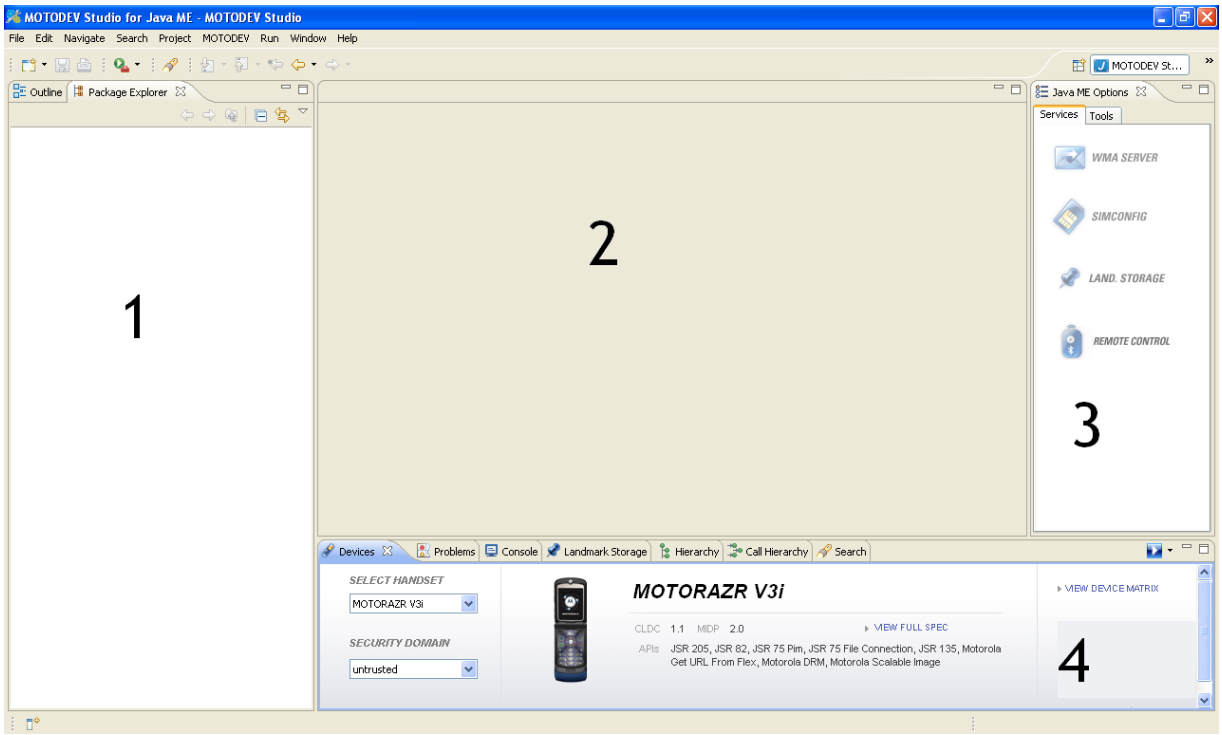
A cette étape, il faut choisir un workspace; entendez par là espace de travail, où seront stockés vos projets et les différents fichiers qu'ils contiendront. Par défaut, celui-ci se trouve dans le dossier d'installation de MotoDev. Vous pouvez très bien changer le chemin et donc choisir un dossier ailleurs, même sur un support de stockage amovible.

MotoDev est, comme Eclipse, capable de gérer plusieurs workspaces. Imaginons que vous désirez avoir deux environnements : le premier pour vos travaux personnels et le deuxième pour vos travaux professionnels. Une fois dans l'application, il est possible de basculer d'un workspace à l'autre en choisissant « File → Switch WorkSpace ». Ainsi, les workspaces s'ouvriront avec leurs projets et préférences respectifs.

4.1.2 La perspective

La perspective par défaut est composée de quatre cadres :

- 1 : l'explorateur de projet (affichage des projets sous forme d'arbre)
- 2 : la feuille de code actuellement ouverte (zone d'édition)
- 3 : les raccourcis vers différents services fournis par Motorola
- 4 : le modèle de téléphone actuellement choisi pour l'émulation

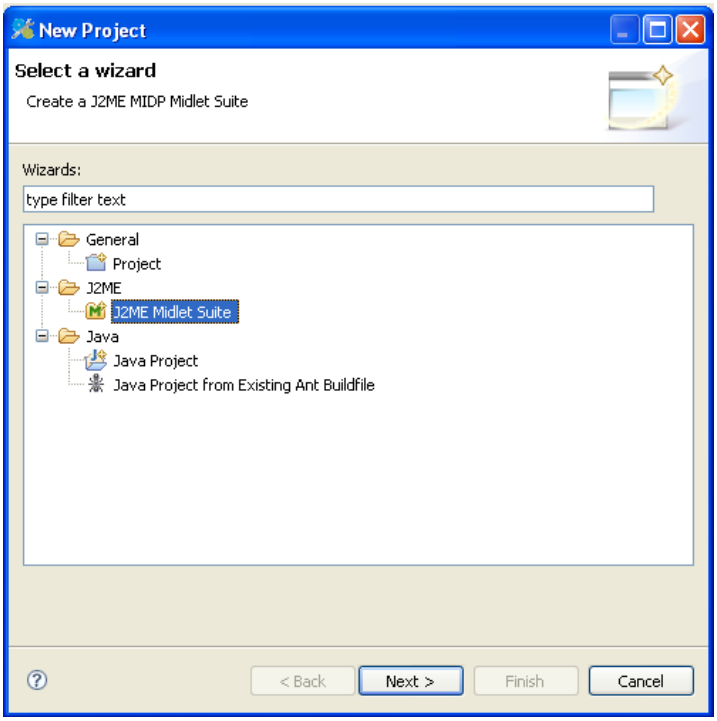


[I_MOTO02]

