

# Déploiement d'applications Java ME

Master MATIS – Management and Technology of Information Systems  
Master en Technologie des Systèmes d'Information

**Hikari WATANABE & Dejan MUNJIN**, Juin 2007

Département des Systèmes d'Information  
Centre Universitaire d'Informatique  
Faculté des Sciences Économiques et Sociales  
**Université de Genève**

Institut d'Informatique et Mathématiques Appliquées  
**Université Joseph Fourier de Grenoble**

Directeur de recherche : Pr. Dimitri KONSTANTAS

## **Remerciements**

Les contributions majeures de notre responsable de projet M. Michel DERIAZ nous ont permis d'avancer dans une forêt de langages et de technologies qui commencent seulement à s'implanter de façon durable et globale.

Le soutien de notre directeur de recherche Pr. Dimitri KONSTANTAS.

Un grand merci à nos familles respectives, aux amis et toutes les personnes ayant apporté leurs encouragements durant cette formation Européenne de IIIe cycle.

# Table des matières

|   |    |
|---|----|
| <b>1. Introduction</b> .....                                | 3  |
| 1.1 Problématique.....                                      | 3  |
| 1.2 Contributions.....                                      | 4  |
| 1.3 Plan.....   | 5  |
| 1.4 Liste des acronymes.....                                | 6  |
| <b>2. État de l'art</b> .....                               | 7  |
| 2.1 Les technologies mobiles.....                           | 7  |
| 2.1.1 Java ME.....  | 7  |
| 2.1.2 Microsoft Windows Mobile.....                         | 7  |
| 2.1.3 Symbian.....  | 8  |
| 2.1.4 RIM et Palm.....                                      | 8  |
| 2.1.5 Le marché du PDA/smartphone.....                      | 8  |
| 2.2 Les différents composants de Java ME.....               | 9  |
| 2.2.1 Les spécifications pour les composants logiciels..... | 9  |
| 2.2.2 « Java Technology for the Wireless Industry ».....    | 10 |
| 2.2.3 « Connected Limited Device Configuration ».....       | 12 |
| 2.2.4 « Mobile Information Device Profile ».....            | 13 |
| 2.2.5 « Wireless Messaging API ».....                       | 15 |
| 2.2.6 « Mobile Media API ».....                             | 15 |
| 2.2.7 Synthèse des spécifications.....                      | 15 |
| 2.3 Les méthodes de déploiement.....                        | 16 |
| 2.3.1 Le déploiement par Bluetooth et IrDA.....             | 16 |
| 2.3.2 Le déploiement par Wireless LAN.....                  | 17 |
| 2.3.3 Le déploiement OTA.....                               | 17 |
| <b>3. Déploiement d'applications mobiles</b> .....          | 19 |
| 3.1 L'implémentation des spécifications Java ME.....        | 19 |
| 3.2 FoxyTest.....   | 20 |
| 3.3 Développement de l'application.....                     | 22 |
| 3.4 Structure de l'application.....                         | 23 |
| 3.4.1 FoxyTest, SplashCanvas, I18n.....                     | 24 |
| 3.4.2 TestConfig, TestLocation, VectorHolder.....           | 24 |
| 3.4.3 HttpProcess.....                                      | 26 |
| 3.5 Réalisation du serveur de données.....                  | 27 |
| 3.5.1 Structure de la base de données.....                  | 27 |
| 3.6 Description de la servlet.....                          | 29 |
| 3.7 Statistique des rapports.....                           | 31 |
| <b>4. Vers un cas pratique</b> .....                        | 33 |
| 4.1 Présentation de FoxyTag.....                            | 33 |
| 4.2 La portabilité théorique et pratique.....               | 34 |
| 4.3 Le temps de développement des versions.....             | 34 |
| 4.3.1 Spécificités des interfaces de programmation.....     | 35 |
| 4.3.2 Modélisation du comportement d'un GPS.....            | 36 |

|            |   |           |
|------------|---|-----------|
| 4.3.3      | Modélisation et réalisation du composant « GPSIntegrated ».....                 | 38        |
| 4.3.4      | Modélisation et réalisation du composant « GPSBluetooth ».....                  | 41        |
| 4.4        | Réutilisation des composants.....   | 49        |
| 4.5        | Méthode de développement des composants retenue.....                            | 49        |
| <b>5.</b>  | <b>Les modèles de distribution du contenu sur les téléphones portables.....</b> | <b>50</b> |
| 5.1        | Modèle de référence pour la distribution des MIDlets.....                       | 50        |
| 5.2        | Déploiement des MIDlets par les réseaux sans fil.....                           | 51        |
| 5.3        | Les besoins de prétests pour le déploiement des MIDlets.....                    | 52        |
| 5.4        | Caractéristiques et regroupement des principaux portails de Java ME.....        | 53        |
| 5.4.1      | Les plateformes de développement collaboratif.....                              | 54        |
| 5.4.2      | Les plateformes de distribution commerciales des MIDlets.....                   | 54        |
| 5.4.3      | Les plateformes utilitaires.....  | 55        |
| 5.4.4      | Les plateformes de prétests des applications Java ME.....                       | 55        |
| 5.5        | Comparaison des plateformes existantes avec le besoin des prétests.....         | 56        |
| <b>6.</b>  | <b>Plateforme collaborative.....</b>  | <b>58</b> |
| 6.1        | Caractéristiques et besoins.....  | 58        |
| 6.2        | Définition des interfaces.....  | 59        |
| 6.3        | Développement du prototype.....   | 60        |
| <b>7.</b>  | <b>Travaux apparentés.....</b>  | <b>61</b> |
| 7.1        | Test de l'environnement d'un téléphone mobile.....                              | 61        |
| 7.2        | Plateforme collaborative.....   | 61        |
| <b>8.</b>  | <b>Évolutions futures.....</b>  | <b>62</b> |
| <b>9.</b>  | <b>Conclusion.....</b>  | <b>63</b> |
| <b>10.</b> | <b>Références.....</b>  | <b>64</b> |
| <b>11.</b> | <b>Figures.....</b>   | <b>65</b> |
| <b>12.</b> | <b>Bibliographie.....</b>   | <b>66</b> |
| <b>13.</b> | <b>Annexes.....</b>   | <b>67</b> |
| 13.1       | Impression d'écran de FoxyTest.....   | 67        |
| 13.2       | NetBeans 5.5 et l'émulateur WTK 2.5 CLDC.....                                   | 70        |
| 13.3       | Schéma de la base de données pour la plateforme de test.....                    | 71        |
| 13.4       | Aperçu de la plateforme collaborative.....                                      | 72        |
| 13.4.1     | Gestion de projet et des versions.....  | 72        |
| 13.4.2     | Évaluations.....  | 73        |
| 13.4.3     | Rapport de test après téléchargement.....                                       | 73        |
| 13.4.4     | Résultats des tests par version.....  | 74        |
| 13.4.5     | Liste des meilleurs testeurs.....   | 75        |

# 1. Introduction

## 1.1 Problématique

La situation actuelle des téléphones mobiles résulte d'une évolution rapide des besoins de ses utilisateurs, couplés à la disponibilité technologique nécessaire dans ce domaine permettant ces avancées. La mobilité des personnes physiques a augmenté, ce qui a demandé une adaptation de la disponibilité des données personnelles, ou communes à un groupe d'utilisateur. Dans ce cadre, la simple fonction vocale d'un téléphone mobile ne suffisait plus, l'échange de données est devenu indispensable.

L'évolution n'est pas des moindres, elle implique la capacité du téléphone mobile à gérer de nouvelles fonctionnalités. Le standard qui s'est imposé pour exécuter ces applications est le Java ME, qui possède l'avantage d'être multiplateforme. Mais il pose certains problèmes d'incompatibilités, notamment dues à la disparité du support des API optionnelles.

Pour employer au mieux ces téléphones mobiles, il faut les agrémenter des applications tierces, permettant d'exploiter au maximum leurs possibilités. Dans le cadre d'un représentant ou d'un consultant en déplacement voulant obtenir une information résidant dans le système d'information de son entreprise, il doit avoir à sa disposition les applications lui permettant de les obtenir. Un autre exemple plus proche est l'utilisateur privé, qui, voulant acheter un jeu, une application de prévisions météo, ou de résultat sportif voudra l'obtenir de la façon la plus simple qui soit. Le déploiement de ces applications nécessite une infrastructure performante et un développement capable de gérer l'hétérogénéité des plateformes mobiles rencontrées. La difficulté réside dans la mise à disposition d'une application pour un grand nombre de téléphones mobiles différents.

## 1.2 Contributions

Le projet de master consiste à réaliser un environnement qui permet de déployer des applications codées en Java ME, sur des téléphones mobiles hétérogènes. Le choix s'est porté sur ce langage de développement, car il gère le multiplateforme grâce à une machine virtuelle présente sur un grand nombre de téléphones mobiles. Notre travail s'est porté sur la création d'une application permettant de définir la compatibilité du client avec les besoins d'une application, la participation active dans un projet de développement d'une application mobile FoxyTag, et l'évaluation d'une plateforme de distribution de ces applications.

Ces étapes de travail permettent d'établir une méthode de développement et effectuer le déploiement des applications Java sur les téléphones mobiles de manière efficace.

### 1.3 Plan

Dans le chapitre deux, nous allons décrire l'état de l'art des technologies impliquées dans le déploiement d'applications mobiles, avec un aperçu détaillé des API et composants utilisés par Java ME dans le cadre des téléphones mobiles limités en ressources. La citation et l'explication des possibilités de méthodes de déploiement sont suivies par le chapitre trois qui présente une solution permettant de faciliter ce déploiement grâce à la programmation et au développement de l'application FoxyTest.

Le chapitre quatre présente le cas de FoxyTag et plus particulièrement la librairie Bluetooth et GPS qui le composent, pour accélérer le développement d'application mobile. Le chapitre cinq propose une description des modèles de distributions actuels des applications orientées mobiles.

Enfin, le chapitre six développe le thème de la plateforme web qui permet de communiquer sur un projet en vue de l'améliorer par un système de versions et la critique des membres de la communauté. Les évolutions futures du chapitre sept permettent d'avoir un aperçu des adaptations du modèle de test proposé par FoxyTest, et les améliorations pouvant être intégrées à la plateforme web.

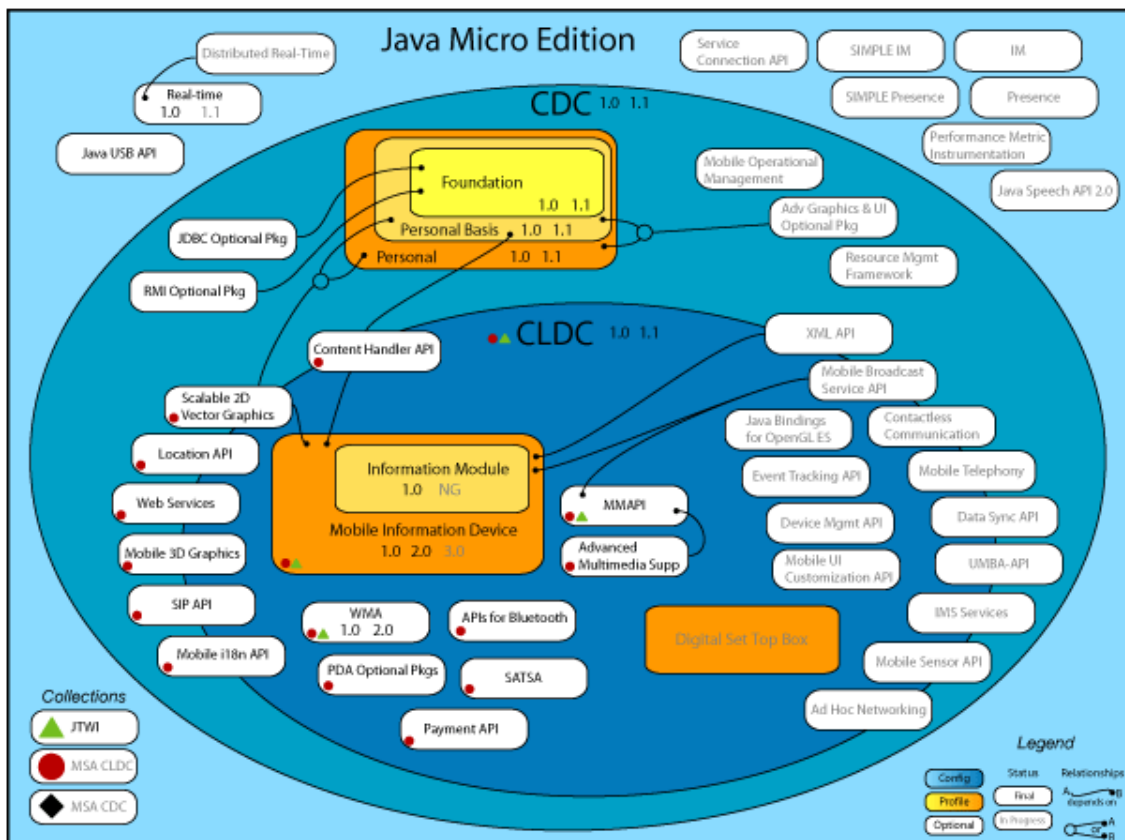


Figure 1.1 : Les composants Java ME  
(Source, Sun Microsystems Java ME)

## 1.4 Liste des acronymes

|       |   |
|-------|---|
| API   | Application Programming Interface                 |
| BTS   | Base Transceiver Station                          |
| CDC   | Connected Device Configuration                    |
| CLDC  | Connected Limited Device Configuration            |
| DBMS  | Database Management System                        |
| EDGE  | Enhanced Data Rates for GSM Evolution             |
| GPRS  | Global Packet Radio Service                       |
| GPS   | Global Positioning System                         |
| GUI   | Graphical User Interface                          |
| HSDPA | High Speed Downlink Packet Access                 |
| HSOPA | High Speed OFDM Packet Access                     |
| HSUPA | High Speed Uplink Packet Access                   |
| HTML  | Hyper Text Markup Language                        |
| HTTP  | Hypertext Transfer Protocol                       |
| IDE   | Integrated Development Environment                |
| IEEE  | Institute of Electrical and Electronics Engineers |
| IETF  | Internet Engineering Task Force                   |
| JAD   | Java Application Descriptor                       |
| JAR   | Java Archive                                      |
| JDBC  | Java Database Connectivity                        |
| JDE   | Java Development Environment                      |
| JCP   | Java Community Process                            |
| JSP   | Java Server Pages                                 |
| JSR   | Java Specification Request                        |
| JTWI  | Java To Wireless Industry                         |
| JUG   | Java User Group                                   |
| L2CAP | Logical Link Control & Adaptation Protocol        |
| LAN   | Local Area Network                                |
| MIDP  | Mobile Information Device Profile                 |
| OBEX  | Object Exchange                                   |
| OFDM  | Orthogonal Frequency Division Multiplexing        |
| PDA   | Personal Digital Assistant                        |
| PHP   | PHP: Hypertext Preprocessor (Personal Home Page)  |
| PKI   | Public Key Infrastructure                         |
| RFC   | Request For Comments                              |
| SQL   | Structured Query Language                         |
| SSL   | Secure Socket Layer                               |
| UIQ   | User Interface Quartz                             |
| UMTS  | Universal Mobile Telecommunications System        |
| URI   | Uniform Resource Identifier                       |
| URL   | Uniform Resource Locator                          |
| WAR   | Web Archive                                       |
| WLAN  | Wireless LAN                                      |

## 2. État de l'art

Les technologies mobiles sont un domaine où les évolutions sont nombreuses et rapide, il est donc difficile de les figer sachant que certaines sont en perte de vitesse et d'autres en plein essor. Cet état de l'art recense donc certains systèmes qui sont voués à évoluer rapidement, ou d'autres qui vont disparaître.

### 2.1 Les technologies mobiles

Les technologies mobiles reposent sur un système d'exploitation souvent propriétaire, sur lequel vont être installées les applications de base. Une machine virtuelle peut également être présente pour permettre l'exécution d'application basée sur cette dernière. Ci-après, nous allons décrire les spécificités des différents systèmes disponibles sur le marché.

#### 2.1.1 Java ME

Java Micro Edition ou Java ME, est une plateforme d'application très présente au niveau des dispositifs mobiles à travers le monde. Basée sur une machine virtuelle, elle fournit un environnement robuste et flexible pour des applications fonctionnant sur une large gamme d'autres dispositifs tel que des téléphones portables, PDA, des boîtiers multifonctions de réception TV, et des imprimeurs. La plateforme Java ME inclut des interfaces utilisateur flexibles, un modèle de sécurité avancé et une large gamme des protocoles de réseau. Java ME repose sur une machine virtuelle présente aujourd'hui dans beaucoup de téléphones mobiles, ce qui le rend populaire. Il présente l'avantage d'être peu gourmand en termes de consommation d'énergie, de capacité mémoire et de puissance de calcul. La majorité des applications, jeux et autres à destination du marché des téléphones mobiles sont développés en Java ME.

#### 2.1.2 Microsoft Windows Mobile

Le système d'exploitation pour PDA et Smartphone de Microsoft en est aujourd'hui à sa sixième version, nom de code Crossbow. De plus en plus de fabricants de téléphones mobiles adoptent ce système, car il possède l'avantage de pouvoir être facilement synchronisé avec la suite bureautique Office de Microsoft, et dispose d'un environnement de développement connu par les développeurs. Ce système requiert néanmoins des ressources qui ne peuvent être fournies par un simple téléphone mobile, il est donc orienté vers les smartphones possédant les capacités pour l'exécuter. La machine virtuelle Java implémentée dans les smartphones équipés de Microsoft Windows Mobile diffère selon le constructeur, il est également possible d'installer une autre machine virtuelle par exemple la Java J9 d'IBM disponible dans sa suite WebSphere Studio.

### 2.1.3 Symbian

Symbian est à l'origine un consortium composé de Psion, Motorola, Sony Ericsson, Nokia et certains autres grands fabricants dans le domaine de la téléphonie mobile et des réseaux de communication. Depuis l'acquisition par Nokia des parts de Psion, ce dernier a pris la majorité du consortium. L'équilibre n'étant plus, l'intérêt des autres constructeurs a diminué, car les décisions sont prises à l'avantage de Nokia. Sony Ericsson maintient sa ligne de smartphones avec ce système d'exploitation.

Concernant la structure du système, Symbian se découpe en deux parties. D'un côté le système d'exploitation Symbian OS (version 9.2) avec son noyau en version EKA2 et une interface utilisateur de référence, et de l'autre l'interface utilisateur modifiée et personnalisée selon les besoins du constructeur. Les interfaces utilisateur sont les suivants, Series 60, 80, 90 utilisé et produit par Nokia, UIQ utilisé et produit par Sony Ericsson, et FOMA utilisé par le Japon. Symbian supporte également les spécifications Java ME de base, avec des API optionnelles et spécifiques.

### 2.1.4 RIM et Palm

Research in Motion ou RIM, la société qui propose le BlackBerry, possède un système propriétaire et propose le « BlackBerry JDE » pour développer des applications Java ME sur leur système qui implémente le CLDC et MIDP. Le BlackBerry requiert une licence d'utilisation pour l'accès au serveur de messagerie électronique, et ce type de licence n'est fourni qu'à des professionnels. L'utilisation par des clients privés n'est donc pas possible, cela limite le BlackBerry à un environnement professionnel.

Palm propose via son entreprise PalmSource Inc. le système d'exploitation PalmOS, qui est utilisé sur les smartphones de type Tréo, qui peuvent également exécuter des applications Java ME. Certains modèles de Palm sont également équipés du système d'exploitation Microsoft Windows Mobile.

### 2.1.5 Le marché du PDA/Smartphone<sup>1</sup>

Selon le cabinet d'étude Gartner, plus de 17 millions de PDA ont été vendus en 2006, ce qui représente une progression de près de 20% par rapport à 2005. Les smartphones ont représenté en 2006 60% des ventes de PDA contre 47% en 2005, ce qui indique l'intérêt très marqué de la préférence de la fusion des fonctionnalités d'un téléphone mobile et d'un PDA. En effet, il est plus intéressant d'avoir un seul appareil pouvant effectuer les tâches d'un PDA et d'un téléphone.

RIM a écoulé près de 3.5 millions d'unités en 2006, soit une progression de 10% par rapport à 2005, et sa part de marché est d'environ 20%. Pour la même période et le même marché, Palm a vu sa part diminuer de 18% à 11%.

Concernant les systèmes d'exploitation, Windows Mobile est en tête sur le marché des PDA avec 56% de part de marché en 2006, une progression de près de 40%. Ce système est employé sur pratiquement 10 millions de PDA.

---

<sup>1</sup> 2007 Gartner, Inc., <http://www.gartner.com>

RIM se place en deuxième position avec 20% de part de marché, et PalmOS et Symbian sont respectivement trois et quatrième.

## 2.2 Les différents composants de « Java Micro Edition »

La première étape de déploiement des applications Java sur les téléphones mobiles est l'étude de faisabilité. Sachant que les ressources sont très limitées sur les téléphones mobiles, la question est de savoir de quelles ressources de programmation Java on peut disposer.

Une solution est d'étudier un modèle de téléphone particulier et développer une application pour ce modèle. Dans ce cas, il est évident que le simple fait d'avoir utilisé le langage Java ne peut garantir aucune portabilité. La taille d'écran d'un appareil, la mémoire disponible, la disponibilité des interfaces de programmation font que ce déploiement reste limité sur ce modèle de téléphone. L'avantage de faire le développement et le déploiement des applications Java de cette manière est leur robustesse et l'exploitation optimisée des ressources.

La deuxième méthode est de savoir s'il existe les interfaces de programmation qui sont implémentées sur tous ou le plus grand nombre des appareils. En utilisant uniquement cette partie commune, la possibilité d'exploiter les ressources de manière optimale est réduite, mais la portabilité est nettement accrue.

Nous avons étudié cette deuxième méthode plus en détail pour essayer de déterminer quelles sont les possibilités de déployer une application sur le plus grand nombre des appareils afin de minimiser les efforts de programmation. Pour le faire, nous allons brièvement recenser les points importants des spécifications des composants disponibles en « Java Micro Edition ». Ça nous permettra d'avoir une vue globale des possibilités de développement d'une part et de définir les limites de portabilité des applications d'autre part.

