MMSCam

Pilotage à distance d'un téléphone MMS

Auteur : Jeanmonod David
Répondant externe : Cecchin Gianpaolo
Prof. Responsable : Robert Stephan
Sujet proposé par : Swisscom Mobile

Travail de diplôme 2003

Département d'électricité et d'informatique

1. Cahier des charges

Données de base

L'idée est d'exploiter le service MMS de tout téléphone portable muni d'une caméra pour capter des images à n'importe quel moment, depuis n'importe quelle position.

Un logiciel pour téléphones portables, ayant la fonction de commander la caméra du terminal et d'envoyer la photo par MMS, sera développé pour satisfaire les fonctionnalités décrites ci-dessus.

Il sera possible d'installer le logiciel sur tout terminal mobile muni d'une caméra.

Cahier des charges

Le cahier des charges de ce projet consiste à :

- Créer une application combinant la réception d'un ordre de « prise de vue » par SMS et d'envoyer cette photo vers une adresse e-mail définie dans le SMS ou via MMS vers le numéro ordonnant.
- Programmer l'application dans un langage qui soit le plus portable possible. L'application doit être simple à utiliser et à installer.
- Programmer une interface utilisateur de haute qualité.
- Ecrire un rapport détaillé (code en annexe) et un mode d'emploi.

Fonction

- Déclenchement par réception d'un SMS avec des paramètres à définir.
- Récupération de la photo prise sur un natel par MMS ou sur une adresse e-mail (destinataire défini dans le SMS).
- Lecture des images à l'écran.
- Déclenchement automatique selon un intervalle de temps selon l'ordre donné par SMS.
- Déclenchement par un appel en absence.
- Possibilité de contrôler l'accès (1. Déclenchement de l'ordre uniquement par des numéros pré-definis 2. Déclenchement de l'ordre par tout le monde)

2. Tables des matières

3. Synthèse

Le but de ce projet était de modifier un téléphone portable, de manière à pouvoir le commander à distance. Le téléphone prendrait alors une photo qu'il nous renverrait par MMS.

La réalisation de ce travail de diplôme s'est découpée en deux phases distinctes :

- Premièrement une analyse des possibilités techniques.
- Deuxièmement la réalisation du travail.

Analyse

Cette partie, la plus importante, à consister en une étude des différentes technologies et protocoles utilisés dans le monde de la téléphonie mobile. Les sujets abordés sont liés à la transmission d'informations d'un téléphone portable à un autre, soit à une communication avec Internet.

L'analyse a permis de révéler des difficultés techniques qui ont influencé la suite du projet, comme par exemple : l'abandon des MMS au profit du courrier électronique.

Réalisation

Le choix a été pris de travailler avec l'environnement de développement Java pour appareils mobiles (J2ME).

Ce qui a été réalisé : trois applications distinctes :

- Une à installer sur un téléphone pour envoyer les ordres à l'appareil muni de la caméra.
- Une à installer sur l'appareil muni de la caméra, qui est chargée de prendre la photo est de l'envoyer à un serveur Web.
- La dernière qui fonctionne sur le serveur Web, qui crée un E-mail contenant la photo et l'envoie à un serveur SMTP.

4. Introduction

L'évolution de la téléphonie

Il y a maintenant une dizaine d'années, apparaissait la téléphonie mobile. Il devenait alors possible de communiquer depuis n'importe quelle zone couverte par le réseau GSM. Cette technologie a rapidement conquis la population puisque actuellement presque tout le monde (du moins en Suisse) possède un téléphone cellulaire.

Lors de leur apparition, les possibilités offertes par les téléphones portables étaient assez restreintes, mais déjà extrêmement pratiques, ce qui a suffi à convaincre.

Comme toutes les technologies, la téléphonie mobile a évolué, et actuellement les possibilités offertes sont bien plus importantes qu'il y a dix ans. Mais bien comme souvent, la majorité des utilisateurs n'utilisent que les fonctions de base, à savoir téléphoner et envoyer des SMS.

Les possibilités actuelles

Aujourd'hui en plus de transmettre du son et des SMS, les téléphones actuels sont capables de se connecter à Internet. Ce qui permet déjà d'envisager une multitude d'applications. Mais en plus est apparu un nouveau standard pour l'envoi de message, le MMS, qui permet de transférer des messages multimédias composés d'images, de sons, de textes, de vidéos, etc.

Une grande partie des appareils commercialisés actuellement possède aussi une caméra incorporée.

De tels progrès permettent d'imaginer de nouvelles applications pour ces appareils qui n'étaient, à la base, que des outils de communication vocale.

L'idée de base du projet

C'est en partant de ce constat, que l'idée de détourner ces appareils de leur fonction de base et d'en faire des outils de surveillance est apparue. Puisque les téléphones actuels nous permettent de prendre

des photos et de les envoyer : Pourquoi ne pas tenter d'améliorer encore un peu le système et d'automatiser le téléphone, de manière à ce qu'il prenne des photos et les envoie seul ?

Ainsi on pourrait placer un téléphone quelque part et voir ce qui s'y passe depuis n'importe où.

Alors qu'une dizaine d'années auparavant, un tel projet aurait nécessité de gros moyens ainsi qu'une grosse infrastructure, aujourd'hui un simple téléphone portable peut suffire.

Une grande partie des appareils commercialisés actuellement possède une caméra incorporée.

5. Analyse et étude de faisabilité

N'ayant jamais réalisé de projet de télécommunication, la première tâche de ce travail a été d'étudier et de tester les différentes technologies existantes. Le contenu de ce chapitre retrace les grandes lignes de cette étude.

Comme vous allez le voir, cette analyse va révéler des problèmes techniques qui vont avoir pour conséquence une modification du cahier des charges. L'ensemble des modifications qui devront être nécessaires sont présentées au chapitre 6.

5.1. Le projet en général

5.1.1. Envoi de l'ordre

La première partie qui devra être réalisée concerne la commande à distance du téléphone muni de la caméra. Il faudra envoyer un message d'ordre à cet appareil.

Pour transmettre un message de téléphone portable à téléphone portable, il n'existe pas énormément de possibilités. En effet, il est possible d'envoyer des SMS, des MMS, des e-mails et de téléphoner.

Sur ces quatre modes de transmission d'informations, deux sont disponibles uniquement sur les téléphones de dernière génération; il s'agit des e-mails et des MMS. De plus, ces deux possibilités se révèlent plus onéreuses que les deux autres. C'est pourquoi elles ne vont pas être abordées.

De plus, en ce qui concerne la transmission d'informations par un appel, il est bien entendu qu'il n'est pas question de dicter vocalement ces ordres au téléphone caméra, mais l'idée serait plutôt de se servir d'un appel en absence pour stimuler l'appareil.

5.1.2. Capture de l'image

Cette partie ne nécessite pas d'étude particulière, si ce n'est la recherche des librairies qui permettent une telle manipulation d'un téléphone portable.

5.1.3. Transmission de l'image

La troisième partie du projet consiste à trouver le meilleur moyen pour renvoyer l'image à celui qui l'a commandée. Là aussi, il existe plusieurs possibilités. On peut transférer une image par MMS, par E-mail, par une connexion réseau sur un serveur, etc…

Ces différentes possibilités font appel à un grand nombre de technologies, qu'il va falloir étudier et tester de manière à évaluer quelle sera la plus adaptée à ce projet.

5.2. Développement sur téléphone

De par la variété des téléphones portables sur le marché, il existe aussi une grande variété d'environnements de programmation pour téléphones portables. Mais désirant réaliser un logiciel qui soit compatible avec une majorité de téléphones portables, il convient de prêter une attention particulière à la sélection de cet environnement.

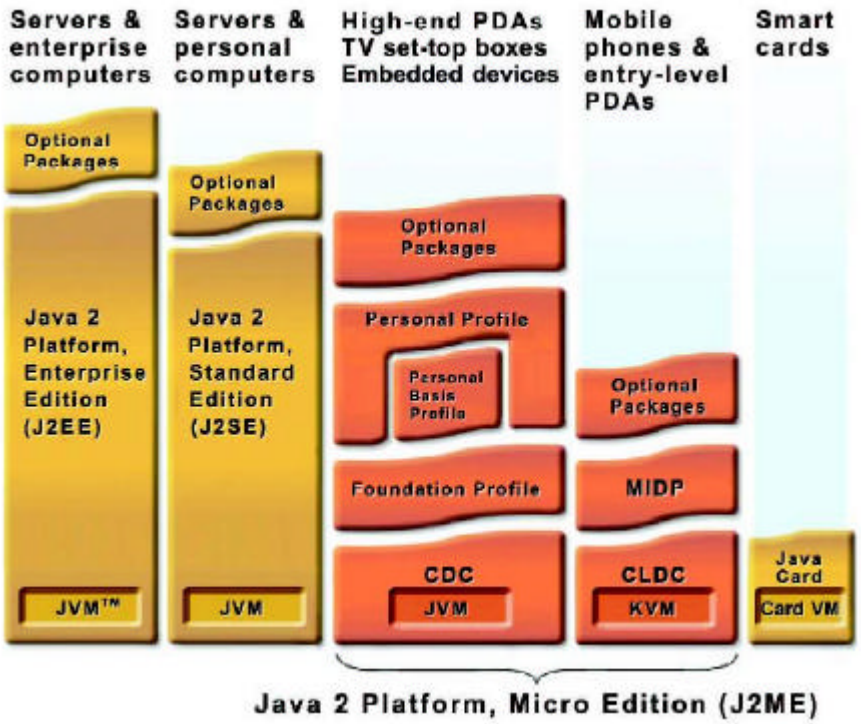
Dans cette «discipline » toute récente, puisqu'il n'y a que 2 à 3 ans qu'il est possible de développer des applications pour mobiles, deux tendances ont émergé et dominent l'univers du développement pour ces petits appareils qui tiennent dans notre poche. Ce sont les environnements de Java et de Symbian.

5.2.1. Java

La technologie Java inclut deux éléments : un langage de programmation et un cadre d'application dans lequel les programmes Java peuvent être exécutés. La syntaxe du langage de programmation Java est similaire au langage C++, tous les deux sont des langages orientés objet. La principale différence entre le C++ et Java, c'est que pour le C++ le code source est compilé dans le code machine natif qui fonctionne uniquement avec la cible spécifique de l'appareil, alors que le compilateur Java ne génère pas de fichier exécutable. Il crée pour chacune des classes d'un fichier Java un fichier qui sera interprété par la Machine Virtuelle Java. C'est le mécanisme qui permet aux applications Java d'être portables, c'est-à-dire qu'une application Java fonctionne dans tous les appareils qui ont une plate-forme Java similaire.

J2ME

Pour pouvoir faire tourner une machine virtuelle Java sur un téléphone portable, il faut bien entendu qu'elle soit plus petite que celle que l'on ferait tourner sur un serveur d'entreprise. Ainsi la plate-forme Java 2 a été déclinée en trois versions. La version d'entreprise Java 2 (J2EE) a été conçue pour déployer les solutions exigeantes des serveurs, la version standard Java 2 (J2SE) est utilisée par les ordinateurs de bureau, et la version Micro Java 2 (J2ME) a été spécialement créée pour les petits appareils électroniques tels que les téléphones mobiles. Cette approche garantit qu'une application appropriée fonctionnera pour différents types d'appareils.



La version J2ME est une collection de technologies et spécifications qui sont conçues pour différentes parties du marché des petits appareils. La partie principale de la plate-forme J2ME est constituée par deux configurations différentes : Connected Device Configuration (CDC) et Connected Limited Device Configuration (CLDC).

Une configuration définit les bibliothèques centrales de la technologie Java et les capacités de mémoire virtuelle de la machine. La configuration CDC s'adresse aux appareils mobiles dernière génération, tels que les organiseurs, téléphones haut de gamme, etc. La configuration CLDC est prévue pour les téléphones mobiles d'entrée de gamme tels que les portables actuels. C'est donc la configuration CLDC qui nous sera utile. A noter encore, que dans le cas de J2ME, la machine virtuelle s'appelle KVM pour Kilobyte Virtual Machine.

Le profil MIDP

En dessus des configurations, il y a les profils qui définissent les fonctionnalités dans chaque catégorie spécifique d'appareils. Le "Mobile Information Device Profile" (MIDP) est un profil pour les appareils mobiles utilisant la configuration CLDC, comme les téléphones mobiles. Le profil MIDP précise les fonctionnalités comme l'usage de l'interface client, la persistance de stockage, la mise en réseau, et l'application modèle.

Sur la plupart des téléphones actuels, la version J2ME est composée de la configuration CLDC et du profil MIDP.

Les packages optionnels

En plus du profil standard MIDP, peuvent, suivant les appareils, être ajouté des packages supplémentaires pour permettre l'utilisation de certaines spécificités des appareils. Par exemple, il existe un package optionnel nommé WMA qui permet l'envoi et la réception de messages du type SMS ou CBS. Comme ces options sont typiquement réservées aux téléphones portables il était naturel de ne pas les intégrer directement dans le profil standard MIDP.

5.2.2. Symbian C++

A la différence de Java qui est une sorte environnement d'exécution pour les programmes, Symbian est un système d'exploitation. Apparu en premier sur des organiseurs de poche tels que, par exemple, les Psions, Symbian a su s'imposer comme le système d'exploitation le plus utilisé actuellement dans les nouveaux téléphones portables. Symbian n'est donc pas un langage de programmation. On trouve d'ailleurs plusieurs compilateurs pour Symbian, ce qui permet de coder en Java, C++, Visual Basic, etc…

C++

Symbian a été écrit en C++. Ce langage est donc le langage natif de Symbian. Une application C++ sera plus rapide qu'une application équivalente dans un autre langage. Le code natif requiert aussi moins d'espace.

On peut donc dire que C++ est « le » langage de Symbian. Il n'y a pas vraiment de raison technique de vouloir utiliser un autre langage pour développer un logiciel sur la plate-forme Symbian.

La plate-forme Série 60

La plate-forme Série 60 a été développée par Nokia, mais est utilisée sous licence par d'autres fabricants. Elle comprend une interface utilisateur graphique (IUG), une suite bureautique et un kit pour développeurs logiciels (SDK) Série 60, tous ces éléments fonctionnant sous Symbian.

Commercialisé au deuxième trimestre 2002, le Nokia 7650 a été le premier produit équipé de la plate-forme Série 60.

Cette plate-forme fournie donc un environnement commun à toute une série de téléphones.

5.2.3. Choix réalisé

C++ possède une plus grande flexibilité que Java. En effet, Java dispose de procédures déjà implémentées qui peuvent ne pas convenir dans certains cas tel que le management des ressources mémoires : le nettoyage automatique de la mémoire RAM en Java peut causer des dysfonctionnements sur certains programmes.

Néanmoins les différents problèmes associés au codage en C++ sur Symbian sont :

- Management de la mémoire : pour la majorité des applications le système Java semble être suffisant.
- Environnement d'exécution : les options proposées sur les exécutables Java comme les protections au téléchargement ou l'exécution sécurisée sont gratuites, alors qu'en C++ il faut les développer, les tester et les maintenir.
- Pérennité : Java semble avoir été accepté pour le développement d'applications sur téléphones mobiles. Les développements futurs rendront Java peut être aussi rapide que le C++.

Pour les jeux où le critère recherché est la vitesse d'exécution, le choix du langage est donc crucial et C++ s'impose comme une solution très valable. Mais dans notre cas, où la vitesse d'exécution n'est pas un critère de choix, cet argument n'as pas d'importance.

Ce qui ressort du cahier des charges est avant tout, une grande compatibilité avec un maximum de téléphones. Et là, c'est sans contexte que Java sait s'imposer. Même si le nombre de téléphones compatibles avec toutes les librairies nécessaires au projet n'est pas encore énorme, il est sûr qu'il va progresser.

C'est donc pour cette raison principale que Java à été choisi pour la réalisation de ce projet. Mais il faut souligner que les deux environnements se valent plus ou moins. Et qu'il aurait probablement été possible de travailler sur les deux plates-formes et d'obtenir le même résultat.

5.3. Le service SMS

5.3.1. Généralité

SMS signifie Short Message Service, que l'on pourrait traduire par service de messages courts. Ce service est disponible sur réseau GSM est permet à un utilisateur de téléphone mobile de composer un message textuel et de l'envoyer à un autre téléphone mobile.

Généralement quand on parle de messages courts on les appelle des SMS, mais ceci est un abus de langage car SMS désigne le service. Un message devrait être désigné par l'acronyme SM mais pour ne pas perturber la majorité des lecteurs, je les appellerai tout de même SMS.

Ces messages peuvent contenir au plus 160 caractères qui sont codés à l'aide d'ASCII 7 bits sur 140 octets. Leur format est défini par une recommandation de GSM [DOC 13].

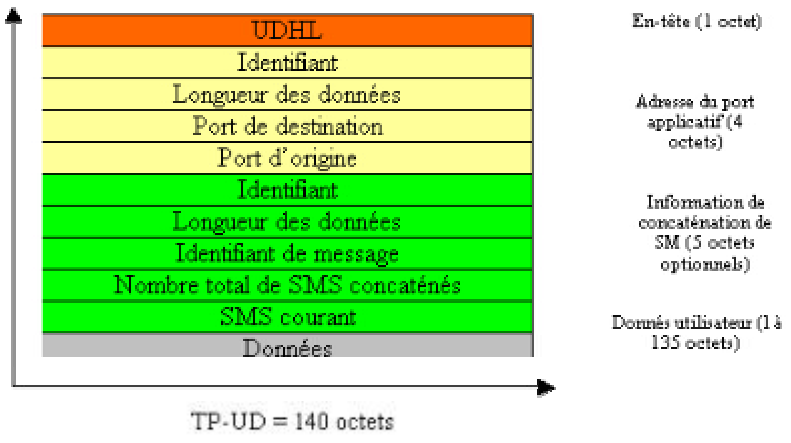
5.3.2. Utilisation dans J2ME

L'envoi et la réception de SMS n'est pas directement une fonctionnalité du profil MIDP. Cela se comprend, car certains appareils compatibles MIDP, n'ont pas la possibilité d'envoyer de tels messages.

Pour pouvoir utiliser le service de message court, il faut que le terminal supporte le paquetage optionnel « Wireless Messaging Api 1.0 JSR 120» (WMA). Grâce à la librairie fournie par Sun la compatibilité avec passablement de téléphones portables modernes est garantie.

5.3.3. Numéro de port

Le schéma suivant montre la structure d'un message SMS.



Comme on peut le voir dans le format du message, les SMS possèdent un numéro de port. Ce numéro de port à exactement le même rôle qu'un numéro de port dans TCP ou UDP. Il sert à différencier l'application destinatrice sur un téléphone. Ainsi plusieurs applications peuvent utiliser le service SMS sur un même téléphone et cela sans risque de voir leurs messages se mélanger.

5.3.4. Réception de message

Lors de tests, il est vite apparu, qu'il n'est pas possible de réceptionner les SMS «standards » qui sont envoyés sur le téléphone. De même qu'une application ne peut pas capter les messages TCP qui ne lui sont pas destinés, il est impossible de réceptionner les SMS qui sont destinés à une autre application.

Si la possibilité d'intercepter les SMS «standards » avait été laissée, on aurait eu des problèmes de concurrence. Imaginer deux applications qui attendent un SMS, comment le système aurait pu savoir à qui délivrer les messages entrants.

La solution consiste à écouter l'arrivée de SMS sur un autre port que le port standard. Comme pour TCP/UDP, il existe aussi 2^16 ports, on n'a donc le choix pour choisir un port pour son application.

Mais bien que l'utilisation de n'importe quel numéro de ports puisse fonctionner, il faut tout de même se référer à la spécification GSM [DOC 13] qui précise que seul les numéros de ports de 16000 à 16999 sont réservés aux applications.

Le gros inconvénient d'écouter sur un autre port, c'est qu'il faut aussi envoyer le message de commande sur ce même port. Et puisque à ma connaissance aucun téléphone ne permet de sélectionner le port lors de l'envoi d'un SMS. Il va falloir créer une deuxième application pour l'envoi des messages de commande.

Ce point est particulièrement ennuyeux, pas du fait qu'il faille développer une deuxième application, mais parce que celle-ci devra être installée sur un téléphone compatible avec la librairie WMA. Ce qui réduit quand même passablement la liste des appareils utilisables.

Ainsi on passe d'une compatibilité complète (tous les téléphones portables) à une compatibilité réduite aux téléphones modernes.

5.4. Le service MMS

5.4.1. Généralité

MMS pour Multimedia Messaging System est un service qui peut à première abord être considéré comme le successeur du SMS. Mais si on étudie un peu plus profondément ce service, on se rend compte que MMS est bien plus complet et du même coup plus compliqué que le service SMS. Là où les SMS étaient principalement utilisés pour une communication de téléphones portables à téléphones portables, les MMS dépassent largement ce cadre et sont destinés à une utilisation beaucoup plus étendue.

Les grandes différences entre ces deux services sont, premièrement le contenu qui peut être transmis et deuxièmement les acteurs qui vont l'utiliser.

Contenu

Le contenu des MMS ne se limite pas au 160 caractères des SMS, comme son nom l'indique, le MMS est destiné à véhiculer un contenu multimédia. Pour le moment le type du contenu est encore passablement limité mais, à terme, on devrait pouvoir transférer tous types de textes, d'images, de sons et de vidéos. De plus ces messages multimédias sont encodés de manière à pouvoir être lus comme des petites présentations qui enchaînent les affichages des éléments du message, un peu comme le fait une présentation PowerPoint.