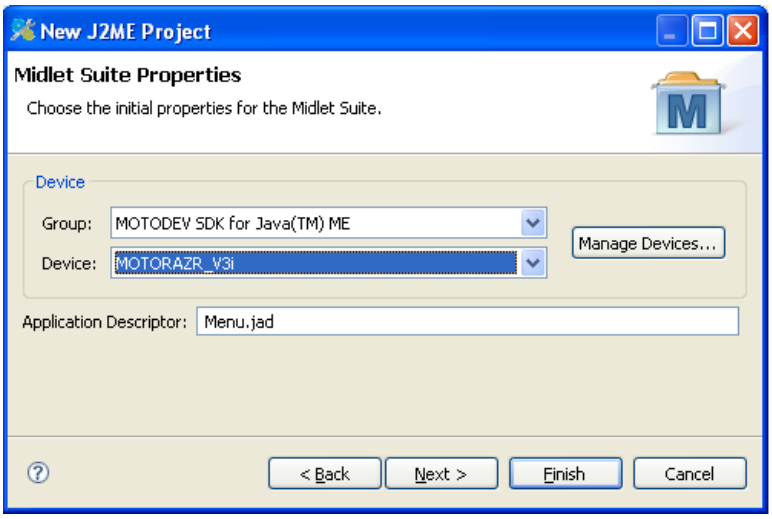
Cette perspective par défaut est entièrement personnalisable et inclut bien évidemment tous les outils de la version de base d'Eclipse. Plusieurs autres perspectives sont affichables suivant le type de travail que vous effectuez comme le débogage par exemple. Pour accéder à ces perspectives, rendez-vous dans le menu « *Windows → Open Perspective… ».*

4.1.3 Création d'un projet

Le cadre 1 « Package Explorer », comme énoncé plus haut, sert à l'affichage des différents projets créés avec MotoDev. Afin d'illustrer son fonctionnement, nous allons décrire la procédure de création d'un nouveau projet nommé : Menu. Pour ce faire, rendez-vous dans le menu *« File → New Project »,* vous obtenez la fenêtre suivante :



[I_MOTO03]

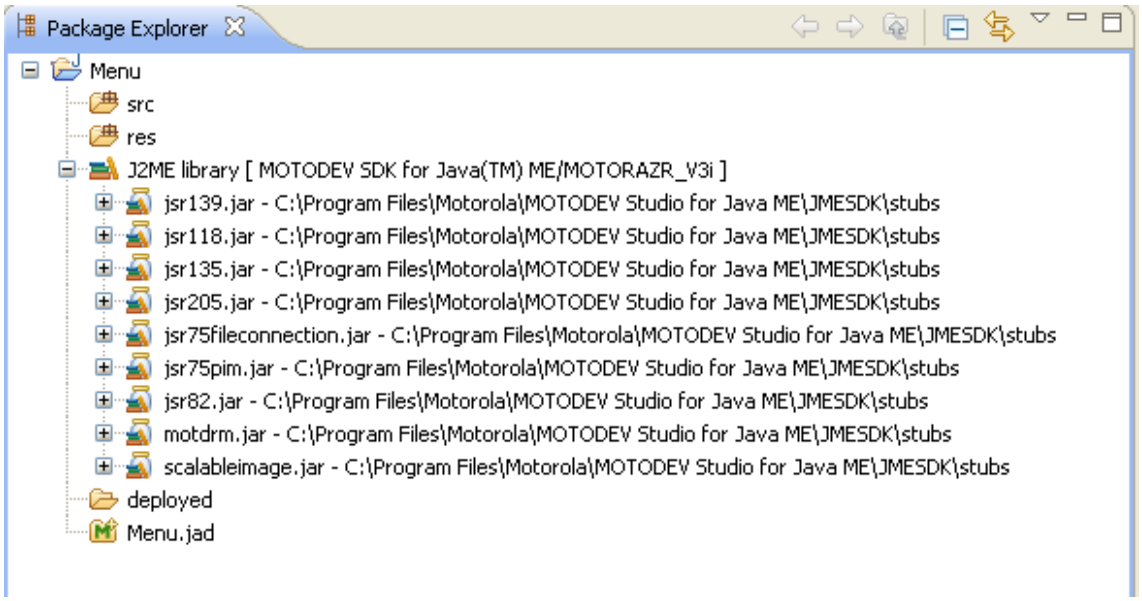


[I_MOTO04]

Ce dossier étant dédié au développement d'application Java pour téléphones mobiles, choisissez J2ME Midlet Suite comme type de projet et nommez-le « Menu ». Cela fait, voici la première fenêtre de configuration spécialement dédiée à l'édition MotoDev [I_MOTO04]. Vous avez la possibilité de choisir le type de téléphone portable (émulé) sur lequel vous désirez que votre application soit exécutée. Cliquez ensuite sur « Finish » pour terminer le processus, ou sur « Next » afin d'importer les différentes librairies de fonctions dont vous avez besoin. Sachez que cette opération peut très bien s'effectuer ultérieurement lors du codage proprement dit en utilisant le menu contextuel de votre projet accessible via « Clique droit → Import ».