### 2.2.1 Les spécifications pour les composants logiciels

Les applications Java pour l'utilisation sur les appareils ayant une capacité mémoire et la capacité de calcul limité sont appelées MIDlets. Une MIDlet est proche d'une applet java. Chaque MIDlet doit étendre la classe MIDlet et réécrire les méthodes suivantes :

```
startApp()  
pauseApp()  
destroyApp(boolean b)
```

Ces méthodes sont ensuite appelées par le gestionnaire des applications pour gérer les états de la MIDlet. Le diagramme à état sur la figure 1.2 montre les états et le cycle de vie d'une MIDlet géré par le gestionnaire d'applications.

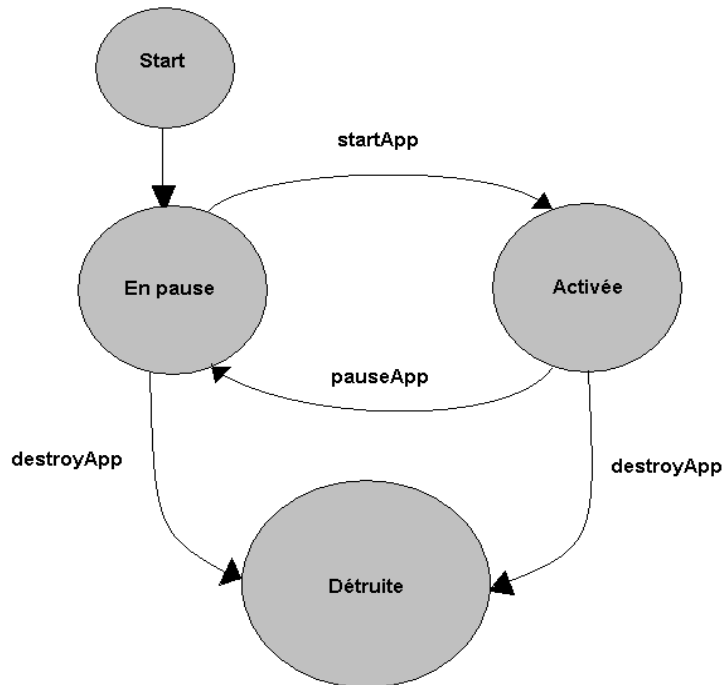


Figure 2.1 : Diagramme d'état du cycle de vie d'une MIDlet

Les MIDlets sont exécutées sur la plateforme « Java Micro Edition » et pour assurer leur portabilité, l'entreprise Sun Microsystems a défini certaines spécifications au sein du « Java Community Process » [2.1]. Java Community Process ou JCP est constitué d'un groupe d'experts issus des entreprises leaders du marché. Ainsi pour assurer la portabilité et l'interopérabilité entre les applications Java sur les appareils avec les ressources limitées (comme les téléphones mobiles, smartphones et assistants digitaux) JCP définit les spécifications qui doivent être respectées à l'implémentation de la plateforme Java ME. Ces spécifications sont appelées JSR ou « Java Specification Request ». Elles sont numérotées et la spécification clé pour Java ME définit par le groupe d'experts porte le numéro JSR-185 et elle est nommée aussi « Java Technology for the Wireless Industry » ou JTWI [2.2].

La prochaine partie du travail est consacrée à l'étude des spécifications. L'intention est de souligner les parties obligatoires des spécifications, afin de pouvoir construire les applications portables entre les différents appareils mobiles.

### 2.2.2 « Java Technology for the Wireless Industry »

La spécification JTWI vise à minimiser la fragmentation des API sur le marché des appareils mobiles et de livrer une spécification claire pour les fabricants des appareils mobiles, opérateurs et les développeurs des applications. JTWI définit les éléments nécessaires qu'un appareil doit respecter si la plateforme Java est utilisée. Elle définit aussi les éléments optionnels, les recommandations et les bonnes pratiques à implémenter.

Cette spécification générale porte donc sur la compatibilité des applications avec les ressources d'un appareil et les versions d'implémentations.

La première des choses à distinguer est la définition de la pile des logiciels. JTWI vise à minimiser et à standardiser la fragmentation des APIs en définissant la pile des logiciels. La pile des logiciels pour un téléphone mobile est définie sur la figure 1.

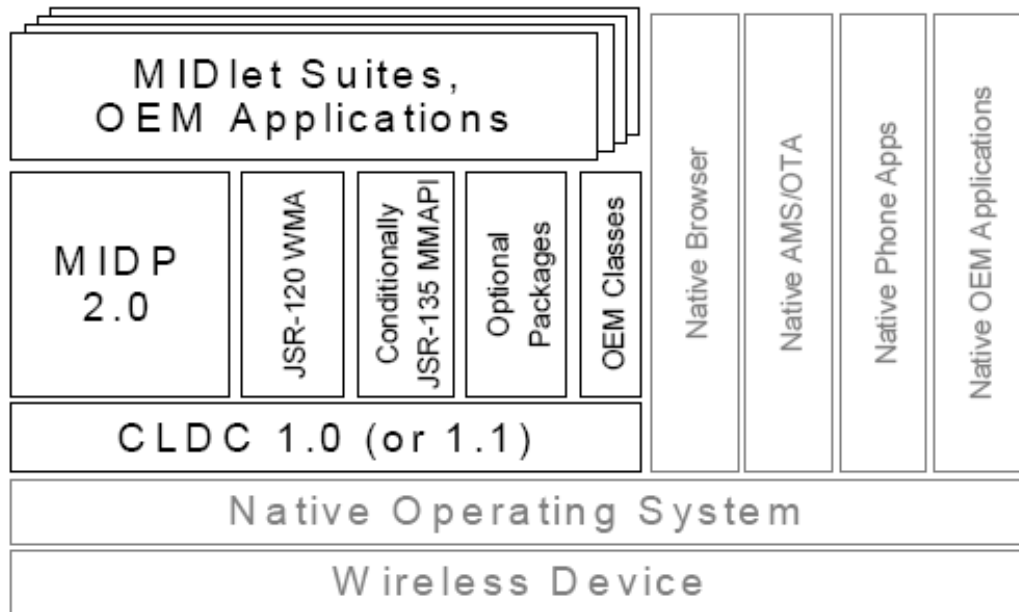


Figure 2.2 : Les composants logiciels du téléphone mobile  
(Source : JSR-185 « Sun Microsystems »)

Comme on peut voir sur la figure 2.2, certains composants sont obligatoires, certains sont conditionnels et certains optionnels. L'architecture de chaque composant est définie dans une spécification de manière détaillée.

La découverte des composants des interfaces de programmation disponibles est prévue à l'aide de la classe système. L'exemple suivant montre son utilisation:

```
class java.lang.System:
public final static String getProperty(String key)
```

Dans ce cas les clés (`String key`) sont définies dans la spécification et représentent les noms des interfaces de programmation disponibles sur le téléphone.

Nous citons ici les clés des API que nous avons utilisées pendant le développement et les tests:

| Clé  | Description   |
|--|---|
| <code>microedition.configuration</code>    | La configuration J2ME supportée par exemple « CLDC-1.1 »  |
| <code>microedition.profiles</code>         | Pour les appareils MIDP-2.0 cette propriété doit être « MIDP-2.0 »  |
| <code>microedition.platform</code>         | Le nom de la plateforme ou d'appareil par exemple « SonyEricssonS700i/4.4.2 »   |
| <code>microedition.location.version</code> | Le paquetage optionnel. S'il est supporté la valeur retournée est la version. Par exemple « 1.0 » sinon la valeur doit être « NULL »                                  |
| <code>bluetooth.api.version</code>         | Le paquetage optionnel. S'il est supporté la valeur retournée est la version. Par exemple « 1.1 » sinon la valeur doit être « NULL »                                  |
| <code>microedition.jtwi.version</code>     | Pour identifier si l'appareil est compatible avec la spécification JTWI. S'il est compatible la valeur retournée doit être « 1.0 » sinon la valeur doit être « NULL » |

En plus de clés définies dans les spécifications de chaque API, les fabricants de téléphones mobiles peuvent définir les clés propres à l'identification des API optionnelles fournies.

La prochaine partie du travail est concentrée surtout sur les spécifications des composants obligatoires, car les applications construites dans le but d'avoir la plus grande portabilité possible, les composants optionnels ne devraient être utilisés sans avoir effectué les tests sur les appareils spécifiques.

### 2.2.3 « Connected Limited Device Configuration »

La spécification « Connected Limited Device Configuration » appelée aussi plus communément CLDC existe actuellement en deux versions : 1.0 [2.3] et 1.1 [2.4]. La version 1.1 doit être implémentée en respectant principalement les spécifications suivantes :

- L'implémentation doit permettre la création simultanée d'au moins 10 objets `java.lang.Thread`.
- La précision au niveau de l'incrémentation du temps ne peut pas dépasser 40 millisecondes.
- Les propriétés des caractères « Basic Latin » et « Latin-1 Supplement » d'Unicode 3.0 doivent être implémentées. Chaque version doit supporter au moins ISO Latin-1 caractères de la version Unicode 3.0.

La configuration spécifie donc surtout un sous-ensemble de noyau du langage Java.<sup>2</sup> Elle définit aussi la puissance minimale d'un appareil en termes de l'unité centrale, de la mémoire vive et persistante allouée à l'exécution des programmes Java.

#### **2.2.4 « Mobile Information Device Profile »**

La spécification « Mobile Information Device Profile » [5] définit les API Java spécifiques aux appareils ayant des ressources limitées. Nous y retrouvons les classes inexistantes dans le monde de « Desktop Java ».

La classe `javax.microedition.rms.RecordStore` est rajoutée par exemple et elle doit permettre la création d'au moins cinq espaces d'enregistrement indépendants.

L'appareil doit implémenter le support du protocole HTTP. Les sockets Java sont optionnels.

Le support du format d'images JPEG (ISO/IEC JPEG avec JFIF) doit être supporté pour les images. Le format PNG est aussi obligatoire et la taille maximale de 32768 pixels par image doit être supportée. Même si l'appareil ne gère pas les couleurs, il doit être capable d'afficher l'image.

L'accès à l'annuaire téléphonique d'un téléphone portable doit être permis aux classes `javax.microedition.lcdui.TextBox` et `javax.microedition.lcdui.TextField` si l'annuaire est disponible.

Le point important de MIDP 2.0 est la spécification d'approvisionnement des MIDlets et leur installation. Le nom donné à cette fonctionnalité est « Over The Air » ou plus communément OTA. OTA spécifie les fichiers nécessaires pour l'installation des MIDlets et les protocoles réseau à utiliser pour cette installation. Les fichiers nécessaires pour l'installation des MIDlets sont les fichiers « Java Archive » et « Java Application Descriptor » ou plus communément JAR et JAD. Le fichier de description ou JAD est toujours associé à un fichier archive ou JAR. Le descripteur contient les données texte décrivant les ressources nécessaires pour le bon fonctionnement de cette archive. L'archive contient une ou plusieurs MIDlets contient l'application elle-même. Dans le prochain travail, nous revenons plus en détail sur les techniques d'installation utilisées couramment.

#### **Politique de sécurité pour les appareils GSM/UMTS**

MIDP 2.0 définit le cadre de sécurité pour l'authentification et autorisation concernant les MIDlet. La spécification JTWI mandate les appareils GSM/UMTS qui implémentent la politique de sécurité de suivre le cadre spécifié dans MIDP 2.0.

Une MIDlet dont on ne peut pas vérifier l'origine et l'intégrité appartient au domaine « sans confiance ». Une MIDlet signée en utilisant les mécanismes de signature MIDP 2.0 valide ne doit pas être installée comme « sans confiance ».

---

<sup>2</sup> Beginning J2ME From Novice to Professional, chapitre 1, pages 6-7

## « Le domaine sans confiance »

Si la MIDlet appartient au domaine « sans confiance » l'utilisateur doit être informé que l'application n'est pas conforme à la source de confiance. Il doit être informé sur la base des informations existantes à propos de la MIDlet avant de lui accorder les permissions.

Les permissions pour les MIDlet sont définies en trois groupes: Le groupe concernant le réseau, celui regroupant les informations de la vie privée de l'utilisateur, et le groupe additionnel.

Le groupe concernant le réseau est subdivisé en quatre sous-groupes :

- Accès aux réseaux : contiens les permissions pour chaque fonction de MIDlet permettant une connexion et transmettant les données sur le réseau.
- Le service des messages : contiens les permissions pour chaque fonction de MIDlet permettant d'envoyer ou recevoir les messages.
- Auto invocation de l'application : contiens les permissions pour chaque fonction qui permet à la MIDlet d'être invoquée automatiquement.
- Connexions locales : contiens les permissions pour chaque fonction de la MIDlet qui utilise une connexion sur le port local.

Le groupe concernant la vie privée de l'utilisateur définit les droits d'accès aux données privées de l'utilisateur comme l'annuaire téléphonique ou la lecture des fichiers sur le téléphone.

Le groupe additionnel spécifie les règles d'utilisation d'autres groupes des permissions définies ailleurs et qui ne reposent pas sur le cadre de MIDP 2.0.

En fonction du groupe, les permissions accordées par l'utilisateur peuvent être valables pour la session ou à chaque accès de la MIDlet au domaine protégé.

## « Domaine de confiance »

À l'aide de la signature numérique, une MIDlet peut passer du domaine « sans confiance » au domaine de « confiance ». Ce domaine de permission va donc permettre à MIDlet d'effectuer les fonctions qui appartiennent au domaine de protection. La MIDlet est authentifiée en signant le fichier JAR. La signature et les certificats sont associés à l'application dans le fichier de description (manifest). L'appareil les utilise pour vérifier la signature. Il complète ensuite l'authentification en utilisant le certificat principal du domaine de protection de l'appareil. Si une implémentation MIDP 2.0 reconnaît les MIDlet signées en utilisant PKI comme MIDlets du « domaine de confiance » elle doit suivre la spécification de MIDP 2.0.

La conformité d'une MIDlet à la spécification de signature numérique assure aussi la grande portabilité d'une application dont on veut assurer l'exécution dans le domaine de confiance.

### **2.2.5 « Wireless Messaging API »**

« Wireless Messaging API » ou WMA est mandaté dans la version 2.0 de MIDP. Cette API est utilisée pour envoyer et recevoir les messages courts sur les réseaux cellulaires par exemple les SMS. WMA doit principalement supporter les messages jusqu'à 120 caractères.

### **2.2.6 « Mobile Media API »**

« Mobile Media API » [6] ou MMAPI est optionnel, mais s'il est implémenté la version 1.1 doit être la version minimale implémentée. Si le support des services média est exposé à l'aide du langage Java, MMAPI doit être implémenté aussi.

### **2.2.7 Synthèse des spécifications**

L'entreprise propriétaire du langage Java ne s'occupe pas d'effectuer les vérifications des spécifications. Ils publient juste une implémentation de référence du « Kit de test de compatibilité ». Ils ne font pas non plus l'implémentation de la machine virtuelle Java comme dans le cas des ordinateurs de bureau. Les fabricants des téléphones mobiles sont entièrement responsables d'effectuer les tests approfondis et de respecter les spécifications.

En résumé de ce chapitre nous pouvons remarquer que de grands efforts ont été investis dans la réalisation des spécifications. La plateforme Java est implémentée par les différents fabricants des téléphones et non pas par le propriétaire du langage. En revanche une définition claire des spécifications a été nécessaire pour assurer la portabilité de ce langage. Il reste tout de même que cette élaboration communautaire des spécifications laisse la place aux exceptions, même après les versions finales. Ainsi, les incompatibilités apparaissent en fonction du marché géographique pour lequel un modèle du téléphone a été fabriqué ou entre les marques d'appareils. Ces incompatibilités entre les spécifications restent mineures et n'affectent pas beaucoup le temps de développement des MIDlets.

Le résultat positif de ces efforts est la protection de la vie privée des utilisateurs grâce à la bonne gestion des domaines de confiance. Concernant les développeurs la bonne définition des interfaces de programmation et la minimisation de leur fragmentation font partie des points forts de « Java Micro Edition ».

Le point négatif est le temps d'élaboration des spécifications. Les interfaces de programmation sont basées sur la sécurité, envoi et réception des messages courts et la connexion basique respectant le protocole http. Le résultat est une portabilité au prix de rester sur les applications basiques et une utilisation des ressources des appareils non efficiente.

L'étude des spécifications reste un bon point de départ pour chaque développeur des MIDlets et elle devrait primer sur les habitudes de développement sur la plateforme « Java Standard Edition » où les ressources de la plateforme sont infiniment plus grandes. En étudiant le déploiement d'une MIDlet utilisant une des interfaces de programmation optionnelle, la bonne connaissance des spécifications permet d'estimer le nombre des téléphones potentiellement compatibles.

## 2.3 Les méthodes de déploiement

Les possibilités de déploiement d'applications mobiles développées sur la plateforme Java ME sont nombreuses, la plus connue étant celle qui se base sur la réception de l'application par le réseau sans fil de l'opérateur. Le matériel sur lequel va être déployée l'application varie en fonction du constructeur, mais la plupart supportent cette méthode de déploiement que nous allons détailler ci-dessous.

La machine virtuelle pour environnement mobile de Java est largement implémentée dans les différents téléphones mobiles, ce qui permet de développer des applications pouvant être exécutées sur un grand nombre de téléphones mobiles. Par contre, il n'existe pas de plateforme simple d'utilisation qui permet de déployer ces applications.

Motorola propose un utilitaire qui s'intitule « mobile Phone Tools » permettant d'accéder au téléphone via un câble USB et les technologies sans fils Bluetooth et infrarouge. Par contre, la visibilité des applications J2ME déployée n'est pas proposée, l'installation de ces applications non plus.

Sony Ericsson et son application « Sony Ericsson Communications Suite » possède les mêmes spécificités que celui de Motorola.

Nokia propose le « Nokia PC Suite » qui ne permet également pas d'accéder aux applications J2ME installées sur le téléphone.

Les autres marques comme Siemens ou Samsung proposent des outils similaires, aux fonctions équivalentes. Les communications entre le téléphone mobile et l'ordinateur sont effectuées par USB, connexion série, Bluetooth ou infrarouge.

Pour simplifier le déploiement, la mise en place d'un serveur web contenant les différentes applications J2ME disponibles est conseillée.

### 2.3.1 Le déploiement par Bluetooth et IrDA

L'API Bluetooth est spécifiée dans la JSR 82, elle contient deux packages (JABWT et BTAPI) qui permettent d'utiliser les classes `javax.bluetooth` pour les communications Bluetooth et `javax.obex` pour le transfert de fichier via cette technologie de communication. Le protocole OBEX est composé de huit commandes gérant le transfert de fichier via Bluetooth. OBEX est spécifié dans la JSR 197, et fait partie du GCF. L'avantage de ce moyen de communication, c'est la possibilité d'atteindre dix mètres de distance, le téléphone mobile peut donc rester dans le sac ou la poche durant un transfert de données avec un ordinateur.

Le déploiement par IrDA est moins commun de nos jours, du fait de la visibilité nécessaire entre le client et le serveur, et la distance maximale d'un mètre. Le Bluetooth a remplacé cette technologie, qui est de moins en moins présente sur les téléphones mobiles.

### 2.3.2 Le déploiement par Wireless LAN

Pour pouvoir effectuer un déploiement par WLAN, il est nécessaire que les clients en soient équipés. Les smartphones actuellement sur le marché possédant une connectique WLAN 802.11 peuvent accéder directement sur le serveur web contenant le JAD ou JAR par HTTP, sans devoir passer par les fréquences GSM/UMTS. Cela permet un déploiement rapide des applications, et également un transfert de données rapide, car les normes 802.11b et 802.11g atteignent respectivement 11 et 54 mégaoctets par seconde.

### 2.3.3 Le déploiement OTA

Le moyen le plus répandu est la méthode OTA par GSM/GPRS/UMTS qui repose sur la connexion sans fil proposée par l'opérateur de téléphonie mobile. Le serveur web contient des pages au format WML qui permettent au navigateur présent dans le téléphone mobile d'afficher les liens vers les JAD et JAR. Cette méthode requiert une passerelle qui permet de traiter correctement les requêtes WAP. Avec les terminaux capables de traiter le XHTML, il est possible d'afficher directement les pages HTML, et donc d'utiliser de manière transparente le standard trouvé sur le web via le protocole HTTP spécifié dans la RFC 2616 de l'IETF.

Les vitesses de transfert varient selon la technologie proposée par l'opérateur, et supportée par le client. Le moyen de base, mais aussi le plus lent est le GPRS, qui permet l'échange de donnée en paquet, à l'image du modèle TCP/IP ou les données sont envoyés par paquet, cela permet d'éviter la surcharge de la station de transmission ou BTS (Base Transceiver Station). Les technologies de transmission plus évoluées sont l'EDGE, qui permet de transmettre les données de manière plus rapide que le GPRS, puis ensuite vient l'UMTS.

C'est le premier protocole de transmission supportant la voix, la vidéo et les données communément appelé 3G pour troisième génération.

Les technologies HSDPA et HSUPA sont implantées actuellement, et permettent des vitesses de transfert encore plus rapide que l'UMTS. Elles font partie du groupe nommé 3G+ pour troisième génération plus. La technologie en développement et successeur du couple HSDPA et HSUPA est le HSOPA ou High Speed OFDM<sup>3</sup> Packet Access. Elle sera intégrée aux téléphones mobiles pour permettre des vitesses d'accès encore supérieures.

Avec ces nouvelles technologies de transmissions, le déploiement OTA sera de plus en plus utilisé, car rapide et plus flexible qu'une synchronisation par Bluetooth par exemple.

La 4G ou quatrième génération tente de faire un rapprochement avec les nouvelles technologies sans fil à venir, à savoir le 802.16 WiMAX qui va permettre d'accéder à la façon WiFi au réseau, mais avec des portées pouvant atteindre les 30km. Elle présente d'ailleurs les mêmes caractéristiques de transmission en OFDM, ce qui

---

<sup>3</sup> OFDM, *Multiplexage par répartition en fréquences orthogonales*

permet d'utiliser la même technologie au niveau des BTS. Le 802.20 ou iBurst étant une implémentation de la norme ANSI HC-SDMA ou High Capacity Spatial Division Multiple Access est pour l'instant déployé en Australie, Afrique du Sud et prochainement le Mexique. Le ralentissement à l'adoption des nouvelles technologies mobiles en Europe est dû aux prix exorbitants des licences UMTS payés par les opérateurs. L'UMTS est une norme présentée depuis les années 1992-1993, et les applications tirant profit de cette technologie ont vu le jour en 2002-2003 soit plus de 10 ans plus tard. La rentabilité est donc un facteur important de l'adoption d'un standard.