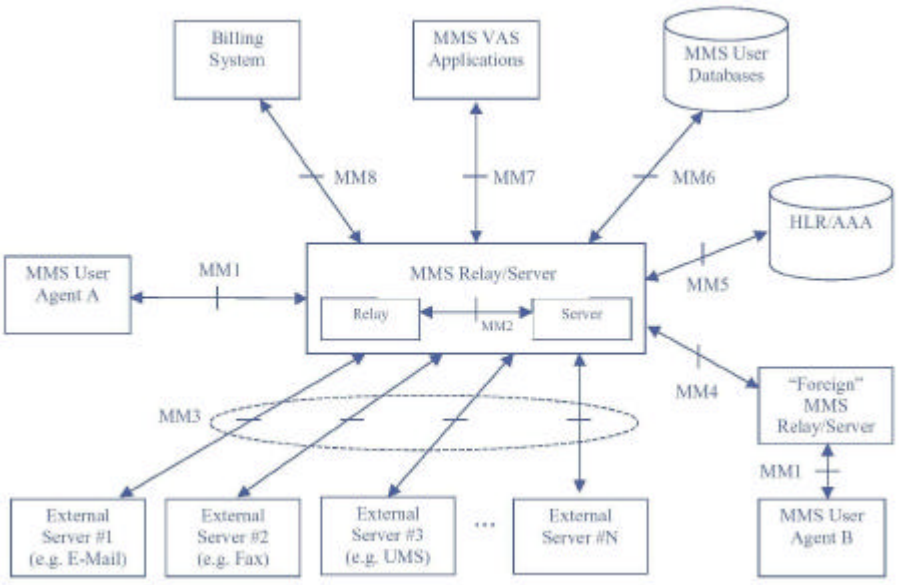
Acteurs

Certes on peut toujours envoyer ces messages de portables à portables, mais les MMS sont prévus aussi pour d'autres utilisations tels que des envois commerciaux, des communications avec des serveurs, etc… Beaucoup de services multimédias vont commencer à utiliser les MMS comme vecteurs de communication. On imagine facilement des services météo, touristique, de divertissement, qui vont utiliser les MMS et, ainsi, repousser la difficulté qui était jusqu'alors la petite taille des SMS.

Ainsi des MMS vont être envoyés depuis un nombre de sources très variée et passer par de nombreux intermédiaires tels que des serveurs web, serveurs d'autres opérateurs, ainsi qu'une multitude de terminaux différents comprenant PC, Agenda électronique, téléphone mobile, etc…

5.4.2. Les différentes interfaces du service MMS

Face à la multitude d'acteurs qui vont utiliser les MMS, il faut garantir la compatibilité des nombreuses interconnections. Pour cela des interfaces ont été définies. On entend par interface, la spécification des communications entre deux entités. Ces interfaces séparent les spécifications en huit « sous-normes » nommées MM1, MM2, … jusqu'à MM8. La figure suivante met en évidence le champ d'activités de chacune de ces interfaces.



La partie centrale de la figure précédente, composée d'un "Relay" et d'un "Server" est ce que l'on nomme le MMSC (Multimédia Message Service Center). C'est le centre du système.

Ainsi par exemple pour une transmission entre deux serveurs MMSC on utilise la norme MM4, entre un téléphone portable et un serveur MMSC se sera la norme MM1 et ainsi de suite…

5.4.3. L'interface MM1

Bien entendu, l'interface qui nous intéresse le plus est MM1, car c'est elle qui décrit comment envoyer un MMS depuis un téléphone portable.

Les informations officielles concernant cette interface sont disponibles dans deux documents publiés par 3GPP. Ces deux parties bien distinctes sont premièrement la structure du message, c'est-à-dire la manière dont il est codé et, deuxièmement, sa transmission, c'est-à-dire les protocoles utilisés ainsi que l'ordre des messages.

Les deux documents publiés par 3GPP sont :

- WAP 206 MMS Client Transactions [DOC 16]
- WAP 209 MMS Encapsulation [DOC 17]

Le support de transmission

Dans le document consacré à la transmission des MMS on apprend que l'envoi d'un MMS est basé sur une communication WSP/http. De quoi s'agit-il ?

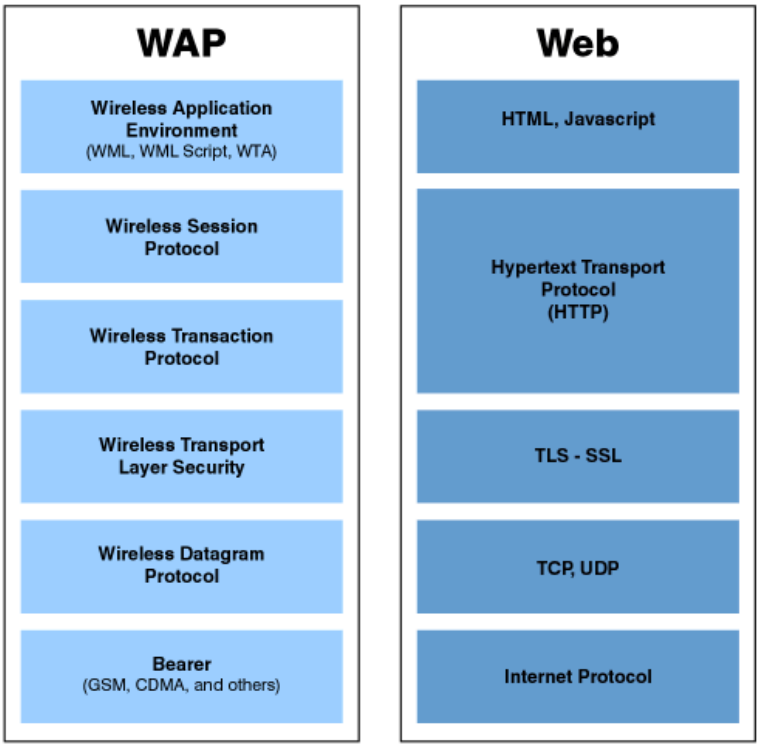
Pour commencer, qu'est-ce que WSP ? En fait, WSP est une couche de la norme WAP. Commençons donc par rappeler ce qu'est le WAP.

WAP

La technologie WAP a pour but de permettre à des terminaux mobiles d'accéder à des documents circulant par des réseaux sans fil. Il s'agit donc de permettre à n'importe quel terminal mobile de pouvoir formater des documents. C'est pour cela qu'un protocole universel a été mis en place : le WAP (Wireless Application Protocol). Il se propose de définir la façon par laquelle les terminaux mobiles accèdent à des services Internet, et cela à un niveau au-dessus de la transmission des données.

Le protocole WAP est défini selon des couches, c'est-à-dire des niveaux d'abstraction des données (dans l'esprit du modèle OSI) afin de séparer les différents traitements des données nécessaires pour effectuer la transaction.

L'illustration suivante montre les 5 couches WAP ainsi qu'en parallèle la correspondance par rapport au modèle OSI :



WSP

Comme son nom l'indique (WSP = Wireless Session Protocol), la couche session permet d'établir une session entre un client et un serveur c'est-à-dire de définir les paramètres de connexion pour effectuer des transactions. La couche WSP permet ainsi à la couche application de bénéficier de deux types de sessions différentes :

- Session orientée connexion dans laquelle la couche session va interagir avec la couche transaction.
- Session orientée non-connexion dans laquelle la couche session va directement agir au niveau de la couche transport pour l'envoi de datagrammes bruts.

WSP est dans son ensemble l'équivalent du protocole HTTP (dans sa version 1.1). On retrouve d'ailleurs un bon nombre d'implémentations identiques au HTTP dans WSP.

Compression des en-têtes

Une différence majeure consiste en la compression des en-têtes connus. Comme nous allons le voir dans le chapitre consacré à http (voir 5.5) la première partie d'un message est composé de un ou plusieurs champs d'en-tête qui ont la forme suivante :

```
nomDuChamp : valeurDuChamp
```

Certains champs sont présents dans presque tous les messages. L'idée de WSP est de remplacer ces champs standards par des valeurs hexadécimales dans le but de réduire la taille du message. On arrive ainsi à augmenter le taux de transfère des informations utiles.

Par exemple l'en-tête suivant dans HTTP:

```
Message type - MMS Message
```

Correspond en WSP à :

```
8C 84
```

5.4.4. L'interface MM7

Cette interface concerne un autre type de communication qui est cette fois la communication entre un client HTTP et un serveur MMSC. Il est donc possible à l'aide de cette interface d'envoyer des MMS non plus uniquement depuis un téléphone portable, mais depuis n'importe quel client HTTP.

Bien entendu, les opérateurs n'ont pas prévu que vous puissiez envoyer gratuitement des MMS depuis une page Web. Et cela est financièrement compréhensible. Mais par contre ce qui à été prévu c'est que l'on puisse ouvrir un compte auprès d'un opérateur, pour obtenir un accès à l'interface MM7. Ainsi l'opérateur pourra tout de même facturer son service.

Taxation

L'avantage de cette interface, c'est qu'une fois un compte ouvert, il est possible de définir le mode de taxation que l'on veut utiliser. On pourrait par exemple spécifier que c'est celui qui reçoit le MMS qui est chargé de payer son prix. Ainsi la personne qui a commandé une photo, la payerait à la réception. Ce système serait pratique pour des caméras qu'on pourrait qualifier de publiques.

Inconvénient

Le principal inconvénient réside dans l'ouverture du compte chez l'opérateur. Puisque que nous voulions une bonne portabilité, il est assez problématique de devoir ouvrir ce compte. En effet, si l'on regarde déjà pour la Suisse, il faudrait ouvrir 3 comptes. De plus l'utilisateur du programme devra définir quel compte utiliser sur son téléphone.

5.5. Le protocole HTTP

Le protocole HTTP (Hyper Text Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990. A partir de la version 1.0 du protocole, on peut transférer des messages avec des en-têtes décrivant le contenu du message.

Le but du protocole HTTP est de permettre un transfert de fichiers, à la base essentiellement des fichiers HTML, mais tout autre type de fichier est possible. Les fichiers sont localisés grâce à une chaîne de caractères appelée URL.

5.5.1. Requète HTTP

La communication entre le client et le serveur se fait en deux temps:

1 Le navigateur effectue une requête HTTP

2 Le serveur traite la requête puis envoie une réponse HTTP

Une requête HTTP est un ensemble de lignes envoyées au serveur par le client. Ces lignes comprennent :

- Une ligne de requête: elle spécifie le document demandé, la méthode qui doit être appliquée, et la version du protocole utilisée. Ex :

```
GET http://www.eivd.ch HTTP/1.0
```

- Les champs d'en-tête de la requête: il s'agit d'un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la requête et le client. Chacune de ces lignes est composée d'un nom qualifiant le type d'en-tête, suivi de deux points et de la valeur de l'en-tête : Ex :

```
Content-Type : text/plain
```

- Le corps de la requête: c'est un ensemble de lignes optionnelles devant être séparé des lignes précédentes par une ligne vide et permettant par exemple un envoi de données par une commande POST.

5.5.2. Méthode de communication

Une requête HTTP peut être de deux types principaux, c'est ce que l'on appelle la méthode de transfert qui est le premier paramètre de la ligne d'en-tête. Ces deux types sont GET et POST.

Une requête du type GET, concerne la simple demande d'un fichier, il n'y a pas de corps de requête. La totalité des informations transmises au serveur se trouve soit dans l'URL, soit dans les champs d'en-tête.

Note : il faut souligner tout de même que des paramètres peuvent être transmis par l'URL Ex : Transmission du paramètre page qui vaut 76 :

```
GET http://www.eivd.ch?page=76 HTTP/1.0
```

Il est ainsi possible de passer des paramètres, mais cela jusqu'à la taille maximum d'une URL soit 256 caractères.

Lorsque plus de données doivent être transmises, on utilise une méthode POST qui permet de transférer un corps à la requête, qui permet de l'envoi d'autant de données que nécessaire. Ce corps de requête doit se trouver après les champs d'en-tête, simplement séparé d'eux pas une ligne vide.

Voici un exemple complet d'une requête du type POST, la demande d'une fiche étudiant :

```
POST http://www.eivd.ch/etudiant/fiche.php HTTP/1.0
Accept : text/html
If-Modified-Since : Monday, 15-January-2000 14:37:11 GMT
User-Agent : Mozilla/4.0 (MSIE 5.0; Windows 95)

Nom=Jeanmonod&Prénom=David
```

5.6. Le courrier électronique

5.6.1. Généralité

Le courrier électronique est considéré comme étant actuellement le service le plus utilisé sur Internet. Ce service, qui fonctionne un peu comme le service de courrier réel, est basé sur l'utilisation de deux protocoles principaux : SMTP et POP.

POP étant utilisé pour consulter le courrier reçu, il ne sera pas abordé dans ce document.

Par contre le protocole SMTP (Simple Mail Transfer Protocol) qui est lui à la base de tout envoi de courrier électronique nécessite un développement.

5.6.2. Le protocole SMTP

Le protocole SMTP est le protocole standard permettant de transférer le courrier d'un serveur à un autre en connexion point à point. Il s'agit d'un protocole fonctionnant en mode connecté, encapsulé dans une trame TCP/IP.

Le courrier est remis directement au serveur de courrier du destinataire. Le protocole SMTP fonctionne grâce à des commandes textuelles envoyées au serveur SMTP (par défaut sur le port 25). Chacune des commandes envoyée par le client (validée par la chaîne de caractères ASCII CR/LF, équivalent à un appui sur la touche entrée) est suivie d'une réponse du serveur SMTP composée d'un numéro et d'un message descriptif.

Ce protocole est extrêmement simple, voici donc un exemple d'envoi d'un mail (Il faut être connecté sur le port 25 d'un serveur SMTP):

Début de communication avec le serveur SMTP, la commande HELO permet de signaler :

```
HELO there
```

Définition de l'adresse de l'expéditeur du mail.

```
MAIL FROM: expediteur@domaine.com
```

Définition de l'adresse du destinataire

```
RCPT TO: destinataire@domaine.com
```

Transmission du corps du mail (Voir point 5.6.3)

```
DATA Corps du mail
```

Sortie du serveur SMTP

```
QUIT
```

5.6.3. Le format du message

Le format d'un message e-mail est défini par la spécification RFC 822 [DOC 14].

En fait on peut distinguer deux types de message : Les messages « Mono-Contenu » et « Multi-Contenu ».

Dans un message du premier type, quelques champs d'en-tête peuvent être spécifiés, mais ne sont pas obligatoires. Ces champs sont par exemple "Return-Path", "Received", "Date", "From", "Subject", "Sender", "To", "cc", etc. Ensuite vient le message texte à proprement dit.

Exemple :

```
Return-Path : <moi@eivd.ch>
Subject : Message d'exemple
To : <toi@eivd.ch>
Texte du message d'exemple !!!
```

Les messages à contenu multiple sont utilisés lorsque le message doit contenir plus qu'un élément. Par exemple si l'on veut envoyer un message ainsi qu'une image et un fichier HTML.

Pour envoyer un tel message, le champ d'en-tête doit contenir 3 lignes supplémentaires :

```
Mime-Version: 1.0
Content-Type: multipart/mixed;
Boundary='---PartieSuivante---'
```

Ces trois lignes précisent :

- La version du type MIME utilisé (voir point 5.6.4)
- Le type du message dans ce cas que c'est un message «Multi-Contenu » avec des documents de plusieurs types.
- La chaine de caractères qui sera utilisée pour séparer deux parties.

Voici un exemple de message qui contient un petit texte et un fichier HTML.

```
Subject : Message multi contenu
To : <toi@eivd.ch>
Mime-Version: 1.0
Content-Type: multipart/mixed;
        Boundary='---Ligne de separation---'

---Ligne de separation---
Content-Type: text/plain;

Message texte

---Ligne de separation---
Content-Type: text/html

<HTML>
  <BODY>
    <B> Message html </B>
  </BODY>
```

```
</HTML>

---Ligne de separation---
```

5.6.4. Les types MIME

Les types MIME (Multipurpose Internet Mail Extensions) sont un standard qui à été crée pour permettre d'étendre les possibilités du courrier électronique. Ces types permettent d'insérer des documents (images, sons, textes, ...) dans un courrier qui jusque-là était réservé à la transmission de texte.

Le type MIME est utilisé d'une part pour typer les documents attachés à un courrier mais aussi pour typer les documents transférés par le protocole HTTP.

La syntaxe d'un type MIME est définie comme suit :

```
Content-type: type_mime_principal/sous_type_mime
```

Une image GIF a par exemple le type MIME suivant:

```
Content-type: image/gif
```

5.6.5. Depuis MIDP 1.0

Le problème de ce protocole, c'est que la connexion avec le serveur SMTP doit se faire en mode connecté, généralement basé sur TCP. La gestion réseau de MIDP 1.0, ne permet pas de telles connexions. Ce type de connexion a été introduit dans la version 2 mais n'est pas supporté par assez de terminaux pour être utilisé actuellement.

Le seul moyen d'envoyer des E-mails depuis notre application MIDP 1.0 serait de se servir d'un serveur que l'on puisse contacter en mode sans connexion, et que ce serveur se charge d'envoyer le mail.

5.7. Essai de l'envoi de MMS

Il y a plusieurs avantages à vouloir envoyer l'image capturée sous forme d'un MMS. Premièrement, cela permet de recevoir l'image soit sur un autre téléphone portable (par exemple celui qui a donné l'ordre) mais aussi directement dans une boîte e-mail, puisque le serveur MMSC peut transférer le MMS sur une adresse e-mail, et cela sans avoir besoin d'implémenter une option particulière.

Deuxièmement, il y a pour le moment un avantage financier. En effet d'après les tarifs actuellement en vigueur chez Swisscom [DOC 10], l'envoi d'un MMS de 100ko coûte -.90 frs alors que l'envoi du même nombre de donnée sur GPRS coûte lui 1.90 frs (3.80 si l'on compte la réception). De plus, la réception d'un MMS est gratuite alors que celle d'un e-mail par exemple est payante.

Cet avantage financier doit tous de même être nuancé, car lors de la transmission d'images de basse résolution (~10ko), le coût est de 20cts alors que le MMS aurait coûté 50cts.

5.7.1. Utilisation d'une librairie Java

Comme nous l'avons déjà vu auparavant, il existe une librairie java qui s'appelle « Wireless messaging Api ». Cette librairie fournit la possibilité d'envoyer des SMS et des CBS, mais étonnement ne permet pas l'envoi de MMS. Pour combler ce manque, la version 2 de cette librairie, la JSR 205, est en cours de développement et permettra l'envoi de MMS.

Le développement de cette librairie touche à sa fin puisqu'elle en est à la révision publique qui a été lancée début novembre 2003.

Il est clair que cette solution constituerait la meilleure option existante, elle permettrait d'allier les avantages des MMS et de ne plus avoir le problème de configuration du serveur MMSC, puisque cela serait automatiquement géré par la librairie.

Mais comme rien n'est parfait, il reste un problème à l'utilisation de cette librairie; en effet, il n'y a encore aucun téléphone sur le marché qui la prennent en charge. Mais cette situation devrait changer dès l'année prochaine, car selon certaines personnes qui travaillent dans le domaine, plusieurs terminaux devraient la prendre en charge à partir du début de l'année 2004

5.7.2. Utilisation de l'interface MM7

Comme cela a été expliqué plus haut, l'utilisation de l'interface MM7 requiert l'ouverture d'un compte chez un opérateur de téléphonie. Ce

compte n'ayant pas été crée, l'envoi de MMS par cette interface n'a pas pu être testé.

5.7.3. Utilisation de l'interface MM1

Pour envoyer des MMS avec l'interface MM1, il faut transférer le message correctement formaté à un serveur MMSC et cela à l'aide d'une connexion WSP/HTTP.

Le serveur MMSC

Quand un utilisateur veut utiliser le service MMS pour la première fois, il doit configurer le service pour son téléphone. Il doit envoyer un message à son opérateur qui lui renvoie les paramètres de connexion.

Comme depuis Java, il n'est pas possible de lire ces paramètres, il va falloir les rentrer à la main. Il faudra donc configurer l'application sur chaque terminal, ce qui, malheureusement, sera un gros frein à la portabilité souhaitée dans le cahier des charges de ce projet.

Chaque compagnie de téléphone possède un voire même plusieurs serveurs MMSC. Les paramètres de connexion dont nous avons besoin, sont simplement les données pour se connecter à un serveur MMSC.

Dans notre cas, le serveur MMSC de Swisscom se trouve à l'adresse http://mms.natel.ch:8079. Malheureusement cette adresse ne suffit pas pour s'y connecter. Voici la liste des paramètres qui doivent être utilisés pour contacter le serveur MMSC :

```
Compte Internet
   Adresse de l'APN :       event.swisscom.ch

Compte WAP
   IP de la passerelle :    192.168.210.2
   Port de la passerelle :  9201

Centre de service
   Adresse du MMSC :        http://mms.natel.ch
   Port du MMSC :           8079
```

Formatage du message MMS

Les messages MMS sont similaires aux messages E-Mail, ils utilisent la même syntaxe et le typage MIME (voir point 5.6.4).

Voici un exemple de message MMS :

```
X-Mms-Message-Type: m-send-req
X-Mms-Transaction-ID: 451048788
X-Mms-Version: 1.0
From: +41797776693
To: david@lombric.ch
Subject: test
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary=\"----=_Part_4691_5946653.1067333561687\"
X-Mms-Message-Class: Personal
X-Mms-Message-Size: 485
```

```
X-Priority: 3
------=_Part_4691_5946653.1067333561687
Content-Type: text/plain; name=Text.txt
Content-Transfer-Encoding: 7bit
Content-ID: <Text.txt>
Content-Location: Text.txt
Content-Disposition: inline

test

------=_Part_4691_5946653.1067333561687--";
```

Comme on peut le voir dans cet exemple, beaucoup de champs de l'en-tête sont spécifique aux MMS (ils commencent par X-Mms). Il serait inutile de commencer à expliquer l'utilité des tous ces champs dans ce document. Le lecteur intéressé trouvera toutes les informations nécessaires dans le document de référence [DOC 17].