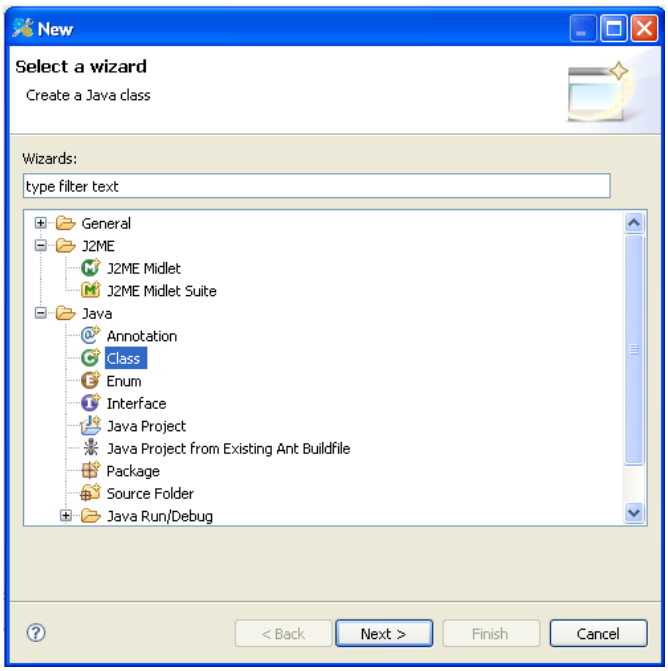
Dans l'arborescence du cadre package explorer, vous voyez apparaître votre projet composé des différents dossiers suivants :

- Src : qui contiendra vos packages et fichiers java
- Res : qui sera le dossier contenant les fichiers compilés
- J2ME library : correspondant aux librairies de fonctions déjà importées
- Deployed : représentant le dossier de sorties de votre application compilée
- Archive Jad : le fichier final de votre application prêt à être installé sur votre téléphone



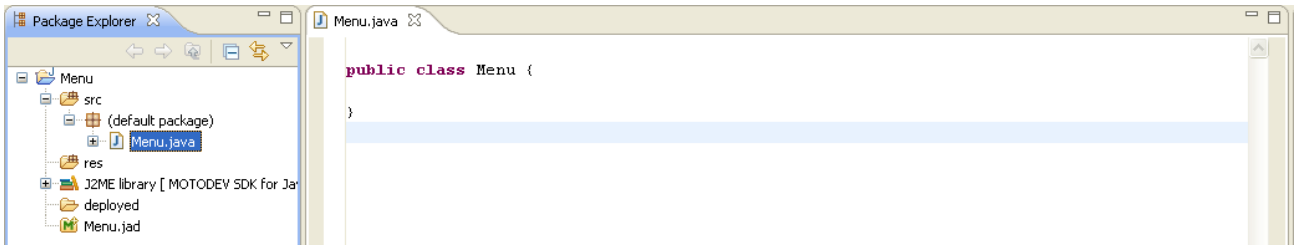
[I_MOTO05]

Pour insérer une nouvelle classe dans votre projet, faites un clic droit sur le dossier src et sélectionner successivement *« New → Other »*. Choisissez ensuite le type de fichier que vous souhaiter créer. Pour l'exemple, on sélectionne une nouvelle Classe [I_MOTO06] que l'on nommera *« Menu »* puis on clique sur *« Next ».*



[I_MOTO06]

Ainsi on obtient un nouveau fichier Java dans le dossier src et le cadre 2 laisse apparaître le code actuel de ce fichier [I_MOTO07] :



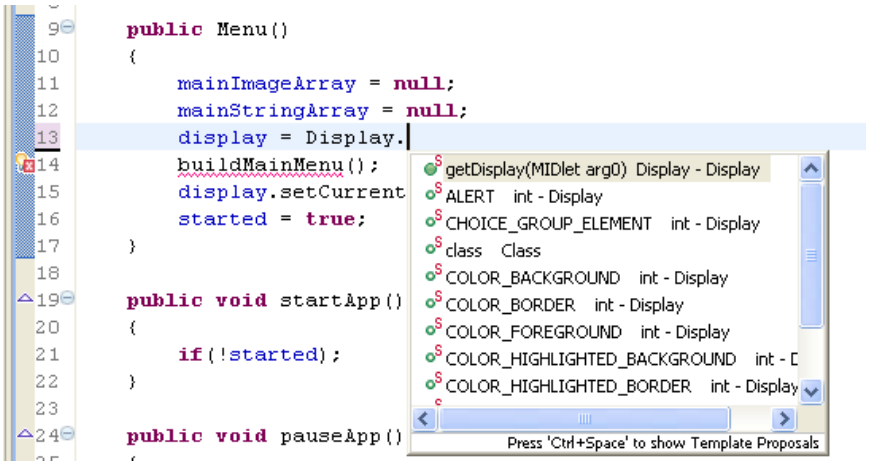
[I_MOTO07]

Dans le cadre de cet exemple, nous passerons sous silence toute la partie de codage proprement dite, le code étant disponible à la partie 1.2.4.

Comme tout IDE, MotoDev possède plusieurs options dont le but est de rendre le codage plus aisé. Les principales sont :

- L'auto complétion : pour la suggestion des classes, fonctions et méthodes à utiliser au moment de la frappe.
- Le correcteur syntaxique : qui vérifie ce que vous écrivez.
- Le vérificateur de code en temps réel : qui compile votre code à chaque nouvelle instruction pour vous permettre de détecter les erreurs et qui vous indique, le cas échéant, une méthode de résolution de problème.