Figure 2.3 : Over The Air

## 3. Déploiement d'applications mobiles

Les téléphones mobiles se multiplient, et sont de plus en plus utilisés dans le milieu professionnel. Pour permettre leur utilisation, il faut les agréments des applications nécessaires aux communications avec le système d'information de l'entreprise, pour qu'ils soient efficaces, qu'ils reflètent les données à jour du système et qu'ils puissent agir sur ces données.

Cette procédure de déploiement des applications sur les téléphones mobiles requiert un travail conséquent de recensement des différentes plateformes employées, d'évaluation des possibilités de déploiement et finalement de test pour vérifier que la procédure se déroule sans problème. Développer une même application sur un nombre conséquent de plateformes est une opération longue et qui demande beaucoup de ressources. En effet, il faut avoir les compétences nécessaires pour développer dans les langages qui sont utilisés pour développer une application qui puisse s'exécuter sur la plateforme, à cela il faut ajouter les règles d'ergonomie, qui varient également compte tenu des différentes tailles d'écran, de la disposition des touches, des méthodes de saisie (tactile, navigateur multidirectionnel, et autre touche spécifique).

Une solution pour réduire le nombre de plateformes consiste à utiliser le langage proposé par Java. Nous nous sommes donc basés sur la technologie Java et plus particulièrement le substrat Mobile Edition, qui présente l'avantage d'être portable. La portabilité d'une application est un avantage certain, car elle est utilisable sur un maximum d'architecture étant donné sa capacité de pouvoir s'exécuter via une machine virtuelle.

### 3.1 L'implémentation des spécifications Java ME

Les grands fabricants du secteur de la téléphonie mobile n'ont pas l'obligation d'implémenter les spécifications Java ME émises par Sun Microsystems. En choisissant une implémentation partielle, il devient difficile de savoir lesquels ont été implantés, et surtout, de ne pouvoir garantir de manière fiable le bon fonctionnement de l'application. Le fait d'avoir des téléphones mobiles intégrant de manière incomplète les spécifications émises par Sun Microsystems pose donc un problème aux développeurs et distributeurs d'applications mobiles. Le but est de trouver une solution qui permet de connaître les spécificités du client, de l'avertir de la compatibilité de son téléphone mobile avec l'application, et l'idéal serait d'avertir également le développeur ou le distributeur de l'application des raisons de l'échec ou de la réussite du test de compatibilité. Ainsi, celui-ci pourra afficher la compatibilité ou non de ses applications aux acquéreurs et utilisateurs potentiels.

Une autre difficulté est la taille variable des écrans, une application doit pouvoir s'adapter en largeur et en hauteur, mais aussi supporter le mode portrait ou paysage de l'affichage.

Le déploiement d'application est donc difficile. Comment valider les fonctionnalités d'une application mobile ?

### 3.2 FoxyTest

Le déploiement d'application mobile dépend non seulement de la version Java embarquée, mais aussi des possibilités d'accès de la machine virtuelle aux composants matériels présent dans le téléphone mobile.

En effet, un téléphone mobile peut contenir les spécificités nécessaires à l'exécution de l'application déployée, mais l'accès à l'un ou l'autre des composants peut être limité par le constructeur. Ces limitations ne sont pas répertoriées sur le site respectif des constructeurs de téléphones mobiles, il nous fallait donc un outil capable de réaliser différents tests sur le téléphone mobile du client pour être certain que ce dernier est supporté par une application nécessitant l'accès aux diverses ressources présentes. Sans la possibilité de vérifier ces paramètres, le risque de déployer une application sans connaître les capacités du client à l'exécuter revient à s'exposer aux erreurs ou blocages de fonctionnement de l'application.

Pour confirmer la possibilité d'utiliser les composants présents dans le téléphone mobile, nous avons développé un logiciel nommé FoxyTest, qui permet de tester l'accès au composant Bluetooth et l'utilisation de ce dernier par une application Java ME. Cette spécification est répertoriée sous la norme JSR 82, elle permet d'accéder et d'utiliser le Bluetooth d'un téléphone mobile.

FoxyTest permet également de tester la présence d'un GPS interne, et l'accès à ce dernier grâce à la spécification JSR 179 qui contient la description des classes de localisation permettant le positionnement de l'utilisateur. L'application repose finalement sur l'accès à l'internet et donc l'utilisation des communications radio GPRS et l'autorisation de transmettre des données via ce canal, pour permettre d'émettre les résultats au serveur qui récupère les données récoltées. Le système est composé d'un client présent sur le téléphone mobile, et d'un serveur qui récupère les données du client.

FoxyTest permet donc de vérifier la présence des composants et de l'accès à ces derniers via une application Java installée sur le téléphone mobile.

Le schéma des connectivités de FoxyTest présente une vue d'ensemble sur les différents acteurs et les technologies de communications engagés depuis le déploiement de l'application jusqu'à son exécution. Les traits continus représentent des liaisons filaires, et les traits interrompus indiquent une communication sans fil.

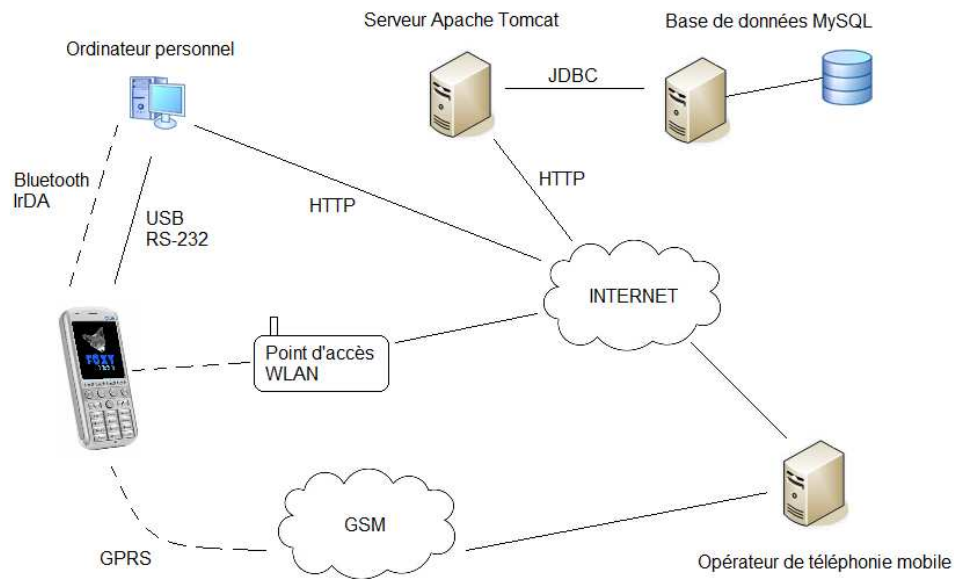


Figure 3.1 : Schéma des connectivités de FoxyTest

La première phase consiste à installer FoxyTest sur le client en employant l'une des méthodes de déploiement spécifié au point 2.3. Soit depuis un ordinateur personnel par USB, infrarouge, Bluetooth ou une connexion série, ou par WLAN via un point d'accès sans fil, ou par GPRS donc OTA. Une fois le client FoxyTest installé et exécuté, les résultats du test sont envoyés par GPRS vers la servlet résidant sur le serveur Apache Tomcat. Ce dernier transmet les données par le connecteur JDBC à la base de données qui insert chaque requête dans la table correspondante.

Les données relayées par l'opérateur de téléphonie mobile sont invisibles au client et au serveur, le fait de transmettre les données de manière standardisée depuis le client en indiquant clairement la cible, l'opérateur achemine les données de manière transparente.

Le point d'accès WLAN ou sans-fils peut être public ou privé, l'accès à ce dernier doit simplement être autorisé, puis les données peuvent transiter par ce moyen. Lors d'un déploiement local depuis un ordinateur personnel, il faut avoir téléchargé le fichier JAR au préalable, pour pouvoir le transmettre au client par l'un des moyens indiqués.

Une fois l'opération réussite, le client à la possibilité de supprimer l'application de son téléphone mobile.

### 3.3 Développement de l'application

Le langage choisi est Java pour les avantages spécifiés dans les chapitres précédents. La plateforme de développement IDE est NetBeans 5.5 et le Mobility Pack 5.5 (CLDC), avec le Sun Java Wireless Toolkit 2.5 for CLDC pour la compilation et Proguard pour l'obfuscation.

NetBeans 5.5 est un environnement de développement entièrement réalisé en Java, et présente l'avantage d'être orienté dans ce même langage. Ce dernier supporte d'autres langages comme le C/C++ via des plug-ins. Le Wireless Toolkit peut être spécifié comme émulateur, donc appelé de manière automatique lors de l'exécution du code compilé ou non, car il intègre également un compilateur. Le Mobility Pack en version CLDC permet d'avoir à portée de main les différentes API spécifiques aux téléphones mobiles, comme le MIDP pour le plus connu d'entre eux.

Le Sun Java Wireless Toolkit est proposé en version CDC ou CLDC, le CDC étant orienté pour les PDA possédant plus de ressources que les téléphones mobiles ou smartphones, le CLDC étant la version limitée et c'est celle que nous utiliserons, et qui est employée dans les téléphones mobiles ayant de faibles ressources. Étant un produit développé par Sun Microsystems, il est d'une excellente qualité pour la compilation et l'émulation de programme développé pour des environnements mobiles. Il a été distingué à plusieurs reprises comme produit de l'année et logiciel d'excellence par les différents acteurs, observateurs et rédacteurs du domaine des outils de développement pour téléphones mobiles. Proguard fait office de librairie optionnelle d'obfuscation, ou de cryptage, pour éviter une découverte du code initial.

Le code est donc compilé par le WTK, résultant par la création de deux fichiers, la ressource et la description. Le Java Application Descriptor ou JAD contient la description du programme, comme l'auteur, le lien vers la ressource, la version et d'autres informations utiles. Le Java Archive contient le programme compilé, qui permet l'exécution de l'application. L'utilisation directe du JAR est possible, mais lors de stratégie de déploiement, le JAD permet d'indiquer à l'utilisateur l'origine du JAR via le fabricant, mais aussi de vérifier la version installée pour éviter d'écraser une version plus récente de l'application. Les quelque 1300 lignes de code sont réparties dans sept fichiers sources que nous allons détailler ci-dessous.

### 3.4 Structure de l'application

Le projet est composé de sept fichiers sources programmés en Java, cinq images et une icône. Une description de leurs rôles et fonctions est décrite ci-après, en respectant le déroulement de l'application. L'arborescence du projet ne peut être reproduite, car dans NetBeans ce dernier mélange les images et les fichiers sources, tandis que le Sun Java Wireless Toolkit place les médias dans un dossier « res » pour ressources.

Le diagramme suivant représente les différentes classes et leurs relations du projet FoxyTest. Il a été généré depuis l'environnement NetBeans:

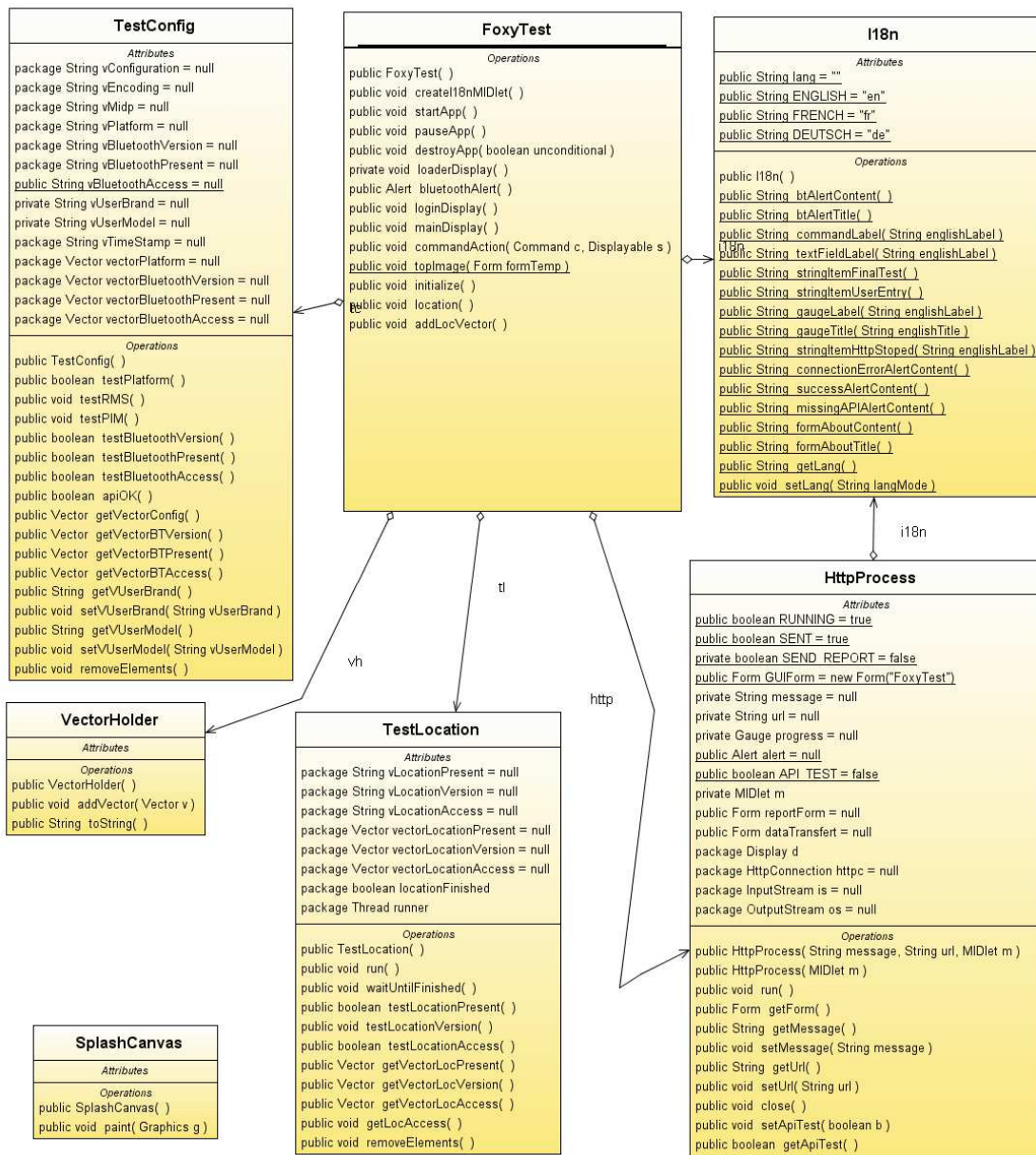


Figure 3.2 : Diagramme des classes de FoxyTest

### 3.4.1 FoxyTest, SplashCanvas, I18n

**FoxyTest.** FoxyTest est la source principale qui contient les appels vers les autres classes et méthodes, et la fonction d'exécution principale. La première étape consiste à importer les paquetages nécessaires à l'exécution du programme, entre autres, `javax.bluetooth.*`. La classe principale est construite comme MIDlet et implémente `CommandListener` pour récupérer les interactions avec le clavier du téléphone mobile. Le standard Java SE bloque l'exécution du programme si l'un des paquetages est manquant. Sous Java ME, l'étape de compilation n'indique pas d'erreur si l'un des paquetages est manquant. Au moment de l'exécution de l'application, le paquetage va simplement être passé.

**I18n.** Une fois les différents écrans et commandes initialisés, l'on affiche le choix des langues, avec la traduction des inscriptions contenue dans le fichier source `I18n.java`. Après le choix de la langue la méthode `bluetoothAlert()` de type `Alert` est appelée pour informer l'utilisateur d'activer le Bluetooth. Sur certains téléphones mobiles, comme par exemple certains de Sony Ericsson, dès que l'application requiert le stack Bluetooth, il est automatiquement activé avec un avertissement, donc il n'est pas nécessaire d'activer le Bluetooth sur ces modèles.

**SplashCanvas.** Après confirmation de lecture de l'alerte, la méthode `loaderDisplay()` est appelée, elle affiche un canevas avec le logo de l'application, le copyright et le nom des développeurs. Ces informations sont récupérées depuis le fichier `SplashCanvas.java` qui contient la méthode de création d'image, depuis le fichier `foxytest.png` pour l'affichage du logo. Dès ce moment, l'utilisateur peut soit choisir le menu « À propos » qui lui donne une brève explication de FoxyTest, ou de démarrer le processus de test. La méthode `loginDisplay()` est affichée, et demande à l'utilisateur d'entrer la marque et le modèle du téléphone mobile qu'il possède, des données qui sont récupérées par un `TextField()`. Ces premiers éléments récupérés que sont la marque et le modèle du téléphone mobile, sont entrés manuellement par l'utilisateur, car les autorisations d'accès aux informations de la plateforme ne sont pas toujours données. En effet, le code suivant ne livre pas toujours la plateforme :

```
System.getProperty("microedition.platform");
```

Il retourne « j2me » pour les téléphones mobiles du fabricant Motorola et également Samsung.

La méthode `mainDisplay()` est appelée, avec la possibilité de tester simplement le téléphone mobile, ou de le tester et envoyer le résultat du test sous forme de rapport au serveur qui est relié à la base de données dans laquelle va être ajouté le test de l'utilisateur. Une fois que l'utilisateur sélectionne « Tester » ou « Rapport », le test de l'application est lancé, les classes concernées sont `TestConfig`, `TestLocation` et `VectorHolder`.

### 3.4.2 TestConfig, TestLocation, VectorHolder

**TestConfig.** `TestConfig` permet de tester les paramètres Bluetooth, RMS et PIM du téléphone mobile. RMS est testé dans un « try » avec la création d'un `RecordStore` comme suit :

```
rs = RecordStore.openRecordStore("test",true);
```

puis la taille du `RecordStore` est vérifiée avec `int size = rs.getSizeAvailable();` Étant initialisé à la valeur nulle, si

la taille est visible, alors l'on considère l'accès et l'écriture au RecordStore possible. Le PIM est testé avec la commande standard :

```
System.getProperty("microedition.pim.version");
```

Si cette dernière retourne une version, elle est ajoutée au vecteur qui va nous permettre de stocker les différents résultats, puis de les envoyer par HTTP.

Au niveau du test Bluetooth, il se décompose en trois étapes.

La première consiste à vérifier la version de l'API Bluetooth installée, et celle de OBEX. La première manière est celle mandatée par la spécification de l'API Bluetooth, mais nous avons constaté qu'elle n'est pas toujours respectée, c'est la raison pour laquelle nous avons dupliqué le test via la méthode suivante :

```
System.getProperty("bluetooth.api.version")
```

Les suivantes sont celles adaptées par rapport à la spécification:

```
LocalDevice.getProperty("bluetooth.api.version");  
LocalDevice.getProperty("obex.api.version");
```

La deuxième étape vérifie la présence du Bluetooth en effectuant une requête simple d'accès aux classes :

```
Class.forName("javax.bluetooth.LocalDevice");  
Class.forName("javax.bluetooth.DiscoveryAgent");
```

Si ces appels ne génèrent pas d'exception, l'on peut considérer que le Bluetooth est bien présent sur le téléphone portable.

La dernière étape permet de tester pleinement les fonctionnalités Bluetooth. Dans un « try », l'on instancie une session Bluetooth via la méthode `DiscoveryAgent` qui permet de lancer une recherche des appareils Bluetooth présents dans un rayon de dix mètres.

```
DiscoveryAgent discoveryAgent;  
LocalDevice localDevice = LocalDevice.getLocalDevice();  
discoveryAgent = localDevice.getDiscoveryAgent();
```

Si cette instance n'est pas récupérée par la fonction « catch », il n'y a donc pas d'exception, alors l'accès et l'utilisation du Bluetooth est autorisé par une MIDlet. Le téléphone mobile est donc considéré comme étant conforme à la spécification JSR 82.

**TestLocation.** `TestLocation` permet de vérifier la présence d'une puce GPS intégrée au téléphone mobile, pour permettre l'utilisation d'application nécessitant le positionnement de l'utilisateur. Comme pour le Bluetooth, l'on vérifie la version, la présence et l'accès à la puce si cette dernière est présente sur le téléphone mobile. La version est donc obtenue avec la même méthode `getProperty()` :

```
System.getProperty("microedition.location.version");
```

Une fois la version obtenue, l'on peut tester la présence du GPS :

```
Class.forName("javax.microedition.location.Location");
Class.forName("javax.microedition.location.LocationProvider");
```

Si ces classes sont présentes, il y a de fortes chances pour que le téléphone mobile soit équipé du matériel permettant la localisation.

Pour en être certain, et pour valider la compatibilité JSR 179, il reste encore à initialiser le GPS.

```
Criteria cr = new Criteria();
cr.setHorizontalAccuracy(500);
LocationProvider lp = LocationProvider.getInstance(cr);
Location l = lp.getLocation(60);
Coordinates c = l.getQualifiedCoordinates();
```

La méthode `Criteria()` et `LocationProvider()` permettent de choisir le type de puce GPS présent, s'il s'agit d'un système GPS pur, ou un système GPS assisté, ou encore un système de localisation à base de triangulation de station de relais (BTS).

La méthode `getQualifiedCoordinates()` permet d'obtenir la latitude et longitude grâce à `getLatitude()` et `getLongitude()`.

Si les coordonnées de `javax.microedition.location.Coordinates` ne sont pas nulles, alors l'accès est possible, et la JSR 179 est supportée.

**VectorHolder.** `VectorHolder` permet de stocker les résultats des différents tests dans un seul vecteur, qui sépare ces derniers par un et commercial « & » pour être récupéré correctement par le serveur de données lorsqu'il recevra le vecteur sérialisé par la classe `HttpProcess`.

### 3.4.3 `HttpProcess`

Une fois les différents tests effectués et le vecteur contenant le résultat des tests, l'application est prête pour donner le résultat de la compatibilité du téléphone mobile avec l'application cible.

Le dernier test consiste à vérifier la présence d'une connexion internet. Pour cela, l'appel de la classe `HttpProcess` présente dans le fichier source `HttpProcess.java` est effectué, pour vérifier la présence d'une connectivité internet par GPRS en ayant comme cible URL `http://cuilxa.unige.ch:8080/foxytagtest/servlet/FoxyTag?type=foxytest`. Si la cible est atteignable, cela signifie que les paramètres GPRS sont correctement installés sur le téléphone mobile, et que la transmission de donnée via ce canal est possible.