Transmission du message

Une fois tous les paramètres de connexion au serveur MMSC définis sur le téléphone, la transmission du message se fait par une requête HTTP Post.

Pour commencer ces tests, il a d'abord été tenté de transmettre un MMS simple, contenant uniquement du texte.

Puisqu'il n'est possible de communiquer avec le serveur MMSC uniquement depuis un téléphone, il n'était pas envisageable de faire ces tests depuis un PC. Il a donc fallu développer une petite application de test avec J2ME qui a été installée sur un téléphone Sony Ericsson P800.

Aucune réponse

Lors du test d'envoi de ce premier MMS, le serveur MMSC n'a donné aucune réponse et aucun MMS n'est arrivé à destination. Il y a donc un problème de transmission, mais il est extrêmement difficile d'identifier son origine puisqu'il est impossible d'installer un analyseur de trame sur le téléphone. Ce problème va demander énormément de temps pour être résolu, le chapitre suivant y est consacré.

5.8. Problème avec HTTP

Suite au problème de contact du serveur MMSC (voir 5.7), il a fallu tenter d'en identifier la source. La première série de tests réalisés à été de controller si la connexion HTTP fonctionnait correctement.

Test avec PHP

Le premier test a été une simple transmission de données à une page PHP pour voir si toutes les données sont transmises correctement.

Pour ne pas commettre une erreur dans le code du programme, il était préférable de prendre un programme plus "sûr", c'est pourquoi le code de test est en fait un code d'exemple fourni par Sun et disponible à l'adresse :

http://developers.sun.com/techtopics/mobility/midp/articles/servlets/PostMidlet.java

La partie importante de se code étant la connexion http, voici un petit aperçu du cœur de l'exemple :

```
c.setRequestMethod(HttpConnection.POST);
c.setRequestProperty("IF-Modified-Since",
   "20 Jan 2001 16:19:14 GMT");
c.setRequestProperty("User-Agent",
   "Profile/MIDP-1.0 Configuration/CLDC-1.0");
c.setRequestProperty("Content-Language", "en-CA");
os = c.openOutputStream();
String str = "name=163748";
byte postmsg[] = str.getBytes();
for(int i=0; < postmsg.length;i++) {
   os.write(postmsg[i]);
}
os.flush();
```

De plus la page PHP qui a été placée sur un serveur WEB contenait uniquement le code suivant :

```
<?php
  mail("djeanmon@eivd.ch","test",$_POST['name']);
  echo "Confirmation";
?>
```

Ainsi, lorsque le téléphone effectuera sa requête POST, un mail confirmera le passage du paramètre « name ».

Résultat du test d'HTTP

Après avoir effectué le test décrit ci-dessus, j'ai été étonné de constater que le mail reçu contenait une valeur "null". Ainsi il y a un problème de transmission du corps de la requête HTTP.

Ne sachant pas comment résoudre ce problème, j'ai tenté d'obtenir de l'aide en postant ce problème sur des forums de développement [FRM1] et [FRM4].

Dans les réponses que j'ai obtenues (voir **Error! Reference source not found.**), 2 types de idées ont tenté de répondre à mon problème.

- Certains prétendent que le problème vient de l'opérateur du réseau mobile qui, pour des raisons de sécurité, tronquerait les requêtes HTTP.
- D'autres prétendent que le problème vient de PHP, et qu'avec d'autre technologie comme Perl par exemple, le problème peut être résolu.

Au premier abord, la première explication me paraît plus crédible que la seconde, mais pour avoir le cœur net, j'ai testé de réceptionner ma requête avec une autre technologie.

Réception de la requête avec une Servelet

Pour le deuxième test, j'ai aussi utilisé un exemple de Sun disponible à l'adresse :

developers.sun.com/techtopics/mobility/midp/articles/socketRMI/

Cette fois la MIDlet effectue aussi une requête HTTP mais c'est une Servlet déposée sur un serveur TOMCAT qui devra récupérer les données.

Surprise, la transmission a fonctionné. Sur le moment, je ne comprenais pas comment la technologie utilisée pour la réception d'une requête HTTP pouvait interférer sur les données transmises. Mais, étant donné que je n'avais pas plus d'informations, j'ai admis que le problème venait de la technologie de traitement de la requête.

L'explication

C'est seulement à 3 semaines de la fin de ce projet, que j'ai découvert un début d'explication à ce problème.

Comme cela a déjà été mentionné plus haut, il semblait bizarre que le destinataire d'une requête http influence la transmission. Mais une évidence était que l'exemple Servlet était le seul à fonctionner.

C'est en retravaillant le code, que j'ai finalement trouvé la différence entre les programmes.

Dans le programme contactant la Servlet, le champ d'en-tête "content-type" était défini avec la valeur suivante :

```
Content-Type : application/x-www-form-urlencoded
```

Cette valeur ne correspondant pas à celle des données transmises, j'ai voulu corriger ce paramètre en remplacent le champ par :

```
Content-Type : text/plain
```

En testant le programme avec la nouvelle valeur, plus rien ne fonctionnait, c'est à ce moment que j'ai compris que la transmission

complète de la requête HTTP dépendait de la valeur du champ Content-Type.

Hypothèse

N'ayant pas pu obtenir de réponse à ce problème, il n'est pas possible de dire qu'elles en sont les causes. Malgré cela, voici tout de même une hypothèse qui n'engage que l'auteur.

Puisque la requête HTTP envoyée par le téléphone est probablement transmise correctement et que la Servlet n'a aucune raison de supprimer le corps de cette requête, cela signifie que la modification se passe entre le téléphone et le serveur.

Comme nous l'avons vu plus haut, les seuls intermédiaires qui touchent aux requêtes HTTP sont la passerelle WAP et le FireWall Swisscom.

Mais pourquoi donc ces éléments supprimeraint le corps de la requête. Pour mieux comprendre, revenons sur le paramètre du Content-Type. La valeur "application/x-www-form-urlencoded" est celle qui est utilisée par défaut par les browsers WEB pour envoyer les données collectées par un formulaire [DOC 17].

L'hypothèse est donc que pour des raisons, probablement, de sécurité les corps des requêtes qui ne proviennent pas d'un trafic WEB standard sont supprimer. En d'autre terme, ces mécanismes de sécurité laissent passer les requêtes générées par un navigateur Internet, mais suppriment celle provenant d'autre programme.

6. Bilan de l'analyse

L'analyse qui a été effectuée à permis de mieux comprendre le fonctionnement des principaux protocoles qui seront utilisés lors du développement de l'application.

Mais cette analyse a surtout révélé certaines difficultés. Ces difficultés vont malheureusement impliquer des modifications du cahier des charges. Modifications qui sont nécessaires à la réalisation du projet.

6.1. Modification du cahier des charges

6.1.1. Utilisation d'E-mail à la place des MMS

Après presque 6 semaines de recherche de documentation puis de tests, la connexion au serveur MMSC ne marchait toujours pas et lors d'envoi de donnée en HTTP le seul moyen de les récupérer était l'utilisation d'une Servlet. Il a donc fallu prendre une décision pour la suite du projet. Continuer à chercher dans la voie des MMS aurait pu être possible, mais au risque que le projet n'aboutisse pas du tout.

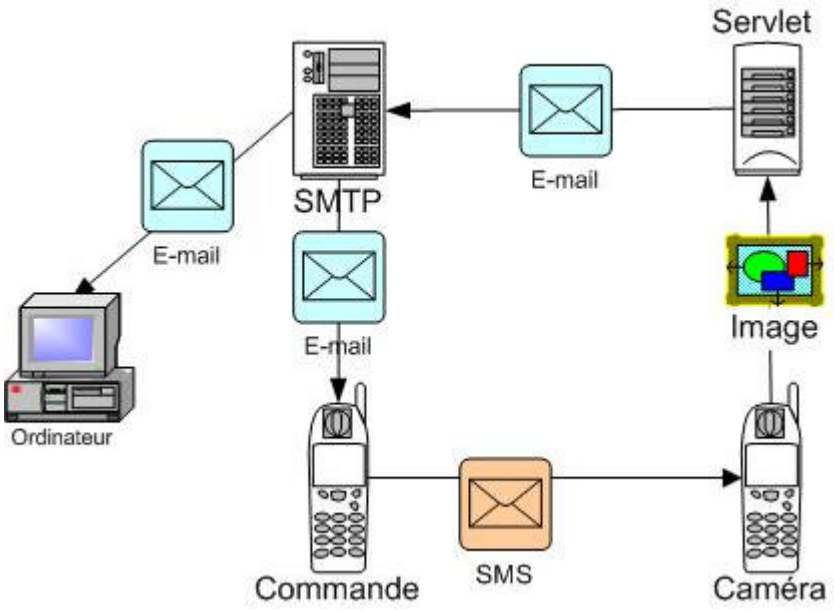
C'est pourquoi, j'ai décidé d'opter pour une autre solution, qui elle avait prouvé qu'elle marchait. J'ai donc décidé d'utiliser la connexion avec la Servlet qui, elle, fonctionnait pour envoyer la photo jusque-là et de la délivrer par mail par la suite. Délivrer l'image par MMS aurait pus être possible depuis la Servelet par le biais de l'interface MM7. Mais les inconvénients de cette interface (discuté plus haut) m'on fait opter pour l'utilisation d'e-mail.

6.1.2. Une deuxième application pour commander

Face au problème de la réception de SMS standard, il n'y a apparemment pas de solution disponible, pour capter ces messages.

Le recours à une seconde application dédiée à l'envoi de SMS est donc nécessaire pour la continuation du projet.

6.1.3. Résumé de ce qui sera réalisé



Ce qui va donc être réalisé est représenté sur le schéma ci-dessus. Le déroulement des opérations sera le suivant :

1) Un utilisateur se servira d'un téléphone avec le programme de commande pour envoyer un SMS à l'appareil de surveillance.
2) Le téléphone de surveillance prendra une photo et en transmettra le code à la Servlet.
3) La Servlet se connectera à un serveur SMTP pour envoyer un E-mail avec l'image en fichier attaché.
4) L'utilisateur se connectera au serveur SMTP soit à l'aide d'un téléphone, soit à l'aide d'un ordinateur. Ainsi il pourra récupérer la photo qu'il a commandé quelques secondes plus tôt.

7. Réalisation

Cette partie du rapport retrace les étapes de réalisations du projet. Je ne voulais pas rentrer trop dans le détail, ainsi chaque chapitre présente les choix qui ont été pris et les problèmes qui ont été rencontrés.

Pour avoir plus d'informations concernant l'implémentation du code, le choix de telle classe plutôt que telle autre, etc… Autant se rendre directement aux listings qui sont commentés de manière assez claire et compréhensible pour répondre aux questions qu'on pourrait qualifier de « bas niveau ».

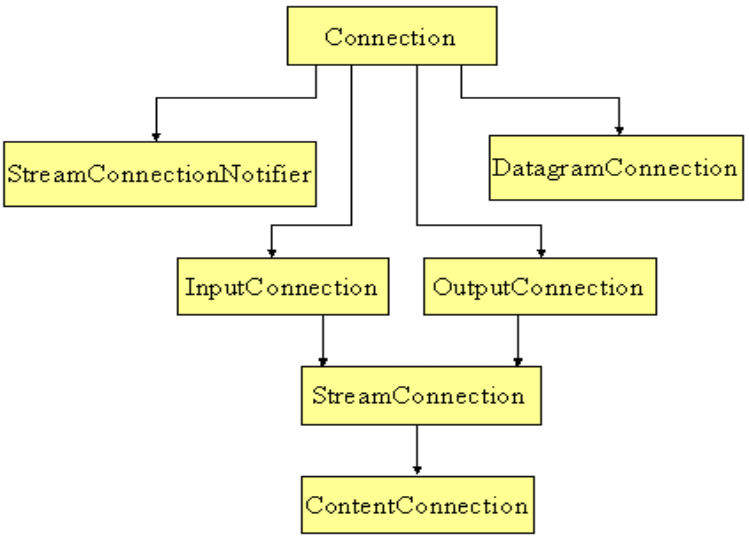
7.1. Environnement Java

Comme nous l'avons vu dans l'analyse, le développement pour téléphones portables est basé sur l'utilisation de la configuration CLDC et sur l'utilisation du profil MIDP. En plus de ces deux éléments standards, nous allons aussi utiliser des paquetages optionnels, un pour la gestion des SMS nommé WMA et un pour l'utilisation de la caméra nommé MMAPI.

7.1.1. Les composants de J2ME

CLDC

Le CLDC (Connected Limited Device Configuration) est constitué d'une API de base nécessaire au langage java (java.lang, java.io, java.util) ainsi qu'une API spécialisée dans l'accès réseau mobile : javax.microedition.io. C'est surtout cette partie réseau qui nous intéresse, car c'est elle qui a déterminé quel moyen utiliser pour communiquer l'image.



Le diagramme ci-dessus correspond aux différentes classes javax.microedtion.io. C'est grâce à ces implémentations que nous pourrons effectuer des requêtes HTTP qui seront réalisées à l'aide de la classe ContentConnection.

MIDP

Le MIDP (Mobile Information Device Profile) est une API JAVA de haut niveau permettant la gestion de l'interface utilisateur, la gestion de l'interface réseau et la gestion d'une base de données sur le mobile.

Cette API est, à ce jour, celle que vous trouvez sur vos mobiles "compatibles J2ME". Si l'on regarde un peu plus dans le détail ce qu'elle contient, on y trouve les trois packages suivants :

- **javax.microedition.lcdui** qui fournit les composants graphiques nécessaires à la création d'applications.
- **javax.microedition.midlet** qui fournit le composant application ainsi que les primitives gérant la vie de l'application.
- **javax.microedition.rms** qui fournit une possibilité de stockage d'informations.

7.1.2. Environnement de développement

Ci-dessous, voici la liste des composants qui ont été nécessaires au développement du projet. Cette liste représente ce qui a été utilisé, mais ne représente pas la seule possibilité. Il existe une multitude d'autres outils qui auraient bien pu être utilisés en complément ou en remplacement de ceux présentés ci-dessous.

J2ME Wireless Toolkit 2.0

WTK est le kit de développement fourni par Sun que l'on peut trouver sur le site leur site[1]. Il permet la création d'applications conçues pour fonctionner sur des appareils mobiles. On y trouve donc toutes les librairies nécessaires ainsi que le compilateur. Il contient aussi un émulateur qui permet de tester la compatibilité avec les différentes configurations et profiles :

- Connected Limited Device Configuration (CLDC), (JSR-039)
- Mobile Information Device Profile (MIDP), (JSR-118)
- J2ME Web Services, (JSR-172)
- Wireless Messaging APIs (WMA), (JSR-120)
- Mobile Media APIs (MMAPI), (JSR-135)

En plus de ces fonctions de base, on y trouve l'outil KToolbar qui permet de créer, modifier, compiler et exécuter des projets J2ME. Finalement, on y trouve aussi toute une série d'exemples qui sont très intéressants pour débuter une étude des possibilités de J2ME.

JBuilder

Certes, l'utilisation de WTK et d'un éditeur de texte standard peut suffire pour développer une application mais le travail peut être extrêmement simplifié avec l'utilisation d'un bon environnement de

[1] http://java.sun.com/products/j2mewtoolkit/download.html

développement. JBuilder étant l'environnement de développement choisi par l'EIVD, c'est donc lui qui sera utilisé pour ce projet.

Comme JBuilder est à la base conçu pour travailler avec la version standard de Java, il faut modifier son environnement pour travailler avec J2ME, cette modification intervient en installant une extension nommée MobileSet.

MobileSet

Ce produit livré gratuitement par Borland est téléchargeable depuis leur site[2]. Cette extension qui s'ajoute à JBuilder offre des outils de conception visuelle pour créer des applications mobiles, un émulateur de périphérique et un débogueur.

L'environnement de développement de JBuilder MobileSet, totalement intégré à JBuilder, permet donc la construction d'applications J2ME utilisant les plates-formes MIDP et CLDC.

Emulateurs supplémentaires

Les émulateurs fournis par Sun dans WTK, sont des émulateurs que l'on pourrait qualifier de génériques. Mais si l'on veut avoir une meilleure visibilité du résultat du développement, il est intéressant d'utiliser les émulateurs d'appareils réels. La plupart des fabricants de téléphones compatibles avec J2ME fournissent de tel émulateurs. Ainsi, pour ce projet, l'émulateur du téléphone Nokia 3650 à été utilisé. Comme pour le reste de ces outils, on peut aussi le trouver sur Internet[3].

[2] http://www.borland.com/products/downloads/download_jbuilder.html

[3] http://www.forum.nokia.com/main.html

7.2. La MIDlet

Tout comme les applets, les MIDlets sont contrôlées par le logiciel qui les lance. Dans le cas d'une applet, le logiciel utilisé est la JVM (Java Virtual Machine) qui est lancée par un navigateur web. Dans le cas d'une MIDlet, c'est la KVM du téléphone.

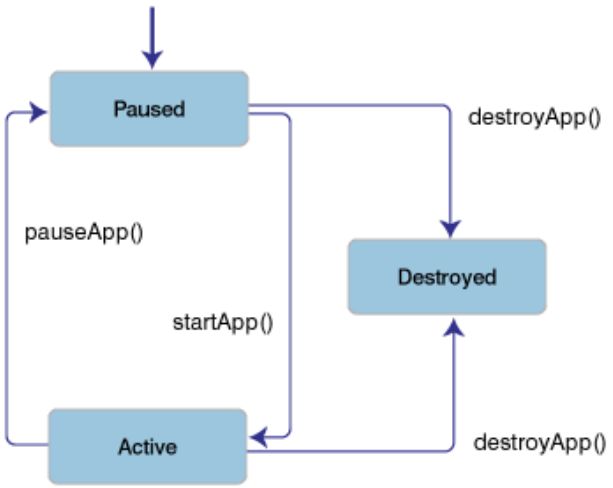
Il faut savoir que toutes les MIDlets dérivent de la classe MIDlet, disponible dans MIDP.

7.2.1. Cycles de vie

Une MIDlet a un cycle de vie un peu différent d'une application standard. En effet, la plupart des téléphones n'ont techniquement pas la possibilité de faire fonctionner plusieurs applications en parallèle. Il a fallu concevoir un système pour pouvoir tout de même passer d'une application à une autre sans qu'elles aient totalement terminé leur exécution. Pour cela chaque MIDlet doit obligatoirement déclarer les 3 méthodes suivantes :

- startApp()
- pauseApp()
- destroyApp(boolean unconditional)

Ces trois méthodes seront, lors de l'exécution, appelée par la KVM et feront passer la MIDlet d'un état à un autre comme le montre le schéma suivant :



Ainsi, à tout moment, notre application pourra être mise en pause. Il sera important à ce moment de procéder à une libération des ressources qui peuvent l'être.

7.2.2. Java Application Descriptor

Lorsque l'on voit les prix pratiqués [DOC 10] pour la transmission de données sur les réseaux mobiles. On se rend vite compte qu'il ne faut pas en abuser.

Comme il existe une multitude de configurations et de profil, il serait dommage qu'après avoir téléchargé un programme, on se rende compte qu'il ne peut pas fonctionner sur son téléphone. Un autre problème pourrait aussi être qu'il n'y a pas la place d'installer le logiciel sur son portable.

L'idéal serait que le téléphone puisse savoir à l'avance les spécificités du programme que l'on aimerait télécharger en terme de composant, de mémoire, etc..

C'est pour cela qu'à été crée le « Java Application Descriptor » (JAD). C'est un fichier qui contient toutes les informations utiles concernant la MIDlet. Il est généralement téléchargé avant le programme, ce qui permet de s'assurer de la compatibilité complète du terminal hôte.

Il est aussi possible d'ajouter des informations complémentaires, qui seront utilisées par le programme. Cela à l'avantage de pouvoir modifier certaines constantes, sans avoir besoin de recompiler les classes.

Pour accéder à ces informations, on utilise la méthode suivante :

```
MIDlet.getAppProperty("Nom du paramètre");
```

Voici ci-dessous le fichier *.jad utilisé dans MMSCam :

```
MIDlet-1: Pilotage, 2.png, MmsCamCommande
MIDlet-2: MMSCam, 1.png, MmsCam
MIDlet-Description: Programme de surveillance pour natel
MMS
MIDlet-Icon: 1.png
MIDlet-Jar-Size: 24076
MIDlet-Jar-URL: MmsCam.jar
MIDlet-Name: MmsCam
MIDlet-Vendor: D. Jeanmonod
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
adresseServlet:
http://www.mycgiserver.com/servlet/jeanmonod.EmailServle
t
portSms: 16753
```

On peut par exemple voir que le numéro de port utilisé pour l'envoi des SMS est stocké dans ce fichier. On y trouve aussi l'adresse du Servlet. On peut ainsi facilement modifier cette valeur.

Remarque

L'idée de ce fichier de description est très bonne, pouvoir éviter ainsi qu'un programme ne s'installe sur un matériel insuffisant à son exécution est une très bonne chose.

La surprise fut grande quand j'ai remarqué qu'il était impossible d'ajouter l'utilisation des paquetages optionnels. Ainsi il n'est pas possible de préciser que j'utilise les deux paquetages WMA et MMAPI. Ce qui est fort dommage car beaucoup de personnes ne savent pas si leur téléphone implémente ces librairies. Elles pourront donc télécharger l'application et constateront lors de l'exécution la non-conformité de leur appareil.