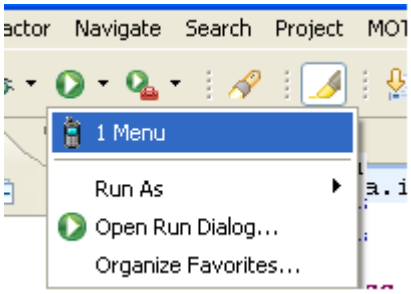
Voici un aperçu :



[I_MOTO08]

La dernière étape consiste à exécuter le code ainsi produit. Le lancement se fait via le bouton « Run » présent dans la barre d'outils de la perspective par défaut [I_MOTO09].

Il permet d'exécuter le programme, soit avec l'émulateur que vous avez choisi lors de la création du projet, soit en spécifiant manuellement les paramètres via la commande « Open Run Dialog… ». Notez que ce menu contextuel est atteignable par « Clic droit → Run as… » sur votre projet.

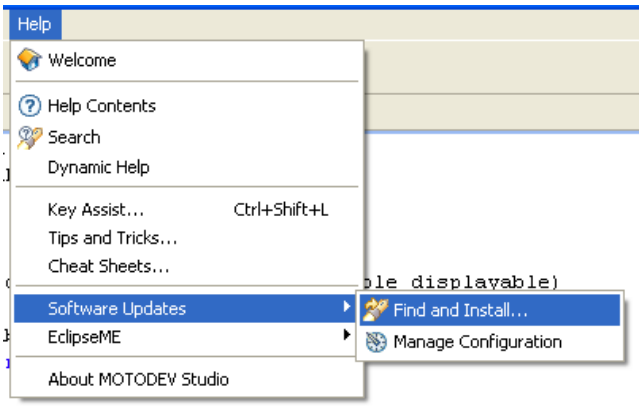


[I_MOTO09]

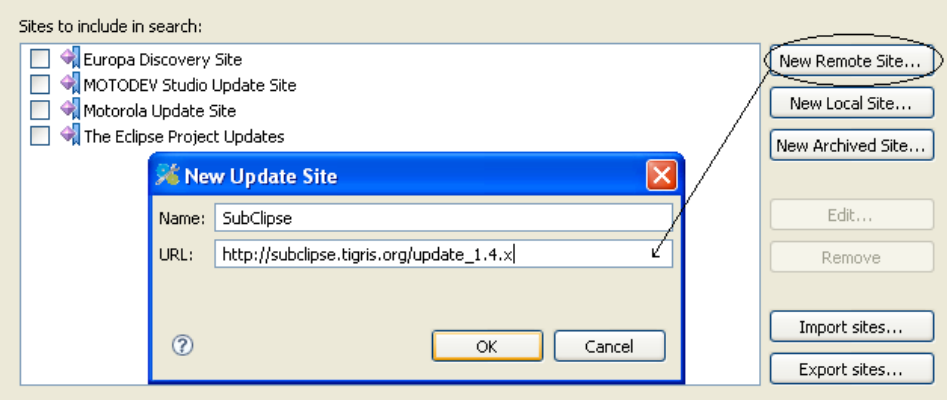
4.1.4 Installation d'un plug-in externe

Comme mentionné au début de ce chapitre, l'environnement Eclipse nous a été utile durant toute la durée du GREP. Afin de sauvegarder notre travail et pouvoir bénéficier d'un suivi de version, nous nous sommes tournés vers le plug-in gratuit Subclipse*. Nous nous servirons de cet outil pour illustrer la procédure d'installation d'un plug-in externe.

Subclipse est fourni par l'Open Source Software Engineering Tools : Tigris, à l'adresse [SUB01] de la bibliographie. Pour l'installer, rendez-vous dans le menu déroulant « Help → Software Updates → Find and install » [I_MOTO10]. Ceci fait, cliquez sur le bouton « New Remote Site » pour désigner un site distant où se trouve le plug-in à installer et entrez l'url [SUB01] [I_MOTO11]. Par défaut, les mises à jour sélectionnables présentes dans la liste ne sont valables que pour la version MotoDev que vous possédez et les plug-ins déjà installés.

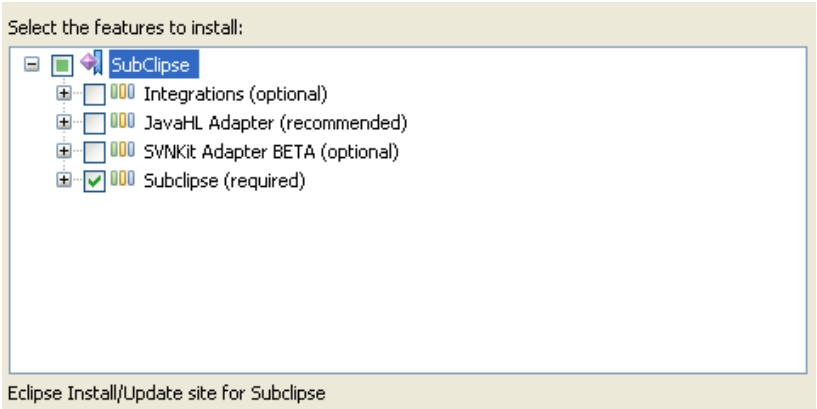


[I_MOTO10]