Pour revenir à l'étape finale de l'application, si la méthode « Tester » est sélectionnée et que les paramètres GPRS permettent la transmission de données, et que l'accès Bluetooth est automatisé, alors l'alerte `getSuccessAlert()` est appelée. Elle affiche l'image `ok.png` et le texte rattaché pris dans la langue sélectionnée par

`setString(i18n.successAlertContent());`. Dans le cas d'une mauvaise configuration GPRS, c'est l'alerte `getConnectionErrorAlert()` qui est affichée avec l'image `error.png` et le texte toujours présent dans le fichier `I18n.java`  
`setString(i18n.connectionErrorAlertContent());`

Dans le cas intermédiaire, à savoir une connexion présente, mais pas de Bluetooth ou ce dernier est inutilisable par une MIDlet, alors c'est `getMissingAPIAlert()` qui est affiché avec le texte `setString(i18n.missingAPIAlertContent());`.

### 3.5 Réalisation du serveur de données

La partie serveur est composée d'un ordinateur faisant office de serveur, ou d'une session virtuelle reposant sur ledit serveur. Les applications nécessaires sont un serveur web gérant les servlets, et une base de données gérée par un SGBD.

Les servlets sont maintenues par Apache Tomcat, un serveur spécialisé dans la technologie des JavaServer Pages ou JSP et l'implémentation des Java Servlets.

Le gestionnaire de base de données est MySQL, c'est l'un des plus répandus dans les PME industrielles, et les projets libres. Basé sur le modèle des bases de données relationnelles, il permet via ses deux outils principaux, le MySQL administrator, et le MySQL Query Browser d'effectuer les différentes tâches d'administration et de gestion de la base de données. Les données sont injectées dans la base de données via la servlet présente sur le serveur Apache Tomcat grâce au connecteur JDBC.

#### 3.5.1 Structure de la base de données

La base de données contient deux tables, l'une est destinée à l'obtention de la notification d'installation de l'application, et l'autre a la récupération du résultat des rapports de test réalisé par le client.

La table « `install_notify` » possède comme champs « `id` », qui est auto incrémentée, « `response_code` » qui contient les codes de réponse renvoyé par l'application, « `region` » pour l'adresse IP, et « `time_stamp` » pour la date d'obtention de la notification renvoyée par le client via un POST.

|                          | Champ                      | Type                     | Interclassement              | Attributs | Null | Défaut                         | Extra                       |
|--------------------------|----------------------------|--------------------------|------------------------------|-----------|------|--------------------------------|-----------------------------|
| <input type="checkbox"/> | <code>id</code>            | <code>int(10)</code>     |                              | UNSIGNED  | Non  |                                | <code>auto_increment</code> |
| <input type="checkbox"/> | <code>response_code</code> | <code>varchar(50)</code> | <code>utf8_general_ci</code> |           | Oui  | <code>NULL</code>              |                             |
| <input type="checkbox"/> | <code>region</code>        | <code>varchar(45)</code> | <code>utf8_general_ci</code> |           | Non  | <code>ch</code>                |                             |
| <input type="checkbox"/> | <code>time_stamp</code>    | <code>timestamp</code>   |                              |           | Non  | <code>CURRENT_TIMESTAMP</code> |                             |

Figure 3.3 : Table `install_notify` de la base de données

La table « testreport » contient les champs suivants : « id » (valeur auto-incrémentée), « userbrand » et « usermodel », les deux champs que l'utilisateur entre dans l'application quand le la marque et le modèle de son téléphone mobile lui son demandé, « platform » pour l'identification du constructeur et de la version du firmware, « midp » « config » et « encoding » pour respectivement la version du profile MIDP utilisé, la version de la configuration CLDC présente et le type d'encodage employé. Au niveau des résultats des tests Bluetooth, les champs « sysbtapi » pour la version de l'API Bluetooth avec un second test au niveau de l'accès nommé « locbtapi » mais renvoyant la même valeur, « btpresent » pour la présence de la puce, « btaccess » pour l'accès à cette dernière et « btobex » pour l'échange de données sont réservés. La localisation présente les même trois champs que le Bluetooth pour la version, la présence et l'accès nommé « locversion », « locpresent » et « locaccess ».

Les champs « memory » indiquent la mémoire disponible, « api » « test » et « actif » sont utilisés pour trier la base de données selon des critères d'accès à la puce Bluetooth.

La « region » récupère l'adresse IP du client, « time\_stamp » indique l'heure à laquelle le test à été effectué, « rms » pour la taille a disposition du RecordStore et « pim » indique la version du PIM installé.

|                          | Champ             | Type         | Interclassement | Attributs | Null | Défaut            | Extra          |
|--------------------------|-------------------|--------------|-----------------|-----------|------|-------------------|----------------|
| <input type="checkbox"/> | <b>id</b>         | int(11)      |                 |           | Non  |                   | auto_increment |
| <input type="checkbox"/> | <b>userbrand</b>  | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>usermodel</b>  | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>platform</b>   | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>midp</b>       | varchar(45)  | utf8_general_ci |           | Oui  | -                 |                |
| <input type="checkbox"/> | <b>config</b>     | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>encoding</b>   | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>sysbtapi</b>   | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>btpresent</b>  | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>locbtapi</b>   | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>btaccess</b>   | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>btobex</b>     | varchar(45)  | utf8_general_ci |           | Non  | -                 |                |
| <input type="checkbox"/> | <b>memory</b>     | varchar(45)  | utf8_general_ci |           | Non  | nok               |                |
| <input type="checkbox"/> | <b>api</b>        | varchar(45)  | utf8_general_ci |           | Non  | nok               |                |
| <input type="checkbox"/> | <b>test</b>       | varchar(45)  | utf8_general_ci |           | Non  | nok               |                |
| <input type="checkbox"/> | <b>actif</b>      | varchar(45)  | utf8_general_ci |           | Non  | no                |                |
| <input type="checkbox"/> | <b>region</b>     | varchar(45)  | utf8_general_ci |           | Non  | ch                |                |
| <input type="checkbox"/> | <b>time_stamp</b> | timestamp    |                 |           | Non  | CURRENT_TIMESTAMP |                |
| <input type="checkbox"/> | <b>rms</b>        | varchar(100) | utf8_general_ci |           | Oui  | NULL              |                |
| <input type="checkbox"/> | <b>pim</b>        | varchar(50)  | utf8_general_ci |           | Oui  | NULL              |                |
| <input type="checkbox"/> | <b>locpresent</b> | varchar(45)  | utf8_general_ci |           | Oui  | NULL              |                |
| <input type="checkbox"/> | <b>locversion</b> | varchar(45)  | utf8_general_ci |           | Oui  | NULL              |                |
| <input type="checkbox"/> | <b>locaccess</b>  | varchar(45)  | utf8_general_ci |           | Oui  | NULL              |                |

Figure 3.4 : Table testreport de la base de données

### 3.6 Description de la servlet

La servlet permet de récupérer les données d'install-notify de l'application FoxyTest, et les envois vers la base de données. Pour permettre la communication entre la servlet et la base de données, le connecteur contenu dans l'archive Java mysql-connector-java-3.0.15-ga-bin.jar est inclus dans le dossier des bibliothèques « lib » du projet développé avec l'IDE NetBeans.

La servlet est composée de trois fichiers sources que nous allons détailler.

**Database.** Database est la classe exécutant les requêtes sur la base données par `SQLService()` permettant d'insérer les données récupérées :

```
execUpdate("INSERT INTO testreport (userbrand, usermodel, platform, midp,
config, encoding, sysbtapi, btpresent, locbtapi, btaccess, btobex, memory,
region, rms, pim, locpresent, locversion, locaccess)
VALUES('"+request.getParameter("userbrand")+"' ,
'"+request.getParameter("usermodel")+"' ,
'"+request.getParameter("platform")+"' , '"+request.getParameter("midp")+"' ,
'"+request.getParameter("config")+"' ,
'"+request.getParameter("encoding")+"' ,
'"+request.getParameter("sysbtapi")+"' ,
'"+request.getParameter("btpresent")+"' ,
'"+request.getParameter("locbtapi")+"' ,
'"+request.getParameter("btaccess")+"' ,
'"+request.getParameter("btobex")+"' , '"+request.getParameter("memory")+"' ,
'"+request.getRemoteAddr()+"' , '"+request.getParameter("rms")+"' ,
'"+request.getParameter("pim")+"' ,
'"+request.getParameter("locpresent")+"' ,
'"+request.getParameter("locversion")+"' ,
'"+request.getParameter("locaccess")+"' );");
```

Elle s'occupe également de renvoyer une chaîne de caractère au client pour confirmer la transaction.

**InstallNotifyListener.** InstallNotifyListener effectue une tâche similaire à la classe précédente, mais pour la table `install_notify`. La requête SQL est la suivante :

```
execUpdate("INSERT INTO install_notify (response_code , region)
VALUES('"+responseCode+"' , '"+request.getRemoteAddr()+"' );");
```

La variable « `response_code` » représente le ou les codes obtenus en réponse par l'application FoxyTest. Chaque code indique un état :

- 900 Application installée
- 901 Mémoire insuffisante
- 902 Annulation d'installation de la part du client
- 903 Loss of Service
- 904 Taille du JAR incohérent
- 905 Attribut incohérent
- 906 JAD invalide
- 907 JAR invalide
- 908 Incompatibilité de la configuration ou du profil
- 909 Erreur d'authentification de l'application
- 910 Erreur d'autorisation de l'application
- 911 Erreur d'enregistrement au système de réception automatique
- 912 Notification de désinstallation

**SQLService.** SQLService se charge de la connexion à la base de données. La première étape consiste à charger le driver du connecteur JDBC en créant une nouvelle instance. La deuxième, c'est d'effectuer l'authentification auprès de la base de données en spécifiant le chemin d'accès au serveur, dans notre cas la base de données se trouve sur la même machine physique que le serveur Apache Tomcat contenant la servlet : `jdbc:mysql://localhost:3306/foxytag`

Lors de la procédure de connexion, il faut également indiquer le nom d'utilisateur et le mot de passe permettant d'effectuer des requêtes sur la base de données :

```
DriverManager.getConnection(PATH, USER, PASSWORD);
```

Finalement, la méthode `close()` permet de fermer la connexion à la base de données.

Une fois le projet compilé, le fichier FoxyTest.war est généré, c'est une archive web ou WAR qui peut directement être déployée sur le serveur Apache Tomcat pour être utilisée.

### 3.7 Statistique des rapports

Le nombre de rapports obtenus depuis la mise à disposition de l'application est d'environ 8000. La requête SQL suivante permet de déterminer le nombre de tests où l'accès à la puce Bluetooth est validé:

```
SELECT * FROM `testreport` WHERE btaccess = 'True';
```

Le résultat obtenu est de près de 5000, soit environ 62.5% de la totalité des rapports obtenus. Le graphique suivant est une représentation classée par marque des modèles ayant le Bluetooth accessible, et donc compatible avec la JSR 82.

La requête suivante nous permet d'obtenir cette classification par marque, dans ce cas il s'agit de la marque Nokia.

```
SELECT * FROM `testreport` WHERE btaccess='True' and platform like '%nokia%'
```

Il suffit de remplacer Nokia par les autres marques pour obtenir les autres résultats.

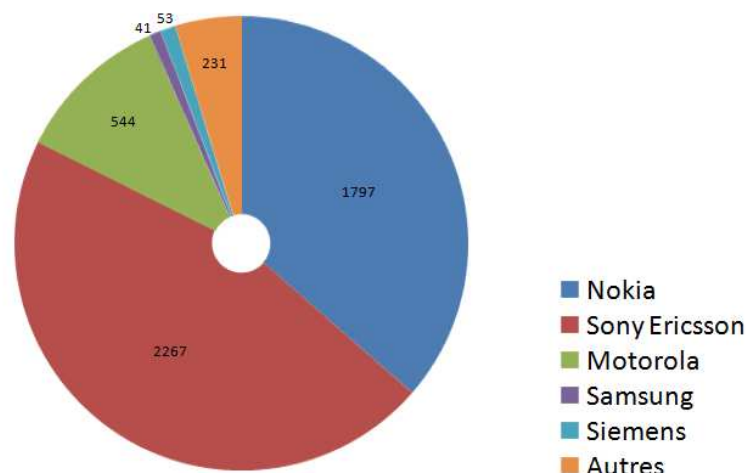


Figure 3.5 : Compatibilité JSR 82 par constructeur

Sony Ericsson et Nokia sont les deux constructeurs respectant le plus la spécification JSR 82, en effet, 2267 téléphones mobiles de Sony Ericsson et 1797 de Nokia sur un total de 5000 forment la majorité.

Le graphique suivant présente les téléphones par constructeur qui ne sont pas compatibles avec la spécification JSR 82.

La requête SQL permettant d'obtenir ces résultats est la suivante, en prenant l'exemple de la marque Nokia.

```
SELECT * FROM `testreport` WHERE btaccess!='True' and platform like '%nokia%'
```

Il suffit de remplacer la requête SQL par le nom du constructeur désiré pour obtenir le résultat par marque.

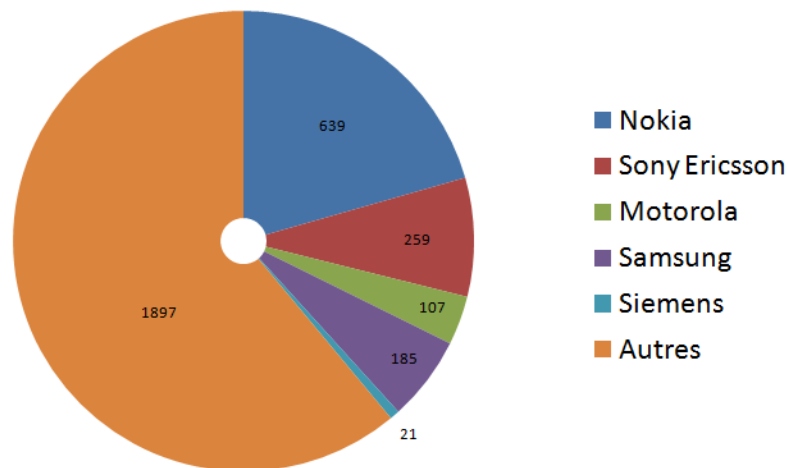


Figure 3.6 : Incompatibilité JSR 82 par constructeur

Le nombre élevé des téléphones incompatibles dans la catégorie « autres » est dû au test rempli de façon incomplète par l'utilisateur au niveau de la marque ou du modèle, avec un nom de plateforme non identifiable, donc générique. Il peut donc s'agir de modèle de la marque Motorola ou Samsung par exemple, mais l'on ne peut pas les identifier.

Le total est d'un peu plus de 3000 pour les téléphones incompatibles, soit le 37,5% du total des tests, qui s'élève à 8000.

## 4. Vers un cas pratique

### 4.1 Présentation de FoxyTag

FoxyTag est un système collaboratif et ouvert permettant aux utilisateurs d'échanger les informations géographiques. L'idée consiste à poser une information géographique appelée aussi tag virtuel près des objets et des endroits pour les signaler aux autres utilisateurs du système. Ainsi, il peut être utilisé pour marquer les endroits dangereux à la montagne et avertir les randonneurs. Les chemins de plongée peuvent aussi être signalés ainsi que les endroits touristiques intéressants à visiter. Les possibilités d'utilisation restent infinies.

D'un point de vue technique, FoxyTag consiste en une MIDlet Java qui sert à capturer la position de l'utilisateur à l'aide d'un GPS, et à transmettre cette information sur le serveur. Le serveur à son tour garde l'information et la distribue entre les utilisateurs en tenant compte des liens de confiance qui s'établissent entre les collaborateurs du système. Développée à l'Université de Genève par Michel Deriaz, cette application a été utilisée comme cas pratique pour étudier la portabilité et le déploiement des MIDlets dans notre travail.

La difficulté principale pour la compatibilité de la MIDlet FoxyTag avec les téléphones portables a été le fait d'utiliser un GPS externe pour capturer la position géographique de l'utilisateur. Étant donné que les appareils GPS ne sont pas encore massivement intégrés dans les téléphones la solution la plus courante est de connecter un appareil GPS externe au téléphone par la technologie Bluetooth. L'utilisation des GPS intégrés dans les téléphones reste une solution d'avenir que nous ne pouvons pas négliger.

En tenant compte de l'apparition des nouveaux GPS intégrés et du fait actuel des GPS Bluetooth nous avons développé une librairie GPS commune facile à réutiliser dans l'élaboration des applications utilisant les manières différentes d'accéder aux données GPS. Le premier but de cette librairie est de diminuer le temps de développement des applications utilisant le GPS. Le second avantage est de pouvoir rapprocher deux API afin de simplifier l'utilisation de GPS Bluetooth. La réutilisation de cette librairie est simplifiée aussi à l'aide d'outils d'obfuscation. Même si le but principal de cet outil est de réduire la taille de l'application et de protéger les droits d'auteur du code source, son avantage dans la réutilisation du code est la fonctionnalité de ne pas intégrer à la compilation les composants non utilisés par l'application. Cela permet de garder une version du code source et d'effectuer les modifications uniquement à l'appel des librairies qui elles accèdent de manière différente aux données GPS. Pour pouvoir effectuer les appels similaires aux librairies GPS elles doivent aussi exposer une interface identique à l'application. De cette manière, les fonctionnalités internes sont cachées au développeur et la réutilisation du code source est facilitée pour préparer les versions du programme pour les différents téléphones.

## 4.2 La portabilité théorique et pratique

L'hypothèse de début de développement d'une application Java utilisant la technologie Bluetooth est que cette dernière est omniprésente et qu'on peut s'en servir dans les applications Java. L'étude préalable des spécifications montre le contraire : l'interface de programmation concernant le Bluetooth est optionnelle. Elle peut aussi être implémentée indépendamment des spécifications, mais dans ce cas elle ne peut pas porter l'indication de conformité. Le fait d'utiliser une API optionnelle a réduit grandement la portabilité de notre application Java. Les résultats des tests reçus à l'aide de FoxyTest nous montrent clairement le nombre de téléphones incompatibles avec cette application.

Si on décide d'utiliser donc deux API optionnelles permettant de recevoir la position en utilisant un GPS intégré ou un GPS Bluetooth l'effort de programmation est double. En rajoutant les différences de code concernant la gestion des objets `java.langage.Thread` et les autres incompatibilités des API entre les différentes marques des téléphones la maintenance des versions et du code source demande un groupe de développeurs beaucoup plus grand. D'autant plus que l'évolution des téléphones portables est rapide et le changement de modèles de téléphones sur le marché est toujours plus rapide ce qui implique les nouveaux changements du code de l'application.

## 4.3 Diminuer le temps de développement des versions

Les composants logiciel tant utilisés dans le monde Java facilitent le développement et réduisent le temps de programmation. L'approche objet, utilisée dans la construction des composants est une des forces principales d'un langage de programmation comme Java. A la question « Quelle est la raison qui rend l'approche objet tellement attractive ? » nous pouvons citer la réponse :

« A cette question, les adeptes de l'objet répondent invariablement que les avantages de l'approche objet sont la stabilité de la modélisation par rapport aux entités du monde réel, la construction itérative facilitée par le couplage faible entre les composants et la possibilité de réutiliser des éléments d'un développement à un autre. » [1]

Dans le point suivant nous allons énumérer les spécificités de programmation sur les téléphones mobiles et nous nous concentrons ensuite sur le développement des composants en utilisant l'approche objet.

### 4.3.1 Spécificités des interfaces de programmation

Si le développement des bibliothèques réutilisables réduit grandement les efforts de la programmation, cette méthode reste difficilement exploitable dans le cas de « Java Micro Edition ». Les différences entre les plateformes visées par Micro Edition font que le recours au développement de composants complexes n'est pas encore très courant.

Les raisons principales sont :

1. La taille de la mémoire vive est limitée sur les téléphones portables
2. La puissance de calcul limitée du processeur intégré
3. L'API de programmation réduite
4. Les différences matérielles des téléphones mobiles
5. La prévérification du code binaire qui est effectuée au moment de la compilation et non pas au moment de l'exécution

Ces caractéristiques font que l'application doit être réduite au maximum et que l'intégration des bibliothèques risque de provoquer des anomalies non prévues au moment de son déploiement.

**FoxyTag MIDlet.** Dans le cas de l'application FoxyTag nous avons besoin d'accéder à un appareil GPS depuis un téléphone portable. Notre application, qui a besoin des données provenant d'un appareil GPS peut exploiter deux possibilités :

1. Utiliser un GPS qui est physiquement intégré au téléphone ou
2. utiliser un GPS externe qui est un appareil indépendant

Dans le premier cas, nous accédons au GPS par une interface de programmation Java tandis que dans le deuxième, la communication entre le téléphone et le GPS s'effectue en transmettant les données par la technologie Bluetooth.

Dans le but de déployer FoxyTag sur le plus grand nombre de téléphones portables, nous avons développé une méthode de développement et d'utilisation des bibliothèques Java.

Pour réaliser ce travail, nous regroupons d'abord les composants disponibles dans la spécification de l'API du téléphone ayant un GPS intégré. Ensuite nous comparons ces composants avec ceux de la spécification de l'API Bluetooth. Le résultat nous donne les classes génériques auxquelles nous rajoutons les méthodes communes. Le but est de produire deux classes génériques avec méthodes et l'interface d'accès identique.

### 4.3.2 Modélisation du comportement d'un GPS

Pour modéliser le comportement d'un GPS et les états dans lesquels il se trouve, nous définissons ses propriétés et méthodes en observant le comportement d'un appareil GPS réel. La figure 4.1 montre une classe abstraite GPS.



Figure 4.1 : Classe abstraite GPS

Nous retenons les propriétés :

STATE\_UNKNOWN, AVAILABLE, TEMPORARELY\_UNAVAILABLE, et OUT\_OF\_SERVICE.

Ces propriétés servent à indiquer l'état actuel d'un appareil GPS générique et nous y accédons avec la méthode `getState()`. Plus précisément, ces propriétés indiquent la visibilité entre les satellites et l'appareil GPS et l'état de connexion entre le téléphone et le GPS.

Les méthodes sont les suivantes avec leur utilité :

```
public void connect()  
public void connect(Display display)
```

Elles établissent la connexion entre le GPS et le téléphone avec ou sans interface graphique

```
public void setLocationUpdateInterval(int intervalSec)
```

Configure la fréquence de réception des données du GPS

```
public String[] getData()  
public void locationUpdated()  
public void stateChanged()  
public void addGPSListener(GPSListener listener)  
public void removeGPSListener(GPSListener listener)  
public void notifyGPSListener()
```

Effectuent les interactions avec le GPS et la réception des données

Comme nous l'avons mentionné, cette classe est générique. Elle va donc communiquer avec les classes contenant les particularités de communication avec un appareil GPS réel. Nous allons en avoir deux et pour assurer la communication entre ces classes et notre classe GPS nous définissons une interface : `InternalGPSListener`. La figure 4.2 montre cette interface.

