Table des matières

1 Avant-propos et remerciements

Le titre de ce travail de diplôme, **Voix sur GPRS** peut faire croire que le contenu de ce document traite d'une solution permettant d'établir une communication téléphonique à l'aide du réseau GPRS. Ceci n'est évidemment pas le cas, les communications téléphoniques sont laissées aux bons soins du réseau GSM qui les réalises très bien.

Voix sur GPRS définit en fait, un service qui permet d'envoyer un message vocal à l'aide du réseau GPRS, une telle application répond beaucoup mieux aux caractéristiques de ce réseau. Ces quelques pages décrivent la solution que nous avons mise en œuvre pour la réaliser.

Ce travail de diplôme fait suite à une première étude que nous avions réalisée lors du projet de semestre. Le document relatant celui-ci est fourni sur le CD, dans le répertoire ProjetDeSemestre.

Par la suite et dans un but de simplification, nous allons utiliser l'acronyme VoGprs pour indiquer Voix sur GPRS.

Maintenant, que nous avons éclairci la signification du titre définissant notre travail de diplôme, nous pouvons remercier les personnes qui nous ont aidées à le réaliser.

Un merci particulier à Mr. Stephan Robert pour nous avoir offert la possibilité de réaliser un travail de diplôme en collaboration avec Swisscom, pour sa disponibilité et son enthousiasme contagieux. Merci à Mr. Gianpaolo Cecchin, pour son humilité à nous expliquer les secrets de GPRS et à nous orienter tout au long de ce travail, pour le nombre de documents qu'il nous a fourni et pour nous avoir mis à disposition divers équipements.

2 Introduction

La téléphonie mobile à connu une formidable progression ces dernières années et son développement ne semble pas s'essouffler, puisque les nouvelles normes la définissant se suivent à rythme soutenu.

Le but avoué du GSM était d'offrir un service de transport de voix à des utilisateurs mobiles. L'évolution du réseau GSM s'est traduit par la mise en place du réseau GPRS qui lui se destine à offrir des services de transfert de données.

La prochaine évolution annoncé et qui a déjà fait parler d'elle est le réseau UMTS. Ce réseau contrairement à GPRS qui est une évolution de GSM, diffère dans sa conception par rapport aux réseaux GSM/GPRS.

Le but commun recherché par ces nouveaux réseaux est d'augmenter le débit offert pour la transmission des données. La téléphonie mobile entre dans une nouvelle ère où le transport de la voix n'est plus le service principal, de plus en plus les constructeurs des réseaux ainsi que les opérateurs téléphoniques mettent en fonction des services qui permettent d'exploiter au mieux les nouveaux débits desquels ils disposent. Le succès des ces nouveaux réseaux est intimement lié à la bonne exploitation de ceux-ci.

D'après les médias spécialisés dans le domaine des nouvelles technologies , la mise en place de GPRS est destiné à être un laboratoire grandeur nature, permettant aux divers acteurs actifs dans le secteur de la téléphonie mobile de se familiariser avec les réseaux haut débits et de tester les différents types d'applications que les clients recherchent. L'emploi de GPRS demande des ressources financières et techniques beaucoup plus faible que UMTS.

A l'heure actuelle personne ne connaît qu'elle est l'application qui permettra de générer du trafic sur le réseaux GPRS, tout le monde est à la recherche de la «killer application». Ce travail de diplôme traite d'un service à offrir aux abonnés GPRS, en espérant que celui-ci soit « la killer application » que tout le monde attend.

3 Objectifs

Le but de ce projet est de réaliser une application qui fonctionne sur un Ipaq et qui permette d'envoyer des messages vocaux après les avoir créés à l'aide du Voice Recorder Tool à un destinataire mobile. Le réseau GPRS est mis à contribution pour effectuer l'envoi des messages vocaux.

Les données doivent être envoyées à un serveur qui lui se chargera de les stocker et envoyer un SMS au destinataire pour l'informer de l'arrivé d'un nouveau message. Le destinataire pourra alors se connecter au serveur pour récupérer son message. Il pourra ensuite effectuer certaines opérations, comme le consulter, le transférer à un tierce ou l'effacer.

La solution d'envoyer le message sur un serveur et non pas directement au destinataire se justifie principalement par le faite de ne plus devoir se préoccuper de savoir si le destinataire à sur lui son Ipaq, lui permettant de lire le message. L'utilisateur peut ainsi prendre connaissances des ces messages au moment où il le juge opportun.

Dans ce document nous traitons d'une solution permettant l'envoi de messages vocaux à l'aide de l'Ipaq. La seule chose qui différencie un message vocal d'un autre type de message est son nom. Il est donc bien évident que notre application permet d'envoyer n'importe quel type de messages que ce soit des viédos, des photos et etc.

4 GPRS

Ce paragraphe n'a pas la prétention d'expliquer dans les moindres détails le réseau GPRS, ceci représenterait certainement le sujet de plusieurs travaux de diplôme. Nous allons juste donner quelques informations permettent de savoir ce qu'est un tel réseau.

4.1 Architecture GPRS

GPRS signifie *General Packet Radio Socket* et constitue une évolution de GSM, par l'utilisation de la communication par paquets afin de réaliser l'envoi de données. La réalisation d'une telle architecture se traduit principalement par l'adjonction de deux nouveaux éléments, le GGSN *Gateway GPRS Support Node* et le SGSN *Serving GPRS Support Node*.

Le GGSN est employé comme passerelle entre GSM et les réseaux informatiques de transmissions de paquets. Le GGSN convertit les protocoles (externes-interne) à la fonctionnalité de routage et peut rassembler les informations de taxation et filtrer les informations non désirées.

Le SGSN est le central pour les données par paquets dans GSM. Il commute les données par paquets, il a fonctionnalité de routage, il a la responsabilité de mobilité permettant au téléphone mobile de se déplacer.

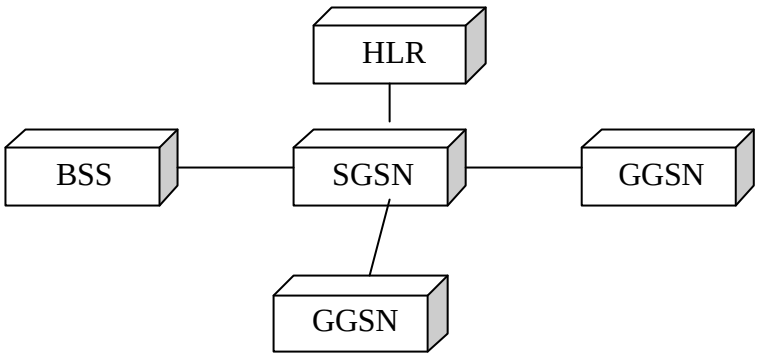
Lorsque un mobile se connecte au réseau GPRS, il se voit attribué de manière dynamique une adresse IP.

Un réseau GPRS à la structure montré en figure 1

Le BSS *Base Station Subsystm* représente le sous-système radio.

Le HLR *Home Location Register* est une base de données contenant tous les profils et les localisations des abonnés du réseau.

Figure 1 *Structure du réseau GPRS*



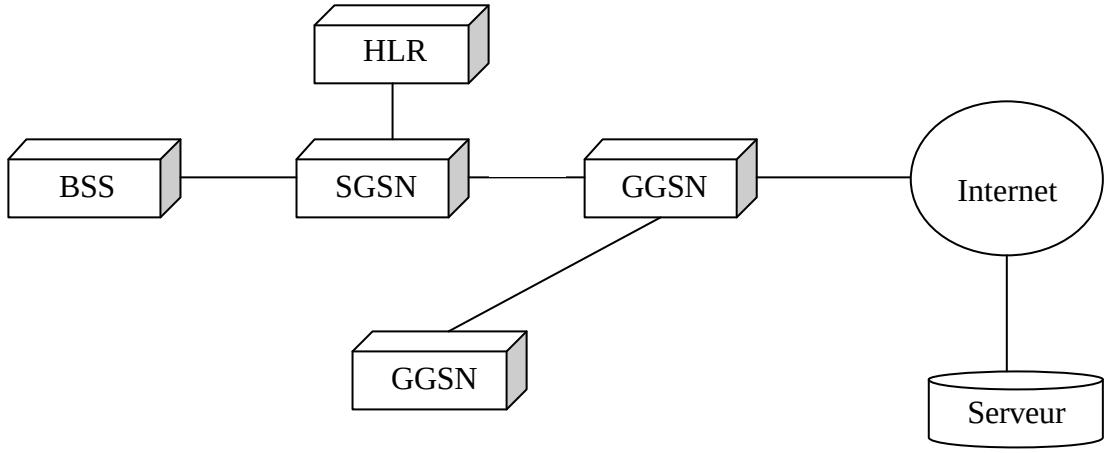
5 Architecture matérielle

Notre service à pour but d'exploiter le réseau GPRS et son emploi doit donc être destiné uniquement aux utilisateurs ayant accès à ce réseau. Pour satisfaire ce point il faut que notre serveur soit directement relié au réseau GPRS. Ceci nous conduit à avoir une architecture où le serveur est directement connecté sur le GGSN.