7.2.3. MIDlet Suite

Pour diffuser une application MIDP, il faut premièrement stocker le programme (c'est-à-dire la classe qui dérive de MIDlet), toutes les classes utilisées ainsi que les fichiers nécessaires au fonctionnement dans une archive compressée JAR.

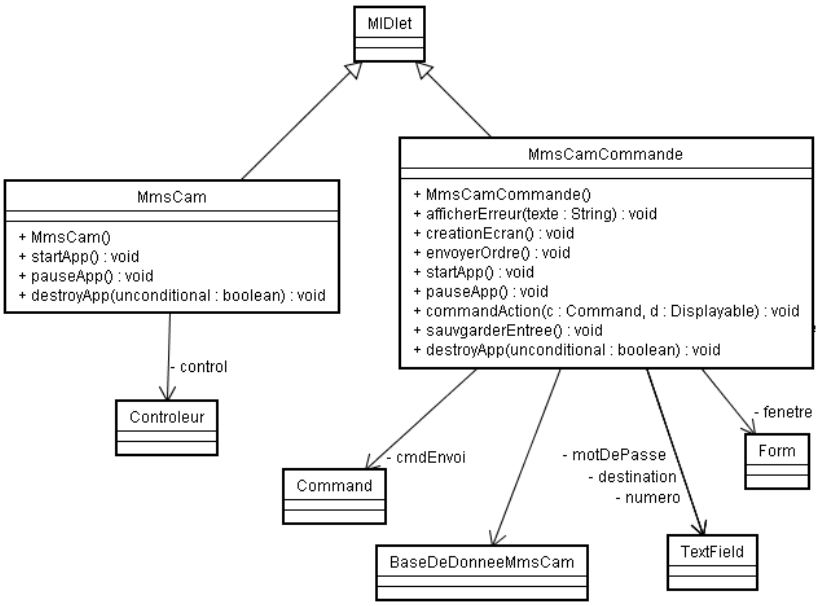
Plusieurs programmes peuvent être stockés dans la même archive, avec la possibilité de se partager les composants du JAR.

Ce qu'on appelle « MIDlet Suite » est simplement une suite de plusieurs MIDlets qui sont rangées dans la même archive JAR. Leur fichier JAD fait aussi partie de la suite.

Pour l'application MMSCam, la suite sera composée des deux applications, celle de commande et l'application principale. Elles se partageront ainsi les fichiers de classes, les fichiers de ressources tels que les icônes et le descripteur de la suite (JAD)

7.3. Structure du programme

Comme il a déjà été précisé l'application est composée de deux programmes. Le premier qui sert à envoyer les ordres possède une structure plutôt simple, le second, de par sa taille, à nécessité un découpage logique plus important. Ci-dessous, un aperçu de la structure générale de l'application :



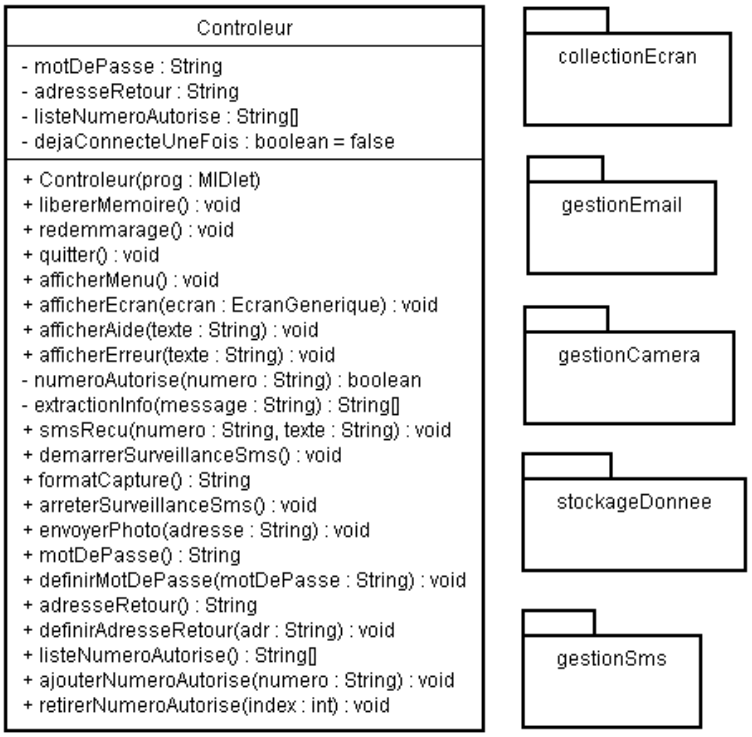
On peut voir que les deux programmes dérivent de la classe MIDLet, condition nécessaire pour qu'ils soient eux-mêmes considérés comme des MIDlets.

On peut aussi voir que le programme de pilotage (MMSCamCommande) utilise toute une série d'objets, tels que des zones de texte, un bouton, une fenêtre et le paquetage de base de données. Cela vient du fait que le corps de ce programme, qui est relativement réduit, est directement implémenté en son sein.

De l'autre côté, on voit que la seule classe qui est utilisée par le programme de surveillance est une classe nommée "controleur". C'est cette classe qui est le centre du programme.

La classe "controleur"

Cette classe qui est le cœur du programme, regroupe toutes les actions que peut effectuer le programme tel que "envoyerPhoto" ou "demmarrerSurveillance".



Pour un meilleur découpage des différentes activités, plusieurs paquetages ont été créés et sont utilisés par le contrôleur.

Les paquetages

Ces différents paquetages regroupent des classes qui ont une même utilité. Voici quelle est l'utilité de chacun d'eux :

gestionSms	Regroupe les classes chargées de détecter l'arrivée d'un ordre et de transmettre les informations au contrôleur
stockageDonnée	Regroupe les classes chargées de conserver les données entre deux utilisations du programme
gestionCamera	Regroupe les classes chargées de capturer une image lorsque nécessaire
gestionEmail	Regroupe les classes qui s'occupent de transférer les données du mail à la Servlet
collectionEcran	Regroupe les différents écrans qui composent l'interface graphique

Les chapitres suivants traitent de l'activité de ces différents paquetages, et tentent d'expliquer les décisions qui ont été prises.

7.4. Pilotage à distance

Ce chapitre retrace le développement du paquetage « gestionSms »

Le but de ce projet est de pouvoir piloter le téléphone muni de la caméra et cela à distance. Par pilotage, on entend lui demander de prendre une photo et de la transmettre à un destinataire. De plus, on aimerait intégrer un minimum de sécurité pour éviter que n'importe qui puisse envoyer des ordres. Ainsi deux systèmes d'identification vont être intégrés.

7.4.1. Informations à transmettre

Lorsque l'on désire commander le téléphone muni de la caméra, on doit bien entendu pouvoir lui dire de prendre une photo. On doit aussi pouvoir lui transmettre l'adresse e-mail du destinataire à qui il devra renvoyer la photo et, pour finir, la possibilité de transmettre le mot de passe qui protège le téléphone caméra. Donc, en résumé, les informations sont :

- L'ordre de prendre la photo
- L'adresse du destinataire de la photo
- L'éventuel mot de passe

7.4.2. Envoi d'un ordre par appel en absence

L'idée de cette option est assez simple. Lorsque l'on désire obtenir une photo de la zone surveillée, il suffit de téléphoner un court instant au téléphone caméra et de raccrocher. Le téléphone détectera cet appel en absence et prendra une photo qui sera envoyée à une adresse prédéfinie.

Le principal avantage de cette méthode est la gratuité de l'opération, mais en contre-partie, on ne pourra pas transmettre d'autres informations que celle de l'ordre de prendre une photo. Ainsi il faudra que l'adresse d'envoi de l'image ait déjà été définie sur le téléphone caméra. On ne pourra pas non plus effectuer de vérification d'un éventuel mot de passe.

Détection d'appel en absence

Pour implémenter la fonction de détection des appels en absence, j'ai commencé par étudier si cette option existait dans les librairies java disponible dans J2ME. Malheureusement, elle n'existe pas. Cela n'est pas très étonnant car, à part dans notre cas spécifique, je ne vois pas trop où une telle fonction pourrait avoir une utilité.

Comme expliqué plus haut, une MIDlet peut avoir plusieurs états (actif, en pause, détruite), une solution à la détection des appels en absence pourrait donc être liée à la transition entre ces différents états.

Ces transitions d'états permettent de libérer la mémoire lorsqu'un autre programme prend la main. L'idée de base serait de détecter le changement d'état qui a lieu lorsque le téléphone reprend la main pour traiter le coup de fil.

Hypothèse fausse

En fait, après plusieurs tentatives, il s'est avéré que le fait de recevoir un coup de téléphone, ne fait pas passer l'application en mode pause. L'application continue son exécution pendant toute la durée de l'appel.

Ces tests ont été réalisés sur un appareil Nokia 3650. Peut-être que pour d'autres types d'appareil, l'idée précédente peut être fonctionnelle, mais faute d'avoir pu essayer, il n'est pas possible de l'affirmer.

Il n'est donc pas possible, à ma connaissance, de détecter qu'un appel ait été reçu. C'est pourquoi l'option de détection des appels en absence à été supprimée du projet.

7.4.3. Activation par envoi de SMS

Avec cette technique, la transmission des informations se fait par SMS. Comme nous l'avons vu dans l'analyse, l'utilisateur ne pourra pas écrire son propre message. Il faut donc développer une application qui s'installera sur le téléphone « commande ». Cette technique permettra de transmettre toutes les informations nécessaires et cela à coût raisonnable, coût correspondant à l'envoi d'un SMS. Les deux applications devront donc avoir un numéro de port commun, qui sera choisi et inscrit dans le fichier JAD. Ce numéro de port devra être compris entre 16000 et 16999, tel que précisé dans la spécification SMS [DOC 13].

7.4.4. Restriction par numéro de téléphone

Puisque lors de la réception d'un SMS, il est possible de connaître le numéro de téléphone de celui qui a envoyé l'ordre, nous pouvons donc réaliser un second contrôle de sécurité. Il suffit de n'autoriser que certains numéros à envoyer des ordres.

Cette seconde sécurité n'entre pas en concurrence avec la sécurité procurée par le mot de passe, mais peut être vue comme un complément.

L'utilisateur devra configurer l'appareil muni de la caméra et préciser les numéros de téléphone qui sont autorisés à commander. Les ordres provenant d'autres numéros seront simplement ignoré.

Constatation n°1 : si aucun numéro n'est configuré, cela devrait signifier que personne n'a le droit d'envoyer un ordre !!!(inutile).

Constatation n°2 : Inversement, si l'on veut autoriser tout le monde, il faudrait introduire tous les numéros existants !!!(impossible).

Face à ces deux constatations, la modification suivante s'impose, il sera défini que si aucun numéro n'est précisé, alors tout le monde aura accès.

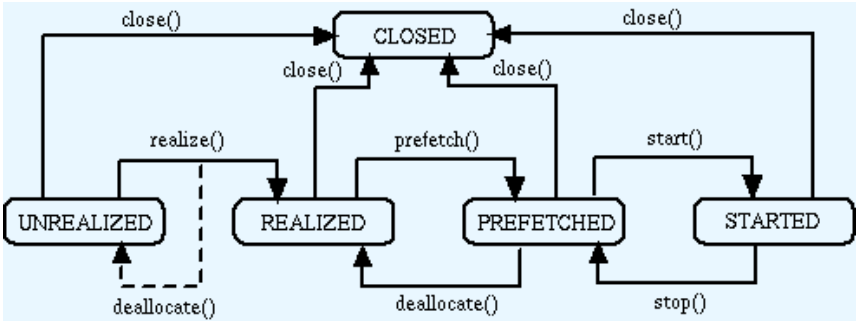
7.5. Capture d'une image

Ce chapitre retrace le développement du paquetage « gestionCamera »

La capture d'image n'étant bien entendu possible qu'avec des appareils munis d'une caméra, il est normal que cette option ne fasse pas partie des fonctions de base de MIDP. En fait, les fonctions se rapportant à la capture d'image, sont disponibles au travers d'un package optionnel nommé Mobile Média API (MMAPI).

7.5.1. Fonctionnement

Pour prendre une photo, il faut premièrement crée un objet du type « Player », que l'on trouve dans la librairie MMAPI. Ensuite il faut l'initialisé selon le schéma ci dessous :



Quand on a crée l'objet, il faut successivement appeler ces méthodes realize(), prefetch(), et start().

Après avoir initialisé les éléments de pilotage de la caméra, il suffit de faire appel à une méthode nommée getSnapshot() pour obtenir une photo.

7.5.2. Problèmes rencontrés

Image floue et bleue

A chaque capture, l'image était de très mauvaise qualité et de couleur bleutée (photo de gauche).

Ce problème venait du fait qu'une fois démarrée (méthode start()), la caméra a besoin d'un petit moment pour faire le point. Il suffisait donc de patienter une seconde entre le moment d'ouverture de la camera et la prise de la photo.

Mémoire pleine

A chaque fois qu'une deuxième photo est prise, l'application se termine en affichant le message d'erreur «Memory full ». J'ai passé passablement de temps à chercher la cause de cette erreur. Finalement, le problème a trouvé sa solution lors de l'ajout du délai entre l'initialisation de la caméra et la prise de vue (voir problème précédant).

Apparemment, le téléphone ne supportait pas un enchaînement trop rapide des opérations.

7.6. Transmission de l'image

Ce chapitre retrace le développement du paquetage « gestionEmail »

Comme nous l'avons vu dans l'analyse, le transfert qui devait à la base être réaliser à l'aide de MMS, a été remplacé par un transfert d'E-mail.

Dans l'analyse, nous avons aussi vu qu'avec le profile MIDP 1.0 il n'était pas possible d'envoyer directement des E-mails. Nous allons donc utiliser un serveur relais.

Pour envoyer un e-mail, il faut contacter un serveur de mail qui se chargera d'acheminer l'image à bon port. La connexion à un tel serveur se fait via le protocole SMTP (Simple Mail Protocol Transfer). Ce sera donc le rôle de la Servlet d'effectuer cette connexion.

Avantage

Le fait de passer par un serveur relais n'as pas que des inconvénients :

La connexion à un serveur SMTP nécessite de connaître un serveur SMTP et d'y avoir accès. Comme ces serveurs sont généralement mis à disposition par les FAI (Fournisseurs d'Accès à Internet), ils ne sont accessibles que pour les gens qui sont abonnés.

Cela signifie que chaque utilisateur devra configurer son logiciel pour lui indiquer un serveur SMTP valide. Une telle configuration peut paraître assez basique pour quelqu'un qui sait de quoi nous parlons, mais c'est loin d'être le cas pour "monsieur tout le monde".

En passant par un serveur relais, on peut utiliser son propre SMTP, et ainsi l'utilisateur n'a plus besoin de configurer quoi que se soit.

7.6.1. Les Servlets

Les servlets sont une technologie qui à été développée par Sun. Elles sont en quelque sorte des applets, mais s'exécutant du coté serveur cette fois. Il est alors possible de gérer les requêtes HTTP qui parviennent au serveur hôte et cela avec du code java, alors que généralement on aurait utilisé un script PHP ou ASP.

L'utilisation de servlets a plusieurs avantages, mais le principal est à mon avis qu'il est possible d'utiliser toutes les classes Java.

Utilisation

Afin de développer une servlet fonctionnant avec le protocole HTTP, il suffit de créer une classe étendant javax.servlet.http.HttpServlet.

Suivant le type de la requête que l'on désire traiter (GET ou POST), il suffit de définir la méthode adéquate :

Si la méthode utilisée est GET, il faut de redéfinir la méthode

```
void doGet(HttpServletRequest r, HttpServletResponse p)
```

Si la méthode utilisée est POST, il faut de redéfinir la méthode

```
void doPost(HttpServletRequest r, HttpServletResponse p)
```

En interrogant le paramètre de r (la requête) on peut obtenir les valeurs des variables que le client a transmises. C'est comme cela que l'on va lire les données de l'image.

Pour donner une réponse au client, il faut la définir en modifiant le paramètre p.

Une servlet de base a donc la structure suivante :

```
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletDeBase extends HttpServlet {

  public void doGet(HttpServletRequest req,
HttpServletResponse res)
     throws ServletException {

     // lecture de la requete
     // traitements
     // envoi de la reponse
  }
}
```

7.6.2. Envoi du courrier électronique

Le rôle de la servlet est très limité dans le cas de ce projet. En effet, l'image lui est transmise donc la seule action qui doit être réalisée est la connexion à un serveur SMTP pour lui communiquer le mail que l'on désire envoyer.

La structure du mail est, elle aussi, assez simple. C'est un mail "multipartie". Il contient d'abord une partie de texte, qui précise à quelle heure à été prise la photo. Et la deuxième partie qui contient la photo.

Ci-dessous, le code du E-mail :

```
Date: Date & heure
From: adresse du destinataire
To: adresse du destinataire
Subject: Photo MMSCam
Mime-Version: 1.0
Content-Type: multipart/mixed;Boundary=-=_NextPart;

---=_NextPart
Content-Type: text/plain;
charset="us-ascii"
Content-Transfer-Encoding: 7bit

Image capturée par un appareil MMSCam le Date & heure

---=_NextPart
```

```
Content-Type: application/octet-stream
Content-Disposition: attachment;
        filename= "nom du fichier"
Content-Transfer-Encoding: base64

Code de la photo

---=_NextPart
```

Note : Les valeurs grisées du mail ci-dessus sont celles qui devront être modifiée.

7.6.3. Serveur hôte

Une fois la classe serlvet réalisée, encore fallait-il trouver la place sur un serveur Web pour pouvoir l'utiliser.

Comme il était obligatoire que ce serveur ait une connexion à Internet (pour que le téléphone puisse le contacter), je n'ai pas pu installer ce serveur en local. J'ai donc dû utiliser un serveur existant.

Après quelques recherches sur Internet, je me suis orienté vers le serveur http://www.mycgiserver.com qui est un des seuls à proposer ce service gratuitement.

De plus, ce serveur a l'avantage de fournir un accès a un serveur SMTP; ainsi, j'ai pu utiliser directement le serveur de courrier fourni.

7.6.4. Le codage de l'image en base64

Dans le code du mail présenté en dessus, on peut voir que le champ suivant :

```
Content-Transfer-Encoding: base64
```

Cette ligne signifie que l'image est codée à l'aide du codec Base64, voyons donc de quoi il s'agit :

Introduction

Que ce soit lors de la transmission d'un mail ou lors de la transmission d'un MMS, ces messages peuvent traverser de nombreux intermédiaires avant d'atteindre leur but. Ces nombreux équipements sont issus de technologies et d'origines très diverses et ne travaillent pas avec les mêmes tables de caractères. Certains utilisent le code ASCII, certains l'ANSI et ainsi de suite. Il arrive donc souvent qu'un message soit modifié en traversant certaines parties du réseau. Qui n'a jamais reçu un mail ou les accents avec été transformé en caractère bizarre ? Ce genre de modification est courant, et nous nous débrouillons tout de même pour lire les messages ainsi déformés.

Le code d'une image

Pour optimiser au maximum la compression d'une image, on n'utilise pas des caractères mais un code binaire. Par contre, lors d'une transmission, il arrive souvent que le code binaire soit interprété comme du texte. Quand on ouvre un fichier binaire en mode texte, on voit que toute la palette des 256 caractères est utilisée. Si l'on ouvre un

fichier du type JPG dans un éditeur de texte, voilà ce que l'on va trouver :

```
Q9 ¦Ç ¯ÎïÄ{ÄGéà»UUºÔ7¦_EÁ0ËÑ¦ä3Ì¦ ü Ð ächÉ¦më?F].#Ükÿ'Æa
Öb× M ãa %íæ$~cë fg¸«+ ÇÍ¾ôYçø~[ËÁë¦ÛÂ¦$:ØÆ3-±-OOÖBÈátj¬
¯)ç>XÄIùj+$NÖ'ÈeƒOÈ¦ÿ¯nyK8Æ7ï¸dð -:Éw¦¦ä+FT ñS)ó0j  ¦],É
£¡(¦Sß i¸°¥Î^¨H7 ¥$ -ðÅä ¡i *ÆUþÍ-9 +t²* jÒ w¦d eºÞ¸WAf¦
í$l²-K ô,ïGOðôã Pü] +- Aù®ª÷{ «`D7¬2tûÚàV-³ À_ë+!Fh-¦i¸#
ÔªBB+w[',èØÙE+Æ8R9íÔÐ¼+ 4AÑ#½
```

Mais alors, comment une image qui, en mode texte composé de tous les caractères, peut-elle garder son intégrité lors d'un transfert si un simple texte accentué peut être altéré par une transmission ?

La norme Base64

La solution qui a été adoptée consiste à recoder les images pour leur transfert, cela avec uniquement des caractères standards et non accentués. On utilise donc la norme Base64 qui est clairement définie dans la RFC 2045 [DOC 15].

Le principe est qu'un un groupe de 3 caractères (3 octets) devient un groupe de 4 ensembles de 6 bits, les 6 bits étant codés selon un sous-ensemble de 64 caractères de l'US-ASCII qui est composé des caractères :

- Les lettres « A » à « Z » majuscule (code 0 à 25)
- Les lettres « a » à « z » minuscule (code 26 à 51)
- Les chiffres de « 0 » à « 9 » (code 52 à 61)
- Les caractères spéciaux « + » et « / » (code 62 et 63)
- Le caractère «= » qui est utilisé uniquement comme caractère de bourrage en fin de message pour que le nombre de caractère soit toujours multiple de 4.

Cela permet d'échanger n'importe quelle donnée binaire à travers des réseaux ne connaissant que le texte non accentué.

Il faut noter tout de même que cet encodage supplémentaire provoque une augmentation de la taille des fichiers encodés. En effet, si 6 bits deviennent 1 caractère (donc 8 bits), alors on a une augmentation de taille de 33%.