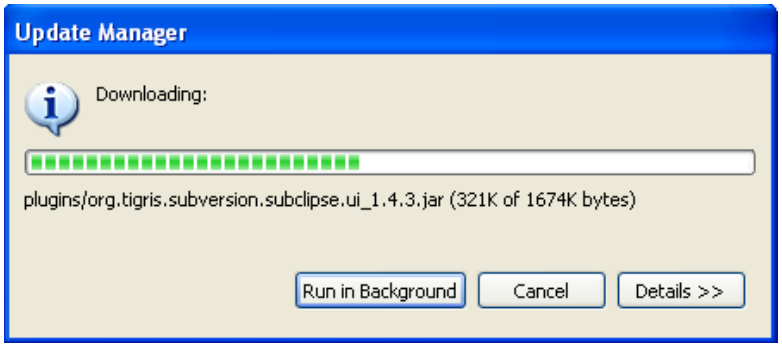


[I_MOTO11]

Cliquez ensuite sur « Next », puis sélectionnez le plug-in qui a été trouvé [I_MOTO12]. Notez que pour terminer l'installation, l'adhésion à un accord de licence est nécessaire. Une fois le plug-in téléchargé [I_MOTO13] et installé, un redémarrage de l'application est nécessaire.



[I_MOTO12]



[I_MOTO13]

5. Java, Bluetooth, GPS et mobilité

Dans ce dernier chapitre, nous allons nous pencher sur la mise en commun des trois technologies java, Bluetooth et GPS au profit de la mobilité. Nous traiterons tout d'abord de ce qui existe comme application sur téléphone portable joignant ces technologies, puis nous verrons une ébauche de projet visant à rendre plus aisée l'utilisation des transports publics dans une grande ville. Entendez par là un service mis à disposition de nouveaux usagers ne connaissant pas forcément l'organisation du réseau et les divers trajets possibles pour se rendre d'un point à un autre.

5.1 Applications existantes

5.1.1 Beacon

Beacon [BEAC01] est une application réalisée par cinq étudiants de la Kent University dans le cadre d'un GREP. Ce GREP possède, dans les grandes lignes, les mêmes contraintes que celui de la HEG aux différences près qu'il est réalisé sur une durée de quatre mois et ne résulte pas de la demande d'un mandant ; c'est l'école qui joue ce rôle.

Ce programme, écrit en Java, consiste en une application pour téléphone portable mettant en relation le Bluetooth, le GPS et la gestion de messages (boîte de réception, temps de validité, anciens messages etc.). Il a été développé avec NetBeans et testé via l'émulateur de Sun : WirelessToolKit.

Les interfaces qui le composent permettent tout d'abord de rechercher un GPS portable, de le coupler au téléphone et de le mémoriser pour une utilisation ultérieure. Puis, une fois le périphérique couplé, il offre la possibilité de mémoriser une ou plusieurs positions géographiques (appelées « Locations ») et de leur associer des messages.

Imaginons que nous voulions enregistrer une liste de courses et l'associer à un magasin donné. Pour ce faire, il faut se rendre dans le menu de création de nouveaux messages, rédiger la liste, lui donner une date limite de validité et l'associer à une position. La position sera donc, dans le cas présent, celle d'un magasin. Au moment où nous entrerons dans le magasin, le message associé sera ajouté à la boîte de réception du téléphone, pour autant que sa date de validité ne soit pas dépassée.

Beacon est donc un bon exemple de ce que peut être le trio Java, Bluetooth, GPS pour la mise à disposition d'un service de mémo géographique mobile.

Voyons maintenant deux applications offrant un service de guidage par GPS.

5.1.2 Nav4All

Disponible gratuitement dans sa version 8.0.6, Nav4All [NAV01] consiste en une application de guidage sur téléphone portable. Via une interface complète, il permet de se guider dans le monde entier comme nous pourrions le faire avec une voiture équipée d'un GPS.

Son installation se fait directement depuis le site web de la société (compatible Wap) et s'adapte donc à quasiment n'importe quel téléphone. L'installation s'effectue de manière transparente pour l'utilisateur et propose l'emploi d'un GPS Bluetooth ou d'un GPS intégré (suivant votre téléphone) pour le guidage. Choisissez votre destination, et par connexion GPRS, l'itinéraire est calculé par les serveurs internet de Nav4all. L'application étant fournie gratuitement, seul le coût de la communication est facturé par votre fournisseur de téléphonie mobile. Une fois l'itinéraire calculé et rapatrié sur le téléphone, la connexion n'est plus nécessaire.

Le guidage est assuré de manière claire via des signaux graphiques et une synthèse vocale (voir les copies d'écrans disponibles sur le site web [NAV02] de la bibliographie) et peut, si vous ne possédez pas de GPS, se faire pas à pas en indiquant au programme à quelle étape précise de l'itinéraire vous en êtes.

5.1.3 Amaze

Amaze assure les mêmes fonctions que Nav4All, et ce toujours gratuitement. Il y ajoute cependant la possibilité de visualiser, via des prises de vues aériennes, les détails d'une position géographique. Notez également la présence de POI – pour Point Of Interest - consistant à informer l'utilisateur des divers lieux ou services intéressants proches de l'endroit où il se trouve et de celui où il souhaite se rendre. Les POI's sont classés par catégories telles que restaurants, hôtels, monuments, services publics etc. Ajoutons enfin qu'un service de prévisions météorologiques est mis à disposition. Ceci, comme le calcul de l'itinéraire, ayant bien évidemment le coût d'une connexion GPRS.

Amaze met à disposition sur son site web [AMA01] la liste des téléphones supportés par la version finale et ceux dont l'application n'est qu'en version Beta, ainsi que plusieurs vidéos de démonstration visant à prendre en main rapidement les capacités du logiciel.