Bien évidemment l'utilisation d'une telle architecture n'est pas réaliste dans la phase de développement, principalement pour une raison, nous ne pouvons pas prendre le risque que notre service puisse nuire de quelques façons que ce soient aux performances du réseau GPRS de Swisscom. Une telle architecture ne sera donc possible que lorsque notre application aura passé avec succès les différents tests de validation définis par Swisscom.

Au vue de cette situation, nous avons pris la décision de mettre notre serveur dans une zone démilitarisée accessible depuis le réseau GPRS. Notre serveur se trouve à l'adresse IP **193.134.216.180**. Ceci nous conduit à avoir l'architecture de la figure 2.

Figure 2 *Architecture de la plate forme pour la phase de développement*



Un point important qu'il faut garder à l'esprit est que tout au long de la phase de développement notre application n'a pas accès aux ressources du réseaux GSM/GPRS.

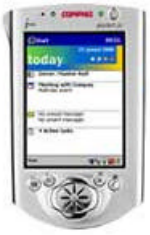
6 Le client

Dans notre application, le client représente la combinaison d'éléments permettant de générer, d'envoyer et de récupérer les messages vocaux. En ce qui nous concerne, ces éléments sont un PDA *Personnal Data Assistant* 'Ipaq et un téléphone mobile le T39 m d'Ericsson.

L'Ipaq a été choisi parce que il possède un enregistreur vocal permettant de créer les méssges vocaux devant être envoyés.

6.1 L'Ipaq

L'iPaq est un PDA développé par Compaq, celui utilisé dans œ projet est le H3630, nous donnons ci-dessous quelques unes de ces caractéristiques à titre d'informations.



Processeur : Intel SArm (*Strong Arm*) de 206 Mhz
Ram : 32 Mo
Système d'exploitation : WinCe 3.0
Communication : Port Infrarouge, IrDA *Infrared Data Association*
Fonction : Possède un enregistreur vocal.

6.2 Le téléphone mobile

Notre application ne demande pas un type de téléphone mobile spécifique, mais le mobile doit posséder au moins deux caractéristiques, la première est qu'il supporte bien évidemment Gprs et la seconde qu'il possède un port de communication infrarouge compatible avec la norme IrDA. Ce dernier sert à communiquer avec l'IPaq.

Le téléphone mobile utilisé dans notre application est le modèle T39 développé par Ericsson. Notre choix s'est porté sur ce téléphone car il nous a gentiment été prêté par Swisscom Mobile comme l'IPaq d'ailleurs. Ce mobile possède les caractéristiques indispensables pour être utilisé dans notre application, à savoir qu'il supporte Gprs et qu'il possède un port de communication infrarouge compatible avec la norme IrDA.

6.3 Communication du client

Le client utilise deux protocoles de communication distincts, l'un pour permettre la communication entre les deux éléments constituant le client et l'autre pour communiquer entre le client et le monde extérieur, c'est-à-dire avec le serveur VoGprs. L'échange d'informations entre les différents éléments constituant le client, donc entre l'IPaq et le T39m est effectuée par IrDA. En ce qui concerne la communication avec le monde extérieur, entre le téléphone et le serveur VoGprs, la norme Gprs est mise à contribution.

Le T39m et l'IPaq possède tout deux des ports IrDA, pour l'IPaq c'est au logiciel se trouvant sur l'Ipaq d'activer la communication par IrDA, pour le T39m il faut mettre le téléphone en mode Infrarouge.

6.4 Le langage de programmation

Le choix d'un langage de programmation dépend de critères tels que la portabilité (capacité à pouvoir fonctionner sur plusieurs plates formes) et la rapidité d'exécution.

Dans le but de rendre notre application utilisable sur le plus grand nombre de PDA possible, l'emploi d'un langage portable comme Java s'impose. Ce qui rend Java indépendant de tout matériel est qu'il exécute le programme dans une machine virtuelle.

Si l'on veut programmer notre Ipaq avec Java, il faut lui trouver une machine virtuelle fonctionnant sous WinCe. Lorsque l'on sait à quel point les relations entre Microsoft et Sun (concepteur de Java) sont hostiles, on imagine facilement que trouver une machine virtuelle pour WinCe sera un vrai défi. Effectivement après de longues recherches nous n'avons trouvé qu'une seule une machine virtuelle adéquate à WinCe, le seul problème est qu'elle coûte 20 US Dollars.

Ceci signifie que pour chaque utilisateur Ipaq, l'on doit débourser 20 dollars. Dans un monde où la guerre des prix est féroce comme c'est le cas dans la téléphonie mobile, demander à un client de payer 20 dollars pour utiliser un service à pour conséquence de réduire fortement les changes de succès de ce dernier.

Le prix que l'utilisation de Java implique, nous a poussé à nous demander si la portabilité est une contrainte obligatoire. Notre application doit envoyer des messages vocaux, il est donc inévitable d'utiliser un PDA permettant de générer ces messages, ceci réduit donc fortement le nombre de candidats. Nous n'avons recensé que l'Ipaq.

Un autre point important qui va à l'encontre de la portabilité, est que le marché des systèmes d'exploitation pour les PDA est dominé par Microsoft, avec son WinCe. Cette domination va encore s'accroître avec la nouvelle version appelée PocketPc 2002 qui remplace WinCe. La meilleure preuve montrant que le marché des PDA n'échappera certainement pas à Microsoft est donnée par tout les grands acteurs du marché comme Compaq, HP, et même Ericsson qui signent des contrats de partenariats avec Microsoft pour l'utilisation de leur système d'exploitation.

Ces arguments nous ont convaincus à ne pas prendre en considération la portabilité dans le choix du langage de programmation.

Microsoft fournit **gratuitement**, une suite d'outils dénommée Microsoft eMbedded Visual Tools permettant de programmer des PDA. Celle-ci contient deux outils de programmation qui sont eMbedded Visual Basic 3.0 et eMbedded Visual C++ 3.0 qui ne sont rein d'autre que des versions allégées de Visual Basic et Visual C++ destiné à la programmation de systèmes possédant peu de ressources. Cette suite d'outils met également à disposition des simulateurs de PDA, dont l'un est destiné à l'Ipaq.

Le choix du langage de programmation s'est porté sur eMbedded Visual Basic 3.0, ceci pour plusieurs raisons : La première est que nous avons déjà de bonnes connaissances de Visual Basic pour l'avoir utilisé dans différents laboratoires, la seconde est qu'il offre d'innombrables fonctions permettant de se connecter à des base de données, de réaliser des modules destinés au monde Internet, etc. eMbedded Visual Basic offre les mêmes possibilités que eMbedded Visual C++, mais son utilisation est nettement plus simple.

7 Gestion des informations

L'une des tâches principales du serveur est de gérer les données représentât les messages vocaux qui lui sont envoyés par les différents utilisateurs. La partie critique de notre travail concerne la gestion de ces messages, car c'est sur cette dernière que repose en grande partie le bon fonctionnement de notre application.