Voici un petit exemple de codage de 4 caractères ASCII :

Chaîne de base	%Z{9				
Caractères de base en ASCII	%	Z	{	9	
Valeur du code ASCII	37	90	123	57	
Valeur binaire (4 x 8 bits)	00100101	01011010	01111011	00111001	
Réorganisation (6 x 6 bits)	001001	010101	101001	111011	001110	010000
Valeur du code	9	21	41	59	14	16

Base64						
Caractère en Base64	J	V	p	7	O	Q
Chaîne encodée	JVp7OQ==					

Note 1 : On remarque que lors de la réorganisation des bits, s'il n'y a pas assez de bits pour le dernier groupe de 6 bits, des zéros donc sont ajoutés pour le compléter.

Note 2 : Une fois la chaîne encodée, il faut qu'elle soit un multiple de 4 caractères, on y ajoute donc des caractères "=" qui servent

Application du codage

Dans notre cas, ce codage en Base64 doit impérativement être réalisé pour créer le corps de l'e-mail. Ainsi ce travail pourrait être réalisé par le serveur relais. Ce choix se justifiait car la communication de l'image entre le téléphone et le serveur était 33% plus petite que si le codage Base64 avait été fait sur le téléphone. Une autre raison de ce choix était que le serveur possédait déjà les codecs pour réaliser cet encodage.

7.6.5. Problèmes : Image corrompue

La solution présentée ci-dessus a été testée à l'aide d'image du type PNG, avec succès, par contre des images de type JPG se retrouvaient corrompues en arrivant sur le serveur relais. Il y avait donc déjà une modification du code lors de ce premier transfert.

Ce problème a été très difficile à résoudre, car il était impossible d'utiliser le débuggeur pour voir ce qui était émis par le téléphone.

Déplacement du codage Base64

Au début, le codage en Base64 était réalisé sur le serveur relais, une des hypothèses aurait donc pu être que la transmission entre le téléphone et le serveur relais déformait l'image. Ainsi, la première mesure qui a été appliqué pour tenter de corriger cette erreur, à été de transférer la tache du codage en Base64 au téléphone et non plus au serveur uniquement.

Pour cela, il a fallu réécrire la fonction de codage, qui n'existait pas sur le téléphone comme cela était le cas sur le serveur.

Malheureusement cette modification n'a pas corrigé le problème des images corrompues.

Des espaces inattendus !

Finalement c'est en étudiant le code en Base64, qu'est apparu le problème. Il avait des espaces dans le code. Pour rappel, le codage Base64 est un codage à l'aide d'un sous-ensemble de l'ASCII, sous-ensemble qui ne contient pas le caractère espace. Mais alors comment donc ces espaces ont pu apparaître dans ce code ?

Après de nombreux tests et suppositions, la source du problème a finalement été trouvée.

Rappelez-vous le problème qui est survenu lors de l'utilisation d'HTTP, pour pouvoir transmettre des paramètres, il avait fallu configurer le champ d'en-tête précisant le type de données transmises comme ceci :

```
Content-Type : application/x-www-form-urlencoded
```

Ce champ signifie que les données transmises proviennent d'un formulaire web. Ceci est faux, mais nécessaire au bon fonctionnement du transfert.

Les formulaires HTTP

Ce que je ne savais pas, et qui aurait pu m'aider à résoudre ce problème c'est que lors de la transmission des données d'un formulaire, le navigateur WEB transforme tous les espaces en caractère +. Ceci vient du fait que généralement ces données sont transmises dans l'URL à l'aide de la méthode GET. Et puisque la syntaxe d'une URL n'accepte pas les espaces, la transformation en un autre caractère était obligatoire.

Dans notre cas, il n'y a pas de navigateur qui procède à ces transformations. Par contre, lorsque la servlet reçoit les données et regarde le type de contenu et puisqu'elle pense que les données proviennent d'un formulaire, elle remplace les caractères + par des espaces de manière à rétablir la situation.

Correction du problème

Le problème vient donc du champ « Content-Type » lors de la transmission de l'image, d'après les spécifications la valeur que je devrais mettre est «text/plain » qui signifie texte sans mise en forme, ce qui correspond au résultat d'un codage en Base64. Mais malheureusement, si cette valeur est précisée comme paramètre, alors la connections HTTP ne marche plus.

Il a donc fallu laisser la valeur fausse et procéder à une correction dans la servlet. Correction qui consiste simplement à reconvertir tous les espaces du texte reçu en caractère «+ ». Après cette modification, les images arrivent enfin dans un état cohérent.

7.7. Interface utilisateur

Ce chapitre retrace le développement du paquetage «collectionEcran»

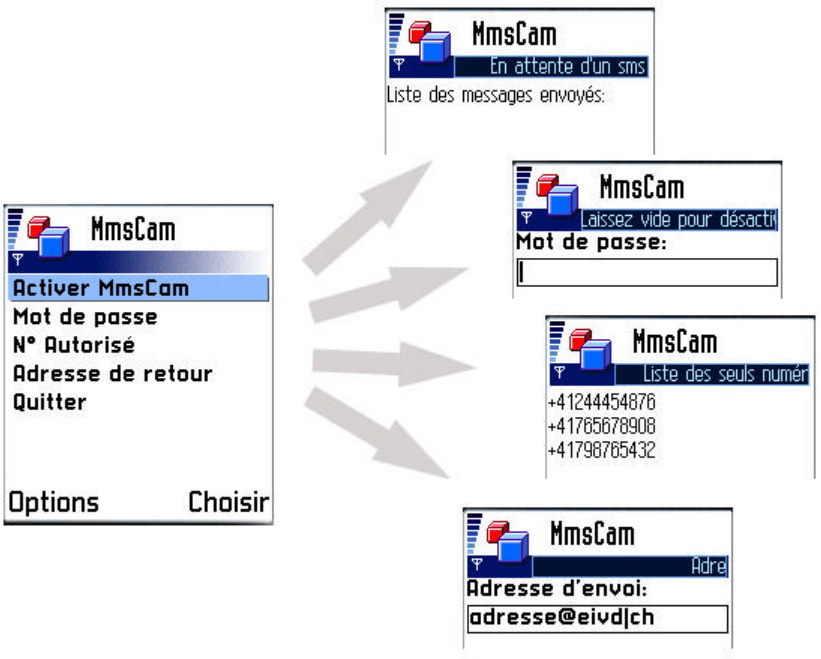
Etant donné la petite taille des écrans et la difficulté d'utilisation de certains claviers de téléphones portables, la création de l'interface utilisateur a une grande importance. Heureusement, au sein du profil MIDP, le paquetage javax.microedition.lcdui fournit les composants nécessaires à la réalisation d'une interface adaptée au téléphone portable.

7.7.1. Conception

Programme de surveillance

Pour cette partie du programme, étant donné le nombre d'options qui doivent être présentes, il faut trouver une structure qui permette de naviguer facilement entre les différentes parties.

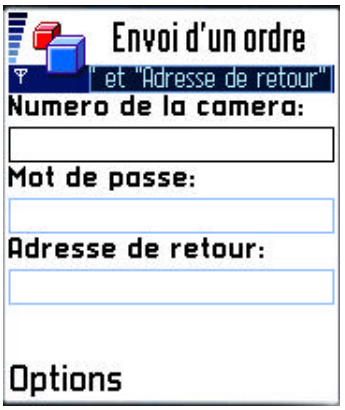
Le choix, qui a été fait, est la création d'un menu principal, qui permet de rentrer dans différents sous-menus. Ensuite, chaque sous-menu est équipé d'un bouton "Retour" qui permet de revenir au menu principal.



De plus, pour aider l'utilisateur qui serait un peu perdu, chacun des différents écrans sera équipé d'un bouton "Aide" qui permettra d'afficher un petit texte d'aide.

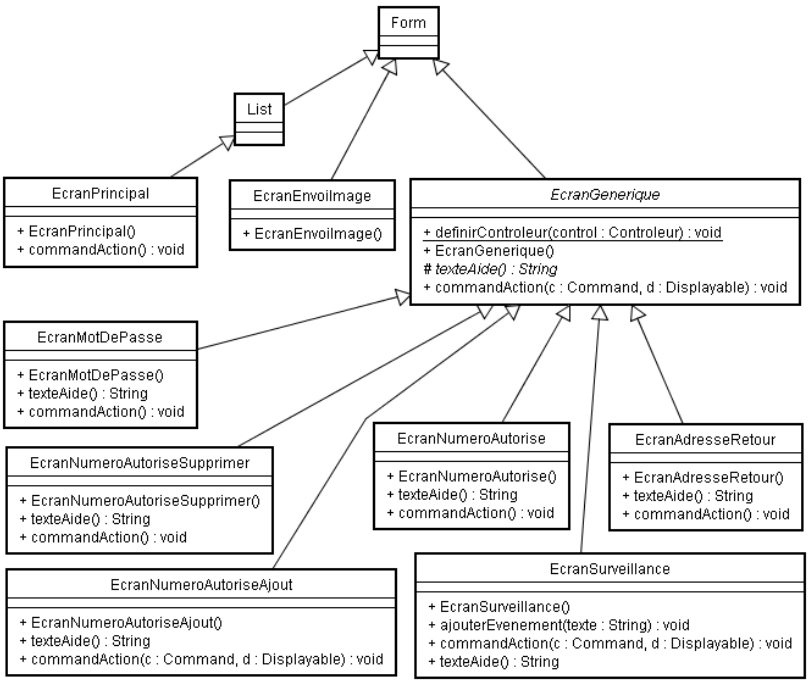
Programme de pilotage

Pour l'interface du programme de commande, rien de très difficile, le peu d'éléments que doit contenir cette partie de l'application permet de placer le tout sur un seul écran. Le résultat est présenté ci-dessous :



7.7.2. Réalisation

Voici le diagramme de classes des écrans de l'interface graphique :



Pour réaliser cette interface, c'est la classe "Form" qui a servi de base à tous les écrans, deux de ces écrans sont particuliers.

Le premier est l'écran principal qui ne dérive par directement de la classe "Form" mais de la classe "List", cette classe permet d'afficher une liste de choix, donc exactement ce dont nous avons besoin: le menu principal.

Le deuxième est l'écran qui s'affiche lors de l'envoi d'images. Car cet écran ne propose aucune interaction avec l'utilisateur, il demande simplement à l'utilisateur de patienter.

Tous les autres écrans dérivent de la classe "EcranGenerique". Cette classe fournit la base commune de tous les écrans, à savoir le bouton retour et l'aide affichable par l'utilisateur.

7.8. Stockage des informations

Ce chapitre retrace le développement du paquetage «stockageDonnee»

Si à chaque fois que l'utilisateur, veut utiliser le programme, il doit réintroduire le numéro de la caméra, l'adresse e-mail, le mot de passe, etc… Il va très rapidement se lasser et finir par ne plus utiliser du tout l'application.

L'idéal serait de pouvoir enregistrer toutes ces informations dans un fichier comme on le ferait sur un ordinateur. Malheureusement, le concept de Java, qui se veut totalement indépendant du support d'exécution, ne permet en aucun cas d'accéder à des fichiers.

Par contre, un autre mécanisme est fourni pour gérer le stockage persistant de données.

7.8.1. RMS

RMS (Record Management System) est une API de stockage persistant sur le terminal. C'est en quelque sorte une base de données indépendante du terminal. Chaque enregistrement est représenté sous forme de tableau d'octets. Les enregistrements sont stockés dans ce que l'on appelle un Record store. Si l'on veut faire un parallèle avec les SGBD relationnels, RMS correspond au SGBD lui-même et le Record store à la table.

Les enregistrements sont identifiés par un nombre entier. La valeur de l'ID du premier enregistrement est 1 et chaque nouvel enregistrement a une valeur ID augmentée de un.

7.8.2. Base de données

Dans le cas de notre programme, il nous faut gérer deux bases de données différentes, une pour l'application de commande et une pour l'application principale.

Etant donné que ces deux bases de donnée sont similaires, ce qui a été réalisé c'est une seule classe de manipulation de la base de données. Comme les deux programmes sont stockés dans la même suite, les deux on donc accès aux classes qui composent la suite, ainsi, on gagne de la place grâce à ce partage des ressources.

8. Bilan de la réalisation

8.1.1. L'application réalisée

Envoi de l'orde

Finalement, ce qui a été réalisé pour commander l'appareil muni de la caméra est un système de communication basé sur SMS. Comme il n'est pas possible de faire cette communication avec des SMS "standards". Il a fallu développer une seconde application qu'il faut installer sur le téléphone qui enverra l'ordre.

De plus, 2 mécanismes pour assurer une certaine sécurité ont aussi été développés. Premièrement, la possibilité de restreindre les numéros de téléphone qui on droit de donner des ordres, en gérant l'ajout et le retrait de numéros d'une liste. Deuxièmement, la possibilité de définir un mot de passe qui devra correspondre à celui transmis dans le SMS.

Transmission de l'image

La transmission est réalisée à l'aide d'un serveur relais. L'image est tout d'abord encodée en Base64, puis transmise par HTTP à un serveur relais connecté à Internet. Ce serveur se charge ensuite d'envoyer cette image par courrier électronique au destinataire désiré.

8.1.2. Problèmes résiduels

Réception des SMS

Il arrive parfois que le SMS de commande reçu, ne soit pas détecté par l'application. Ce problème se produit dans 10% des cas environ. Le message est donc placé dans la boîte de réception standard, l'ordre n'est pas exécuté.

Si l'on arrête la surveillance et qu'on la redémarre, le message est alors généralement consommé à ce moment là et l'application commence par répondre à l'ordre qui était resté en attente.

Perte de la connexion GPRS

Le problème vient du fait que lors de la connexion a GPRS, le téléphone demande à l'utilisateur de choisir le point d'accès qu'il veut utiliser. Si le téléphone est placé en surveillance et que personne n'est là pour choisir ce point d'accès, le téléphone restera en attente indéfiniment.

Pour corriger partiellement ce problème, il a été ajouté une "pré-connexion" de manière à ce que cette question soit posée avant que l'utilisateur ne place sont téléphone pour une surveillance.

Le problème se pose lorsque le téléphone perd la connexion GPRS car le téléphone reposera la question lors de la réouverture.

Ce problème ne se pose pas souvent puisque généralement le téléphone garde sa connexion indéfiniment. Le cas peut se produire, par exemple, si l'appareil caméra est placé sur un véhicule et que celui-ci quitte momentanément la zone de couverture GSM.

Blocage de la transmission

Dernier problème, il arrive que lors de l'envoi de l'image au serveur Web, la transmission s'arrête pendant un temps variable qui peut durer jusqu'à 5 minutes. Cette interruption se produit aussi dans environ 10% des cas.

Il n'a pas été possible d'identifier pourquoi. Peut-être le problème vient-il du serveur Web public qui est utilisé (mycgiserver.com). Quoiqu'il en soit, ce problème ne perturbe pas l'envoi de l'image puisque l'E-mail parvient bien à son destinataire. Le seul désagrément est le retard de la réception du courrier électronique.

9. Evolution future

Le programme qui a été réalisé dans le cadre de ce projet, devra évoluer rapidement pour avoir un avenir commercial. Je vais tenter de présenter les évolutions qui devront avoir lieu dans un futur proche.

9.1. Passage à MIDP 2.0

Le passage à la nouvelle version du profil MIDP va apporter de nouvelles possibilités qu'il serait très intéressantes d'exploiter.

9.1.1. Meilleure interface graphique

Les grandes nouveautés de MIDP 2.0 sont surtout présentes au niveau des composants d'interface qui deviennent plus complets et donc permettent de créer des interfaces utilisateurs plus conviviales et plus simples à utiliser.

9.1.2. Meilleure interactivité entre les téléphones

Avec la version 1.0 du profil MIDP, les opérations qui ont un coût comme l'envoi de SMS par exemple, nécessitent toujours une confirmation de l'utilisateur avant d'être effectuées. Ceci pour éviter qu'une application abuse des fonctionnalités payantes du téléphone au dépens de son propriétaire, ce qui est fort compréhensible.

Malheureusement, cette façon de faire limite considérablement l'activité des programmes qui, comme MMSCam, doivent être conçus pour fonctionner sans la présence d'un utilisateur pour donner sans arrêt l'autorisation d'envoyer ou non des messages.

Avec cette nouvelle version, une nouvelle notion est intégrée, celle d'application de confiance qui peut, si l'utilisateur à donné son autorisation au début, utiliser toutes les fonctions payantes sans qu'une confirmation soit demandée à chaque fois.

Ceci est très intéressant actuellement lorsque l'utilisateur envoie un SMS de commande. Il ne reçoit aucun message de confirmation en retour. Si le mot de passe était erroné, il ne le saura pas, alors qu'avec MIDP 2.0 on pourrait automatiquement lui signaler son erreur par SMS. On arriverait ainsi à une meilleure interactivité entre les deux téléphones et on pourra piloter l'appareil MMSCam non plus à l'aveugle, mais en ayant un retour des actions effectuées.

Soulignons quand-même que ce retour à un coût, puisque dans ce cas il est quand même du prix d'un SMS par message transmis.

9.1.3. Abandon du serveur relais

Dernier avantage de ce nouveau profil, la possibilité d'effectuer des communications en mode connecté. Alors qu'avec la version 1.0 on ne pouvait effectuer que des transmissions de type UDP, on peut donc maintenant établir des connections TCP.

L'avantage de cette évolution, pour la transmission de l'image, est qu'on va pouvoir se passer du serveur relais entre le téléphone et le serveur de mail.

Comme nous l'avons vu précédemment, le protocole SMTP, requiert une connexion TCP\IP. C'est pourquoi nous avons dû mettre un intermédiaire entre le serveur de mail et le téléphone.

Avec cette évolution de la version 2.0, le téléphone va pouvoir lui-même directement communiquer avec le serveur SMTP.

Cette modification aura deux conséquences. La première plutôt négative, est que pour utiliser MMSCam, l'utilisateur devra commencer par fournir les coordonnées à un serveur SMTP dont il a l'accès. De la même manière qu'il fournit ces données lors de la configuration d'un compte de messagerie.

La deuxième conséquence, beaucoup plus intéressante: le programme devient dès lors complètement autonome et ne requiert plus l'utilisation du serveur relais. L'application devient donc utilisable sans qu'une infrastructure spécifique soit nécessaire à son bon fonctionnement.

9.2. Utilisation de la librairie MMS JSR 205 (WMA 2.0)

A la base, ce projet devait être basé sur une utilisation des MMS. Suite aux problèmes techniques rencontrés et décrits dans ce document, la transmission des images par MMS à été abandonnée et remplacée par une transmission par e-mail.

Cette modification à été nécessaire pour contourner certains problèmes et le manque de support des MMS dans la version java utilisée lors du développement.

Ce manque de support devrait être comblé rapidement puisqu'un groupe d'experts de la communauté java est en train de développer un package d'extension devant prendre en charge la communication par MMS.

En fait, ce package sera simplement l'extension de la librairie WMA (JSR 120). Renommé en WMA 2.0 (JSR 205), on aura donc la possibilité d'envoyer et de recevoir des MMS facilement depuis une application J2ME, comme on peut déjà le faire actuellement avec les SMS et les CBS.

A l'heure actuelle, c'est-à-dire décembre 2003, le développement de cette librairie est presque terminé puisqu'on peut déjà la trouver en version bêta. La version finale devrait sortir d'ici quelques mois et on devrait trouver rapidement des téléphones compatibles avec cette version 2.0 de WMA.

Selon une communication non-officielle d'un employé de Swisscom, les premiers terminaux compatibles avec cette librairie devraient sortir au premier semestre 2004.

A ce moment, il serait très intéressant de modifier le projet en conséquence, ce qui permettrait ainsi de retrouver l'idée de base du projet, c'est-à-dire, une communication par MMS, qui est plus adaptée entre deux téléphones qu'une communication par e-mail.

10. Conclusion

Avant de commencer ce travail, je pensais qu'il serait facile d'atteindre les objectifs fixés par le cahier des charges. Mais au fur et à mesure de l'avancement de l'analyse, je me suis rendu compte que de nombreux problèmes allaient me rendre la tâche difficile.

C'était la première fois que je réalisais un logiciel destiné à fonctionner ailleurs que sur un ordinateur de bureau. Ainsi de nombreuses difficultés que je n'avais pas prévues me sont apparues. Parmi ces problèmes il faut souligner :

Déboguage à l'aveugle

Le plus gros problème qui s'est posé à été l'identification des problèmes réseau. Sur un ordinateur, il est possible d'installer un programme d'espionnage pour voir exactement quelle est l'activité du réseau et ce qui s'y passe. On peut ainsi contrôleur que les trames émises correspondent bien à ce que l'on désire. Mais en travaillant avec un téléphone portable, il est impossible d'installer un tel programme puisque le réseau est sans fil et donc ne possède par de point de connexion.

Manque d'information

Certes, dès le début, je savais que j'allais devoir rechercher énormément d'informations. Premièrement, pour combler le manque du fait de ma formation en orientation logiciel et non télécom, ce projet étant destiné à un étudiant de télécom.

Mais aussi parce que le domaine téléphonie mobile évolue extrêmement vite; de nouvelles technologies, spécifications, etc. sont continuellement mise en service.

Mais tous cela restait dans le domaine du prévisible. Par contre, ce que je n'avais pas imaginé, ont été les problèmes liés à l'utilisation du réseau de téléphonie. Le plus gros problème, a été cette histoire de requête HTTP tronquée. Quelques personnes ont pu me donner un début de réponse dans divers forums. Par contre je n'ai trouvé aucune informations officielles qui ont pu m'aider.