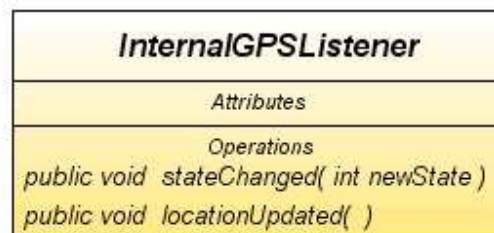


Figure 4.2 : Interface InternalGPSListener

À l'aide de cette interface, notre classe GPS générique peut suivre l'état des classes de communication avec le GPS physique intégré ou Bluetooth et être notifiée des changements de localisation.

### 4.3.3 Modélisation et réalisation du composant « GPSIntegrated »

**API de localisation.** Cette API est de haut niveau et elle est modélisée sans préoccupation des détails de communication avec le GPS. Le GPS est supposé être intégré dans le téléphone et l'interaction entre les deux est supposée être implémentée dans les couches logicielles de plus bas niveau que le Java.

La figure 4.3 montre une MIDlet utilisant l'API de localisation.

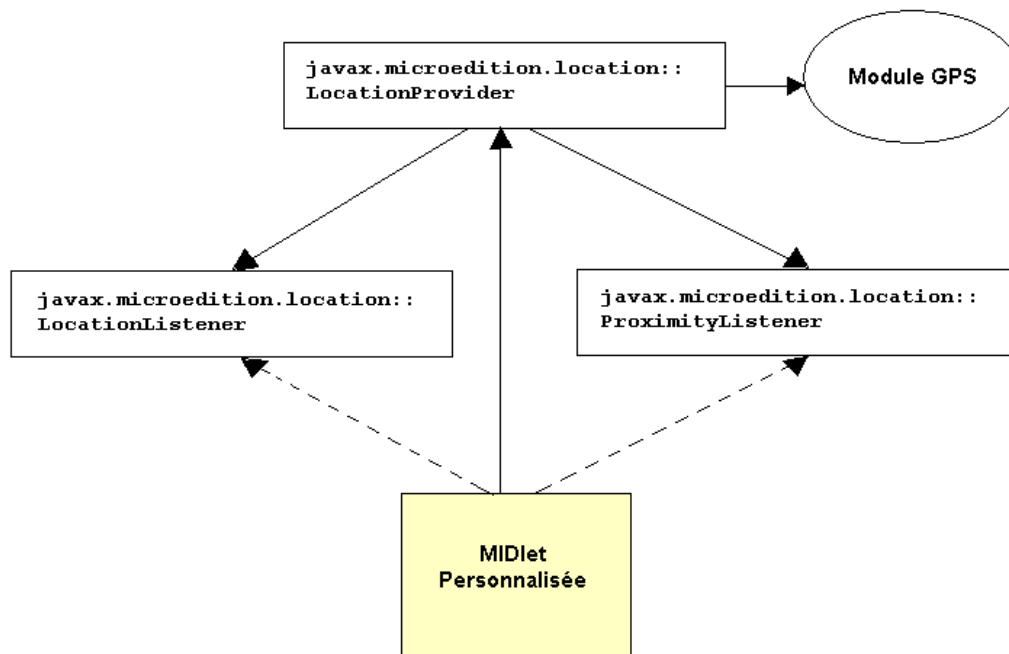


Figure 4.3 : MIDlet utilisant l'API de localisation

Le modèle de base est alors la spécification des classes et méthodes pour accéder au GPS intégré au téléphone.

Dans cette étape nous modélisons une classe qui va communiquer avec notre classe GPS générique. Elle possédera des méthodes en fonction du paquetage employé lors de la communication avec un GPS intégré au téléphone mobile. La communication entre deux classes s'effectue en utilisant une interface interne.

La figure 4.4 montre notre nouvelle classe nommée « GPSIntegrated ».

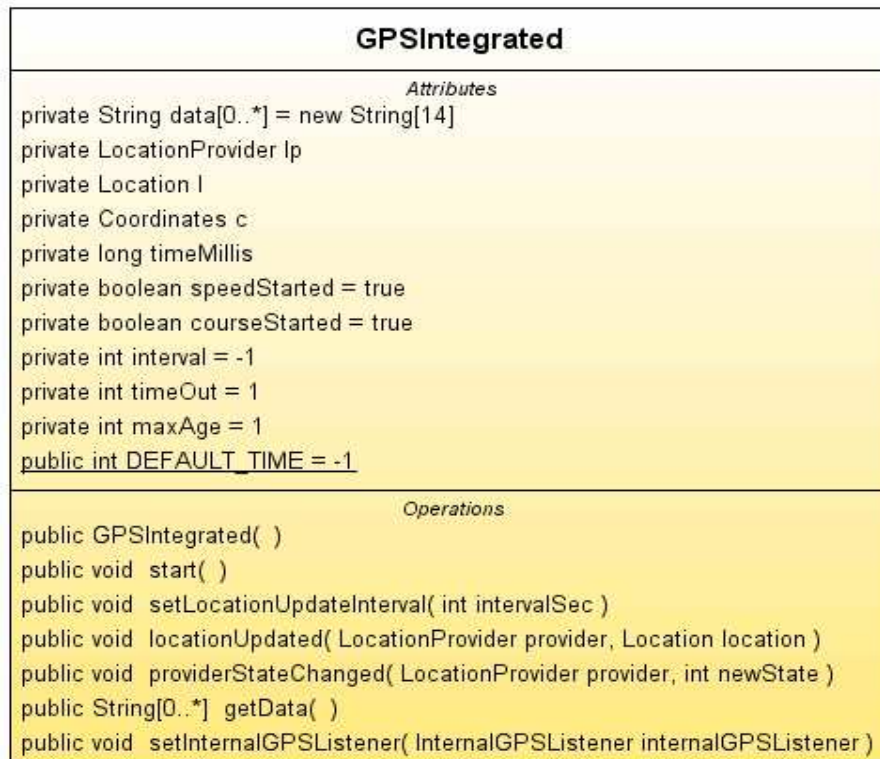


Figure 4.4 : La classe GPSIntegrated

Comme nous pouvons le remarquer sur cette figure, certaines méthodes sont différentes des méthodes de notre classe `GPS`. Leur comportement est différent aussi. Les propriétés de cette classe sont très loin des propriétés de la classe `GPS`. L'intégration de ces propriétés et méthodes est nécessaire dans cette classe, car elles assurent son bon fonctionnement avec les composants de l'API de localisation. Au cours de la modélisation et du développement de cette classe, nous avons déjà essayé d'approcher les noms des méthodes avec les noms des méthodes de la classe `GPS` générique même si leur fonctionnement reste différent. Les classes et les interfaces dont le « `IntegratedGPS` » se sert pour fournir l'information à propos de l'appareil GPS physique sont alors propres à l'utilisation d'un appareil GPS intégré.

Nous pouvons énumérer les méthodes de notre classe « `GPSIntegrated` » et expliquer brièvement leur fonctionnement comme ceci: `GPSIntegrated` :

`getData()`

Pour récupérer des données GPS

`locationUpdated()`

Pour notifier le changement de la position

`providerStateChanged()`

Pour notifier le changement de l'état de fournisseur de localisation

`setInternalGPSListener()`  
Pour rajouter un écouteur de GPS interne

`serLocationUpdateInterval()`  
Pour définir un intervalle de réception des données GPS

`start()`  
Pour démarrer l'écoute de GPS

Le diagramme UML du paquetage `integrated` que nous avons produit et qui décrit les relations principales est montré sur la figure 4.5.

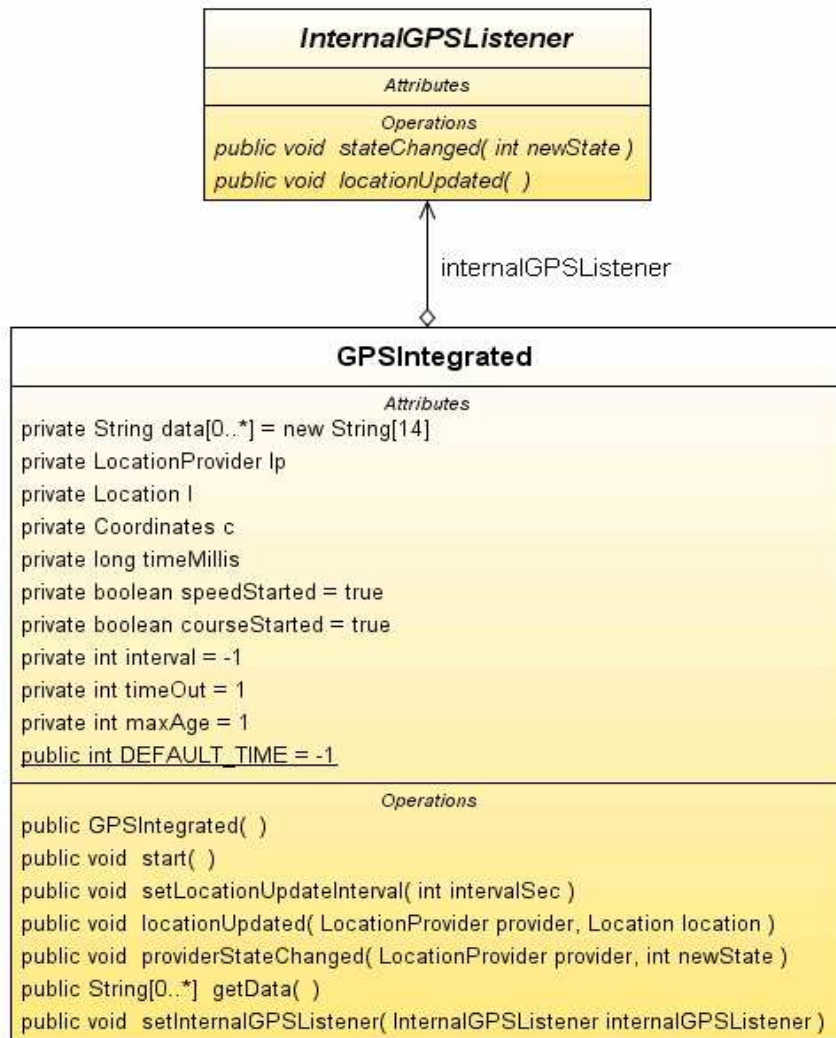


Figure 4.5 : Le diagramme du paquetage `integrated` généré avec l'outil de développement « Netbeans »

À la fin de cette étape, nous avons deux composants :

1. Composant permettant l'utilisation d'un appareil GPS physique intégré au téléphone
2. Composant établissant le comportement d'un appareil GPS générique

La prochaine étape consiste à modéliser et développer un composant permettant l'utilisation d'un appareil GPS Bluetooth externe. Ce composant va exploiter la technologie Bluetooth et va contenir les spécificités par rapport à cette technologie.

#### 4.3.4 Modélisation et réalisation du composant « GPSBluetooth »

**Technologie Bluetooth.** La technologie Bluetooth est conçue pour la communication sans fil avec les appareils à la proximité. Les fonctionnalités qu'on trouve dans la spécification d'API Bluetooth pour « Java Micro Edition » sont montrés sur la figure 4.6 :

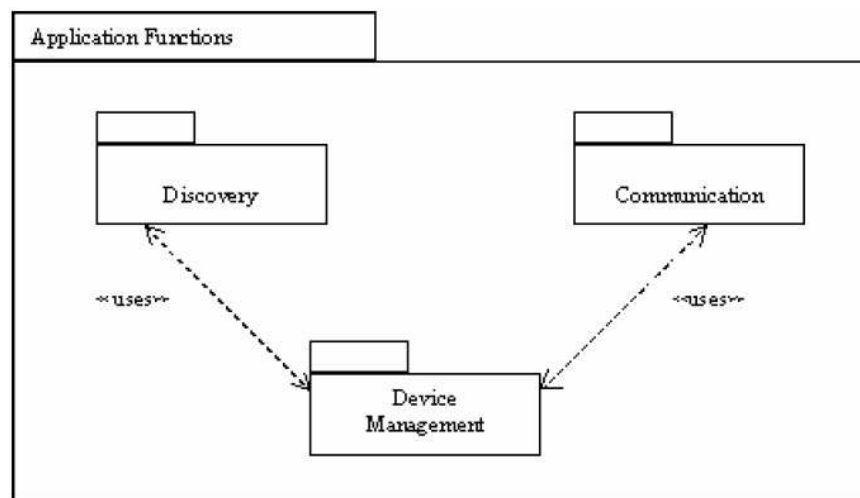


Figure 4.6 : Les fonctionnalités de l'API Bluetooth

(Source : « Java Specification Request JSR-82 »)

La figure montre les fonctionnalités très génériques et la communication est subdivisée en plusieurs protocoles.

La partie de découverte (« Discovery » sur la figure) suppose plusieurs étapes :

- Initiation de l'appareil Bluetooth sur le téléphone mobile
- La découverte des noms des appareils à proximité
- La découverte des caractéristiques et des services que ces appareils nous mettent à disposition

La communication (« Communication » sur la figure) à son tour s'établit de manière suivante :

- L'échange des adresses physiques entre les appareils
- Envoi et réception des données par une des couches des protocoles de communication

L'envoi et la réception des données peuvent passer par plusieurs couches de communication. Nous exploitons dans notre travail uniquement la couche L2CAP du protocole qui sert à envoyer les données binaires.

L'API de programmation Bluetooth en « Java Micro Edition » est modélisée sur cette architecture. Trois paquetages composent cette API, ils sont montrés sur la figure 4.7.

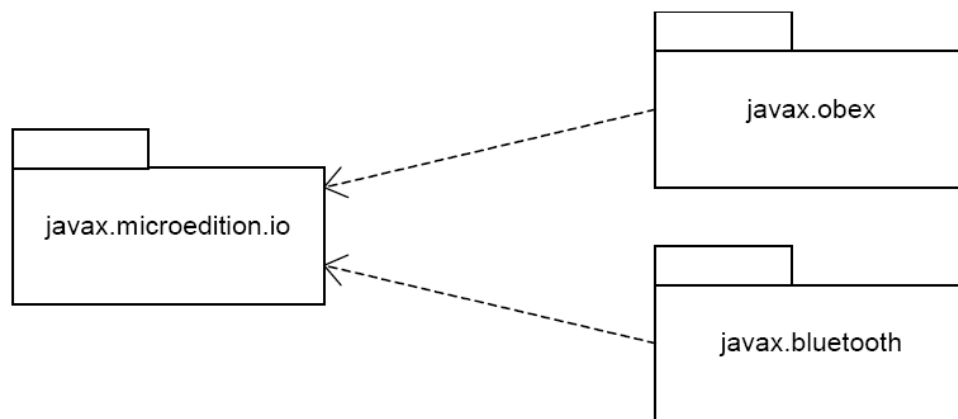


Figure 4.7 : Les paquetages de l'API Bluetooth

(Source : « Java Specification Request JSR-82 »)

Pour la communication avec un GPS Bluetooth nous utilisons `javax.microedition.io` et `javax.bluetooth`.

Nous résumons alors les classes et leurs méthodes principales contenues dans le paquetage `javax.bluetooth` en groupes suivants :

- Les classes et interfaces de découverte des appareils distants
- Les classes et interfaces d'enregistrement de services des appareils
- Les classes et interfaces de gestion de l'appareil Bluetooth local

Le diagramme d'une application utilisant cette API Bluetooth est montré sur la figure 4.8.

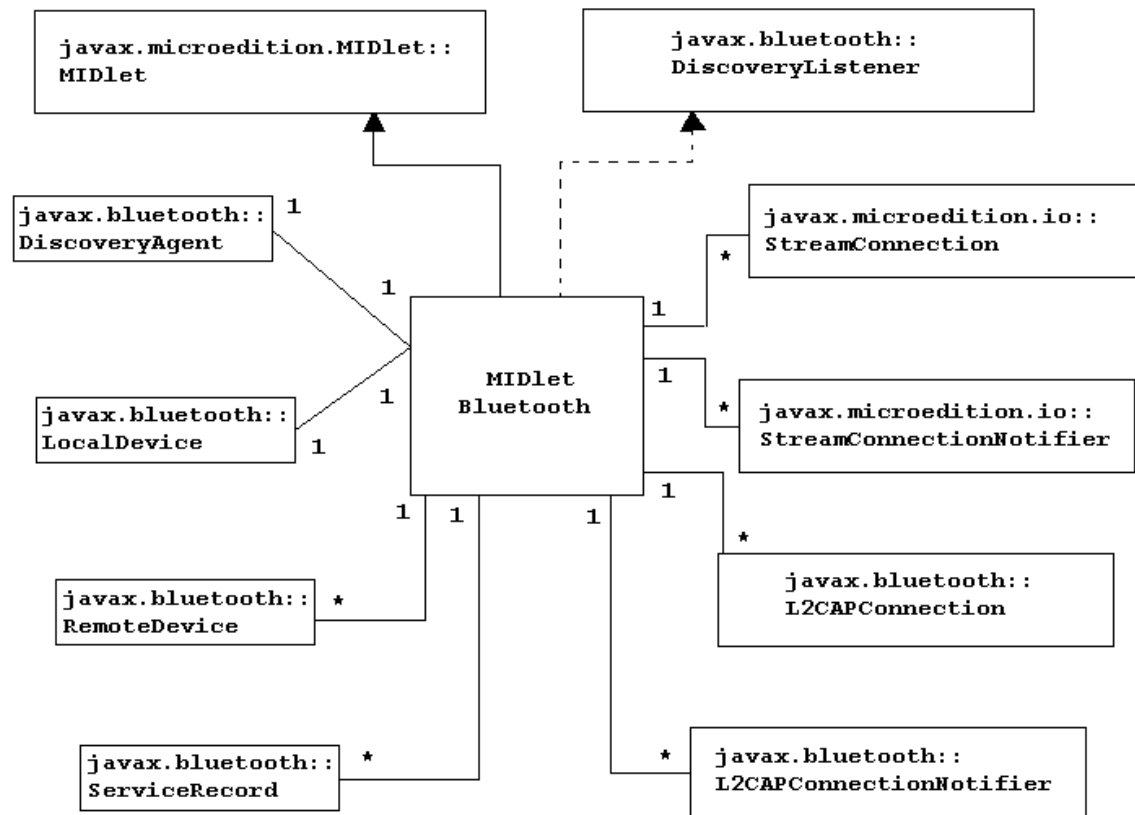


Figure 4.8 : MIDlet utilisant l'API Bluetooth

A priori même si les données géographiques dont nous avons besoin sont les mêmes si on utilise un GPS intégré ou un GPS Bluetooth, il s'agit de deux spécifications d'API complètement différentes. Le protocole de communication Bluetooth est conçu pour transmission de n'importe quelles données par les ondes radio alors que la spécification d'API de localisation est centrée sur les appels aux données GPS.

Nous avons commencé la prochaine étape par modélisation et développement des composants utilisant les protocoles Bluetooth pour accéder au GPS physique. Ces composants vont effectuer les opérations suivantes :

1. Découverte de GPS physique
2. Découverte des services fournis par cet appareil GPS
3. Établissement de la connexion avec cet appareil GPS
4. Récupération des données géographiques du GPS en continu
5. La détection des changements des états du GPS physique

Les informations récupérées du GPS physique par ces composants seront transmises à la nouvelle classe que nous avons nommée « BluetoothGPS ». Nous allons éviter les explications détaillées sur tous les composants dans ce travail. Les explications s'avèrent trop longues et concernent plus le travail technique de développement de la technologie Bluetooth que le déploiement des applications. Ces composants et leur relation avec la nouvelle classe « BluetoothGPS » sont montrés sur la figure 4.9. Leur code source et la documentation sont rajoutés en annexe. La classe que nous avons produite à partir des spécificités de la technologie Bluetooth est présentée plus en détail dans ce travail.