La gestion de ces messages se fait à l'aide d'une base de données. Cette solution offre l'avantage de résoudre tous les problèmes relatifs aux accès concurrents, aux verrous mortels et autres que nous pouvons rencontrer dans les applications client-serveur. Effectivement les environnements de base de données gèrent eux-mêmes les cas d'accès concurrents, ainsi, si plusieurs clients accèdent en même temps à la base de données, la gestion des priorités sera prise en charge par la base de données elle même.

Cette solution offerte par les bases de données est un grand avantage, qui évite de devoir mettre en place de lourds mécanismes de gestion de priorité. Il faut tout de même rappeler que de laisser la gestion des accès concurrent à la base de données est possible uniquement parce que la seule ressource à laquelle les clients vont accéder de manière concurrente est la base de données. Si les clients ont besoin d'accéder à d'autres ressources que la base de données, il faudrait mettre en œuvre des mécanismes de gestion de la concurrence.

Un autre grand avantage qu'offre l'utilisation d'une base de données dans la gestion de nos messages est qu'elle permet de standardiser les opérations. Effectivement toutes les opérations de gestion des messages se résument à des commandes du langage SQL (langage de requêtes). De plus, une base de données peut facilement être accédée depuis différents langages de programmation grâce à l'utilisation de passerelles réalisant le lien entre le langage de programmation et la base de données elle-même.

8 Structure de la base de données

La base de données doit permettre de gérer les utilisateurs et tous les messages échangés entre-eux. La conception d'une base de données est une démarche d'abstraction qui nécessite l'utilisation de modèle. Le modèle Entité-Association EA est un des modèles conceptuel les plus utilisés. Une entité définit une chose concrète ou abstraite qui intéresse le système d'information et à propos de laquelle on veut enregistrer de l'information. Une association est une correspondance entre deux ou plusieurs entités. L'attribut est une caractéristique d'un type d'entité ou d'un type d'association. Un autre point important dans les bases de données est la notion de clé. Celle-ci est un attribut ou un ensemble d'attributs qui permettent de définir de manière unique chaque tuple d'une entité.

Cherchons les entités et les relations que contiennent notre modèle. On constate facilement la présence de deux entités, une définissant les **utilisateurs** et l'autre définissant les **messages**. Le lien entre ces deux entités définit l'association dans notre modèle. Désormais cette association sera définit par le nom **communication**.

Notre modèle, bien qu'il paraisse assez simple à première vue, cache une petite subtilité qui n'a pas été des plus évidente à résoudre. Le problème est que, un utilisateur peut être soit l'expéditeur, soit le destinataire d'un message. Ceci nous oblige a insérer dans l'association communication l'utilisateur représentant l'expéditeur et l'utilisateur représentant le destinataire du message.

Une fois notre modèle EA trouvé, définissons les attributs pour les entités et l'association. Ceci revient à introduire les caractéristiques avec lesquelles nous allons définir les utilisateurs, les messages et l'association reliant les deux entités.

8.1 L'entité Utilisateurs

La plus simple manière de définir un utilisateur est d'utiliser son MSISDN *Mobile Station ISDN Number* qui ne représente rien d'autre que son numéro de téléphone. Ce seul attribut permet de définir de manière unique un utilisateur car le MSISDN n'existe qu'à un seul exemplaire. Nous allons encore ajouter un attribut à notre utilisateur, nous lui insérons un mot de passe *Password*. Ce nouvel attribut permet d'accroître la sécurité de notre système, ainsi on peut s'assurer qu'un utilisateur indiquant un MSISDN en soit bien le propriétaire.

Nous devons également dire que le choix du MSISDN sait imposé par lui-même, effectivement c'est le seul élément qui permette au serveur de réaliser l'envoi de SMS sans avoir besoin d'accéder aux ressources du réseau GSM/Gprs plus particulièrement le HLR pour réaliser ce dernier. Ceci n'aurait pas été possible si nous avions utilisé le IMSI *International Mobile Subscriber Identity* par exemple, là un accès au HLR aurait été indispensable pour réaliser la correspondance IMSI-MSISDN.

8.2 L'entité Messages

Les attributs que nous avons définis pour l'entité messages sont, le contenu qui représente le nom du message, sa taille et sa date de création, c'est-à-dire la date d'arrivée du message sur le serveur. La date de création n'est pas la date de création physique du message par l'utilisateur : La date qui lui est attribuée par le Ipaq, mais la date définie par le serveur dès la

réception du message. Cette solution est adoptée dans le but d'assurer une cohérence au niveau des dates entre tous les messages échangés par les différents utilisateurs. On ne peut pas utiliser la date de création physique du message, car il existe d'innombrables Ipaq dans la nature et il est impossible de s'assurer que tous soient synchronisés. Cet état de fait nous oblige à utiliser une horloge qui soit indépendante de tous les clients et la meilleur solution est d'utiliser celle se trouvant sur le serveur.

Il nous reste encore à définir la clé de cette entité. On ne peut pas utiliser l'un ou l'ensemble des attributs définis jusqu'à présent, car on ne peut pas assurer qu'il n'y ait pas deux contenus identiques, au contraire il est fort probable que deux utilisateurs nomment leur message de la même façon. Utiliser la date de création comme clé, n'est pas possible, car même si c'est peu probable il est tout à fait possible de recevoir deux messages en même temps, si on ajoute à ça la dérive de l'horloge, deux messages reçu à quelques milisecondes d'intervalles peuvent être interprétés comme deux messages reçus en même temps. Les deux cas mentionnées ci-dessus ne nous permettent pas de définir un message de manière univoque.

Après avoir réfléchi longuement à ce problème sans trouver de solution satisfaisante, fonctionnant dans tous les cas de figure, nous avons décidé de mettre en œuvre une solution insérant comme clé un numéro de message. Ce numéro est incrémenté de un à chaque insertion dans la base de données d'un nouveau message. Cette solution n'est pas optimale, car il arrivera un moment où le numéro de message atteindra sa capacité maximale et ne permettra plus d'insérer de nouveaux messages. Bien que nous indiquions que cette solution ne soit pas idéale, nous nous permettons de l'utiliser quand même, et ceci pour diverses raisons : La première est que nous n'avons pas trouvé d'autres solutions, la seconde est que la plus part des systèmes de base de données permettent de stocker des chiffres à neuf caractères, ce qui nous donnes une plage de valeurs allant de 0 à 999999999 ceci nous permet de stocker un milliard de messages. Ce chiffre nous semble assez imposant pour notre application et pour son utilisation dans le cadre d'un travail de diplôme.

Bien que pour une utilisation commerciale, cette solution ne soit pas adéquate, elle laisse tout de même un certain temps de manœuvre pour trouver une autre solution. Effectivement en admettant que dans le meilleur des cas il s'échange un million de messages par jour, cela nous laisserait mille jours, (un peu moins de trois ans) pour trouver une clé adéquate. De plus le nombre de jours peut facilement augmenter car les bases de données professionnelles sur lesquelles serait implémentée notre application comme Oracle ou Sybase permettent de stocker des chiffres un peu plus de quatre milliards de messages.

8.3 L'association Communication

Comme nous l'avons dit précédemment, il faut que l'association communication contienne l'expéditeur, le destinataire et bien évidemment le numéro de message qui est échangé entre les deux utilisateurs. Le fait qu'un utilisateur puisse transférer un message qu'il a reçu à un tierce, nous oblige a insérer un nouvel attribut dans cette association, une date d'envoi. Celle-ci définit le moment auquel le transfère du message est effectué. Cette date diffère de la date de création contenu dans l'entité Messages, la date de création définit la date à laquelle le message est reçu par le serveur. La date d'envoi définit dans l'association Communication indique l'heure à laquelle l'envoi du message au destinataire s'effectue réellement. Comme pour la date de création, la date d'envoi est définie par le serveur.

L'association communication doit contenir encore deux attributs, le premier permet d'indiquer si le SMS indiquant la réception d'un nouveau message à été envoyé et le second permet de savoir si un message reçu par un destinataire à déjà été consulté.