11. Référence & Bibliographie

Ce projet traite d'un sujet à la pointe de l'actualité et nécessite d'être informé au maximum des avancées technologiques effectuées dans le développement pour appareils portatifs. C'est pourquoi il est très difficile, voir même impossible, de s'appuyer sur des ouvrages écrits. Ainsi la majeure partie des références qui vont être citées ci-dessous, sont des sites Internet spécialisés ou des forums de discussion.

11.1. Livres

[LIV 1] « Introduction to 3G Mobile Communications 2nd edition »
Juha Korhonen 2003
Artech House Publishers Norwood USA

[LIV 2] « Learning Wireless Java »
Qusay H. Mahmoud Jan. 2002
O'Reilly Sebastopol USA

11.2. Documents techniques

[DOC 1] « Using the Nokia 3650 Messaging Api »
Version 1.0 06 sept. 02
Exemple de programmation en C++ qui crée des MMS et les envoies, document trouvé sur le site de Nokia [SIT 1].

[DOC 2] « Using the Nokia 3650 Camera »
Version 1.0 06 sept. 02
Exemple de programmation en C++ qui permet de prendre des images avec la caméra et de les stocker sur le natel, document trouvé sur le site de Nokia [SIT 1].

[DOC 3] « Datasheet : Java 2 Platform Micro Edition »
Présentation générale des buts et fonctionnalités deJ2ME
http://java.sun.com/j2me/j2me-ds.pdf

[DOC 4] « Brief Introduction to MIDP Programming »
Version 1.0 18 nov. 02
Ce documents propose deux exemples simples d'introduction à la programmation de MidLet, document trouvé sur le site de Nokia [SIT 1].

[DOC 5] « The Wireless Messaging API »
by C.Enrique Decembre 2002
Article de présentation de la librairie WMA, qui permet d'envoyé ou de recevoir des messages (MMS ou SMS), article trouvé sur le site de Sun [SIT 2]

[DOC 6] « How to Create MMS Service »
Version 4.0 26 juin 2003
Document qui explique les principes de la création et d'envoi de MMS, document trouvé sur le site Nokia [SIT 1]

[DOC 7] « Mobile Media Api Overview »
de Jonathan Knudsen Juin 2002
Documents de présentation des fonctionnalités de la librairie MMAPI, documents trouvé sur le site de Sun [SIT 2]

[DOC 8] « Camera MIDLet : A Mobile Media API Example »
Version 1.0 24 janvier 03
Document qui explique, au travers d'un exemple, l'utilisation de la librairies MMAPI dans le buts de capturer une image avec la caméra d'un téléphone, documents trouvé sur le site de Nokia [SIT 1]

[DOC 10] Les tarifs de communication mobile appliqués par Swisscom en novembre 2003
http://www.swisscom-mobile.ch/Objekte/vipsystemeuploadcharges_11_01-fr.pdf

[DOC 11] Un résumé de la technologie java pour les téléphones portables.
http://www.nokia.ch/french/phones/technologies/java/faq.html

[DOC 12] Rapport de stage de Nicolas Frick
Juillet 2001
Implémentation de nouveaux services à valeur ajoutée sur les réseaux radio-mobiles GSM
http://www.chez.com/jaaayyy/html/Radiomobiles/stage/Sommaire.html

[DOC 13] Spécification ETSI 3.40 de GSM
Technical realization of the Short Message Service (SMS)
http://www.3gpp.org/ftp/Specs/archive/03_series/03.40/0340-600.zip

[DOC 14] RFC 822 Crocker, D., "Standard for the Format of ARPA Internet Text Messages," Department of Electrical Engineering, University of Delaware, August 1982.
http://www.w3.org/Protocols/rfc822/rfc822.txt

[DOC 15] RFC 2045 Multipurpose Internet Mail Extensions (MIME) Part One :
Format of Internet Message Bodies
http://www.ietf.org/rfc/rfc2045.txt

[DOC 16] WAP 206 : WAP MMS Client Transactions, Version 15-Jan-2002
http://www.openmobilealliance.org/wapdocs/WAP-206-MMSCTR-20020115-a.pdf

[DOC 17] Wireless Application Protocol : MMS Encapsulation Protocol, Version 05-Jan-2002
http://www.openmobilealliance.org/wapdocs/WAP-209-MMSEncapsulation-20020105-a.pdf

Beaucoup d'autres documents ont été consultés lors de ce travail de diplôme, mais uniquement dans une optique de «culture générale » que j'ai eu besoin d'acquérir.

11.3. Site Internet de référence

[SIT 1] Site d'aide aux développeurs de Nokia, ce site très complet fourni des exemples de programmes, des documents de références, un forum de discussion et bien d'autres outils d'aide.
http://forum.nokia.com

[SIT 2] Site de Sun, ce site fourni des toute la documentation possible sur les librairies Java J2ME crée par Sun, dont MIDP, MMAPI, WMA, etc…
http://java.sun.com/j2me/

[SIT 3] Site d'information en français concernant les technologies de développement sur téléphones portables
http://www.cellconcept.com

[SIT 4] Site de banalisation des technologies informatiques, très utiles pour obtenir des résumés de technologie.
http://www.commentcamarche.net/

[SIT 5] Travail d'Etude et de Recherche de Laurent BLIN
Novembre 2001
http://www.lirmm.fr/~ajm/Cours/01-02/DESS_TNI/TER9/index.htm

[SIT 6] Recommandation W3C
Documents traitant des formulaires dans les documents HTML
http://www.w3.org/TR/1999/REC-html401-19991224/interact/forms.html

[SIT 7] Divers articles sur J2ME et sur le développement en général
http://developpeur.journaldunet.com/

11.4. Forum de discussions

[FRM 1] Forum de discussion de Nokia, ce forum constituera probablement la meilleure source d'aide. Très fréquenté, les questions trouvent réponse moins d'une journée, soit par des autres développeurs, soit par le personnel professionnel de Nokia

http://discussion.forum.nokia.com/forum/

[FRM 2] Forum de discussion de Siemens, beaucoup moins fréquenté et moins sérieux que celui de Nokia, mais pouvant apporter une aide tout de même.

https://communication-market.siemens.de/portal/ puis cliquer sur Developer Portal

[FRM 3] Forum de discussion en français, traitant du monde de Java embarqué.

http://www.cellconcept.com/phpBB2/index.php

[FRM 4] Forum de Now SMS, entreprise active dans l'offre d'accès à des serveurs MMSC et SMSC

http://www.nowsms.com/messages/

Il existe bien d'autres forums, mais ces 4 sont les principaux que j'ai utilisés pour la rédaction de ce document.

12. Listes des abréviations utilisées

API	Application Programming Interface
ASP	Active Server Page
C++	Extension du langage C
CBS	Cell Broadcast Service
CDC	Connected Device Configuration
CLCD	Connected Limited Device Configuration
GIF	Graphics Interchange Format
GSM	Global System for Mobile Communication
HTML	HyperText Makeup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
J2ME	Java 2 Micro Edition
JAD	Java Application Descriptor
JAR	Java Archive
JPEG	Joint Picture Expert Group
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	Kilobyte Virtual Machine
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extansion
MMAPI	Mobile Media API
MMS	Multimedia Message Service
MMSC	Multimedia Message Service Center

PHP	PHP: Hypertext Preprocessor
PNG	Portable Network Graphics
POP	Post Office Protocol
RMS	Record Management System
SMS	Short Message Service
SMSC	Short Messaging Service Center
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WMA	Wireless Messaging API
WSP	Wireless Session Protocol
WTK	Wireless ToolKit

13. Annexes

13.1. Mode d'emploi du programme

13.1.1. Configuration minimale

Pour pouvoir utiliser le logiciel MmsCam, il faut deux téléphones portables, qui ont les caractéristiques suivantes :

- Support du profile MIDP 1.0 et de CLDC 1.0
- Support de l'api WMA (JSR 120)
- 30ko d'espace mémoire disponible

Uniquement pour l'appareil qui sera utilisé pour prendre les photos :

- Support de l'api MMAPI (JSR 135)

13.1.2. Diffusion du logiciel

Pour diffuser ce logiciel, il existe plusieurs possibilités, mais dans chacun des cas, les deux seuls fichiers qui doivent être fourni à l'utilisateur sont :

- MmsCam.jad
- MmsCam.jar

Ces fichiers doivent être transféré sur le téléphone, pour cela il y a plusieurs solutions :

- Téléchargement sur le téléphone d'un E-mail contenant les deux fichiers en pièces jointes.
- Accès aux fichiers placés sur un serveur Web (Accessible temporairement sur www.lombric.ch/mmscam)
- Transfert à l'aide de l'infrarouge
- Etc…

Dans tous les cas, les deux fichiers doivent se trouver dans le même répertoire ou le même E-mail.

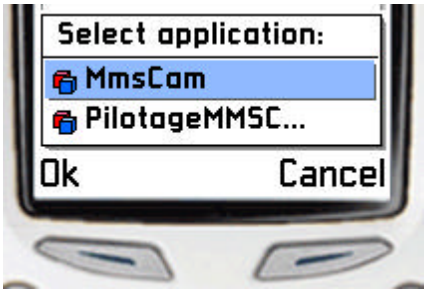
13.1.3. Installation du programme

Etant donné que l'installation diffère selon les téléphones, il n'est pas possible de donner une marche à suivre précise. Mais normalement, le simple fait d'ouvrir le fichier MmsCam.jad suffit à installer l'application.

13.1.4. Exécution de la MIDlet

Lors du démarrage, vous devez choisir quelle application vous désirez; vous avez le choix entre :

- MMSCam : qui est l'application de surveillance
- PilotageMMSCam : qui est l'application d'envoi des ordres



ATTENTION : Ne jamais envoyer un ordre avec l'application de pilotage, si l'application de surveillance est activée sur le même téléphone. Cette action aura pour effet de rendre l'application de surveillance inopérante.

13.1.5. L'application de surveillance

Le premier écran qui s'affiche lors de l'ouverture de l'application de surveillance est le menu principal.

Vous pouvez vous déplacer avec les flèches et sélectionner une option à l'aide du bouton "Choisir".

Activer MMSCam

C'est option permet d'activer le mode de surveillance; à partir du moment ou vous aurez activé MMSCam, le programme sera sensible à l'arrivée de SMS de commande.

A chaque fois qu'un SMS de commande sera reçu, l'écran affichera l'action effectuée (image envoyée, problème de transmission, mot de passe erroné, etc…) Ainsi en récupérant le téléphone qui à effectué une surveillance, vous pourrez savoir quelles sont les actions qui on été effectuées.

Mot de passe

C'est là que vous pourrez définir un mot de passe. Si vous laissez le champ mot de passe vide, alors aucun mot de passe ne sera nécessaire pour ordonner la prise d'une photo.

Si vous définissez un mot de passe, celui-ci devra impérativement être transmis dans le SMS de commande.

N° Autorisé

Cet écran affiche la liste des numéros des téléphones qui peuvent envoyer des ordres. En ajoutant ou en enlevant des numéros vous pouvez ainsi définir précisément qui a droit de commander des photos.

Pour ajouter ou enlever des numéros, il suffit d'utiliser les options du menu.

Si aucun numéro n'est spécifié, alors tous les numéros sont autorisés.

Note : Les deux options de sécurité, à savoir la protection par mot de passe et la limitation des numéros autorisés à donner des ordres, sont complémentaires. Vous pouvez très bien définir une liste de numéros et quand même demander un mot de passe.

Adresse de retour

Ici vous pouvez spécifier une adresse E-mail par défaut. Si un SMS de commande sans adresse E-mail est reçu, c'est à cette adresse que sera transmise la photo.

Quitter

La dernière option permet de quitter le programme.

ATTENTION : Utilisez la touche rouge de votre téléphone n'a pas le même effet. Si vous sortez de l'application à l'aide d'une autre méthode que cette option "Quitter" l'application sera toujours active.

13.1.6. L'application d'envoi des ordres

Cette application, beaucoup plus simple, vous permet d'envoyer les ordres de prise de photos.

Les trois champs que vous devez remplir vous permettent de préciser :

- Le numéro de l'appareil à qui envoyer le message.
- Le mot de passe éventuel.
- L'adresse E-mail à qui envoyer la capture.

Une fois que vous avez rempli les informations nécessaires, choisissez "envoyer" dans le menu.

A ce moment, le téléphone va vous demander si vous voulez envoyer le message.

Choisissez Oui. Si vous venez à choisir non, le message ne sera pas envoyé.

13.2. Communication sur les forums

13.2.1. Discussion du problème HTTP [FRM 1]

Developer Discussion Boards
(http://discussion.forum.nokia.com/forum/index.php)
- **Networking**
(http://discussion.forum.nokia.com/forum/forumdisplay.php?forumid=56)
-- **http post is not working on 3650**
(http://discussion.forum.nokia.com/forum/showthread.php?threadid=33407)

Posted by jeanmonod on 11-17-2003 10:00 AM:

http post is not working on 3650

Hello,

I try to send some data to a web serveur using the http post method and to get the data with a PHP page. The problem is that when the php script try to read the variable, it's empty.

The midlet code, is the one that is provide by sun on the address :
http://wireless.java.sun.com/midp/a...PostMidlet.java
And the code of the php page is like follow :
<?php
mail("info@lombric.ch",$_POST['name'],$_POST['name']);
echo "Confirmation";
?>
I'm using a nokia 3650 to make the test.
I know that the connection is ok and that the script had been executed, because I get the mail, but the variable 'name' is empty !!!
If someone add an idee why is not working please tell me?

Posted by shmoove on 11-17-2003 10:42 AM:

Sometimes the network carrier has things set up so that POST's are cut off. This sounds like what your problem may be.
Try installing a sniffer on your server to see exactly how the request your sending looks, and I think you'll see that your POST data is not arriving, and maybe some of your headers have been mangled with.
The solution for this is usually to use an Internet access point instead of a WAP access point (by setting the gateway IP to 0.0.0.0 in the connection settings of the phone).

shmoove

Posted by imarenic on 11-17-2003 11:40 AM:

HTTP POST seems to be common pitfall for developers these days.

I had same the same problem. Just look at thread:
http://discussion.forum.nokia.com/f...&threadid=33254

What shmoove told you is probably write about carriers cutting POST's. But it shouldn't happen to local machine on local server using the emulator.

I hope there is sollution in PHP for fetching POST data. I couldn't find it. And I've searched it for days.

PHP works fine when classic web clinets are POSTing data. Maybe we should give fake info for user agent.

However, PERL and ASP fetch posted data pretty well.

You may try them as an alternative.

Greetings,
Ivan

Posted by aled.morris on 11-18-2003 12:20 AM:

maybe due to tcpip chunking...

we had similar problems in a Java servlet on tomcat 4.1, where seemingly the parameters were being stripped out of the request. however, in that case, we noticed that there was some body content - lo and behold, the parameters were actually in the message body, but had not been parsed by tomcat to be presented through the servlet interface. so i think actually this was a tomcat problem but perhaps common to other servers?

so the solution would be to not rely on your server to extract the parameters, but try and pick up what it regards to be the message body which would look something like:

action=email&subject=j2me_network_access&message=all_screwed_up&recipients=operators

which you can now parse to breakdown into parameter & value pairs. hopefully!

after this we helped another developer fix the same problem with a SE t610 -> tomcat. on closer investigation (using Ethereal), the chunking of the request was different to other phones - the request was coming in 3 chunks (it was something like (1) HTTP POST line (2) headers and (3) parameters) whereas other phones were posting 2 chunks.

this might help explain why you can see this behaviour through one network's gateways and different behaviour on other networks - it is down to the gateways (and their configurations) as to whether they keep the same chunking or run them together in a buffer before forwarding? or whatever other grim messing around as shmoove says...

btw, for direct internet there may be more settings to change than just

gateway IP, but usually problems i've had with WAP APN vs Internet APN is whether the request gets through at all.

good luck,
dan.

Posted by jeanmonod on 11-26-2003 10:27 AM:

Still not working with php...

First Thank all of you for the answers...

I have been trying many things to make working php and midlet post, but without any result.
At the end I'm using a servlet, that can fetch the information that I need.
If someone, find something new for php, please tell-it but at least you know that with a servlet, all is working fine

13.2.2. Discussion de la transmission MM1 [FRM 4]

Hello,
I'm trying to send a MMS from my Midlet using the HttpConnection. Like I' have been looking in some document, sending an MM1 is based on a WSP/HTTP Post method.
But when I try to do it, I didn't get any answers from the MMSC Proxy.
Maybe I did a mistake with the encapsulation of my MM, but I would like to have a confirmation.
If someone has already did that or if you think that what I'm trying is impossible for some reasons, please let me know… Thanks

---

Some MMSCs will simply ignore or disconnect a very badly formatted PDU.

Are you sure that you're actually getting a connection to the MMSC over HTTP? How are you physically connected to the MMSC ... can the MMSC identify who you are? (Many MMSCs require that you connect through a WAP proxy.)

Since so many people want to see examples ... a simple MMS session is shown in the following thread:

http://www.nowsms.com/discus/messages/12/522.html

13.2.3. Discussion de la transmission MM1 [FRM 4]

Hi,
I'm looking for a sample with explanation How to build MMS Message. I can't find anything usefull.

I have created header part, text header part and I have added text data. But now I need to now how to build Image Header part. I have sample of Image header part, but I don't understand it.

8C 84 .... Message type - MMS Message
8D 90 .... MMS Version
85 04 .... datum and Lenght
3F 27 B3 43 .... datum value in sec from 01-01- 1970

84 A3 01 Content type (1 part - image or text only) - I don't understand what does A3 mean
---
now starts Image Header:

1C 82 C6 1F ...this I don't understand
I thing that there must be lenght and maybe size of the picture

---

Petr,

A3 is the content type. MMS uses content type codes from the WAP WSP (Wireless Session Protocol) specification. Actually, the codes are published separately at http://www.wapforum.org/wina/wsp-content-type.htm. In this case, the content type is "application/vnd.wap.multipart.mixed", which has a code of 0x23 according to the WINA link, but WSP encoding OR's this value with 0x80.

The content type field is ALWAYS the last field of the MMS header of an "m-retrieve-conf" (0x8C 0x84 header) or "m-send-req" format MMS message.

Immediately after the content type field comes the content itself.

In this case, you're looking at data encoded in the "application/vnd.wap.multipart.mixed" format. This format is defined in section 8.5 of the WAP WSP specification. Basically it is a binary encoding for a multipart MIME message.

See this link here, as there's more discussion of this in another message: Explain MMS binary code

---

Thanx, I read that samples.

Now I have specific problem. Octec 1C means the lenght of header (counted from 9E to the beginning of the Data).

I found some examples of MMS's and I thing that the next 3 Octecs (82 C6 1F) means the lenght of te image file (jpeg). Because in another MMS, thare was only 2 octecs and these 2 Octecs were the same in two different MMS with the same image added (text message was different etc).

But it is strange, that there was value 85 09 and the image was only 649B long.
In another MMS, there was F0 09 and the image was 14KB long.

Im confused of that.

Lengths are encoded as variable length unsigned integers, as defined by section 8.1.2 of the WAP Wireless Session Protocol (WSP) specification.

Basically for each byte that has the high bit set, this indicates that the length continues for another byte.

Your example of 82 C6 1F would decode to (0x02 * 0x80 + 0x46) * 0x80 + 0x1F = 0xA31F = 41759.

85 09 would decode to: 0x05 * 0x80 + 0x09 = 0x289 = 649.

Can you explain further about this decoding 85 09 would decode to: 0x05 * 0x80 + 0x09 = 0x289 = 649. logic.

I read the WSP specification 8.1.2.. but could'nt understand! Perhaps a simpler explaination would be help ful.

Also how can i get hex of the filesize e.g. i have file of 4123 bytes, how to reverse into hex..

Sure.

If the high bit of one of the bytes in a length field is set (or with 0x80), then this indicates that the field continues for at least another byte. So you ignore this bit in computing the actual value.

Let's take 85 09 ...

In binary, this is:

10000101 00001001 (85 09)

To decode this, you remove the high bit from each byte, which gives you this:

0000101 0001001

Or, converting the bits, to bytes, you get:

000010 10001001

02 89

And in decimal, 649.