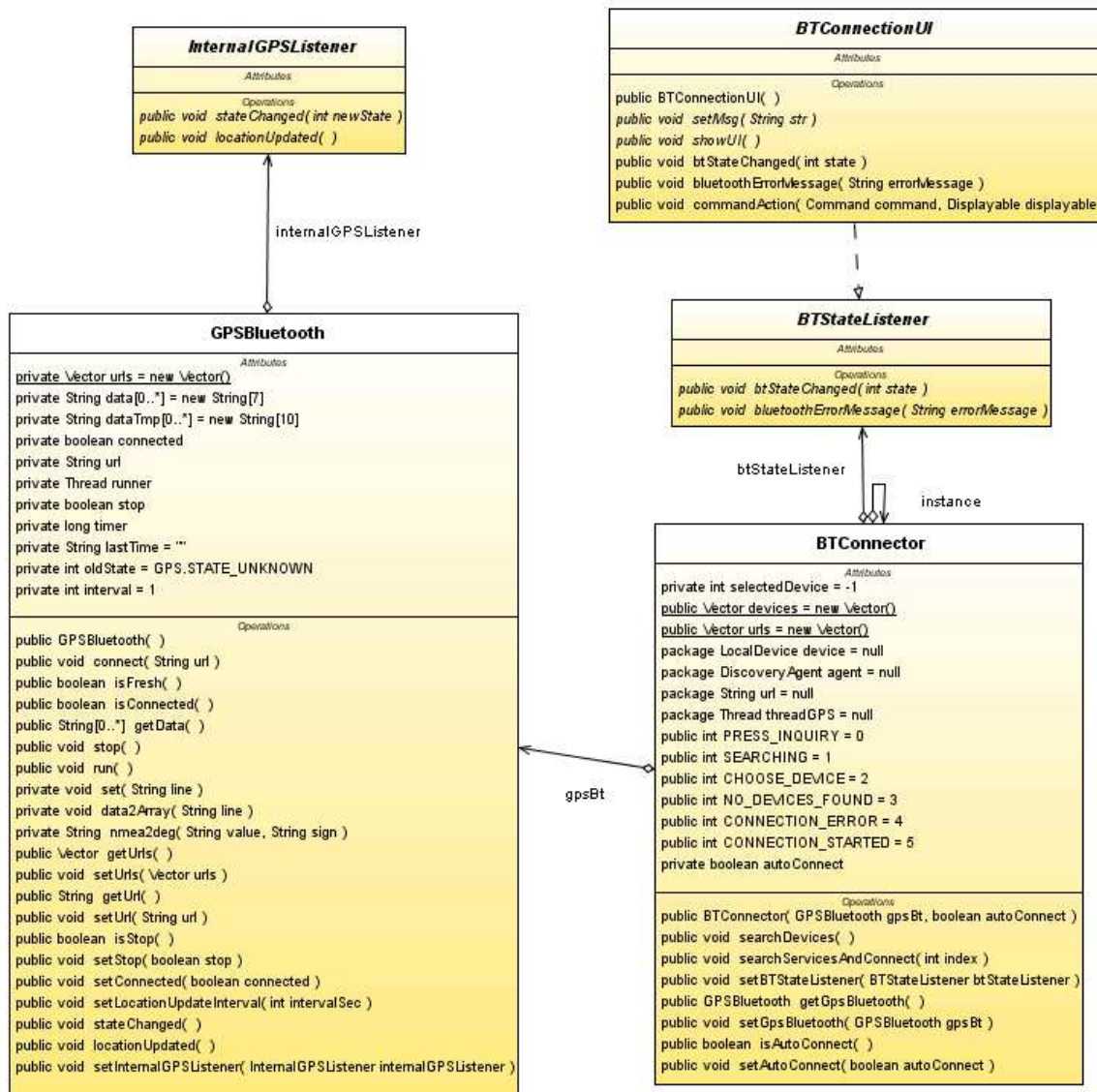


Figure 4.9 : Les composants réalisés et regroupés dans le paquetage bluetooth

La classe d'accès au GPS intégré et ses méthodes n'existent pas dans la spécification de connexion au Bluetooth car cette dernière est générique et peut être utilisée pour recevoir n'importe quels type et format de données. D'autre part, nous avons besoin d'une interface graphique pour permettre à l'utilisateur de choisir l'appareil GPS auquel il veut connecter le téléphone. Dans un premier temps, nous préparons les classes utilitaires qui nous permettront de réaliser une classe la plus proche possible de la classe de référence. Ensuite nous modélisons et implémentons la classe spécifique pour un appareil GPS Bluetooth. Nous présentons uniquement la classe nouvellement produite et ses méthodes ici :

```
class GPSBluetooth:
```

```
connect(String url)
```

Pour afficher l'interface graphique et pour se connecter à un URL de GPS

```
getData()
```

Pour recevoir les données du GPS

```
isConnected()
```

```
isFresh()
```

```
isStop()
```

Trois méthodes pour accéder à l'état du GPS

```
locationUpdated()
```

Pour notifier le changement de la position

```
run()
```

Pour démarrer le Thread de connexion

```
setInternalGPSListener()
```

Pour rajouter un écouteur de GPS interne à cette classe

```
setLocationUpdateInterval()
```

Pour définir l'intervalle de réception des données GPS

```
setUrl(String url)
```

Pour rajouter un URL de connexion

```
stateChanged()
```

Pour notifier les changements d'état du GPS

```
stop()
```

Pour arrêter le processus d'écoute de GPS

Même si nous voulions rapprocher les deux composants au cours de modélisation il reste toujours des différences au niveau de méthodes entre ces deux classes.

Notre classe `GPSBluetooth` doit utiliser l'interface `java.lang.Runnable` d'où provient la plus grande différence. L'utilisation de cette interface ne peut pas être évitée en raison de l'utilisation des opérations de connexion.

Chaque connexion doit être démarrée dans un processus différent pour éviter les blocages de l'interface graphique. L'implémentation de cette interface nous introduit la méthode `run()` et nous rajoutons aussi une méthode `stop()` pour arrêter la connexion. La nécessité de la méthode `stop()` provient aussi de l'incompatibilité entre les téléphones portables. Nous avons détecté au cours de développement que la méthode `join()` prévue pour arrêter un processus démarré dans un objet `java.lang.Thread` n'existait pas sur les téléphones Nokia.

Après avoir tenté d'intégrer les similarités au cours de développement des deux classes `GPSIntegrated` et `GPSBluetooth` il reste difficile d'utiliser l'une ou l'autre de manière interchangeable dans une application. L'étape suivante consiste donc à regrouper les fonctionnalités des deux composants dans un seul qui cachent au développeur ce fonctionnement différent de ses sous composants. Nous retenons donc les méthodes principales qui sont communes et nous implémentons l'interface commune pour assurer la communication identique avec les deux composants. L'interface déjà mentionnée avec les fonctionnalités des méthodes :

Interface `InternalGPSListener` :

`locationUpdated()`

Pour notifier les changements de la position

`stateChanged()`

Pour notifier les changements d'état du GPS

Cette interface contient uniquement les méthodes indiquant l'état de GPS et les changements de position.

Après avoir réalisé les composants pour la communication spécifique avec les appareils GPS nous pouvons implémenter la classe GPS générique qui peut être utilisée par les développeurs des applications. Nous implémentons cette classe de manière suivante :

`class GPS implements InternalGPSListener`

`addGPSListener()`

Pour rajouter un écouteur des états de GPS

`connect()`

Pour établir la connexion au GPS

`getData()`

Pour récupérer les données GPS

`getState()`

Pour découvrir l'état du GPS

`locationUpdated()`

Pour recevoir les notifications sur les mises à jour de localisation

`notifyGPSListeners()`

Pour notifier les objets écouteurs du GPS

```
removeGPSListener()
```

Pour désabonner un objet qui écoute les événements

```
setLocationUpdateInterval()
```

Pour définir l'intervalle des mises à jour de localisation

```
stateChanged()
```

Pour être averti des changements d'états du GPS

La relation entre les classes `GPS`, `GPSIntegrated` et `GPSBluetooth` est montrée sur la figure 4.10.

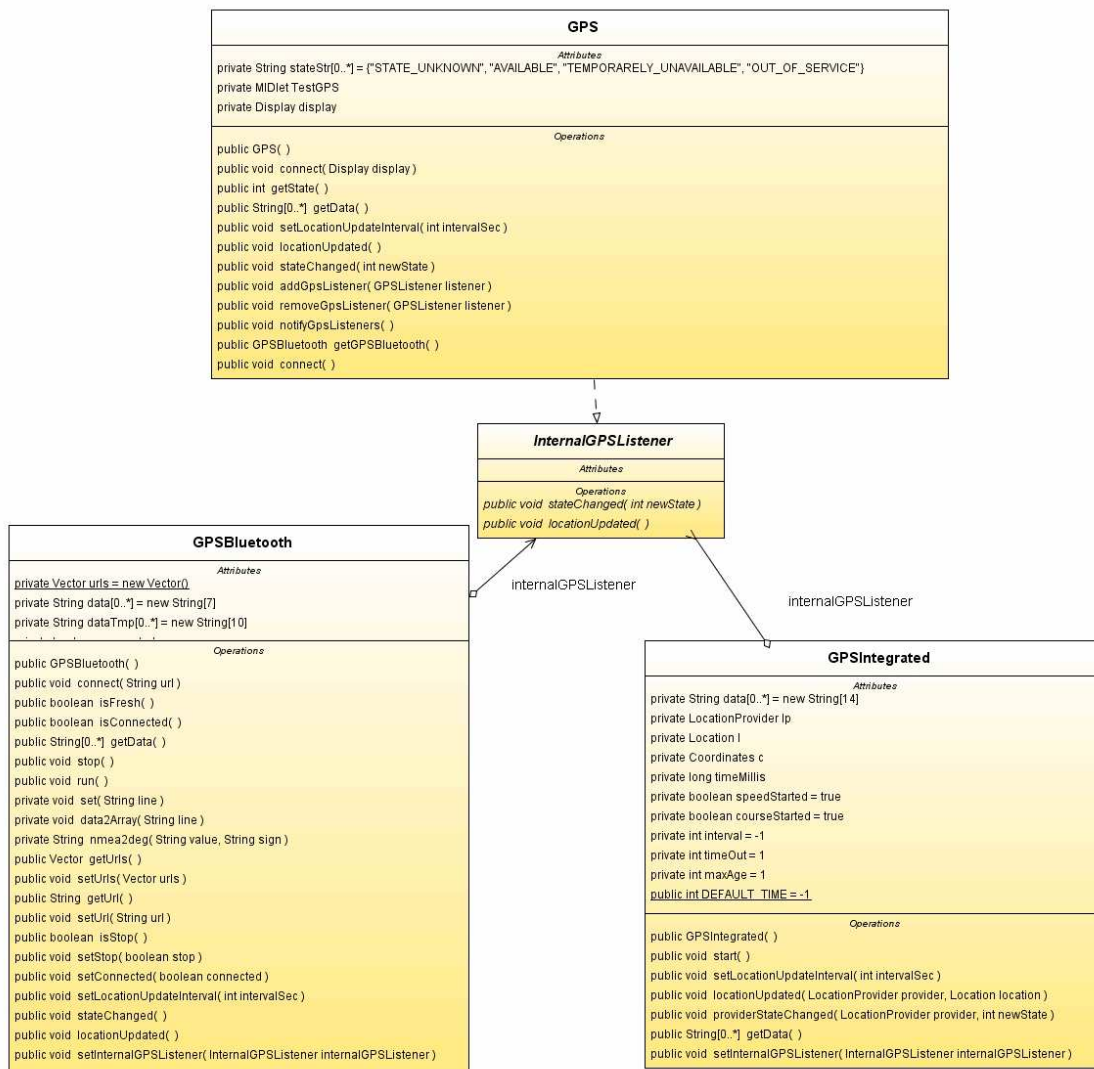


Figure 4.10 : Relation entre les classes `GPS`

Notre classe GPS nouvellement produite regroupe ainsi que les fonctionnalités communes des deux composants `GPSIntegrated` et `GPSBluetooth`. Du point de vue de la modélisation, c'est un composant utilisable dans une application. Coté pratique nous ne pouvons pas encore l'utiliser dans une MIDlet. Le problème provient des API disponibles sur un téléphone mobile. Si ce dernier n'intègre pas l'API de localisation par exemple, ce code sera considéré comme invalide à l'exécution et l'application ne pourra pas s'exécuter.

La prochaine étape consiste à faire deux paquetages : un pour chaque API disponible. Nous les appelons paquetage « Bluetooth » et paquetage « Integrated ». Chaque paquetage regroupe les classes propres au type de connexion et une copie de notre classe générique GPS.

De cette manière, l'utilisation de cette API est la même depuis l'application principale et la seule modification à effectuer est l'importation des paquetages en fonction des téléphones ciblés. L'illustration d'utilisation est suivante :

a) Pour les téléphones avec un GPS intégré

```
import com.foxytag.integrated.GPS;  
GPS gps = new GPS();  
//appel aux méthodes du GPS
```

b) Pour les téléphones avec un GPS Bluetooth

```
import com.foxytag.bluetooth.GPS;  
GPS gps = new GPS() ;  
//appel aux mêmes méthodes du GPS
```

Comme nous l'avons mentionné, la dernière étape consiste à utiliser cette librairie en fonction d'API du téléphone ciblé. Ainsi, les applications différentes peuvent avoir le rôle de gérer l'affichage de données GPS ou de les exploiter dans une autre forme de l'interface graphique. Le point important à souligner est l'exclusivité de choix de paquetage si les deux API de base à savoir « Bluetooth » et « Localisation » ne sont pas présents sur un téléphone. Le cas d'intégration de cette librairie GPS avec l'application FoxyTag est présenté à l'aide du diagramme UML sur la figure 4.11.

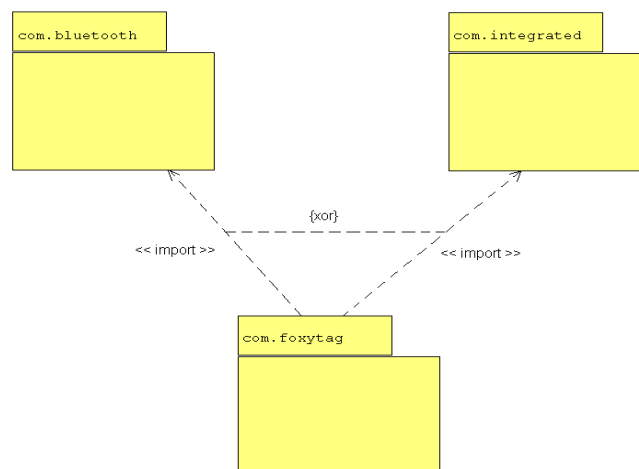


Figure 4.11 : Diagramme UML de l'utilisation de la librairie GPS avec FoxyTag

#### **4.4 Réutilisation des composants**

Dans le monde de développement en Java standard, la meilleure pratique serait de produire une librairie unique réutilisable. La plus grande différence entre ces deux éditions Java est détectée au niveau de gestion des exceptions. Le code source pour « Java Micro Edition » est pré vérifié durant la compilation. L'existence des objets utilisant les API inexistantes provoque l'arrêt immédiat de l'exécution de la MIDlet même si les objets en question sont références à l'aide des conditions logiques déterminant l'existence des interfaces de programmation.

Concevoir une librairie en un paquetage unique a été encore impossible pour des raisons suivantes:

- La taille d'une application doit rester petite en raison de ressources limitées des appareils
- Impossibilité de prévoir les exceptions et de les récupérer – le code inconnu est traité comme invalide et le manager des applications gère ce code de manière imprévisible
- L'introduction du code inutilisé augmente le risque des « bugs » imprévus

La librairie produite et utilisée de cette manière a tout de même des grands avantages au niveau de gestion des versions et de réutilisation des composants. Dans le cas de développement d'une application qui vise les appareils avec une des deux API disponibles il est préférable de garder le code source dans le même projet de l'environnement de développement et d'utiliser l'outil d'obfuscation. Son rôle dans ce cas est d'intégrer uniquement les librairies références par l'application principale au moment de compilation. De cette manière, le développeur gère une version de développement en effectuant les modifications minimales au moment de générer une distribution de l'application.

#### **4.5 Méthode de développement des composants retenue**

En conclusion de ce chapitre, nous pouvons souligner que la pratique de réutilisation du code source sous la forme des librairies pour assurer une plus grande portabilité des applications Java sur les téléphones portables est une manière de développement très spécifique. La réalisation de ces librairies doit s'effectuer en connaissant les spécifications des interfaces de programmation et en tenant compte de pré vérification du code au moment de la compilation. À l'aide d'abstraction du comportement d'un appareil GPS il est également possible de produire une classe abstraite et indépendante du mode de connexion. L'apport de cette librairie et la vitesse accrue de développement des applications de géolocalisation et la simplicité de gestion des versions en fonction des interfaces de programmation disponibles sur les différents appareils.

## 5. Les modèles de distribution du contenu sur les téléphones portables

Les modèles de distribution du contenu sur les téléphones portables sont nombreux, mais toujours dépendants du modèle de téléphone. Ces modèles supportent souvent l'installation des applications comme le déplacement des fichiers à l'aide des différentes technologies. Les plus courantes sont : « Bluetooth », « Infrarouge », « MMS » et téléchargement à l'aide d'un navigateur web. Même si l'utilisation de ces technologies est très répandue rien ne peut garantir leur disponibilité sur les différents téléphones. La gestion des droits d'auteur sous la forme de « Digital Right Management » rajoute encore une complexité à la distribution du contenu. Ce sujet de gestion de droits numériques reste en dehors de ce travail et nous nous focalisons uniquement sur la spécification et les standards de distribution des applications « Java Micro Edition ».

Dans ce chapitre nous commençons par résumer les besoins d'effectuer les prétests d'une application sur les différents téléphones portables avant de commencer son déploiement massif.

Seconde étape résume le modèle de référence de distribution des MIDlets. Ensuite nous essayons de regrouper les différentes plateformes web représentatives selon leurs caractéristiques principales et en relation avec « Java Micro Edition ». Finalement, nous comparons ces plateformes existantes avec les besoins de développeurs.

Notre motivation est de découvrir et spécifier les solutions permettant d'effectuer les prétests d'une MIDlet avant de commencer son déploiement.

### 5.1 Modèle de référence pour la distribution des MIDlets

La recherche des plateformes qui nous aident à effectuer les prétests des MIDlets commence d'abord par étudier les spécifications existantes pour cette tâche.

La seule spécification publiée par « Sun Microsystems » et inchangée depuis 24 octobre 2003 est nommée « J2EE Client Provisioning Specification ». En même temps l'entreprise a donné à disposition de la communauté de développeurs une implémentation de référence. De nos jours il existe une seule implémentation de cette spécification rajoutant les fonctionnalités « peer to peer ». Il s'agit de la plateforme « JVending » nommée par analogie de la machine de distribution des produits. Cette spécification est basée exclusivement sur la distribution des MIDlets et d'autre contenu numérique. La distribution des applications non vérifiées et leur test s'avèrent impossibles. L'architecture de l'implémentation l'interdit en intégrant les fichiers de description qui doivent contenir le modèle du téléphone ciblé. Ce modèle ne couvre en aucun cas le besoin de prétests.

## 5.2 Déploiement des MIDlets via les réseaux sans fil

En réalité la distribution des MIDlets s'effectue plutôt depuis les sites web traditionnels destinés à l'utilisation avec un ordinateur de bureau. Même si la spécification de distribution utilisant la technologie Java n'a pas eu le grand succès, le modèle de distribution sous le nom « Over The Air » ou plus souvent OTA est respecté par tous les fabricants de téléphonie mobile et reste le seul standard assurant l'installation d'une MIDlet. Un schéma simplifié de la distribution des applications utilisant OTA est présenté sur la figure 5.1 :

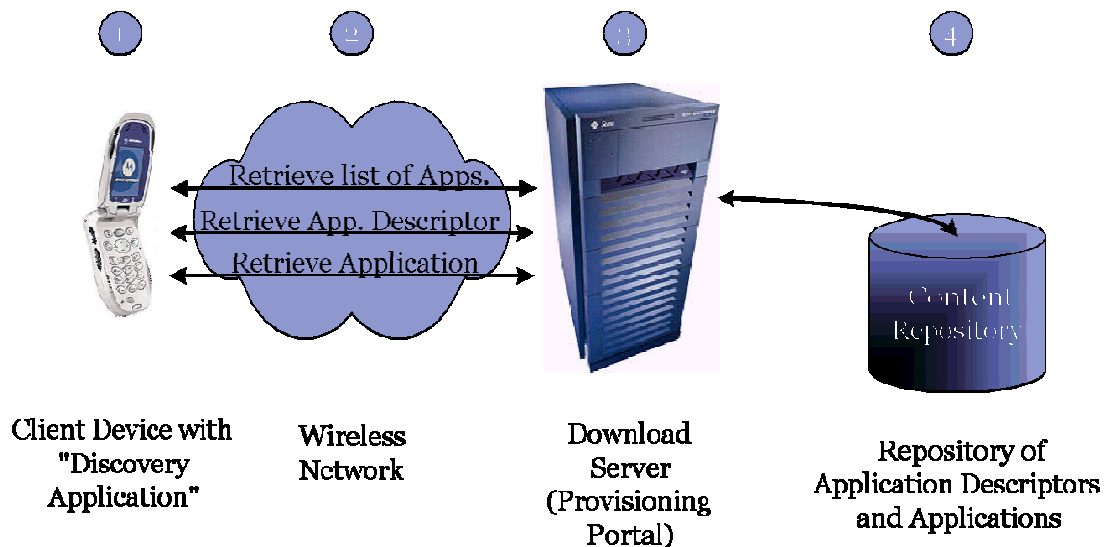


Figure 5.1 : Schéma de déploiement des MIDlets « Over The Air »

(Source : « Sun Microsystems »)

La première étape du protocole de téléchargement OTA suppose la découverte des applications à l'aide du navigateur web du téléphone. En pratique on recourt à l'utilisation des navigateurs des ordinateurs de bureau en raison de petite taille de l'interface d'utilisateur sur les téléphones portables.

Deuxième étape est la lecture du fichier de description de l'application par le gestionnaire des applications du téléphone portable. Le gestionnaire des applications compare les possibilités du téléphone avec leur description.

Si les ressources pour le bon fonctionnement de l'application demandée dans le fichier de description peuvent être satisfaites, le gestionnaire des applications sur le téléphone procède à leur téléchargement et l'installation.

Dans le cas contraire l'application ne doit pas être téléchargée ni installée sur le téléphone par le gestionnaire des applications.

Cette architecture suppose que les fichiers de description et le Java Archives ne sont pas forcément hébergés sur le même serveur. Le gestionnaire des applications sur un téléphone portable peut lire un fichier de description et télécharger l'archive Java depuis un autre site. Elle ne spécifie pas non plus des détails sur les portails ni sur

les serveurs des fichiers. Alors, cette architecture peut être utilisée avec n'importe quel serveur web et/ou serveur des fichiers.

Le concept de développement d'une application pour une plateforme de téléphone spécifique est au cœur de l'architecture. Ce fait augmente le besoin d'effectuer les prétests d'une application avant son déploiement.

Nous avons alors de nombreux sites web hébergeant des applications « Java Micro Edition ». Ils regroupent des caractéristiques et des fonctionnalités parfois très différentes. Dans le but de retrouver les possibilités actuelles d'effectuer les prétests, nous étudions plus en détail les plateformes web existantes et en relation avec « Java Micro Edition ».

### 5.3 Les besoins de prétests pour le déploiement des MIDlets

Nous avons souligné dans l'étude des spécifications de « Java Micro Edition » l'importance de bien connaître les disponibilités des interfaces de programmation Java sur les téléphones mobiles. Après cette étape d'étude, nous découvrons la disponibilité des interfaces de programmation optionnelles à l'aide des applications de test. Ces applications de test nous permettent d'explorer les possibilités de développement de l'application principale et affinent le choix des téléphones sur lesquels nous voulons la déployer.

Pour faciliter le développement, nous disposons de nombreux outils de développement intégrant le simulateur du téléphone. Il s'agit d'un simulateur du téléphone générique qui est donc différent de chaque téléphone spécifique. Après avoir développé une application sur un émulateur la méthode traditionnelle consiste à effectuer les tests sur les téléphones ce qui implique d'avoir en possession ces téléphones. Il va de soit qu'il est impossible d'avoir tous les téléphones portables du marché à n'importe quel moment. Les différences imprévues de comportement d'une application sur un modèle du téléphone augmentent aussi significativement les efforts de correction de l'application.