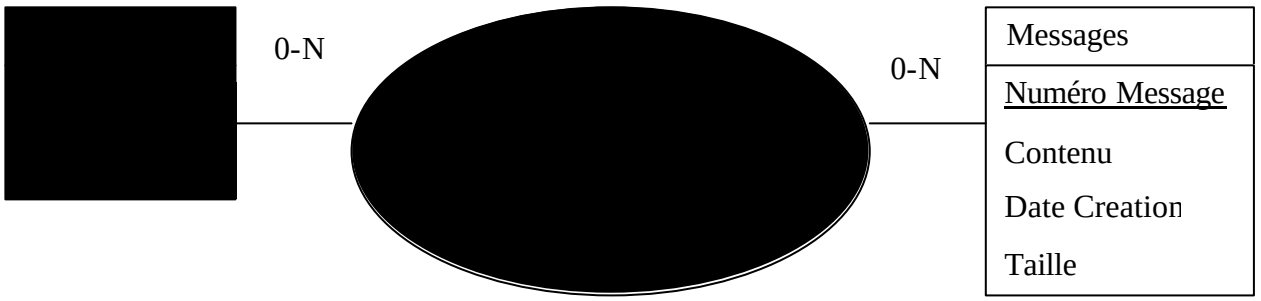
La clé de cette association est définit par l'ensemble des attributs, expéditeur et destinataire.

Pour que notre association soit complètement définit il faut encore indiquer de quel type d'association il s'agit. Il existe trois types d'associations, un à un (1 :1), un à plusieurs (1:n) et plusieurs à plusieurs (n :n). Les types d'association définissent le nombre d'occurrences mises en jeu de part et d'autre de l'association. Dans notre cas cela consiste à indiquer le nombre de message qu'un utilisateur peut envoyer, ce nombre peut varier entre 0 et n, nous avons donc une association de 0 à n. Il faut également indiquer à combien d'utilisateurs un message peut être destiné. Dans notre cas un message peut appartenir à 0 ou n utilisateurs. Nous avons donc encore une fois une relation 0 à n.

8.4 Représentations graphique

Les entités Utilisateurs, Messages et l'association communication que nous venons de définir entraînent la représentation graphique du modèle Entité-Association définie en figure 3.

Figure 3 *Entité-Association de la base de données*



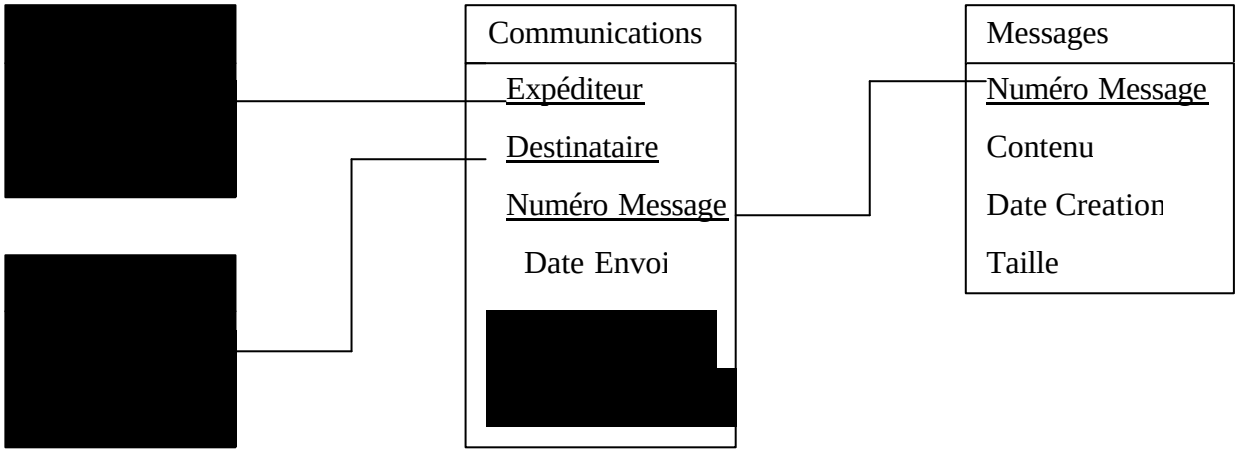
Rappelons que les attributs permettent de modéliser la réalité, par conséquent il est évident que l'on pourrait définir différemment les entités et l'association. Lors de la définition des attributs nous avons essayé de garder à l'esprit que plus le nombre d'attributs est grand et plus la mise à jour, c'est-à-dire l'insertion, l'effacement ou la modification d'éléments de la base de données est lente vu que le nombre d'attribut concernée est plus important. De plus la gestion et la maintenance de la base de données s'en retrouvent alourdies.

Nous n'avons donc inséré que les attributs indispensable permettant de respecter le cahier des charges tout en garantissant une bonne exploitation de la base de données.
Le modèle Entité-Associaiton ne donne qu' une représentation logique de la base de données, et ne donne pas la représentation finale, sous forme de tables de la base de données. Cette forme est dénommé Schéma de relation. Pour établir ce schéma, il faut appliquer un certains nombres de règles au modèle Entité-Association. Nous n'allons pas énoncer toutes les règles, nous allons juste indiquer celle qui s'applique à notre situation, c'est-à-dire que si on a des deux cotés de l'association une relation de type 0 à n il faut insérer une nouvelle table portant

le nom de l'association et y rajouter comme attribut les clés des entités reliant l'association. Le dédoublement de l'entité utilisateur s'explique car un utilisateur peut être à la fois expéditeur et destinataire, de plus des règles d'intégrités obligent à ce que chaque clé soit reliée avec sa clé équivalente sur une autre table.

La figure 4 ci dessous montre le schéma de relation.

Figure 4 *Schéma de relation de la base de données*



Notre base de données se caractérise donc pas trois tables, qui sont les tables, Utilisateurs, Messages et Communications.

9 Choix de la base de données

Pour choisir une base de données, il faut prendre en considération les critères suivants : Le nombre de table, le nombre de tuples et la fréquence des accès à la base de données. Comme nous l'avons déjà vu, nous avons trois tables, le nombre en est donc faible, en ce qui concerne la quantité de tuples, elle dépend directement du nombre de messages échangés entre les utilisateurs. Cette valeur dépendra du succès que le service rencontrera, mais pendant la phase de développement que caractérise ce travail de diplôme, nous pouvons considérer que ce nombre est faible, quelques dizaines de tuples puisqu'il n'y a qu'un seul utilisateur. Enfin, pour ce qui concerne la fréquence des accès à la base de données, elle dépend directement du nombre d'utilisateurs. Comme pour le nombre de tuples, cette valeur dépendra du succès que le service rencontrera. Pendant la phase de développement le nombre d'utilisateur est également faible, puisqu'il est de un.

Comme nous pouvons le constater, les contraintes que subit la base de données changent en fonction que nous soyons dans la phase de développement ou dans la phase d'exploitation commerciale du service. Dans la première phase, comme nous l'avons vu , les contraintes sont faibles, par contre pour ce qui concerne la seconde phase les contraintes sont intimement liées au succès du service et donc au nombre d'utilisateurs. Il est donc probable que ces contraintes puissent être très élevées, elles laissent donc supposer que l'utilisation d'une base de données professionnelle "robuste" telle que Oracle ou Sybase soit indispensable.

Les contraintes, les difficultés et les ressources qu'imposent l'utilisation et la gestion d'une base de données professionnelle ne sont pas justifiées dans notre phase de développement, elles pourront le devenir par la suite, mais pour l'instant nous préférons utiliser une base de données d'exploitation plus simple répondant parfaitement à nos besoins actuels.

Nous connaissons principalement deux bases de données demandant peu de ressources pour leur utilisation, l'une est Access de Microsoft et l'autre est mySql. Cette dernière est un produit libre d'utilisation, ne demandant aucune licence. Elle est très utilisée sur Internet, principalement couplée avec des serveurs fonctionnant sous Linux, système d'exploitation libre lui aussi.