13.3. Code source

13.3.1. Programme de pilotage

```
import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.wireless.messaging.*;
import stockageDonnee.*;

/**
 * Programme de pilotage par sms de MMSCam, c'est ce programme qui crée les sms
 * de commande envoyé à MMSCam.
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
public class MmsCamCommande extends MIDlet implements CommandListener,
   AfficheurErreur {

   /**
    * La fenêtre qui contiendra les composents graphiques
    */
   private Form fenetre;

   /**
    * Le champ texte avec le numéro de l'appareil executant MMSCam
    */
   private TextField numero;

   /**
    * Le champ texte avec le mot de passe de l'appareil executant MMSCam
    */
   private TextField motDePasse;

   /**
    * Le champ texte avec l'adresse désirée de réception de l'image
    */
   private TextField destination;

   /**
    * Un bouton pour envoyer le sms de commande
    */
   private Command cmdEnvoi;

   /**
    * Base de donnée servant a donner une persistance aux information
    */
   private BaseDeDonnee baseDonnee;

   /**
    * Constructeur, il ouvre la base de donnée
    */
   public MmsCamCommande() {
      baseDonnee = new BaseDeDonnee("mmscamcommande", this);
   }

   /**
    * Méthode qui affiche l'erreur passée en paramètre
    * @param texte Le message d'erreur
```

```
     */
    public void afficherErreur(String texte) {
       Display.getDisplay(this).setCurrent(new Alert("Erreur",
             texte, null, AlertType.ERROR));
    }

    /**
     * Méthode qui met en place les composents graphiques
     */
    public void creationEcran() {
       // Création de la fenêtre
       fenetre = new Form("Envoi d'un ordre");
       // Création et placement des champs textes
       numero = new TextField("Numero de la camera:", baseDonnee.numeroCamera(),
                              20, TextField.PHONENUMBER);
       motDePasse = new TextField("Mot de passe:", baseDonnee.motDePasse(), 12,
                                  TextField.PASSWORD);
       destination = new TextField("Adresse de retour:",
                                   baseDonnee.adresseReponse(), 100,
                                   TextField.EMAILADDR);
       fenetre.append(numero);
       fenetre.append(motDePasse);
       fenetre.append(destination);

       // Création et placement des boutons
       cmdEnvoi = new Command("Envoyer", Command.OK, 1);
       fenetre.addCommand(cmdEnvoi);

       // Création de la ligen d'aide
       fenetre.setTicker(new Ticker(
          "Les champs \"mot de passe\" et \"Adresse de retour\" sont optionnels"));

       // Activation de l'écoute d'évenements
       fenetre.setCommandListener(this);

       // Placements de la fenetre à l'écran
       Display.getDisplay(this).setCurrent(fenetre);
    }

    /**
     * Méthode qui envoye l'ordre par sms
     */
    public void envoyerOrdre() {

       // La connection et le message
       MessageConnection mc = null;
       TextMessage tm = null;

       // Création de la ligne d'adresse
       String adresse = "sms://" + numero.getString() + ":" +
          this.getAppProperty("portSms");

       // Création du texte du sms
       String texte = "";
       if (motDePasse.getString() != "") {
          texte = texte + "m" + motDePasse.getString() + " ";
       }
       if (destination.getString() != "") {
          texte = texte + "a" + destination.getString() + " ";
       }

       // Envoi du sms
       try {
          mc       =       (MessageConnection)       Connector.open("sms://:16030");//
this.getAppProperty("portSms"));
          tm = (TextMessage) mc.newMessage(mc.TEXT_MESSAGE);
          tm.setAddress(adresse);
          tm.setPayloadText(texte);
          mc.send(tm);
```

```
            } catch (Exception e) {
            }

            // Fermeture de la connection
            try {
               if (mc != null) {
                  mc.close();
               }
            } catch (IOException ex) {
               afficherErreur("La connection SMS ne s'est pas fermée, vous devrez " +
                              "sans doute éteindre vous téléphone pour pouvoir " +
                              "utiliser à nouveau ce programme");
            }

            // Sauvegarde des infos et fermeture du programme du programme
            sauvgarderEntree();
            baseDonnee.fermer();
            notifyDestroyed();
         }

         /**
          * Méthode appelée au démarrage de l'application
          */
         public void startApp() {
            baseDonnee.ouvrir();
            creationEcran();
         }

         /**
          * Méthode appelée juste avant que l'application soit mise en pause
          */
         public void pauseApp() {
            sauvgarderEntree();
            baseDonnee.fermer();
         }

         /**
          * Méthode appellée lorsque qu'un événement est détecté
          * @param c L'objet command qui à déclanché l'événement
          * @param d L'objet affichable sur lequel est déclaré l'objet c
          */
         public void commandAction(Command c, Displayable d) {
            // Déclanchement de l'envoi
            if (c == cmdEnvoi) {
               envoyerOrdre();
            }
            sauvgarderEntree();
         }

         /**
          * Sauvgarde les valeur entrée par l'utilisateur
          */
         public void sauvgarderEntree(){
            baseDonnee.definirNumeroCamera(numero.getString()) ;
            baseDonnee.definirMotDePasse(motDePasse.getString()) ;
            baseDonnee.definirAdresseReponse(destination.getString());
         }

         /**
          * Métode appellée une seule fois au moment de la fermeture de la MIDLet
          * @param unconditional spécifie si la MIDLet doit effectuer un nettoyage
          */
         public void destroyApp(boolean unconditional) {
         }
      }
```

13.3.2. Programme principale MMSCam

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

```
import collectionEcran.*;
import controleur.*;

/**
 * MIDLet, classe de base, qui est le point d'entrée du programme
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
public class MmsCam extends MIDlet {

   /**
    * Le controlleur du programme
    */
   private Controleur control;


   /**
    * Constructeur de la MIDLet
    */
   public MmsCam() {
      control = new Controleur(this);
      EcranGenerique.definirControleur(control);
   }

   /**
    * Méthode appellée à chaque fois que l'application reprends la main
    */
   public void startApp() {
      control.redemmarage();
   }

   /**
    * Méthode appellée à chaque fois que l'application perd la main
    */
   public void pauseApp() {
      control.libererMemoire();
   }

   /**
    * Métode appellée une seule fois au moment de la fermeture de la MIDLet
    * @param unconditional spécifie si la MIDLet doit effectuer un nettoyage
    */
   public void destroyApp(boolean unconditional) {
      control.quitter();
   }
}
```

13.3.3. Classe controleur

```
package controleur;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import collectionEcran.*;
import gestionCamera.*;
import gestionEmail.*;
import gestionSms.*;
import stockageDonnee.*;

/**
 * Le controlleur du programme MMSCam
 * <p>Titre : Projet MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
```

```
 * @version 1.0
 */

public class Controleur implements LecteurSms, AfficheurErreur {

   /**
    * Lien sur la MIDLet en cours d'execution
    */
   private MIDlet programme;

   /**
    * L'affichage courant
    */
   private Display affichage;

   /**
    * L'objet charger de détécter l'arrivée de sms
    */
   private GestionnaireSms serveurSms;

   /**
    * La zone de stockage persistante des données
    */
   private BaseDeDonnee donnee;

   /**
    * Le mot de passe courant
    */
   private String motDePasse;

   /**
    * L'adresse de retour courante
    */
   private String adresseRetour;

   /**
    * La liste des numéros d'ou peuvent venir les ordres
    */
   private String[] listeNumeroAutorise;

   /**
    * Spécifie si l'application c'est déjà connecter une fois au GPRS
    */
   private boolean dejaConnecteUneFois = false;

   /**
    * Le constructeur de ce controlleur
    * @param prog la MIDLet qui declare le controlleur
    */
   public Controleur(MIDlet prog) {
      programme = prog;
      donnee = new BaseDeDonnee("mmscam", this);
   }

   /**
    * Méthode qui libère un maximum de mémoire, généralement appellée avant
    * la mise en pause de l'application
    */
   public void libererMemoire() {
      donnee.fermer();
   }

   /**
    * Méthode qui est appellée après une mise en pause de l'application
    */
   public void redemmarage() {
      //Chargement des données
      donnee.ouvrir();
      motDePasse = donnee.motDePasse();
      adresseRetour = donnee.adresseReponse();
      listeNumeroAutorise = donnee.listeNumero();
```

```
        //Placemment du menu principal
        affichage = Display.getDisplay(programme);
        afficherMenu();
    }

    /**
     * Démarre la procedure de fermeture de la MIDLet
     */
    public void quitter() {
        if (serveurSms != null) {
            serveurSms.terminerLEcoute();
            serveurSms = null;
        }
        programme.notifyDestroyed();
    }

    /**
     * Affiche le menu principal
     */
    public void afficherMenu() {
        affichage.setCurrent(new EcranPrincipal(this));
    }

    /**
     * Affiche l'ecran passé en paramètre
     * @param ecran l'écran à afficher
     */
    public void afficherEcran(EcranGenerique ecran) {
        affichage.setCurrent(ecran);
    }

    /**
     * Affiche une petite fenêtre avec un texte d'aide
     * @param texte le texte qui ser affiché
     */
    public void afficherAide(String texte) {
        Alert message = new Alert("Aide", texte, null, AlertType.INFO);
        message.setTimeout(Alert.FOREVER);
        affichage.setCurrent(message, affichage.getCurrent());
    }

    /**
     * Affiche une erreur
     * @param texte le descriptif de l'erreur
     */
    public void afficherErreur(String texte) {
        Alert message = new Alert("Erreur", texte, null, AlertType.ERROR);
        message.setTimeout(Alert.FOREVER);
        affichage.setCurrent(message, affichage.getCurrent());
    }

    /**
     * Fonction qui verifie si un numéro est autorisé à donnée un ordre par sms
     * @param numero le numéro à vérifier
     * @return vrai si le numéro à le droit de donner un ordre
     */
    private boolean numeroAutorise(String numero) {
        // Si la liste des numéros autorisé est vide, alors on a le droit
        if (listeNumeroAutorise == null) {
            return true;
        }
        // Sinon, on verifie que le numéro soit dans la liste
        for (int i = 0; i < listeNumeroAutorise.length; i++) {
            // Si il y est, alors c'est bon
            if (numero.equals(listeNumeroAutorise[i])) {
                return true;
            }
        }
        // Si on ne l'as pas trouvé on retourne faux
        return false;
```

```
}

/**
 * Analise un message recu et retourne un tableau de string avec dans la
 * première case le mot de passe transmis et dans la seconde l'adresse de
 * retour transmise
 * @param message le message à analiser
 * @return tableau de string avec les infos du message
 */
private String[] extractionInfo(String message) {
   String texte = message;
   String[] info = {
      "", ""};
   // Tant que le message n'est pas totalement parcouru
   while (texte.length() > 0) {
      // Si ça commence par un 'm', alors on a à faire à un mot de passe
      if (texte.charAt(0) == 'm') {
         texte = texte.substring(1);
         // Lecture du mot de passe
         while (texte.compareTo("") != 0 && texte.charAt(0) != ' ') {
            info[0] = info[0] + String.valueOf(texte.charAt(0));
            texte = texte.substring(1);
         }
         // Si ça commence par un 'a', alors on a à faire à une adresse d retour
      } else if (texte.charAt(0) == 'a') {
         texte = texte.substring(1);
         // Lecture de l'adresse de retour
         while (texte.compareTo("") != 0 && texte.charAt(0) != ' ') {
            info[1] = info[1] + String.valueOf(texte.charAt(0));
            texte = texte.substring(1);
         }
      } else {
         texte = texte.substring(1);
      }
   }
   // transfère du tableau
   return info;
}

/**
 * Méthode qui traite la reception d'un sms
 * @param numero le numéro d'origine du sms
 * @param texte le texte qui était contenu dans le sms
 */
public void smsRecu(String numero, String texte) {

   // Si le numéro à le droit de donner un ordre
   if (numeroAutorise(numero)) {

      //Extraction des informations du message
      String[] info = extractionInfo(texte);

      // Si le mot de passe correspond
      if (motDePasse.equals("") || info[0].equals(motDePasse)) {

         // Arret de l'écoute sms pendant l'envoi
         serveurSms.terminerLEcoute();

         // Envoie d'une photo soit à l'adresse standard, soit à celle envoyée
         if (info[1] != "") {
            envoyerPhoto(info[1]);
         } else {
            envoyerPhoto(adresseRetour);
         }
      } else {
         ( (EcranSurveillance) affichage.getCurrent()).ajouterEvenement(
            "Reçu message de " + numero + "\nMot de passe incorrect");
      }
   } else {
```

```
            ( (EcranSurveillance) affichage.getCurrent()).ajouterEvenement(
               "Reçu message de " + numero + "\nNuméro non accepté");
         }
      }

      /**
       * Place le téléphone en mode surveillence, c'est à dire sensible aux sms
       */
      public void demarrerSurveillanceSms() {

         // Préparation d'un écran patientez"
         Form ecranPatience = new Form("MMSCam");
         ecranPatience.append("Initialisation, patientez SVP");
         affichage.setCurrent(ecranPatience);

         // Effecture une préconnexion si il n'y a jamais eu de connexion
         //  auparavant, cela pour eviter le message bloquant "Select access point"
         if (!dejaConnecteUneFois) {
            CommunicationServelet.preConnexion(programme.getAppProperty(
               "adresseServlet"));
            dejaConnecteUneFois = true;
         }

         // Démmarage de l'ecoute de sms
         serveurSms = new GestionnaireSms();

         // Si la détection est possible
         if ( serveurSms.demarrerUneEcoute(this, programme.getAppProperty("portSms"))) {
             // Création et affichage d'un écran de surveillance
             afficherEcran(new EcranSurveillance());
         } else {
            // Affichage de l'erreur et retour au menu principal
            Alert message = new Alert("Erreur", "Impossible de détécter l'arrivée"+
                                      " de sms. Redémarrer votre téléphone SVP",
                                      null, AlertType.ERROR);
            message.setTimeout(Alert.FOREVER);
            affichage.setCurrent(message, new EcranPrincipal(this));
         }
      }

      /**
       * Retourne le meilleur format possible pour la capture
       * @return le format soit "jpeg" soit "png"
       */
      public String formatCapture() {
         // Lecture des encodages possibles
         String codages = System.getProperty("video.snapshot.encodings");
         // Recherche de jpeg
         while (codages.length() > 3) {
            if (codages.startsWith("jpeg")) {
               return "jpeg";
            }
            codages = codages.substring(1);
         }
         return "png";
      }

      /**
       * Quitte le mode surveillence et revient au menu principal
       */
      public void arreterSurveillanceSms() {
         serveurSms.terminerLEcoute();
         serveurSms = null;
         afficherMenu();
      }

      /**
       * Envoie une photo à l'adresse spécifiée
       * @param adresse L'adresse ou la photo va être envoyée
```

```
     */
    public void envoyerPhoto(String adresse) {
       // Création d'une caméra et prise de la photo
       Camera cam = new Camera(this);
       byte[] temp = cam.prendrePhoto(formatCapture());
       cam = null;
       String resultat = "";
       EcranSurveillance actuel = (EcranSurveillance) affichage.getCurrent();

       // Si la photo à été prise
       if (temp != null) {

          // Affichage de l'image sur l'écran
          Gauge barre = new Gauge("Progression de l'envoi", false, 1, 0);
          Image image = Image.createImage(temp, 0, temp.length);
          affichage.setCurrent(new EcranEnvoiImage(image, barre));

          // Envoi de l'image à l'adresse spécifiée
          resultat = CommunicationServelet.transmettre(programme.getAppProperty(
             "adresseServlet"), adresse, "image." + formatCapture(), temp, barre);

          temp = null;
       }
       // Affichage du résultat
       actuel.ajouterEvenement("Envoi à " + adresse + " : " + resultat);
       // Retour au mode surveillance
       afficherEcran(actuel);
       if ( ! serveurSms.demarrerUneEcoute(this, programme.getAppProperty("portSms"))
){
          actuel.ajouterEvenement("La détection de SMS ne marche plus: " +
                                  "Redémmarage nécessaire");
       }
    }

    /**
     * Retourne le mot de passe défini par l'utilisateur
     * @return Le mot de passe
     */
    public String motDePasse() {
       return motDePasse;
    }

    /**
     * Modifie le mot de passe
     * @param motDePasse le nouveau mot de passe
     */
    public void definirMotDePasse(String motDePasse) {
       this.motDePasse = motDePasse;
       donnee.definirMotDePasse(motDePasse);
    }

    /**
     * Retourne l'adresse de retour définie par l'utilisateur
     * @return L'adresse de retour
     */
    public String adresseRetour() {
       return adresseRetour;
    }

    /**
     * Modifie l'adresse de retour ou l'image sera envoyée
     * @param adr La nouvelle adresse
     */
    public void definirAdresseRetour(String adr) {
       adresseRetour = adr;
       donnee.definirAdresseReponse(adr);
    }

    /**
```

```
    * Retourne la liste des numéros autorisé à donnée des ordres par sms
    * @return La liste de numéros
    */
   public String[] listeNumeroAutorise() {
      return listeNumeroAutorise;
   }

   /**
    * Ajoute un numéros à la liste des numéros autorisé
    * @param numero Le numéro à ajouter
    */
   public void ajouterNumeroAutorise(String numero) {
      // Liste temporaire
      String[] temp;
      // Si la liste est vide, alors on crée une liste de taille 1
      if (listeNumeroAutorise == null) {
         temp = new String[1];
         // Sinon, on allonge la liste existente
      } else {
         temp = new String[listeNumeroAutorise.length + 1];
      }
      // On place le nouveaux numéros en début de liste temporaire
      temp[0] = numero;
      // On décale tout les autres dans la liste temporaire
      for (int i = 1; i < temp.length; i++) {
         temp[i] = listeNumeroAutorise[i - 1];
      }
      // Remplacement de la liste courante par la nouvelle
      listeNumeroAutorise = temp;
      donnee.definirListeNumero(listeNumeroAutorise);
   }

   /**
    * Retire un numéro de la liste des numéros autorisé à donner des ordres
    * @param index L'emplacement du numéro à supprimer
    */
   public void retirerNumeroAutorise(int index) {
      String[] temp;
      // Si la liste à une taille supérieur à 1
      if (listeNumeroAutorise.length != 1) {
         temp = new String[listeNumeroAutorise.length - 1];
         // On garde tous les numéros situé avant l'indice à supprimer
         for (int i = 0; i < index; i++) {
            temp[i] = listeNumeroAutorise[i];
         }
         // On décale tous les numéros situé après
         for (int i = index; i < temp.length; i++) {
            temp[i] = listeNumeroAutorise[i + 1];
         }
         // Si la liste a une taille de 1, on la supprime
      } else {
         temp = null;
      }
      // Remplacement de la liste courante par la nouvelle
      listeNumeroAutorise = temp;
      donnee.definirListeNumero(listeNumeroAutorise);
   }
}
```

13.3.4. Paquetage de la collection d'écran

Pour ce paquetage, je présente ici uniquement le code de 3 écrans, les autres étant tous basés sur le même principe de construction, il serait inutile de les présenter ici.

```
package collectionEcran;

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

```
import controleur.*;

/**
 * <p>Titre : Projet MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */

public class EcranPrincipal extends List implements CommandListener {

   private Controleur control;
   private Command cmdChoisir;

   public EcranPrincipal(Controleur control) {

      super("MmsCam", ChoiceGroup.IMPLICIT);

      this.control = control;

      append("Activer MmsCam", null);
      append("Mot de passe", null);
      append("N° Autorisé", null);
      append("Adresse de retour", null);
      append("Quitter",null);

      cmdChoisir = new Command("Choisir", Command.BACK, 1);
      addCommand(cmdChoisir);

      setCommandListener(this);
   }

   /**
   * Méthode appellée lorsque qu'un événement est détecté
   * @param c L'objet command qui à déclanché l'événement
   * @param d L'objet affichable sur lequel est déclaré l'objet c
   */
   public void commandAction(Command c, Displayable d) {
      if (c == cmdChoisir) {
         switch (getSelectedIndex()) {
            case 0:
               control.demarrerSurveillanceSms();
               break;
            case 1:
               control.afficherEcran(new EcranMotDePasse());
               break;
            case 2:
               control.afficherEcran(new EcranNumeroAutorise());
               break;
            case 3:
               control.afficherEcran(new EcranAdresseRetour());
               break;
            case 4:
               control.quitter();
               break;
         }
      }
   }
}

package collectionEcran;

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

public class EcranSurveillance extends EcranGenerique {
```

```
   Command cmdArreter;

   public EcranSurveillance() {
      removeCommand(cmdRetour);

      removeCommand(cmdAide);
      cmdArreter = new Command("Arreter", Command.BACK, 1);
      addCommand(cmdArreter);
      addCommand(cmdArreter);
      setCommandListener(this);
      setTicker(new Ticker("En attente d'un sms d'ordre"));
      ajouterEvenement("Début de la surveillance");
   }

   public void ajouterEvenement(String texte){
      Calendar calendrier = Calendar.getInstance();
      // Affichage de la date et de l'heure
      append(String.valueOf(calendrier.get(Calendar.DAY_OF_MONTH)) +
             "/" +  String.valueOf(calendrier.get(Calendar.MONDAY)) +
             "/" +  String.valueOf(calendrier.get(Calendar.YEAR)) +
             " " +  String.valueOf(calendrier.get(Calendar.HOUR_OF_DAY)) +
             ":" +  (calendrier.get(Calendar.MINUTE )<10?"0":"") +
                    String.valueOf(calendrier.get(Calendar.MINUTE)));
      // Affichage de l'événement
      append("\n" + texte + "\n\n");
   }

   /**
    * Méthode appellée lorsque qu'un événement est détecté
    * @param c L'objet command qui à déclanché l'événement
    * @param d L'objet affichable sur lequel est déclaré l'objet c
    */
   public void commandAction(Command c, Displayable d) {
      super.commandAction(c, d);
      if (c == cmdArreter) {
         control.arreterSurveillanceSms();
      }
   }

   public String texteAide() {
      return "Mode surveillance, votre téléphone est actuellement sensible " +
         "aux sms de commande. Cet écran affiche toutes les opérations qui" +
         " ont déjà été effectuées";
   }
}


package collectionEcran;

import javax.microedition.lcdui.*;
import controleur.*;

/**
 * Ecran générique qui sera à la base des écrans de commande de MMSCam
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
abstract public class EcranGenerique extends Form implements CommandListener{

   /**
    *  Référence sur le controleur du programme
    */
   protected static Controleur control = null;
```

```
    /**
     * Trois boutons de commande normalement disponible sur tout les ecrans
     */
    protected Command cmdRetour;
    protected Command cmdAide;

    /**
     *  Sert à définir le controleur avec lequel communiqueront les ecrans
     *  !!! Doit être defini avant la création d'un écran
     * @param control Le controleur de l'application
     */
    public static void definirControleur(Controleur control){
        EcranGenerique.control = control;
    }

    /**
     * Constructeur d'un écran générique
     */
    public EcranGenerique() {
        // Affichage du titre
        super("MmsCam");
        // Création et placement des boutons
        cmdRetour = new Command("Retour",Command.BACK,1);
        cmdAide = new Command("Aide",Command.HELP ,1);
        addCommand(cmdRetour);
        addCommand(cmdAide);
    }

    /**
     * Methode qui devra retourner un texte d'aide pour l'utilisation de l'écran
     * @return Le texte d'aide
     */
    abstract protected String texteAide();