5.1.4 MGMaps

Au contraire des deux autres applications ci-dessus, Mobile GMaps [MGM01] permet uniquement la visualisation de cartes et d'images aériennes sur téléphone mobile soit manuellement en sélectionnant une adresse, soit automatiquement en affichant votre déplacement grâce à un récepteur GPS. MGMaps est gratuit et utilise les banques de photos de Yahoo ! Maps™, Windows Live Local™, Ask.com™ et Open Street Map™. Précisons également que le programme permet de suivre ses déplacements sur une carte via l'inscription gratuite à GMap-Track [GMT01].

5.2 Ébauche de projet

Pour faire suite aux présentations des applications ci-dessus, voici une ébauche de projet de services GPS pouvant être mis à la disposition des usagers de transports publics. Il pourrait offrir aux usagers des informations telles que : horaires - calcul d'itinéraire - temps d'attente - POI, directement sur leur téléphone mobile.

5.2.1 Présentation

Le système consisterait en une application Java qui, couplée ou non à un récepteur GPS, permettrait aux usagers piétons d'avoir facilement accès à toutes les informations nécessaires à un déplacement en ville (ou en périphérie suivant les dessertes offertes par les transports publics).

Afin de mieux comprendre le fonctionnement typique de cette application, voici un exemple concret :

Une personne ne connaissant pas la ville (ou étant très pressée) veut se rendre à un point donné :

- Il lance l'applicatif et entre sa destination.
- Le logiciel fournit alors la position actuelle, l'itinéraire complet, l'arrêt de bus (ou autre) le plus proche, les différentes étapes, une estimation du temps de parcours (et donc de l'heure d'arrivée), le coût du trajet.
- Le temps de parcours se base sur : le temps nécessaire pour se rendre (à pied) à l'arrêt le plus proche permettant d'atteindre la destination, le temps d'attente à cet arrêt ainsi qu'aux arrêts futurs, si le parcours nécessite un changement, le temps normalement effectué par le moyen de transport pour effectuer le trajet (cumulatif s'il y a des changements), et enfin le temps pour se rendre du dernier arrêt à la destination finale.
- Pendant l'utilisation, l'utilisateur peut à tout moment connaître sa position et voir son déplacement sur une carte.

Du point de vue technique, on notera la nécessité de pouvoir se connecter via GPRS à un serveur dédié à la mise à disposition des horaires et des cartes ainsi qu'aux calculs d'itinéraires et à la délimitation de zones autour des arrêts pour permettre le calcul correct du temps de parcours restant (en cours de déplacement).

Comme pour Nav4All du point 5.1.2, il serait bon qu'une fois l'itinéraire calculé la connexion GPRS ne soit plus nécessaire; cela limiterait les coûts d'utilisation.

De plus, des services associés peuvent être inclus comme : la consultation des horaires sans guidage, l'état du trafic et les difficultés que l'utilisateur pourrait rencontrer, la réservation (paiement) du ou des billets nécessaires pour ainsi ne plus avoir à se soucier d'avoir l'appoint en monnaie.

L'affichage de POI prédéfinis et la mémorisation de POI personnels peuvent également être inclus.

5.2.2 Mise en place

Avant de se lancer dans un tel projet, l'entreprise doit connaître la capacité technique des employés qu'elle désignera pour le mener à bien. Si la formation d'un groupe de projet possédant les qualités requises n'est pas possible, alors l'intégrité du projet pourrait en être fortement compromise.

Les étapes suivantes pourraient constituer une démarche à suivre avant et après le lancement du projet.

Tout d'abord, savoir si les utilisateurs (clients) seront au rendez-vous, si le service doit fournir au départ un éventail conséquent de possibilités ou s'il doit se cantonner au

strict minium. En résumé, y a-t-il une attente de la part des usagers (et des non usagers) de voir ce service se mettre en place ?

Afin de répondre à ces questions, l'envoi d'un questionnaire par la poste aux usagers déjà abonnés pourrait être une solution; ajouter à cela un questionnaire en ligne pour les non-abonnés, ce qui serait un bon moyen de connaître les attentes des personnes dans une plus large mesure. Une certaine somme d'argent devant être allouée pour cette démarche, elle est à fixer au départ pour ne pas avoir de surprise quant à son coût.

Des résultats de ces enquêtes, l'équipe de projet devra déterminer s'il est effectivement bon de lancer la réalisation.

Prévoir un panel de beta testeurs, que l'on pourrait motiver en proposant une réduction sur leur abonnement général ou sur la gratuité du nouveau service sur le long terme. Ces personnes auraient pour mission de définir si des aspects, comme le design adopté ou la facilité d'utilisation, sont satisfaisants.

Ce nouveau service s'intègre parfaitement dans la logique actuelle à savoir, réduire la nécessité des personnes à se servir de leur propre véhicule vu le prix des carburants, et les difficultés d'accès au centre ville aux heures de pointe.

Conclusion

Les applications J2ME se sont multipliées depuis l'essor des terminaux mobiles multimédias et sont pour la plupart tournées vers les services aux utilisateurs.

Java a permis cet état de fait, en donnant la possibilité de créer des applicatifs légers et portables ne nécessitant que peu de ressources matérielles. Les constructeurs ont amélioré la manière de créer, d'installer et de se servir de ces programmes pour les rendre plus simples et accessibles à tous. Il faut cependant dissocier la phase de création de la phase d'utilisation.

Nous avons pu constater que la création de telles applications nécessite des connaissances assez poussées. Il ne serait pas judicieux de se lancer dans le codage sans avoir des bases solides sur le langage et une vue générale de ce qu'implique un développement logiciel dans le contexte mobile.