Les deux bases de données, offrent des caractéristiques équivalentes. Généralement lorsque deux produits ont les mêmes caractéristiques, on choisit le moins cher donc dans ce cas mySql qui est gratuit, mais pas cette fois et ceci pour plusieurs raisons : La première est que nous sommes déjà familiers avec Access pour l'avoir utilisé dans divers projets de laboratoire, la seconde est que notre serveur sur lequel doit être installé la base de données, fonctionne sous Windows NT, et comme tout le monde le sait, la firme de Mr. Gates est hostile à tout ce qui n'est pas "made by Microsoft". Nous préférons privilégier l'option tout Microsoft pour s'assurer que la compatibilité entre les différents produits soit respectés. Nous pensons que les quelques points ci-dessus feront oublier que Access n'est pas gratuit surtout que sa licence ne vaut qu'une centaine de francs et que nous n'ayons pas de besoin de les débourser puisque nous avons une licence d'utilisation.

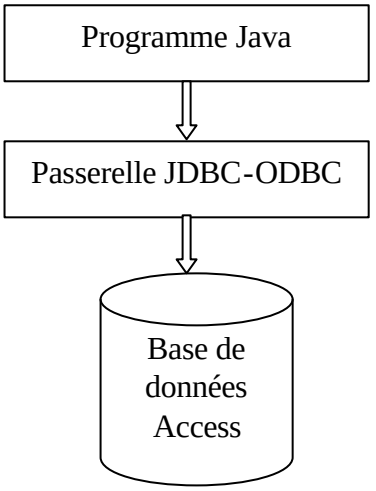
En résumé nous allons utiliser Access pour le développement et il faudra passer à une base de données professionnelle pour l'exploitation commerciale. Pour faciliter au maximum cette migration, il faut que toutes les opérations effectuées sur la base de données se fassent en

langage SQL standard, qu'il n'y ait pas de requêtes embarquées directement dans la base de données, ainsi la seule chose à changer lors de la migration de la base de données sera la passerelle entre l'application tournant sur le serveur et la base de données. Cette passerelle permet de pouvoir associer des applications développées dans un certain langage à différentes bases de données sans avoir à modifier la moindre ligne de code.

Par exemple le langage Java offre comme passerelle l'interface *JDBC Java Database Connectivity* qui permet de faire le lien entre des applications développées en langage Java et différents types de base de données. Cette passerelle est généralement fournie par l'éditeur de la base de données.

La figure 5, ci-dessous montre l'utilisation de la passerelle JDBC-ODBC permettant de connecter une application écrite en Java à une base de données Access.

Figure 5 *Schéma permettant de se connecter à une base de données Access*



10 Facturation du service

Lors de l'envoi ou du téléchargement d'un message à l'aide du client, (de l'Ipaq et du réseau Gprs), les informations permettant d'établir la facturation à savoir principalement le nombres de données transférées sur le réseau sont fourni par le réseau lui-même. Le problème de la facturation ne se poserait pas si l'on était certain que lors de l'envoi d'un message à un destinataire le message transite physiquement par le réseau. Ceci se pose lorsqu'au lieu d'effectuer l'envoi d'un nouveau message, on transfert un message se trouvant déjà sur le serveur à un tierce utilisateur.

Etant donné que le message est déjà sur le serveur, il n'est pas judicieux que l'utilisateur le renvoie au serveur, puisque les seules informations desquelles nous avons besoin sont celles concernant le destinataire et l'émetteur, il est donc plus économique en terme d'utilisation des ressources du réseau GPRS de n'envoyer que ces informations là. Dans ce cas la facturation est faussée puisqu'un utilisateur peut transférer à une tierce personne, un message faisant plusieurs megaoctect en ne faisant transiter sur le réseau qu'un faible nombre de données, l'équivalent d'un fichier de commande.

Pour résoudre ce problème, il faut donc que le serveur enregistre toutes les informations concernant les messages échangés entre les utilisateurs. La solution que nous avons mise en place consiste à insérer une nouvelle table que nous appelons **Facturation** dans notre base de données. Celle-ci est très proche de la table Communication, a l'exception près que les attributs SMS Envoyé et nouveau Message sont supprimés.

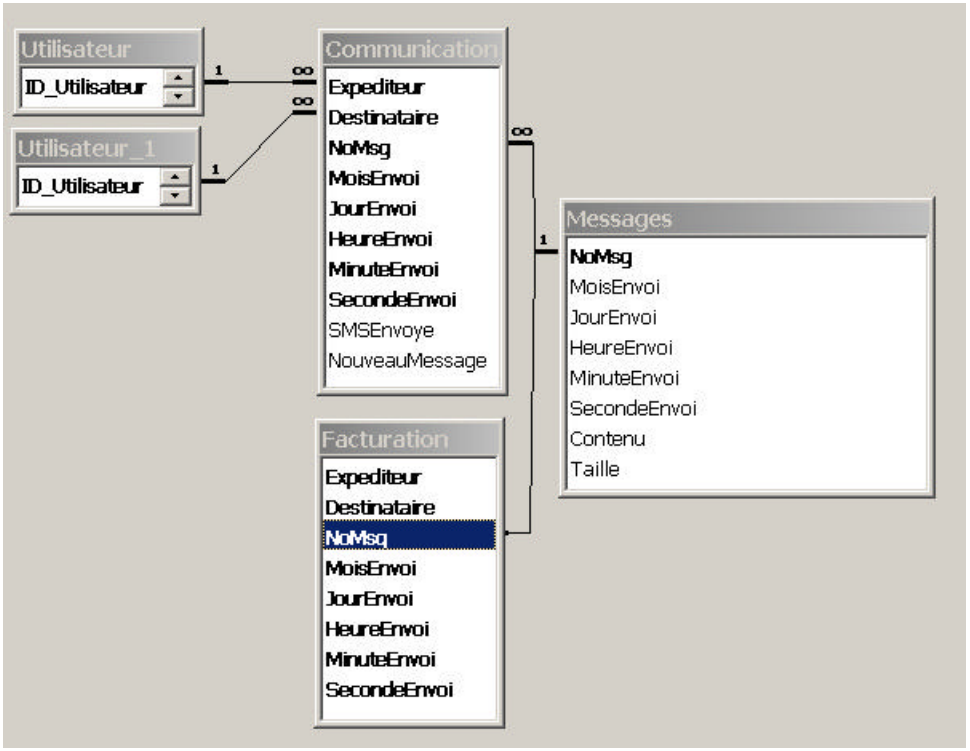
L'utilisation de la table Facturation s'impose bien qu'elle soit très proche de la table Communication qui existe déjà. Effectivement on ne peut pas utiliser la table Communication pour garder une trace de tous les messages échangés et ceci pour une raison simple, c'est que les utilisateurs ont accès à cette table en mode lecture et écriture, ils peuvent donc y effacer des informations, ce cas se présente lorsque un utilisateur veut supprimer un message de son compte. Les informations sur les messages échangés sont donc perdues.

L'application, plus précisément le serveur, ne peut réaliser qu'une seule opération sur la table Facturation, qui consiste à insérer les informations indiquant le transfert d'un message entre les utilisateurs. Le serveur ne peut réaliser aucune autre opération sur cette table : Consulter, effacer des informations, etc . Le traitement des informations contenues par cette table ne doit être permis que à l'exploitant du service. Pour l'heure nous ne réalisons aucune opération sur cette table à part l'insertion des informations relatives à la réception de la part d'un utilisateur d'un nouveau message, mais l'on peut facilement réaliser des opérations indiquant la façon dont le service est exploité par les utilisateurs, on pourrait savoir le nombre de messages envoyés par utilisateur, les heures d'envoi, la taille moyenne des messages envoyés, etc… Toutes ses fonctions sont facilement réalisable puisque ce ne sont rien d'autres que de simples requêtes SQL.

Dans le but de rendre notre application le plus indépendante possible de la base de données, nous avons préférer ne pas utiliser pour représenter les dates et les heure les formats offert par Access qui sont propriétaire. Nous avons opté par une solution qui n'utilise que le type Integer qui lui existe dans toutes les base de données.

La figure 6 ci-dessous montre le schéma de relation de la base de données Access utilisé dans notre application.