    /**
    * Méthode appellée lorsque qu'un événement est détecté
    * @param c L'objet command qui à déclanché l'événement
    * @param d L'objet affichable sur lequel est déclaré l'objet c
    */
    public void commandAction(Command c, Displayable d){
        // Si bouton retour alors on retourne au menu principal
        if (c == cmdRetour){
            control.afficherMenu();
            System.gc();
        }
        // Si bouton d'aide on affiche l'aide
        if (c == cmdAide){
            control.afficherAide(texteAide());
        }
    }
}
```

13.3.5. Paquetage de transmission de l'ordre

```
package gestionSms;

/**
 * Interface qui permet de transmettre un sms recu à celui qui l'implémente
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */

public abstract interface LecteurSms {

    /**
     * Signale la réception d'un sms
     * @param numero Le numéro de celui qui a envoyé le sms
```

```
     * @param texte Le texte du sms
     */
    void smsRecu(String numero, String texte);

    /**
     * Affiche une erreur à l'écran
     * @param texte Le message d'erreur à afficher
     */
    void afficherErreur(String texte);
}
```

```
package gestionSms;

import java.io.*;
import javax.microedition.io.*;
import javax.wireless.messaging.*;

/**
 * Classe chargée de la réception d'sms de commande
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
public class GestionnaireSms implements MessageListener {

    /**
     * La connection pas laquelle le message va arriver
     */
    private MessageConnection connexionDEcoute;

    /**
     * Le demandeur de sms, a qui on transmettra le sms recu
     */
    private LecteurSms lecteur;

    /**
     * Méthode qui démarre l'écoute de sms
     * @param destinataire L'objet à qui il faudra transmettre le sms
     * @param port Le port sur lequel il faut écouter l'arrivée de sms
     */
    public boolean demarrerUneEcoute(LecteurSms destinataire, String port) {
        this.lecteur = destinataire;
        // Ouverture de la connection
        try {
            //lecteur.afficherErreur("Debut d'écoute sur : sms://:" + port);
            connexionDEcoute = (MessageConnection) Connector.open("sms://:" + port);
            connexionDEcoute.setMessageListener(this);
            return true;
        } catch (IOException e) {
            return false;
        }
    }

    /**
     * Méthode qui ferme la connexion
     */
    public void terminerLEcoute() {
        if (connexionDEcoute != null) {
            try {
                connexionDEcoute.close();
            } catch (IOException ex) {
            }
        }
    }
```

```
    int bug = 1110;
    /**
     * Méthode appelée lors de la réception d'un sms
     * @param mc La connection sur laquelle est arrivé le message
     */
    public void notifyIncomingMessage(MessageConnection mc) {
        TextMessage smsRecu = null;
        String numero = "";
        // Lecture du message et transmission de celui-ci à son destinataire
        try {
            smsRecu = (TextMessage) mc.receive();
            numero = smsRecu.getAddress();
            numero = numero.substring(0,numero.lastIndexOf(':'));
            lecteur.smsRecu(numero+String.valueOf(bug), smsRecu.getPayloadText());
        } catch (IOException e) {
            lecteur.afficherErreur("Impossible de lire le message reçu");
        }
            bug++;
    }
}
```

13.3.6. Paquetage de capture de l'image

```
package gestionCamera;

import javax.microedition.media.*;
import javax.microedition.media.control.*;
import controleur.*;

/**
 * Classe chargée de la capture d'une image
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
public class Camera {

    /**
     * Le controlleur du programme
     */
    Controleur control = null;

    /**
     * Constructeur de la camera
     * @param control Le controlleur du programme
     */
    public Camera(Controleur control) {
        this.control = control;
    }

    /**
     * Capture une image et la retourne sous la forme d'un tableau de byte
     * @return L'image capturée
     */
    public byte[] prendrePhoto(String format) {
        // La camera fournie par le telephone
        Player enregistreur = null;
        // Le controlleur de la camera
        VideoControl controleurVideo = null;
        // Capture d'une image au format spécifié
        byte[] imagePng = null;

        try {

            // Le manager crée l'instance de la camera
            enregistreur = Manager.createPlayer("capture://video");
```

```
            // Préparation et initialisation de la camera
            enregistreur.realize();
            enregistreur.prefetch();

            // Création du controleur vidéo
            controleurVideo = (VideoControl) enregistreur.getControl(
               "VideoControl");
            controleurVideo.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);

            // Démarrage de l'activité de l'enregistreur
            enregistreur.start();

            // Mise en veille d'une seconde pour que la caméra fasse le point
            Thread.sleep(1000);

            // Capture de l'image
            imagePng = controleurVideo.getSnapshot("encoding=" + format +
                                                   "&width=200&height=150");
            // Fermeture des toutes les instances
            controleurVideo.setVisible(false);
            controleurVideo = null;
            enregistreur.stop();
            enregistreur.deallocate();
            enregistreur.close();
         } catch (Exception e) {
            // Affichage de l'erreur
            control.afficherErreur("Impossible de capturer une image, la caméra" +
                                   " est peut-être déjà utilisée");
         }
         // Retourne l'image
         return imagePng;
      }
   }
```

13.3.7. Paquetage de stockage des données

```
package stockageDonnee;

import javax.microedition.rms.*;

/**
 * Classe qui s'occupe de stocker les donnée du programme de manière à éviter
 * une réintroduction complète entre chaque utilisation
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
public class BaseDeDonnee {

   /**
    * La case de donnee
    */
   private RecordStore base = null;


   /**
    * Le nom de la base de donnee
    */
   private String nomBaseDonnee;

   /**
    * Une contante représentant un champ texte vide
    */
   private String champVide = "@vide@";

   /**
    * Lien sur le controlleur du programme
```

```
 */
private AfficheurErreur afficheur;

/**
 * L'emplacement des différents champs dans la bse de donnée
 */
private int positionMotDePasse = 1;
private int positionadresseReponse = 2;
private int positionListeNumero = 3;
private int positionNumCamera = 4;

/**
 * Constructeur
 * @param con Le lien sur le controlleur
 */
public BaseDeDonnee(String nom, AfficheurErreur con) {
   nomBaseDonnee = nom;
   afficheur = con;
}

/**
 * Modification d'une valeur de la base de donnée
 * @param position La position de la valeur à modifier
 * @param valeur La nouvelle valeur
 */
private void modifier(int position, String valeur) {
   byte[] temp;
   // Si la nouvelle valeur est une chaine vide, on la remplace par la
   // constante chaine vide
   if (valeur.compareTo("") == 0) {
      temp = champVide.getBytes();
   } else {
      temp = valeur.getBytes();
   }
   // Placement de la valeur à la position demandée
   try {
      base.setRecord(position, temp, 0, temp.length);
   } catch (Exception ex) {
      afficheur.afficherErreur("Impossible de sauvgarder, les données seront"
                               + " perdues lors de la fermeture");
   }
}

/**
 * Lis les données située à une position donnée
 * @param position La position desirée
 * @return Les données qui on été lues
 */
private String lire(int position) {
   String temp = null;
   // Lecture de la valeur
   try {
      temp = new String(base.getRecord(position));
   } catch (Exception ex) {
      afficheur.afficherErreur("Impossible de charger les données");
   }
   // Si c'est la constante chaine vide, alors on la remplace
   if (temp.compareTo(champVide) == 0) {
      temp = "";
   }
   return temp;
}

/**
 * Retourne le mot de passe sauvgardé
 * @return le mot de passe
 */
public String motDePasse() {
   return lire(positionMotDePasse);
}
```

```
/**
 * Modifier le mot de passe sauvgarder
 * @param motDePasse Le nouveau mot de passe
 */
public void definirMotDePasse(String motDePasse) {
   modifier(positionMotDePasse, motDePasse);
}

/**
 * Retourne l'adresse de réponse sauvgardée
 * @return L'adresse de réponse
 */
public String adresseReponse() {
   return lire(positionadresseReponse);
}

/**
 * Modifie l'adresse de réponse
 * @param adresseReponse La nouvelle adresse de réponse
 */
public void definirAdresseReponse(String adresseReponse) {
   modifier(positionadresseReponse, adresseReponse);
}

/**
 * Retourne le numéro de l'appareil avec la camera
 * @return Le numéro de la camera
 */
public String numeroCamera() {
   return lire(positionNumCamera);
}

/**
 * Modifie le numéro de l'appareil avec la camera
 * @param Le nouveau numero
 */
public void definirNumeroCamera(String numero) {
   modifier(positionNumCamera, numero);
}

/**
 * Retourne un tableau de numéro de téléphone autorisé à donnée des ordres
 * @return La liste de numéro de téléphones
 */
public String[] listeNumero() {
   String chaine = lire(positionListeNumero);
   String[] temp = null;

   // Si la chaine contient des numéros
   if (chaine.length() > 0) {
      int nombre = 1;
      int i = 0;
      // On compte le nombre de numéos
      while (i < chaine.length()) {
         if (chaine.charAt(i) == ';') {
            nombre++;
         }
         i++;
      }
      // Création du tableau
      temp = new String[nombre];
      i = 0;
      int position = 0;
      // Placement des numéros dans la liste
      while (i < chaine.length()) {
         if (chaine.charAt(i) == ';') {
            temp[position] = chaine.substring(0, i - 1);
            chaine = chaine.substring(i + 1);
            position++;
```

```
            i = -1;
         }
         i++;
      }
      temp[position] = chaine;
   }
   // Transmission de la liste
   return temp;
}

/**
 * Defini une nouvelle liste des numéros autorisé à donner des ordres
 * @param liste La nouvelle liste
 */
public void definirListeNumero(String[] liste) {
   String chaine = "";
   // Création des chaine en concatenant les numéros de la liste
   if (liste != null) {
      for (int i = 0; i < liste.length; i++) {
         if (i == 0) {
            chaine = liste[i];
         } else {
            chaine = chaine + ";" + liste[i];
         }
      }
   }
   // Inscription de la nouvelle chaine
   modifier(positionListeNumero, chaine);
}

/**
 * Ouverture de la base de donnée
 */
public void ouvrir() {
   // Ouverture
   try {
      base = RecordStore.openRecordStore(nomBaseDonnee, true);
   } catch (Exception ex) {
      afficheur.afficherErreur("2" + ex.toString());
   }

   try {
      // Si c'est la première utilisation de la base de donnée alors on
      // initialise les 3 champs avec une chaine vide
      if (base.getNextRecordID() == 1) {
         byte[] a = champVide.getBytes();
         base.addRecord(a, 0, a.length); //Mot de passe
         base.addRecord(a, 0, a.length); //adresseReponse
         base.addRecord(a, 0, a.length); //ListeNumero
         base.addRecord(a, 0, a.length); //NumCamera

      }
   } catch (Exception ex1) {
      afficheur.afficherErreur("Problème lors de l'ouverture de la" +
                               " base de donnée");
   }
}

/**
 * Ferme la base de donnée
 */
public void fermer() {
   try {
      base.closeRecordStore();

   } catch (Exception ex1) {
      afficheur.afficherErreur("Problème lors de la fermeture de la base de" +
                               " donnée");
```

```
        }
    }
}
```

13.3.8. Paquetage de transmission de l'image

```
package gestionEmail;

/**
 * Classe d'encodage d'un tableau de byte en base64
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod, basé sur le code de Dr. Mark Thornton
 * @version 1.0
 */
final public class Base64 {

   // Table des caractères utilisés dans le code base64
   static private char[] carBase64 =
      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
      .toCharArray();

   /**
    * Encode la chaine de bytes passée en paramètre en code Base64
    * @param entree La chaine de bytes à encoder
    * @return La chaine de caractère encodée
    */
   static public String encoder(byte[] entree) {
      boolean quadruplet = false;
      boolean triplet = false;
      // Valeur formée par la lecture de 3 caractères de 8 bits
      int valeur;

      // Chaine de caractère resultat (taille = 4/3 de l'entrée
      char[] sortie = new char[ ( (entree.length + 2) / 3) * 4];

      //Parcourt de toutes la chaine d'entrée
      for (int i=0, index=0; i<entree.length; i+=3, index+=4) {

         // Lecture du première caractère
         valeur = (0xFF & (int) entree[i]);
         valeur <<= 8;

         // Lecture du second caractère si il y en a un
         if ( (i + 1) < entree.length) {
            valeur |= (0xFF & (int) entree[i + 1]);
            triplet = true;
         }
         valeur <<= 8;

         // Lecture du troisième caractère si il y en a un
         if ( (i + 2) < entree.length) {
            valeur |= (0xFF & (int) entree[i + 2]);
            quadruplet = true;
         }

         // Les 3 caractères de 8 bits lu soit 24 bits, sont converti en 4
         // caractères de 6 bits pris dans la table des caractères Base64
         // Dans le case ou il n'y en avait que 1 ou 2, on bourre avec des '='
         sortie[index + 3] = carBase64[ (quadruplet ? (valeur & 0x3F) : 64)];
         valeur >>= 6;
         sortie[index + 2] = carBase64[ (triplet ? (valeur & 0x3F) : 64)];
         valeur >>= 6;
         sortie[index + 1] = carBase64[valeur & 0x3F];
         valeur >>= 6;
         sortie[index + 0] = carBase64[valeur & 0x3F];
      }
```

```
      return String.valueOf(sortie);
   }
}
```

```
package gestionEmail;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;

/**
 * Classe qui permet de transmettre les données à la servelet chargée
 *  d'envoyer un mail
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de diplôme D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
final public class CommunicationServelet {

   /**
    * Fonction qui réalise une préconnexion, c'est à dire qui active
    *  l'utilisation du réseau, mais ne transmet rien.
    * @param adresse Une adresse ou effectuer une préconnexion
    */
   static public void preConnexion(String adresse) {
      try {
         HttpConnection c = (HttpConnection) Connector.open(adresse);
         c.setRequestMethod(HttpConnection.GET);
         OutputStream os = c.openOutputStream();
         os.close();
         c.close();
      } catch (IOException ex) {
      }
   }

   /**
    * Transmet les paramètres de l'e-mail à la servelet
    * @param adresseServlet L'adresse de la servlet
    * @param adrDestination L'adresse de destination
    * @param nomFichier Le nom du fichier image
    * @param codePhoto Le code de la photo
    * @param barre La barre de progression de l'envoi
    * @return Le code de retour de la servlet
    */
   static public String transmettre(String adresseServlet,
                                    String adrDestination, String nomFichier,
                                    byte[] codePhoto, Gauge barre) {

      // Encodage de la photo en Base64
      String photo = Base64.encoder(codePhoto);
      // 3 objets de connexion, La connexion http, le flux d'entrée et de sortie
      HttpConnection c = null;
      InputStream is = null;
      OutputStream os = null;
      // Le resultat de la requete
      String resultat = "";


      // Envoi des données aux serveur
      try {
         // Initialisation des paramètres de communication
         c = (HttpConnection) Connector.open(adresseServlet);
         c.setRequestMethod(HttpConnection.POST);
```

```
            // Paramètre faux, mais essentiel pour un bon fonctionnement
            c.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");

            // Préparation de la barre d'avancement
            barre.setMaxValue(photo.length() / 72);

            // Ouverture du flux de sortie et écriture des données
            os = c.openOutputStream();
            os.write( ("pour=" + adrDestination).getBytes());
            os.write( ("&nomFichier=" + nomFichier).getBytes());
            os.write( "&photo=".getBytes());

            // Envoie la photo en ligne de 72 caractère comme spécifier dans la RFC
            while (photo.length() > 72) {
               os.write( (photo.substring(0, 72)+"\r\n").getBytes() );
               photo = photo.substring(72);
               barre.setValue(barre.getValue()+1);
            }
            os.write( photo.getBytes() );

            // Modification de la barre
            barre.setLabel("Création du mail par la Servlet");
            barre.setValue(barre.getValue() / 2);
         }
         catch (Exception e) {
            resultat = "Impossible d'envoyer les données";
         }

         // Ouvertue du flux d'entrée et lecture du resultat
         try {
            StringBuffer lecture = new StringBuffer();
            is = c.openDataInputStream();
            int caractere;
            while ( (caractere = is.read()) != -1) {
               lecture.append( (char) caractere);
            }
            resultat = lecture.toString();
         }
         catch (Exception e) {
            resultat = "La serveur Web ne donne pas de réponse";
         }

         // Fermeture des connexions
         try {
            is.close();
            os.close();
            c.close();
         } catch (Exception e) {
         }

         // On retourne le resulat de l'envoi qui devrai être OK
         return resultat.toString();
      }
```

13.3.9. Code de la servlet

```
// Packetage nominale pour être déposé sur le site mycgiserver.com
package jeanmonod;

import java.io.*;
import java.text.*;
import java.net.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;

/**
```

```
 * La servlet qui récéptionne les donnée transmise par le téléphone
 * <p>Titre : Projet de diplôme MmsCam</p>
 * <p>Description : Pilotage à distance par SMS d'un téléphone MMS avec caméra</p>
 * <p>Copyright : Travail de dîplome D. Jeanmonod Hivers 2003</p>
 * <p>Société : EIVD Yverdon</p>
 * @author D. Jeanmonod
 * @version 1.0
 */
public class EmailServlet extends HttpServlet {

   /**
    * Méthode qui est appellée lors de la pré-connexion, pour s'assurer que la
    *  connexion soit possible
    * @param requete
    * @param reponse
    */
   public void doGet(HttpServletRequest requete,
                     HttpServletResponse reponse) {
      // Préaparation de l'envoi de la réponse
      try {
         reponse.setContentType("text/plain");
         PrintWriter out = reponse.getWriter();
         out.print("OK");
      } catch (Exception e) {
      }
   }

   /**
    * Méthode qui est appellée lorsque le téléphone post ces donnée
    * @param request La réponse qui va être transmise au téléphone
    * @param response La requête envoyée par le téléphone
    * @throws IOException
    * @throws ServletException
    */
    public void doPost(HttpServletRequest requete,
                       HttpServletResponse reponse) throws IOException,
      ServletException {

      // Lecture des paramètres transmis par le téléphone
      String pour = requete.getParameter("pour");
      String photo = requete.getParameter("photo");
      String nomFichier = requete.getParameter("nomFichier");

      // Correction car lors de l'envoi les espace sont transformé en +
      photo = photo.replace(' ', '+');

      // Envoi de l'e-mail avec la photo
      String texteRetour = envoyerImage(pour, nomFichier, photo);

      // Envoi de la confirmation
      // Préaparation de l'envoi de la réponse
      reponse.setContentType("text/plain");
      PrintWriter out = reponse.getWriter();
      out.print(texteRetour);
      out.close();
   }


   /**
    * La Méthode chagrée d'envoyé un e-mail avec une photo
    * @param from Permet de spécifié qui a envoyé l'e-mail
    * @param to L'adresse de destination de l'e-mail
    * @param subject Le sujet de l'e-mail
    * @param photo Le code de la photo
    */
   private String envoyerImage(String pour, String nomFichier, String photo) {
      Socket smtpSocket = null;
```

```
DataOutputStream os = null;
DataInputStream is = null;

// Ouverture d'une connexion sur un serveur SMTP
try {
   smtpSocket = new Socket("mycgiserver.com", 25);
   os = new DataOutputStream(smtpSocket.getOutputStream());
   is = new DataInputStream(smtpSocket.getInputStream());
}catch (Exception e) {
   return "Impossible de contacter le serveur SMTP";
}

try {
   os.writeBytes("EHLO there" + "\n");
   os.writeBytes("MAIL FROM: " + pour + "\n");
   os.writeBytes("RCPT TO: " + pour + "\n");
   os.writeBytes("DATA\n");
   os.writeBytes("Date: " + new Date() + "\n");
   os.writeBytes("From: " + pour + "\n");
   os.writeBytes("To: " + pour + "\n");
   os.writeBytes("Subject: Photo MMSCam \n");
   os.writeBytes("Mime-Version: 1.0\n");
   os.writeBytes("Content-Type: multipart/mixed;");
   os.writeBytes("Boundary=-=_NextPart\n");

   // Première partie, le texte du mail
   os.writeBytes("---=_NextPart\n");
   os.writeBytes("Content-Type: text/plain;\n");
   os.writeBytes("charset=\"us-ascii\"\n");
   os.writeBytes("Content-Transfer-Encoding: 7bit\n\n");
   os.writeBytes("Image capturée par un appareil MMSCam le " +
                 new Date() + "\n\n");

   // Seconde partie l'image
   os.writeBytes("---=_NextPart\n");
   os.writeBytes("Content-Type: application/octet-stream\n");
   os.writeBytes("Content-Disposition: attachment;\n");
   os.writeBytes("        filename=\"" + nomFichier + "\"\n");
   os.writeBytes("Content-Transfer-Encoding: base64\n\n");
   os.writeBytes(photo);

   os.writeBytes("\n\n");
   os.writeBytes("---=_NextPart\n\n");
   os.writeBytes(".\n\n");
   os.writeBytes("QUIT\n");
}
catch (Exception e) {
   return "Communication interompue avec le SMTP, Verifiez " +
          "l'adresse E-mail";
}

String texte = "L'e-mail n'as pas pu être delivré au destinataire, " +
               "verifiez l'adresse svp";
try {
   String ligne;
   BufferedReader buffer = new BufferedReader(new InputStreamReader(is));
   while ( (ligne = buffer.readLine()) != null) {
      if (ligne.matches(".*delivery")) {
         texte = "OK";
         break;
      }
   }
}catch (Exception e){
   return "Connexion perdue avec le serveur SMTP, l'E-mail est peut-être perdu";
}

// Fermeture des connexipn aux serveur smtp
```

```
            try {
               os.close();
               is.close();
               smtpSocket.close();
            } catch (Exception ex){
            }
            return texte;
         }
      }
```