Dans le cas de l'application FoxyTag nous rencontrons la gestion différente des objets `java.lang.Thread` détectés entre les téléphones Sony Ericsson et Nokia. Ces objets sont nécessaires pour établir n'importe quel type de connexion et l'application comme FoxyTag ne peut pas fonctionner sans eux. La spécification MIDP 2.0 caractérise les deux modèles du téléphone et même si les méthodes des objets susmentionnés sont y obligatoires, sur le modèle Nokia nous ne disposons pas des méthodes `stop()` et `join()`.

L'implémentation personnalisée de ces méthodes ne s'avère pas difficile et nous pouvons décider de concentrer les efforts de correction et d'amener notre application à supporter les différences entre deux téléphones.

Dans le cas de FoxyTag nous rencontrons un autre exemple avec le téléphone Motorola RAZR V3. Même s'il implémente la spécification de l'API Bluetooth ce téléphone rajoute une complexité imprévue : la MIDlet qui accède à l'API Bluetooth doit être signée. Le coût de certificat de signature numérique s'avère trop cher pour

le projet universitaire et nous décidons d'abandonner les corrections de l'application FoxyTag pour ce téléphone.

En fonction du nombre des utilisateurs potentiels ayant un certain modèle de téléphone et des caractéristiques de ces derniers nous pouvons donc décider d'effectuer les corrections de l'application. Ces corrections sont dès fois trop grandes et nous pouvons les abandonner. Dans certains cas le nombre des utilisateurs avec un modèle de téléphone et plus significatif et nous décidons de développer une version de l'application pour ce modèle.

On ne peut toujours pas garantir que notre application sera bien déployée sur tous les téléphones que nous avons ciblés au préalable. Dans le déploiement d'une application, nous avons alors un besoin clair d'analyser les différences imprévues sur les téléphones afin de pouvoir prendre la décision d'effectuer des corrections ou d'abandonner certains modèles de téléphones.

La question que nous pouvons nous poser dans cette étape est : comment détecter ces différences imprévues de comportement d'une application sur les différents téléphones ? Nous pouvons chercher la réponse à cette question sur les différents portails de développeur ou les portails de distribution des MIDlets.

#### **5.4 Caractéristiques et regroupement des principaux portails de Java ME**

Nous proposons de regrouper les sites web destinés au « Java Micro Edition » dans des catégories en fonction de leur conception et utilité. Pour le faire, nous retenons les critères suivants :

1. Le contenu pour les téléphones mobiles
2. La possibilité d'échanger les commentaires entre les utilisateurs de la plateforme
3. La possibilité pour le développeur de distribuer une application
4. L'impacte du modèle de financement de la plateforme sur le déploiement des applications
5. La possibilité d'effectuer les prétests d'une application

Dans cette comparaison nous prenons les plateformes destinées à la distribution du contenu pour les téléphones mobiles. Nous excluons les plateformes qui mélangent le contenu pour les applications de bureau avec celles pour les téléphones mobiles. Nous retenons uniquement les plateformes dites représentatives définies selon leur notoriété sur le moteur de recherche.

Selon ces critères nous estimons qu'on peut regrouper les plateformes dans 4 catégories:

1. Les plateformes de développement collaboratif
2. Les plateformes de distribution commerciale des MIDlet
3. Les plateformes utilitaires
4. Les plateformes de prétest d'applications Java

### 5.4.1 Les plateformes de développement collaboratif

*Le contenu pour les téléphones mobiles* sur ce type de la plateforme est très spécialisé. Il s'agit de communauté de développeurs qui sont créés pour soutenir le développement d'un certain langage de programmation. Concernant le langage Java les plus connus sont : le site de développeurs de « Sun Microsystems », le site de « Micro Java Network » ou encore « J2ME Polish ». Le contenu sur ces plateformes est sous la forme du code source des applications et pour les déployer il faut une connaissance des outils de compilation Java.

*La possibilité d'échanger les commentaires entre les utilisateurs de la plateforme* est offerte sous la forme de commentaires sur les tutoriaux.

Les utilisateurs peuvent y laisser leurs commentaires libres à propos des applications open-source et tutoriaux qui les accompagnent. De plus, les forums de support sont intégrés dans ce type de plateforme.

*La possibilité pour le développeur de distribuer une application* est possible uniquement pour les applications open-source. La distribution OTA n'est pas supportée.

*L'impacte du modèle de financement de la plateforme sur le déploiement des applications* est négligeable. Les plateformes sont financées par les publicités et le déploiement n'est pas facilité. Il s'effectue en téléchargeant le code source et en le compilant sur les ordinateurs de bureau.

*La possibilité d'effectuer les prétests d'une application* est possible indirectement. Les commentaires des utilisateurs sont le seul indicateur sur le comportement de l'application sur les différents téléphones mobiles.

### 5.4.2 Les plateformes de distribution commerciale des MIDlets

Les plateformes de distribution commerciale des MIDlets sont surtout les sites des opérateurs de la téléphonie mobile. Les applications sont proposées par intermédiaire de l'opérateur réseau et une charge est toujours prélevée de leur part pour chaque téléchargement. Dans ce groupe appartiennent les sites web d'opérateurs Orange, Swisscom ou Sunrise.

*Le contenu pour les téléphones mobiles* est très divers. Il est regroupé surtout selon les différentes marques et modèles des téléphones mobiles. On y trouve le contenu numérique comme les fonds d'écran, les thèmes, les sonneries, les MIDlets ou les applications dépendant de la plateforme mobile. Le bon fonctionnement des MIDlets après le paiement doit être assuré avant leur téléchargement ce qui amène les opérateurs à faire un tel regroupement et de distribuer les MIDlets surtout par MMS.

*La possibilité d'échanger les commentaires entre les utilisateurs* n'existe pas sur ce type de plateforme.

*La possibilité pour le développeur de distribuer une application* est assurée par l'intermédiaire de l'opérateur réseau.

*L'impacte du modèle de financement de la plateforme sur le déploiement est positif. L'opérateur réseau doit s'assurer du bon fonctionnement d'une application avant le paiement.*

*La possibilité d'effectuer les prétests d'une application n'existe pas sur ce type de la plateforme. Les prétests doivent s'effectuer par développeurs sans relation avec ce type de la plateforme.*

### **5.4.3 Les plateformes utilitaires**

Le groupe des plateformes hébergeant les MIDlets et les informations sur la possibilité de leur développement sur les différents téléphones portables. Ce type de plateforme héberge surtout les MIDlets de « benchmark » développées par le propriétaire de la plateforme. Dans ce groupe nous prenons comme exemple le site « MIDlet Review » et « JBenchmark ».

*Le contenu pour les téléphones mobiles sur ce type de plateforme est orienté vers les applications Java en raison de spécificités de l'installation sur les téléphones qui compliquent la tâche d'utilisateur d'une part et en raison de méconnaissance des API disponibles sur les différents téléphones de la part de développeurs. À part les applications Java on y retrouve les informations sur les API Java des téléphones.*

*La possibilité d'échanger les commentaires entre les utilisateurs sur les applications Java n'existe pas.*

*La possibilité pour le développeur de distribuer une application est inexistante sur ce type de la plateforme.*

*L'impacte du modèle de financement de la plateforme sur le déploiement des applications ces plateformes offrent souvent les espaces membres contre les paiements. Ainsi, les membres disposent des informations sur les API Java ce qui a un impacte positif sur le déploiement des applications.*

*La possibilité d'effectuer les prétests d'une application n'existe pas sur ce type de la plateforme.*

### **5.4.4 Les plateformes de prétests des applications Java ME**

Parmi ces plateformes nous pouvons retrouver un seul représentant : « Get Jar ». Il s'agit d'une seule plateforme qui propose explicitement les prétests des MIDlets.

*Le contenu pour les téléphones mobiles est concentré sur les applications pour les téléphones mobiles. On y trouve les applications pour à peu près tout système d'exploitation des téléphones mobiles.*

*La possibilité d'échanger les commentaires entre les utilisateurs est directe pour chaque application. De plus, les utilisateurs peuvent mettre une note sur chaque application. Tous les utilisateurs doivent être enregistrés pour commenter les applications.*

*La possibilité pour le développeur de distribuer une application est facile est entièrement gérée par la plateforme. Il suffit d'être un développeur enregistré sur la plateforme pour envoyer n'importe quelle application. Les utilisateurs de cette application disposent de téléchargements par téléphone portable ou le navigateur web d'un ordinateur de bureau. La distribution par OTA est très bien supportée.*

*L'impacte du modèle de financement de la plateforme sur le déploiement des applications est d'une grande importance. Le site est financé par les publicités. Les publicités sont concentrées sur les applications pour les téléphones mobiles. Le modèle de prétests attire toujours des nouveaux développeurs qui continuent à distribuer leur application sur la même plateforme après les prétests. Ainsi, le site attire les développeurs d'une part et les utilisateurs dits testeurs prêts à tester une application.*

*La possibilité d'effectuer les prétests d'une application est très sophistiquée, mais ne laisse pas le moyen de gérer les prétests pour plusieurs versions d'une même application. Cette possibilité d'effectuer les prétests est disponible uniquement pour les applications Java. Un test s'effectue en trois phases :*

1. Le développeur met à disposition une MIDlet et il indique la période de test qui est d'une à deux semaines, et les modèles de téléphone à utiliser pour le test
2. Les testeurs enregistrés sur la plateforme téléchargent cette application et reçoivent le formulaire de test avec une obligation de le remplir
3. Une semaine après l'expiration de la période de test le développeur reçoit les résultats de test

## **5.5 Comparaison des plateformes existantes avec le besoin de prétest**

D'après les besoins de prétests d'une application, nous pouvons remarquer que les plateformes actuelles ne sont pas adaptées pour cette tâche. Selon les critères identifiés dans le point précédent nous pouvons résumer les défauts des plateformes existantes.

*Le contenu pour les téléphones mobiles est centré sur les versions finales. La seule exception est la plateforme « Get Jar » qui offre la possibilité d'effectuer un test par application. Le défaut de cette plateforme est le manque de possibilité de tester plusieurs versions de la même application.*

*La possibilité d'échanger les commentaires entre les utilisateurs sur les plateformes actuelles est basée sur l'opinion des utilisateurs finaux. Le contact entre les développeurs et les testeurs ou utilisateurs finaux est effectuée sur les forums de discussion de manière informelle. Rien n'oblige de spécifier le modèle du téléphone utilisé avec cette application. De cette manière, les opinions restent très générales et n'aident pas beaucoup le développeur de l'application.*

*La possibilité pour le développeur de distribuer une application est bonne sur les plateformes actuelles. Néanmoins dans la plupart des cas rien n'oblige les testeurs*

de donner leur opinion sur l'application et le retour de commentaires est insuffisant dans ce cas. La seule plateforme de test identifiée « Get Jar » possède cette fonctionnalité.

*L'impacte du modèle de financement de la plateforme sur le déploiement des applications* favorise uniquement les applications finies sans la possibilité de faire les tests. Dans le cas des plateformes commerciales tout est prévu pour la bonne distribution alors que dans la distribution gratuite même une installation via OTA n'est pas disponible. Pour certaines marques de téléphone comme Samsung cela complique la tâche d'installation est les testeurs sont souvent incapables d'installer les applications depuis la plateforme.

*La possibilité d'effectuer les prétests d'une application* est absente sur les plateformes sauf sur « Get Jar ». Le défaut de cette plateforme sont les formulaires de test qui sont basés uniquement sur la question d'acceptabilité : « Est-ce que je suis prêt à payer cette application ? ». L'impossibilité de tester plusieurs versions d'une application est le deuxième défaut de cette plateforme.

À la fin de ce chapitre nous pouvons conclure que les prétests des applications Java Micro Edition sont importants pour maximiser le nombre de téléphones compatibles avec une application. Dans certains cas nous pouvons décider d'abandonner les modifications pour un modèle de téléphone, mais le plus souvent les modifications sont mineures et on peut les corriger. Actuellement nous n'avons pas la possibilité d'effectuer les prétests de manière efficace sur les plateformes de distribution existantes. La solution la plus sûre est d'avoir tous les téléphones qui existent sur le marché pour effectuer les prétests ce qui s'avère difficile en raison des avancements rapides dans la téléphonie mobile est le nombre de nouveaux modèles de téléphone.

## 6. Plateforme collaborative

L'idée d'avoir à disposition une plateforme permettant de proposer des applications sous forme de projet avec la possibilité d'agréger des versions et l'évaluation de ces dernières par une communauté nous a semblé utile dès le début. Les plateformes actuellement disponibles permettent uniquement de déposer une application finie, destinée à l'utilisateur finale, sans pouvoir partager des connaissances pour améliorer l'une des versions du projet. En somme, pas de retour d'information de la part de l'utilisateur, si ce n'est celui de l'acceptation, ou le rejet de l'utilisation de l'application, avec une perte de confiance si le choix se porte sur la seconde option.

### 6.1 Caractéristique et besoins

La plateforme web est donc composée d'un groupe de développeurs spécialisés dans le domaine des applications mobiles, avec des connaissances dans le langage de programmation Java, et les API rattachées à l'environnement mobile. Les membres de ce groupe vont pouvoir mettre à disposition de la communauté leurs projets, pour qu'ils puissent être téléchargés et testés. Après une utilisation standard, le membre ayant téléchargé l'application pourra remplir un formulaire lui permettant d'effectuer une critique constructive de l'application. Ce rapport est directement transmis au développeur qui peut ainsi corriger son application, puis évaluer par une note le rapport de l'utilisateur. Le concept de cette évaluation des comptes rendus est utilisé pour évaluer la qualité de l'analyse et de la critique faite par l'utilisateur. Ainsi, une liste des utilisateurs proposant les meilleures analyses peut être appelée, ce qui permet un classement mettant en valeur les utilisateurs. Les différentes solutions d'évaluations rencontrées ne valorisent pas ou rarement les utilisateurs, au profit des projets et des applications.

Les besoins d'une plateforme d'échange d'idées et de partage de projets nous ont interpellés au court de notre travail notamment après les résultats obtenus lors de notre implication dans le déploiement du projet FoxyTag. Nous avons pu constater que pour assurer une portabilité maximale d'une application il ne suffisait pas de respecter les spécifications émises par Sun Microsystems, mais il était également nécessaire d'effectuer des tests détaillés sur les téléphones mobiles ciblés. La difficulté dans cette approche de test sur des plateformes ciblées est qu'un développeur doit avoir à disposition les marques et modèles des téléphones, car les émulateurs ne reproduisent que très rarement la situation réelle. Ce nombre de téléphones mobiles peut facilement être doublé si l'on prend en compte les modèles modifiés et préconfigurés par les opérateurs des réseaux téléphoniques.

Au cours du développement et de la distribution de FoxyTag, le forum technique qui était conçu pour fournir une aide aux utilisateurs, nous a permis de recenser un groupe d'utilisateurs voulant participer à l'évolution du programme, et se prêtant au jeu du testeur prêt à indiquer les problèmes rencontrés. Notre avantage était d'avoir un utilisateur très réactif, et indulgent. Son avantage est évidemment la compatibilité de l'application avec son téléphone mobile.

Avec l'avancement du travail, nous avons réussi à créer une petite communauté de testeurs possédant des téléphones mobiles différents. On ne peut pas qualifier ces utilisateurs comme des experts ni comme des développeurs en raison de leur qualification, mais comme des « utilisateurs précoces », motivé et passionné par les applications mobiles.

D'autre part, la distribution d'une application nécessite sa présence sur plusieurs plateformes de distribution pour atteindre le maximum de téléchargements pendant le développement où l'on n'a aucune possibilité de créer une telle communauté.

Après avoir fait l'audit des différentes plateformes de téléchargement hébergeant des applications Java ME, aucune ne satisfaisait l'idée de base, à savoir le test et l'évaluation d'une application mobile.

Nous retenons donc les fonctions suivantes à implanter dans le prototype :

- Chaque développeur doit être le testeur des applications
- La plateforme doit avertir les utilisateurs inscrits à propos de chaque nouvelle version de l'application dans leur catégorie préférée
- Un résultat de test doit être rempli et envoyé après chaque téléchargement de l'application
- Les développeurs peuvent noter les résultats de tests
- Par rapport aux notes données sur les résultats de tests, les développeurs peuvent classer leurs « meilleurs testeurs »
- Chaque présentation de l'application doit afficher les téléphones de tests indiqués par développeur
- Un développeur doit pouvoir gérer plusieurs versions de l'application sur la plateforme

D'après les spécifications nous décidons de stocker les applications Java et leurs versions dans un système des fichiers. Cela permet d'intégrer plus simplement les téléchargements OTA. Les noms des répertoires contenant des versions sont générés à partir de la base de données relationnelle [annexe 13.2].

Les données sur les utilisateurs et les résultats de tests seront implémentés dans une base de données relationnelle. La gestion de droit d'accès est aussi générée à partir de la base de données.

## **6.2 Définition des interfaces**

Le site présente une page d'accueil sur laquelle figure un menu situé sur la gauche incluant les catégories des applications disponibles, le classement des meilleurs utilisateurs, et la section privée de l'utilisateur après qu'il se soit identifié.

Le profil de cette section privée lui permet d'indiquer les téléphones mobiles qu'ils possèdent pour indiquer aux autres membres de la communauté sur quelles plateformes il peut tester leurs applications. Un suivi des catégories favorites est également disponible, pour le prévenir des nouveaux projets inscrits dans les catégories définies.

Dans la section des projets, une liste des projets initiés est affichée, avec la possibilité de les éditer. L'on peut rajouter une version au projet, effacer une version, ou effacer le projet dans sa totalité. [Annexe 13.4.1]

Lorsqu'un utilisateur télécharge une application, une demande d'évaluation est automatiquement insérée dans la section des évaluations du menu privé. [13.4.2]

Cela permet au testeur de faire rapport de test de l'application, ainsi le développeur peut en améliorer le fonctionnement ou la compatibilité. [Annexe 13.4.3]

Le développeur va donc recevoir le rapport de test et noter ce dernier de 1 à 10. Il est également indiqué avec quel téléphone mobile le test a été effectué. [Annexe 13.4.4]

La plateforme permet un classement des meilleurs utilisateurs, basé sur le résultat des rapports de tests qu'ils ont effectués. [Annexe 13.4.5]

### **6.3 Développement du prototype**

Du point de vue technique, nous avons décidé de développer la plateforme web en PHP, un langage souple qui permet de gérer les accès à une base de données, et donc de pouvoir générer des pages de manière dynamique d'après une requête SQL spécifique. Un fichier de style CSS (Cascading Style Sheet) permet d'avoir une homogénéité des différentes interfaces qui composent la plateforme web. Le développement a été effectué avec l'IDE Eclipse 3.2 agrémenté du plug-in PHP permettant de travailler directement sur le code source avec l'affichage simultané du résultat.

## 7. Travaux apparentés

Dans ce chapitre, nous allons présenter les différents travaux apparentés au sujet du déploiement d'application mobile.

### 7.1 Test de l'environnement d'un téléphone mobile

Les travaux similaires à FoxyTest sont deux sites de recensement des téléphones mobiles compatible avec la JSR 82. Il s'agit d'abord javablueetooth.com qui liste des téléphones mobiles sans proposer de client de test, et le projet TastePhone 0.6.0 créé par le Java User Group ou JUG. Le projet est disponible sur une plateforme collaborative mise en place par Sun Microsystems, via le portail java.net. Le lien du projet est : <http://tastephone.dev.java.net>. Le site allemand AreaMobile propose une liste de téléphone mobile avec les spécifications Java ME.

### 7.2 Plateforme collaborative

La plateforme web proposée par Sun Microsystems est la « Mobile and Embedded Community », réalisée et mise à disposition vers la fin de l'année 2006. C'est lors de la conférence JavaOne à San Francisco que Sun annonce le passage du code source de Java en « Open Source », et la création de cette communauté. Il s'agit d'une plateforme collaborative permettant de proposer des projets, et d'en discuter dans un forum. Le projet le plus relevant est phoneME qui est la version Open Source de l'implémentation de la plateforme Java ME de Sun Microsystems basée sur les spécifications CDC et CLDC.

## 8. Évolutions futures

Concernant l'application FoxyTest, cette dernière peut être intégrée comme module à une application qui ne permet pas d'être exécutée si l'un des critères n'est pas rempli. Une variable de type RecordStore avec comme nom `testok=True` ou `testok=False` et qui serait modifiée seulement si le téléphone mobile passe les tests de compatibilité. Cela permet d'éviter des problèmes d'exécution en garantissant que le client possède tous les prérequis nécessaires à la bonne utilisation de l'application.

Il est également possible d'ajouter des paramètres de test pour que d'autres éléments d'un téléphone portable soient vérifiés. Le concept peut également être porté vers la plateforme Microsoft Windows Mobile, en effet, il possède également des paramètres très différents en ce qui concerne la pile Bluetooth employée pour piloter la puce en elle-même. Cela dit, l'environnement de développement est .NET, et le passage d'un code vers l'autre nécessite certaines adaptations.

La plateforme peut évoluer pour supporter par exemple un système de crédit permettant aux membres d'en acquérir après avoir réalisé des évaluations de projet, pour à leur tour les faire valoir sur des projets qu'ils ont créés. Une liste des testeurs favoris est également intéressante pour un développeur. Un forum est intéressant à mettre en place pour permettre des discussions sur un projet, ou pour des remarques sur des versions d'un projet.

## 9. Conclusion

L'évolution des nouveaux modèles est rapide, ce qui réduit la portabilité des applications. Les besoins d'une méthode simple et reproductible de déploiement sont nécessaires. Ce travail recense les étapes à mettre en place lors du développement et déploiement d'une application sur les téléphones mobiles différents avec un minimum d'effort.

Les fabricants des téléphones mobiles définissent les spécifications des API de programmation en Java avec lesquels il faut traiter lors de la création d'une application. Ces spécifications peuvent contenir des exceptions, car elles ne sont pas supportées par tous les téléphones mobiles. L'étude de ces spécifications est la première étape pour maximiser la portabilité des applications Java ME.

Ce travail de master nous a permis d'appréhender le problème du déploiement d'applications mobiles sous l'angle de la théorie, mais également celui de la pratique. Après avoir programmé une application en Java ME permettant d'analyser et de recenser l'environnement matériel et l'accès à ce dernier par la machine virtuelle Java présente dans les téléphones mobiles, nous avons utilisé les résultats de ces tests pour observer quel téléphone mobile présentait les caractéristiques nécessaires au fonctionnement de l'application nécessitant l'accès au Bluetooth.

La réalisation de l'évolution de la librairie GPS nous a permis d'obtenir de meilleurs résultats quant à la détection et la communication avec des boîtiers GPS/Bluetooth externe. La librairie traite un maximum d'exceptions concernant les incompatibilités des API Java présentes sur le téléphone mobile. Cette construction de la librairie permet, en respectant les spécificités de Java ME, de limiter le nombre de versions pour les différents téléphones mobiles.