Figure 6 *Schéma de relation de la base de données Access utilisée*



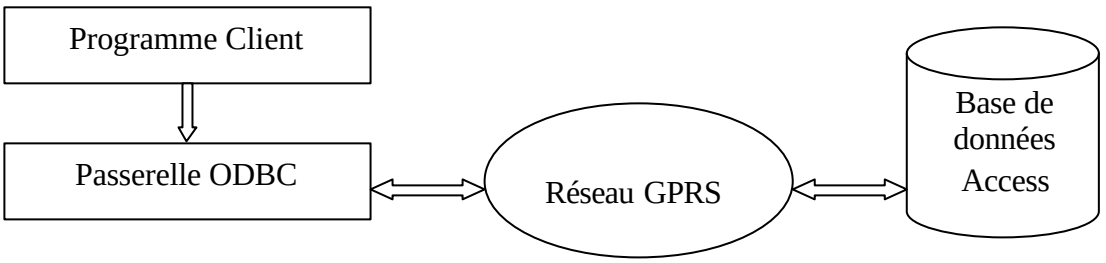
11 Accès à la base de données

La mise à jour de la base de données, c'est-à-dire l'insertion ou l'effacement de messages, peut principalement se faire de deux manières différentes. La première consiste à ce que les modifications se fasse directement depuis le client, cela signifie que chaque client à un accès direct à la base de données. La seconde est que les modifications soient faites depuis l'application tournant sur le serveur, c'est-à-dire que seul une application a accès à la base de données.

11.1 Depuis le client

Cette solution consiste à dire que toutes les mises à jour se font directement depuis le client, ainsi les requêtes SQL sont envoyées depuis l'Ipaq. Le système d'exploitation WinCe tournant sur l'Ipaq permet l'utilisation d'une base de données dénommée Microsoft SQL Server 2000 WinCe qui fournit tous les outils pour se connecter à une base de données distantes. L'architecture de cette solution est montrée en figure 7.

Figure 7 *Schéma montrant la connexion à la base de données depuis le client*



L'utilisation d'une base de données fournie par Microsoft comme Access ou SQL Server nous pose quelques problèmes, notamment pour l'insertion d'un nouveau message, parce que ces bases de données ne permettent pas d'utiliser un fichier comme type de données, l'on ne peut insérer un fichier comme information dans la base de données, mais uniquement un lien sur celui-ci, c'est-à-dire une chaîne de caractères pointant sur un fichier. L'emploi d'une telle solution ne serait possible que si chaque client Ipaq a une adresse IP fixe et qu'il soit toujours online de façon à ce que chaque utilisateurs puissent retrouver un message à l'aide du lien contenu dans la base de données. Il est bien évident que ces conditions sont impossibles à garantir et ceci pour plusieurs raisons : La première évidemment est qu'il est impossible de demander à un utilisateur de laisser son Ipaq toujours allumé, la seconde est qu'il est peu réaliste de disposer d'une adresse IP fixe même avec l'introduction "prochaine" de IPV6. Effectivement actuellement les adresses IP obtenues lors d'une connexion au réseau Gprs sont fournies par DHCP, les adresses ne sont donc pas constantes.

L'architecture décrite ci-dessus serait possible si nous disposions d'une base de données qui permette d'utiliser des fichiers de préférences de types binaires comme types de données. Après quelques recherches nous avons trouvé une base de données qui le permet, il s'agit de mySql. Cette base de données permet d'utiliser un type de données dénommé Blob qui permet

de stocker des données binaires de taille maximum d'un peu plus de 4Goctets. Cette taille est plus que suffisante pour les besoins de notre application, surtout lorsqu'on sait que l'enregistrement d'environ 10 secondes avec le Voice Recorder Tools de l'Ipaq occupe une vingtaine de koctes. Ces quelques 4 Goctets offrent assez d'espace pour l'envoie d'autres types de données comme par exemple des photos, des vidéos ou des fichiers musicaux MP3.

On peut même dire que dans notre cas ces 4 Goctets représente un nombre trop élevé par rapport à la bande passante dont nous disposons. Effectivement pour le moment Gprs, offre un débit tout à fait honorable de quelques 30 Koctets/seconde, pour transférer 4 Goctets il faut environ 9 heures de temps, autant dire qu'il faut être extrêmement patient.

L'utilisation de mySql comme base de données, nous oblige à lui trouver une passerelle fonctionnant sur l'Ipaq. Lorsque l'on sait que l'Ipaq fonctionne avec le monde Microsoft (WinCe) et que mySql est très utilisé sous Linux, on peut facilement imaginer que la cohabitation ne va pas être des plus faciles.

Comme nous pouvions le penser, la seule passerelle que nous avons trouvée fonctionnant sous WinCe et permettant de piloter la base de données mySql est une passerelle JDBC-mySql. Cela signifie que l'on doive imposer le langage Java comme langage de développement de l'application client. Le langage Java n'a pas été retenu comme langage de développement pour l'application à cause des raisons citées en page 9. Ces raisons nous empêchent d'utiliser mySql, et ce type d'architecture, mais ce n'est pas la raison la plus fondamentale.

Même si nous trouvions une passerelle pour mySql utilisant le langage de développement eVB fonctionnant sous WinCe, cette architecture ne serait toujours pas utilisable dans notre type d'application, principalement pour une question de sécurité. Effectivement dans notre cas tous les clients ont un accès direct à la base de données, cette perspective ne provoquerait pas trop de sueurs froides si les clients avaient un accès en lecture uniquement, puisque ils n'auraient pas la possibilité de modifier les données de la base de données. Dans notre cas les clients doivent pouvoir modifier des données, c'est-à-dire en insérer ou en effacer de façon à indiquer l'envoi ou l'effacement d'un message, ceci signifie que tous les clients ont un accès total à la base de données en mode écriture et lecture. Dans une telle situation on peut facilement imaginer qu'un utilisateur mal veillant pourrait détruire notre base de données sans grande difficulté, il est bien évident qu'une telle situation va à l'encontre des niveaux de sécurité que notre application doit garantir et n'est donc pas applicable. La seule solution garantissant un niveau élevé de sécurité est celle où la base de données est accéder par une seule application.

11.2 Depuis le serveur

Cette solution consiste à dire que toutes les mises à jour se font uniquement depuis le serveur, c'est-à-dire qu'il n'y a que l'application serveur qui ait un accès total à la base de données. L'architecture d'une telle solution est représentée en figure 5, page 16.

Ce type d'architecture offre de bons critères de sécurité, pour plusieurs raisons : La première est évidemment qu'une seule application a un accès total à la base de données et la seconde est que toutes les opérations que l'application peut effectuer sur la base de données sont définies à l'avance. Cela veut dire que l'application est programmée pour effectuer certaines opérations principalement l'insertion ou l'effacement de message et rien d'autre.

Dans la situation ou la base de données était accessible directement depuis le client, les opérations effectuées sur la base de données ne pouvaient pas être garanties. Effectivement chaque utilisateur un peu malveillant pourrait programmer son client pour qu'il réalise n'importe quel type d'opération même des opérations non définies par notre serveur.

Une telle tentative de piratage est beaucoup plus difficile à réaliser dans la situation ou uniquement le serveur possède un accès à la base de données, puisque l'élément à accéder est l'application se trouvant sur le serveur lui-même et non plus l'application tournant sur le client qui est à disposition de chaque utilisateur.

En plus de la sécurité, cette solution offre une plus grande souplesse dans le choix de la base de données se trouvant sur le serveur. Il est beaucoup plus facile de trouver des passerelles pour tous les types de base de données devant fonctionner sur des systèmes d'exploitation tels que Windows (NT, 2000, etc.), Solaris, Unix et autre parce que ce sont les éditeurs de base de données eux-mêmes qui fournissent ces passerelles, alors que avec WinCe comme nous l'avons vu, le choix de la base de données est plus que limité.

Nous avons donc décidé d'utiliser cette architecture pour réaliser notre application.

12 Le Serveur

Par définition un serveur est un processus fonctionnant de manière infinie et offrant un service demandé par un client. Le principal service que doit offrir notre serveur est de mettre à jour la base de données en fonction des demandes qu'il reçoit, tel l'insertion ou la suppression d'un messages. Lors de la réception d'une demande d'envoi d'un message, le serveur doit envoyer un SMS permettant d'avertir un utilisateur qu'il a reçu un nouveau message.

12.1 Choix du langage de programmation

Les critères pour définir le langage sont les mêmes que ceux cités lors du choix du langage de programmation pour l'application client à savoir la portabilité et la rapidité.