Ajoutez à cela que le code Java développé pour un terminal donné n'est pas forcément totalement compatible avec un autre. Des différences subsistent et peuvent amener à une utilisation impossible sur des mobiles de diverses marques. Pour y remédier en partie, certains constructeurs, dont par exemple Motorola avec son IDE MotoDev Studio présenté dans ce document, permettent de tester nos créations en émulant la quasi-totalité des appareils produits. On peut ainsi se rendre compte facilement que des problèmes de compatibilité peuvent survenir même sur des terminaux de marque identique.

De manière générale, nous avons réuni les informations liées aux technologies Bluetooth et GPS pour en comprendre le fonctionnement et avons présenté le code Java requis pour les mettre en œuvre.

L'ébauche de projet de la dernière partie présente une idée d'utilisation conjointe des technologies Java – Bluetooth – GPS pour fournir un service qui n'existe pas encore et dont les caractéristiques coïncident avec la tendance actuelle d'une mobilité facilitée et moins onéreuse.

Glossaire

A

API : Application Programming Interface. C'est une interface qui permet de définir de quelle manière un composant peut communiquer avec un autre. Elle correspond à une bibliothèque de fonctions de bas niveau destinées à être réutilisées dans diverses applications afin de réaliser des traitements de haut niveau.

ASCII : American Standard Code for Information Interchange. C'est une norme d'échange des fichiers informatiques de type texte simple codée sur 256 caractères (de A à Z, de a à z, ainsi que la ponctuation, les chiffres etc..).

B

Baud : Désigne la mesure du nombre de symboles transmis par seconde par un signal modulé dans le domaine des télécommunications.

C

Checksum : C'est la somme de tous les bits d'un message, effectuée dans le but de détecter des erreurs de transmission. Si la somme calculée après réception est différente, c'est qu'il y a eu une erreur de transfert.

Cibi : Terme francisé qui correspond au nom anglais « citizen band » ou CB. Il s'agit d'une bande de fréquence allouée au trafic radio et ouverte à tous.

F

Firmware : En francais « micrologiciel » désignant donc le logiciel interne qui est intégré dans un composant matériel.

FireWire : Nom commercial de l'ILINK d'Apple qui définit une norme de bus fonctionnant en mode série. Il permet le branchement de périphériques externes et supporte le plug-and-play (débit de 800 Mbits/seconde).

G

GPRS : Global Packet Radio Service. C'est une évolution du standard de téléphonie mobile GSM qui permet des transferts de données par paquets, comme sur l'internet. On l'utilise pour transmettre des photos, des vidéos et des fichiers audio via un téléphone portable.

H

Hexadécimal : Désigne un système de numérotation en base 16 qui utilise les chiffres arabes pour les dix premiers chiffres (0 à 9) et les lettres de A à F pour les six suivants. Exemple : A = 10.

Horloge atomique : C'est actuellement l'horloge la plus précise au monde qui utilise la « fréquence du rayonnement émis lors de la transition atomique entre deux niveaux d'énergie particuliers de l'atome de Césium 133 » [HORL01].

I

IEEE 802.15.1 : L'Institute of Electrical and Electronics Engineers est une organisation regroupant un grand nombre d'ingénieurs, d'électriciens, d'informaticiens et de professionnels du domaine des télécommunications. Elle a notamment pour but d'établir des normes dans ce domaine. La 802.25.1, adaptée aux spécifications sans fil Bluetooth, en est une.

Instanciation : Action de création d'une instance de classe, c'est-à-dire d'un objet.

IrDA : Infrared Data Association. Les rayons infrarouges sont ici utilisés pour le transfert de données à courte distance entre deux appareils.

J

JSR : Java Specification Request. Requête de modification / amélioration du langage et / ou d'une bibliothèque.

Jumper : Dans le contexte de la programmation il s'agit de sauter d'un point du code pour aller directement à un autre.

P

Parité : Dans le contexte électronique cela s'applique à un bit de contrôle de la qualité d'une transmission. Il est obtenu par l'analyse d'un caractère, et peut être pair ou impair selon le nombre de bits à 1 présents dans le code du caractère.

R

RAM : Random Access Memory ou mémoire vive en Français. Elle permet de stocker un programme en cours d'exécution et ses données.

Routeur : Généralement matériel, il sert à diriger les données à travers plusieurs réseaux. Dans le contexte présent il s'agit de dispatcher, de façon logicielle, les données entre plusieurs appareils.

Runtime Java : Ensemble d'éléments logiciels nécessaires à l'exécution d'un programme Java. Cet ensemble comprend une machine virtuelle Java et les classes de la bibliothèque de base.

S

Subclipse : Outil de versionning pour projet. Il permet la sauvegarde de fichiers sur un repository en ligne en incluant à chaque modification d'un fichier un numéro de version et le nom de la personne qui l'a modifié. Des actions, comme le retour à une version ultérieure ou la comparaison entre deux versions, font partie de ses nombreuses possibilités.

Switch : Généralement matériel, il désigne ici la capacité de diriger les données de manière optimale entre plusieurs appareils. Contrairement au hub, le switch permet de ne transmettre le trafic réseau qu'entre les ports impliqués dans une communication.

T

Temps processeur : c'est la somme des intervalles de temps pendant lesquels le système d'exploitation donne accès aux CPU pour permettre aux processus de s'exécuter.

Trame : Il s'agit d'une unité d'information transportée sur un réseau. Elle est constituée d'une série de bits de taille variable.