Les besoins de tests sont importants avant la distribution sur le marché d'une application Java. Pour éviter au développeur la possession de la majorité des téléphones mobiles du marché, nous avons proposé une plateforme collaborative spécialisée. La finalité du travail nous a donc portés vers le développement d'une plateforme web, proposant la création de projet et de version pour les développeurs d'applications orientées mobiles, agrémentées d'un système d'évaluation et de critique des projets permettant au développeur de les améliorer.

## 10. Références

- [1] Sun Microsystems, *Java Community Process*, 2004  
Accessible en mai 2007 sur <http://jcp.org/en/procedures/jcp2>
- [2] Sun Microsystems, *Java™ Technology for the Wireless Industry*  
Accessible en mai 2007 sur <http://jcp.org/en/jsr/detail?id=185>
- [3] Sun Microsystems, *J2ME™ Connected, Limited Device Configuration*, 2000  
Accessible en mai 2007 sur <http://jcp.org/en/jsr/detail?id=30>
- [4] Sun Microsystems, *Connected Limited Device Configuration 1.1*, 2002  
Accessible en mai 2007 sur <http://jcp.org/en/jsr/detail?id=139>
- [5] Sun Microsystems, *Mobile Information Device Profile 2.0*, 2002  
Accessible en mai 2007 sur <http://jcp.org/en/jsr/detail?id=118>
- [6] Sun Microsystems, *Mobile Media API*, 2003  
Accessible en mai 2007 <http://jcp.org/en/jsr/detail?id=135>
- [7] Wikipedia, *Java (Langage)*, 2007  
Accessible en mai 2007 [http://fr.wikipedia.org/wiki/Java\\_\(langage\)](http://fr.wikipedia.org/wiki/Java_(langage))
- [8] FoxyTag  
Accessible en mai 2007 <http://www.foxytag.com>

# 11. Figures

Figure 1.1 : Les composants Java ME

Figure 2.1 : Diagramme d'état du cycle de vie d'une MIDlet

Figure 2.2 : Les composants logiciels du téléphone mobile

Figure 2.3 : Over The Air

Figure 3.1 : Schéma des connectivités de FoxyTest

Figure 3.2 : Diagramme UML de FoxyTest

Figure 3.3 : Table install\_notify de la base de données

Figure 3.4 : Table testreport de la base de données

Figure 3.5 : Compatibilité JSR 82 par constructeur

Figure 4.1 : Classe abstraite GPS

Figure 4.2 : Interface InternalGPSListener

Figure 4.3 : MIDlet utilisant l'API de localisation

Figure 4.4 : La classe GPSIntegrated

Figure 4.5 : Le diagramme du paquetage integrated généré avec l'outil de développement « Netbeans »

Figure 4.6 : Les fonctionnalités de l'API Bluetooth

Figure 4.7 : Les paquetages de l'API Bluetooth

Figure 4.8 : MIDlet utilisant l'API Bluetooth

Figure 4.9 : Les composants réalisés et regroupés dans le paquetage bluetooth

Figure 4.10 : Relation entre les classes GPS

Figure 4.11 : Diagramme UML de l'utilisation de la librairie GPS avec FoxyTag

Figure 5.1 : Schéma de déploiement des MIDlets « Over The Air »

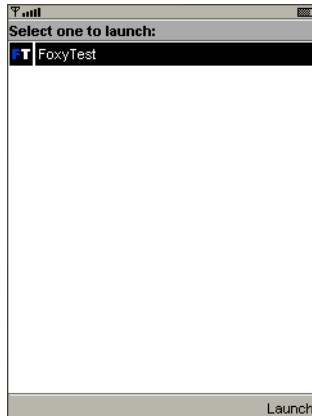
## 12. Bibliographie

- [1] Pierre-Alain Muller, *Modélisation objet avec UML*, page 10, Eyrolles Deuxième tirage 1997
- [2] Kim Topley, *J2ME in Nutchell*, O'Reilly First Edition 2002
- [3] Bruce Hopkins, Ranjith Antony, *Bluetooth for Java*, Apress 2003
- [4] Ramez Elmasri et Shamkant Navathe, *Conception et architecture des bases de données (4<sup>e</sup> édition)*, Pearson Education 2004
- [5] Pat Niemeyer & Jonathan Knudsen, *Introduction à Java (2<sup>e</sup> édition)*, O'Reilly 2002
- [6] Bruno Delb, *J2ME Applications Java pour terminaux mobiles*, Eyrolles 2002
- [7] Robert Virkus, *Pro J2ME Polish*, Apress 2005
- [8] FoxyTag, <http://www.foxytag.com>
- [9] Bruce Hopkins, Ranjith Antony, *Bluetooth for Java*, Apress 2003
- [11] Sing Li and Jonathan Knudsen, *Beginning J2ME From Novice to Professional*, Apress Third Edition 2005
- [10] Jonathan Knudsen, *Wireless Java Developing with J2ME*, Apress 2003
- [12] Orange Suisse, <http://www.orange.ch>
- [13] Swisscom Mobile, <http://www.swisscom-mobile.ch>
- [14] Sunrise Suisse, <http://www.sunrise.ch>
- [15] Midet Review, <http://www.midlet-review.com>
- [16] Jbenchmark, <http://www.jbenchmark.com>
- [17] GetJar.com, <http://www.getjar.com>
- [18] JavaBluetooth.com, <http://www.javablueetooth.com/jsr82devices.html>
- [19] Micro Java Network, <http://www.microdevnet.com/>
- [20] J2ME Polish, <http://www.j2mepolish.org>

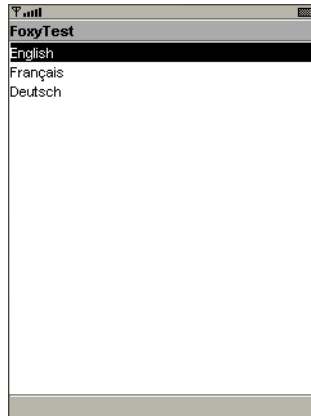
# 13. Annexes

## 13.1 Impression d'écran de FoxyTest

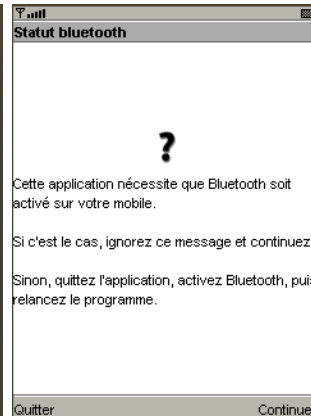
1



2



3



Sélection de l'application, de la langue et alerte Bluetooth.

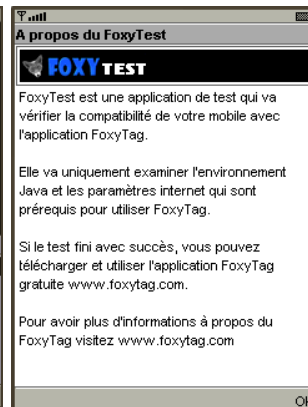
4



5



6

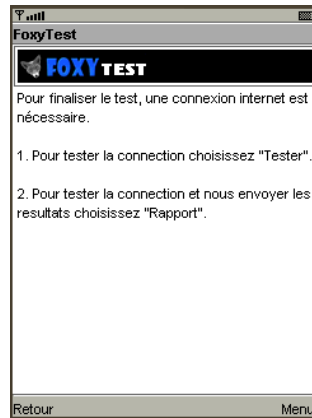


Affichage de l'invite, sélection « Menu » puis « A propos » revenir avec « Ok » puis « Démarrer ».

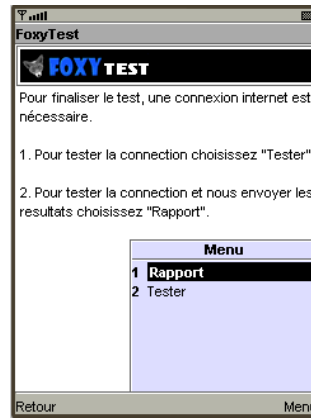
7



8

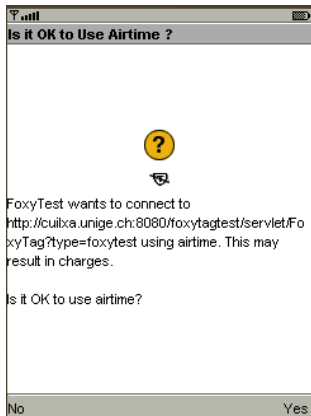


9

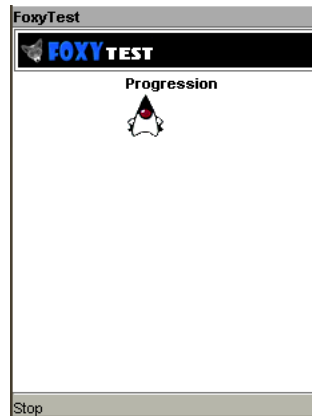


Indication de la marque et du modèle du téléphone mobile. Sélection du « Tester » par le menu ou « Rapport ».

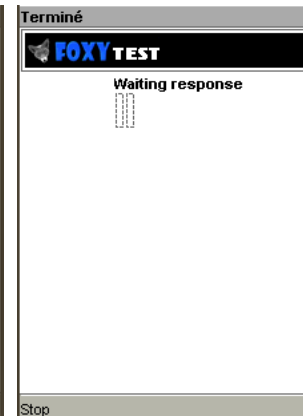
10



11

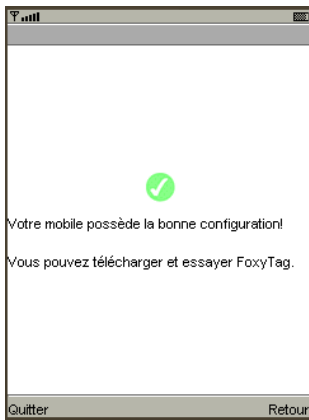


12

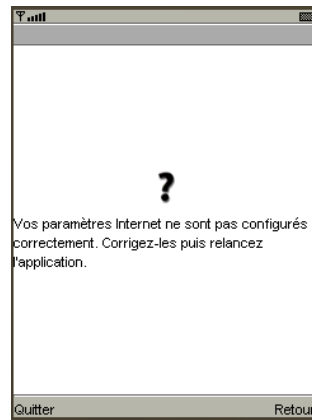


Alerte d'utilisation d'une connexion internet, et demande d'acceptation par l'utilisateur. Progression de la requête HTTP, attente d'une réponse.

13



14



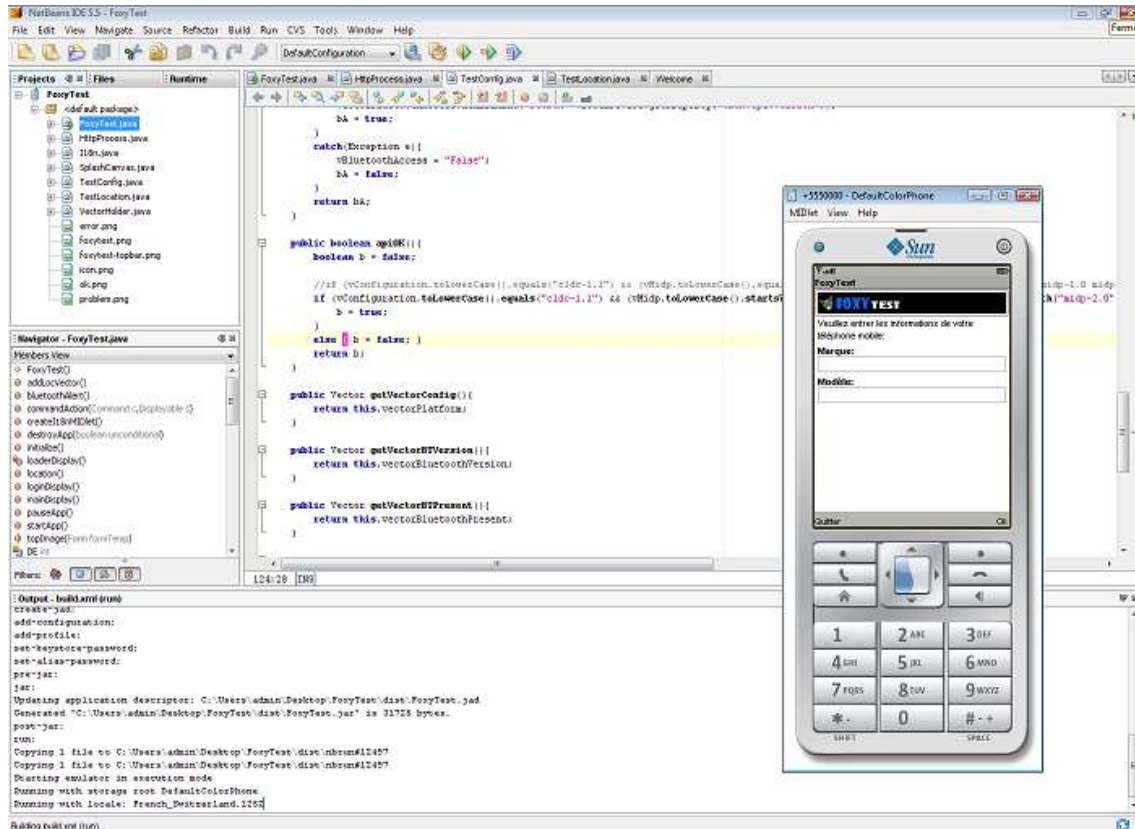
15



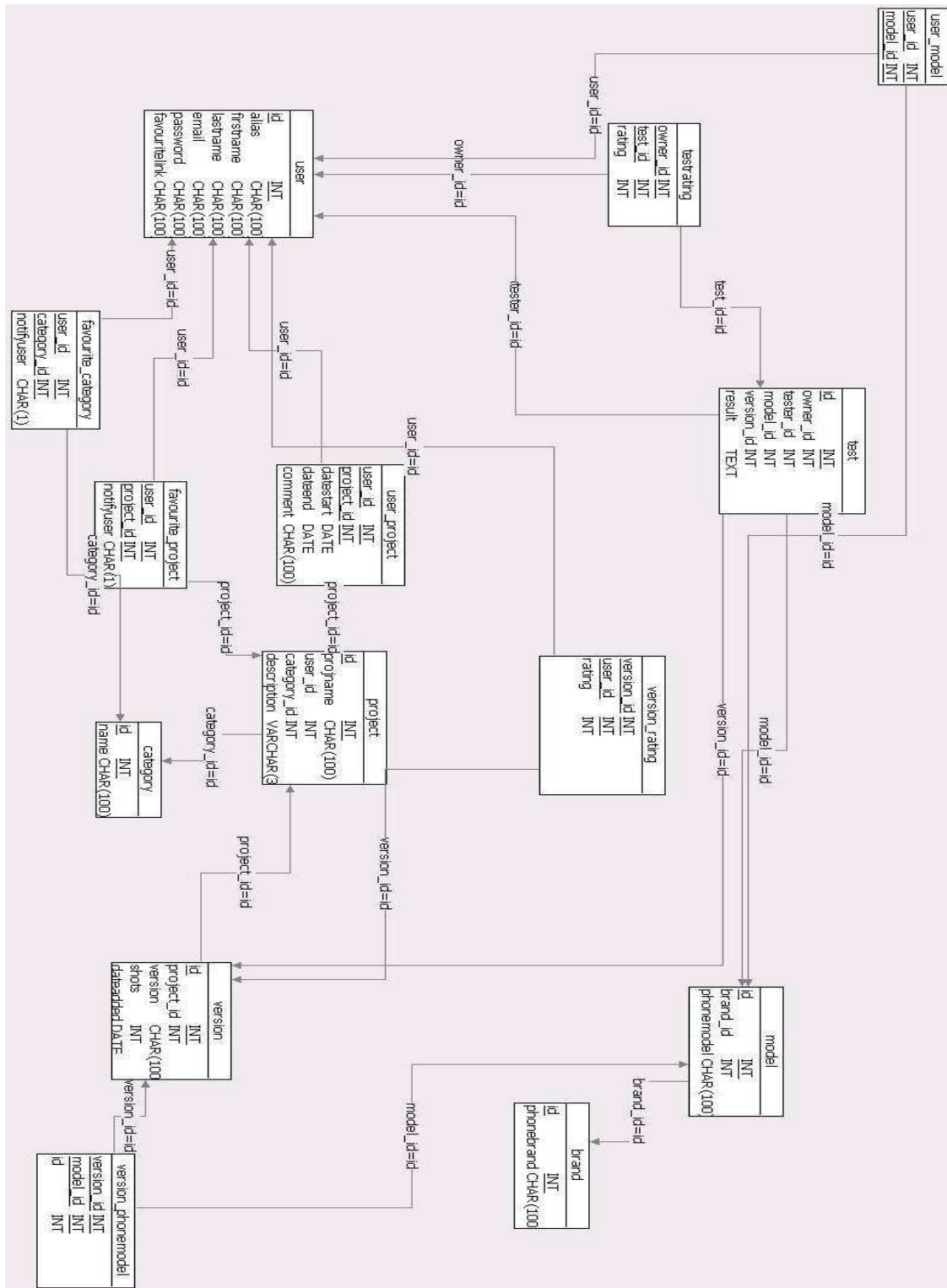
Réussite de l'accès Bluetooth, ou problèmes de paramètres GPRS, ou échec de l'utilisation du Bluetooth.

## 13.2 NetBeans 5.5 et l'émulateur WTK 2.5 CLDC

Une impression d'écran de l'application avec le logo inspiré de celui de FoxyTag [8], et en arrière plan NetBeans 5.5 avec le fichier TestConfig.java en édition.



### 13.3 Schéma de la base de données pour la plateforme de test



## 13.4 Aperçu de la plateforme collaborative

### 13.4.1 Gestion de projet et des versions

**Catégorie**

Localisation

Jeux

Utils

Autres

---

Best users

**Mon profile**

Profile

Mes projets

Evaluations

## Projet - edition


Nom du projet:  
**FoxyTest**

Editer Description (max 300 char) :

Desc

296 characters restants

Editer Image



Catégorie:

Localisation:

### Versions

Version: 1.0 [Editer / Rajouter les téléphones de test](#) [Effacer cette version](#)

### Rajouter une version

Fichier jar:

Fichier jad:

## 13.4.2 Rapport de test après téléchargement

Accueil Logout

Catégorie

- Localisation
- Jeux
- Utils
- Autres

Best users

Mon profile

- Profile
- Mes projets
- Evaluations

### Rapport de test pour: FoxyTag Version: 1.0

```
L'application s'arrete au moment de recherche Bluetooth.  
Une améloiration de l'interface est la bienvenue.
```

Testé avec: Nokia 6680 Enregistrer tout

Authors: Hikari & Dejan

## 13.4.3 Évaluations

Accueil Logout

Catégorie

- Localisation
- Jeux
- Utils
- Autres

Best users

Mon profile

- Profile
- Mes projets
- Evaluations

### Evaluations

**Les tests à effectuer et à envoyer:**

Propriétaire: michel Application: FoxyTag Version: 1.0 [Telecharger jad](#) [Telecharger jar](#) [Remplir le test](#)  
A tester avec: Nokia 6280.

Propriétaire: michel Application: FoxyTag Version: 1.0 [Telecharger jad](#) [Telecharger jar](#) [Remplir le test](#)  
A tester avec: Nokia 6280, Nokia 6280.

**Nouvelles applications:**  
Catégorie: Autres  
Catégorie: Jeux  
Catégorie: Localisation  
Catégorie: Outils

Authors: Hikari & Dejan

## 13.4.4 Résultats des tests par version

|                             |   |
|-----------------------------|---|
| <a href="#">Autres</a>      | <b>Version:</b> 1.0   |
| <a href="#">Best users</a>  | <b>Catégorie:</b> Localisation  |
| <b>Mon profil</b>           | <b>Téléphones de test:</b>  |
| <a href="#">Profil</a>      | Marque      Modèle  |
| <a href="#">Mes projets</a> | Nokia      6680 <a href="#">effacer</a>   |
| <a href="#">Evaluations</a> | SonyEricsson S700i <a href="#">effacer</a>  |
|                             | <b>Rajouter un téléphones de test:</b>  |
|                             | Marque: <input type="text"/> Modèle: <input type="text"/> <input type="button" value="Rajouter"/> |
|                             | <a href="#">Revenir au projet</a>   |

---

### Résultats de tests:

Par: raph  
le: 2007-05-31  
Testé avec: Samsung sgh  
test réalisé  
Note pour ce test:

Par: hikari  
le: 2007-05-25  
Testé avec: Motorola E1000  
test hikari  
Note pour ce test:

### 13.4.5 Liste des meilleurs testeurs

|                              |                                      |
|------------------------------|--------------------------------------|
| <a href="#">Accueil</a>      | <a href="#">Logout</a>               |
| <hr/>                        |                                      |
| <b>Catégorie</b>             | <b>Best rated users:</b>             |
| <a href="#">Localisation</a> |                                      |
| <a href="#">Jeux</a>         | <b>1 - hikari</b>                    |
| <a href="#">Utils</a>        | Average rating: 10<br>Total tests: 1 |
| <a href="#">Autres</a>       | <b>2 - dejan</b>                     |
|                              | Average rating: 9<br>Total tests: 2  |
| <a href="#">Best users</a>   | <b>3 - raph</b>                      |
| <b>Mon profile</b>           | Average rating: 0<br>Total tests: 1  |
| <a href="#">Profile</a>      |                                      |
| <a href="#">Mes projets</a>  |                                      |
| <a href="#">Evaluations</a>  |                                      |
| Authors: Hikari & Dejan      |                                      |

## **Partition**

Cette annexe définit la partition du travail entre Hikari WATANABE et Dejan MUNJIN, ci-après HW et DM. Les points d'introduction et de conclusion ont été rédigés communément.

### **Chapitre 2 :**

Point 2.1        HW  
Point 2.2        DM  
Point 2.3        HW

### **Chapitre 3 :**

Point 3.1, 3.7   HW

### **Chapitre 4 :**

Point 4.1, 4.5   DM

### **Chapitre 5 :**

Point 5.1, 5.5   DM

### **Chapitre 6 :**

Point 6.1, 6.3   HW et DM

### **Chapitre 7 :**

Point 7.1, 7.2   HW

### **Chapitre 8 :**

Point 8            HW et DM