Dans cette situation la portabilité est un point important, puisque dans le monde des serveurs le marché n'est pas dominé de manière aussi franche par un seul acteur comme c'est le cas avec les PDA, mais il est réparti entre plusieurs concurrents. L'utilisation de Java permet de rendre notre application serveur utilisable sur n'importe quel système d'exploitation.

Dans le monde des serveurs les machines virtuelles Java se trouvent beaucoup plus facilement, la meilleure preuve est que même Microsoft en met une à disposition avec Windows (NT, 2000).

Java en plus d'être portable, offre d'innombrable paquetage permettant par exemple de se connecter à une base de données, de communiquer avec un réseau IP, de réaliser des interfaces graphiques, etc. Nous avons donc choisi Java comme langage pour la réalisation de notre application serveur.

12.2 Communication avec le serveur

La question est de savoir comment nous allons communiquer avec le serveur dans le cas ou un client souhaite modifier l'état de la base de données, notamment pour signifier l'envoie ou l'effacement d'un message. Ceci consiste à savoir quel mécanisme nous allons mettre en œuvre pour permettre au client d'indiquer au serveur les opérations qu'il veut réaliser.

Puisque dans notre service, le but est d'envoyer des messages binaires (vocaux) entre les clients et le serveur, le plus simple est également d'envoyer un fichier de commande indiquant au serveur les opérations que le client souhaite exécuter.

12.3 Fichier de commande

Nous avons défini trois versions pour le fichier de commande, celle permettant l'insertion d'un nouveau message, celle indiquant le transfert d'un message déjà existant sur le serveur à une tierce personne et finalement celle permettant l'effacement d'un message se trouvant sur le serveur. Nous montrons la structure des fichiers de commande, ci-dessous en figure 8.

Figure 8 *Représentation des trois format de fichier de commandes*

Delete Destinataire 0792814219 Message 2 Mois 11 Jour 24 Heure 15 Minute 41 Seconde 36	Insert Emetteur 0796190370 Destinataire 0792814219 Message Enregistrement1.wav	Forward Emetteur 0796190370 Destinataire 0792814219 Message 2

Les fichiers de commande sont de simples fichiers texte et portent le nom suivant FileCmdMSISDN.cmd, ce qui donne par exemple FileCmd792814219.cmd. Le fait d'insérer le MSISDN permet de différencier les fichiers de commande entre-eux. Ceci est obligatoire puisque tous les clients envoient leur fichier de commande au même endroit sur le serveur.

Pour l'instant notre application ne permet d'envoyer qu'un seul message à un seul destinataire, de transférer un seul message à un seul destinataire et de n'effacer qu'un seul message à la fois. Cette situation est due à une question de délai, nous n'avons pas eu le temps d'implémenter d'autres possibilités comme l'envoi multiple. Mais nous sommes convaincus que la réalisation de ces nouvelles fonctions ne demande qu'un investissement en temps, car nos applications que ce soit du côté client que serveur offrent les fonctions logicielles de base.

12.4 Envoi des fichiers

L'envoi des fichiers, que ce soit ceux de commandes ou les fichiers représentant le message lui-même, entre les clients et le serveur équivaut à l'envoie de fichiers entre deux ordinateurs relié par un réseau IP. C'est exactement la même situation. Pour réaliser un tel échange d'information, nous avons à disposition plusieurs solutions différentes.

Nous pouvons utiliser des solutions travaillant au niveau quatre du modèles OSI comme TCP ou UDP. L'emploi de ces deux protocoles permet de réaliser des connexions sur des machines à l'aide de l'adresse IP et d'un port de communication.

TCP *Transmission Control Protocol* est un protocole avec connexion et incorpore le contrôle d'erreur.

UDP *User Datagram Protocol* est un protocole sans connexion et n'incorpore pas le contrôle d'erreur.

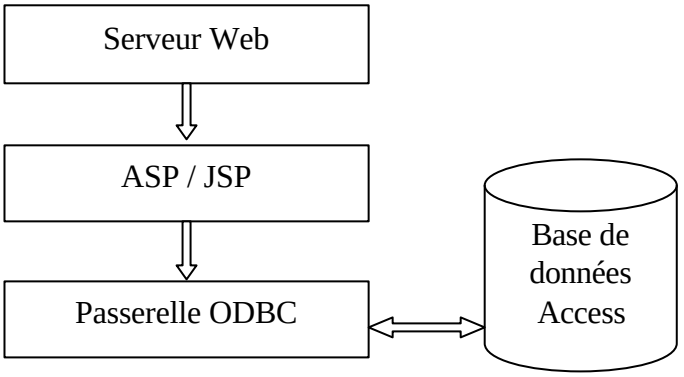
Nous pouvons également utilisé des solutions exploitant la couche application qui correspond à la couche numéro sept du modèle OSI. L'avantage d'utiliser de telles solutions est qu'elles prennent en charge toute la gestion des couches inférieurs. Ces solutions prennent donc en charge TCP ou UDP.

12.4.1 HTTP

HTTP, *Hyper Text Transfer Protocol* est un protocole qui permet de se connecter à un serveur Web et de transférer des données. L'utilisation de HTTP implique obligatoirement l'utilisation d'un serveur Web (permettant la connexion) du côté de notre serveur et évidemment que nos clients puissent générer et interpréter les requêtes HTTP.

L'utilisation d'un serveur Web, oblige l'adoption d'un moyen actif faisant le lien entre le serveur Web et la base de données, comme ASP *Active Server Pages*, JSP *Java Server Pages* , etc. et donc l'adjonction d'une couche logiciel supplémentaire pour accéder à la base de données. Ceci nous conduirait à avoir l'architecture montré en figure 9.

Figure 9 *Schéma montrant la connexion à la base de données depuis le client*



La couche active réalisant l'interface entre le serveur Web et la base de données peut être supprimée, à condition que notre application implémente un module permettant de traiter les requêtes HTTP en écoutant sur le port 80. Ce dernier est le port désigné pour les serveur Web. Cette solution n'est pas des plus simples à mettre en œuvre. Les éventuels avantages qu'apportent ce module, à savoir la suppression du serveur Web et du module actif (ASP, JSP), sans compter le temps de développement qu'il impose, ne se justifient nullement.

Effectivement le temps de conception d'une telle réalisation est certainement supérieur au temps de développement incluant le serveur Web et le module actif (ASP, JSP).

L'emploi de HTTP demande de gérer des tampons du coté de nos clients, par exemple lors du téléchargement d'un message, il faut indiquer au tampon de lecture la taille que nous voulons lire et tester à intervalle régulier si tous le message à été téléchargé. Il est évident que le même problème se pose lors de l'envoi d'un message à destination du serveur.

Cette brève analyse de HTTP, nous montre que son utilisation impose des contraintes qui ne sont pas des plus triviales à résoudre.

12.4.2 FTP

FTP *File Transfer Protocol* est comme son nom l'indique, un protocole permettant de transférer des fichiers entre différents ordinateurs sur des réseaux IP. L'emploi de FTP demande l'utilisation d'un serveur FTP (permettant la connexion) du coté de notre serveur et évidemment que nos clients puissent générer et interpréter les requêtes FTP. On trouve facilement des serveurs FTP gratuits sur Internet. Le port par défaut pour l'utilisation de FTP est le 20.

Contrairement à HTTP, FTP ne demande pas la gestion de tampons pour savoir si un fichier à été complètement transféré, FTP gère cette situation automatiquement. De plus l'utilisation de FTP permet de se décharger complètement de tous les problèmes qui se posent lors de l'envoi de fichiers entre ordinateurs, à savoir la gestion des erreurs, les contrôle de flux, etc…tous ces mécanismes sont incorporés dans FTP.

Par contre FTP ne permet pas au contraire de HTTP d'accéder directement à une base de données, même en y rajoutant différentes couches logicielles. Ce point n'est pas primordial, puisque ce qu'on demande à FTP c'est de transférer des fichiers entre nos clients et notre serveur.