U

Upload : Désigne la capacité d'envoyer des informations d'un nœud (machine connectée à un réseau) vers un autre.

URL : Uniform Resource locator. Symbolise l'adresse d'un hôte sur un réseau ou plus couramment l'adresse internet unique d'une page web.

USB : Universal Serial Bus. Norme de bus fonctionnant en mode série, et permettant la connexion de périphériques externes, compatible plug-and-play. Exemple une clé USB, un scanner, une webcam etc. Dans sa version 2.0, l'USB permet des débits de l'ordre de 480 Mbits/sec.

W

Wifi : Wireless Fidelity. Réseau sans fil de type ethernet qui permet d'atteindre actuellement des débits de l'ordre de 240 Mibts/sec maximum théoriques.

Bibliographie

[AMA01] Amaze – Guidage par GPS et services associés sur téléphone portable http://www.amazegps.com/index.php?page=home&language=fr

[BEAC01] Kent University – Beacon: A context aware Messaging Application http://www.cs.kent.ac.uk/pubs/ug/2006/co600-projects/soggycomp/report.pdf

[BLTH01] Fujitsu – Extrait de « Strategy for business » numéro 7, été 2001 http://www.fujitsu.com/ca/fr/news/publications/articles/bluetooth.html

[BLTH02] The Bluetooth Special Interest Group – About SIG http://www.bluetooth.com/Bluetooth/SIG/

[BLTH03] Sun Microsystems – J2ME – JSR-82 Bluetooth API overview http://java.sun.com/javame/reference/apis/jsr082/

[BLTH04] Oscar Vivall – Sony Ericsson – Developer Support https://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_java_gps_receiver_bluetooth.jsp

[CLAS01] Sun Microsystems – « VM Spec, the class file format » http://java.sun.com/docs/books/jvms/second_edition/html/ClassFile.doc.html

[CLCD01] The Java Community Process – Java Specification Request http://jcp.org/en/jsr/detail?id=30

[DYN01] TECH-FRESH / The first cell phone Motorola DynaTac 8000x http://cellphones.techfresh.net/cell-phones/motorola/motorola-dynatac-8000x-the-first-portable-cellphone/

[ECLI01] IDE Eclipse for Java / Page de téléchargement http://www.eclipse.org/downloads/

[FHSS01] Frequency Hopping Spread Spectrum (FHSS) – Wikipedia http://en.wikipedia.org/wiki/Frequency-hopping_spread_spectrum

[GAR05] Market Focus - Bluetooth in mobile devices, 2004-2009 http://www.gartner.com/DisplayDocument?ref=g_search&id=486634

[GAR07] IT-SCAN / ABE-BAO Agence Bruxellois pour l'emploi http://www.itscan.be/fr/articles/tag/Gartner

[GMT01] GMap-Track BETA – Track your position on Google Maps™ http://www.gmap-track.com/register.php

[GPS01] Wikipedia / Récepteur GPS : temps d'acquisition à froid, chaud http://fr.wikipedia.org/wiki/R%C3%A9cepteur_GPS#D.C3.A9marrage_.C3.A0_froid

[HORL01] Ministère de l'enseignement supérieur et de la recherche / Galileo http://www.sciences.gouv.fr/index.php?qcms=article,view,2849,archives,159,3,,,,

[JSR30] Java Community Process / CLDC – The JSR 30 Final Release http://jcp.org/aboutJava/communityprocess/final/jsr030/index.html

[JSR37] Java Community Process / MIDP – The JSR 37 Final Release http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html

[JSR82] Java Community Process / Bluetooth API – The JSR 82 Final Release http://jcp.org/aboutJava/communityprocess/final/jsr082/index.html

[JSR118] Java Community Process / Mobile Information Device Profile 2 http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html

[MGM01] MGMaps – View maps from various sources on your mobile phone http://www.mgmaps.com/download.php

[MAT01] MathFP for KVM – Fixed Point Integer Math http://home.comcast.net/~ohommes/MathFP/

[MIDP01] Sun Microsystems – Mobile Information Device Profil http://java.sun.com/products/midp/overview.html

[MOTO01] Motorola – The Motorola Developper Network http://developer.motorola.com/docstools/motodevstudio/– Introducing MotoDev

[NMEA01] Remember GPS - The NMEA-0183 protocol details http://www.remember.ro/dl/nmea0183.pdf

[NAV01] Nav4All Install – Navigation sur téléphone mobile dans le monde entier http://www.nav4all.com/site2/www.nav4all.com/fre/index_install.php

[NAV02] Nav4All Visuels – Navigation sur téléphone mobile dans le monde entier http://www.nav4all.com/site2/www.nav4all.com/fre/index.php

[PAN01] About.Com Wireless Network – Personal Area Network http://compnetworking.about.com/od/networkdesign/g/bldef_pan.htm

[RUP01] Summation Generator – Rainer Rueppel 1985 - Cryptography http://en.wikipedia.org/wiki/Summation_generator

[RUP02] Hermelin Research Index – Cryptographic properties of Bluetooth http://citeseer.ist.psu.edu/271733.html

[SIRF01] The SiRF Technology, Inc. North America – Home Page http://www.sirf.com/Default.aspx

[SUB01] Tigris Open Source Software Engineering Tools – Subclipse 1.4 http://subclipse.tigris.org/update_1.4.x

[THRD01] Sun Microsystems – Java 2 Plateform – Threads http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Thread.html

[THRD02] Eric Larcher – Site web personnel – Chapitre VII : les Threads http://www.larcher.com/eric/guides/javactivex/VII.htm