Un point important à garder à l'esprit, est que avec l'utilisation de FTP les communications sont toujours générées par le client. Dans notre cas c'est l'Ipaq qui est le seul capable de lancer des requêtes FTP à destination du serveur FTP. Si le client veut envoyer un fichier à notre serveur ou en récupérer un depuis ce dernier, c'est à lui de lancer l'opération.

Le serveur FTP permet de créer des comptes avec des noms d'utilisateurs, des mots de passe et d'attribuer des droits sur ces comptes, ce qui permet de limiter son accès uniquement à nos clients, et permet d'augmenter ainsi notre niveau de sécurité.

Après avoir pris de toutes les possibilités qu'offre FTP, nous avons décidé de l'adopter pour le transfert de nos fichiers.

Nos clients sont configurer de sorte qu'il se connecte sur notre serveur se trouvant à l'adresse 192.134.216.180. L'utilisateur et le mot de passe pour le compte FTP sont Pasquale. L'utilisateur et le mot de passe sont complètement transparents à l'utilisateur, il n'a pas à les connaître.

L'application fonctionnant sur nos clients, n'utilise que deux fonctions à savoir celle qui permet de mettre un message sur le serveur et celle qui permet de récupérer un message de ce dernier, c'est-à-dire FTPPutFile et FTPGetFile.

13 Gestion des fichiers

Sur notre serveur FTP, nous recevons deux types de fichiers, les messages vocaux et les fichiers de commandes. La différenciation est faîte au niveau de leur nom, comme nous l'avons déjà vu les fichiers de commande portent le nom FileCmdMSISDN.cmd. Pour ces derniers leur nom est facilement gérable puisque c'est l'application client qui le définit, on peut dire à l'application client d'insérer le nom que l'on veut.

Par contre en ce qui concerne les messages, leur nom n'est pas imposable puisque c'est l'utilisateur qui le définit, il est libre de l'appeler comme il l'entend. Le seul point sur lequel nous pouvons agir est l'extension, en effet celle-ci dépend de l'application, par exemple Microsoft Word génère des *.doc alors que Adobe crée des *.pdf, etc. Dans notre cas les messages vocaux ont l'extensions *.wav puisque c'est l'extension utilisée pour les fichiers crées par le Voice Recorder Tools se trouvant sur l'Ipaq. Ainsi on sait que tous les fichiers ayant une extension *.wav représentent les messages.

En informatique on ne peut pas avoir deux fichiers portant le même nom dans le même répertoire, c'est pourtant ce qui risque d'arriver avec notre solution. Pas cependant au sujet des fichiers de commandes puisque chaque client possède son propre nom, mais au niveau des messages. Il est tout-à-fait possible que des utilisateurs envoient deux messages différents mais portant le même nom.

Puisqu'on ne peut pas avoir deux fois le même nom de fichier dans le même répertoire, le fichier ne sera pas transféré. Effectivement FTP contrairement à Windows n'écrase pas l'ancien fichier par le nouveau s'il porte le même nom, mais il ne le transfert tout simplement pas.

Le meilleur moyen pour contourner ce problème est comme pour les fichiers de commande et de s'assurer que les messages émis par des clients différents ne puissent pas avoir le même nom. Cette solution est possible grâce à une fonction offerte par FTP qui permet de transférer un message se trouvant sur le client et portant le nom X au serveur en modifiant son nom sans en changer le contenu. Dans notre cas un message, se nommant Enregistrement1.wav se nommera MSISDNEnregistrement1.wav une fois qu'il se retrouvera sur le serveur.

13.1 Comment traiter les fichiers de commandes

On reçoit les fichiers de commandes sur le serveur FTP, on les retrouve physiquement sur la machine dans le répertoire où a été installé le serveur FTP, dans notre cas sous D:\VoGprs \Commande\. Les fichiers de commande sont facilement reconnaissables puisqu'ils portent l'extension *.cmd. Notre application doit traiter le fichier de commande dès qu'il en reçoit dans le répertoire, idéalement l'application devrait être avertit à l'arrivé d'un fichier de commande. En Java il n'est pas possible «d'écouter un répertoire », d'être informé par exemple en levant une exception lors du changement d'état du répertoire.

Par contre on peut créer une petite application, qui ne fait que surveiller le répertoire D:\VoGprs\Commande\, avertir notre serveur dès l'arrivée d'un fichier de commande **et lui passer ce dernier en paramètre.**

Java permet de réaliser de tels mécanismes à l'aide de la notion de Thread. Les Thread permettent à une application de définir plusieurs unités d'exécution se déroulant en quasi simultanéité à l'intérieur de cette application.

En ce qui nous concerne, notre serveur est défini par deux Threads, un qui surveille le répertoire D:\VoGprs\Commande et qui avertit le cas échéant le second Thread qui lui a pour tâche de traiter le fichier de commande, d'accéder à la base de données et d'effacer le fichier de commande une fois celui-ci traité.

Les Threads permettent d'implémenter un mécanisme dénommé Observable, Observateur, c'est-à-dire qu'un thread Observateur peut regarder l'état d'un thread défini Observable. Dans notre situation nous définissons le thread qui surveille le répertoire comme Observable et celui qui a pour tâche de traiter le fichier de commande comme Observateur.

En résumé nos deux threads définissant notre serveur ne font qu'exécuter des programmes en boucle. Le thread Observateur est endormi tant qu'il ne reçoit pas de fichier de commande de la part du thread Observable. Dès qu'il en reçoit un, il bloque le thread Observable, il traite le fichier de commande, puis il le supprime, il se remet en état de veille et débloque le thread Observable.

Lors de l'observation du répertoire D:\VoGprs\Commande, il se peut et il est même probable qu'il y ait plusieurs fichiers de commande en attente d'être traités. Le choix du fichier de commande à traiter se fait à l'aide du mécanisme FIFO *First In First Out* , c'est-à-dire que l'on traite en premier lieu le fichier de commande le plus ancien (celui qui est arrivé en premier). Même si ces fichiers de commande proviennent de plusieurs clients différents et que ces clients ne sont certainement pas synchronisés entre eux au niveau de leur horloge, ceci ne pose pas de problèmes dans la définition du plus vieux fichier de commande se trouvant sur le répertoire, puisque les propriétés de date et heure de tous les fichiers reçus par le serveur sont automatiquement convertis à l'horloge du serveur.

14 Avertir les destinataires

Comment notre application doit avertir un client qu'il a reçu un nouveau message. Dans le cahier des charges il est clairement indiqué que cette fonction doit être réalisée à l'aide d'un SMS. Rappelons brièvement pourquoi la solution SMS a été choisie.

Puisque notre application est destinée à des équipements mobiles, l'avertissement d'un nouveau message doit leur être destinés. Ceci ne laisse la porte ouverte qu'à des solutions pouvant directement atteindre le téléphone mobile et il en existe principalement deux. La première est le SMS et la seconde est l'Email.

14.1 Email

La plupart des téléphones mobiles modernes **(pas tous)** offrent la possibilité d'envoyer et de recevoir des Emails. Ceci nous fait comprendre que l'Email n'est pas la solution a privilégier, puisque celle-ci ne nous permettra pas d'atteindre tous les clients potentiels.

14.2 SMS

L'avertissement par SMS sous GSM offre à notre avis la meilleure solution, ceci pour plusieurs raisons: Le SMS est défini par la norme GSM est donc tous les téléphones mobiles l'offre, il est déjà implémenté et fonctionne parfaitement, permet d'atteindre directement le mobile, gère tous les problèmes liés aux mobiles non accessible (éteint). Le mobile recevra le SMS lorsqu'il se reconnectera au réseau. De plus GSM prend en charge l'itinérance (roaming), cela signifie qu'un client se trouvant momentanément sur un autre réseau GSM recevra quand même le SMS qui lui est destiné.

L'emploi du SMS peut se justifier également par d'autres critères que par ceux purement techniques, il faut être conscient que le SMS remporte un très grand succès auprès des clients de téléphonie mobile et que son emploi est familier à tous les abonnées. Les clients ne seront donc pas surpris ou perturbé de recevoir un SMS.

Bien que nous réalisions une application GPRS, nous estimons qu'il est préférable que l'avertissement se fasse à l'aide de SMS sous GSM et non sous GRPS. Effectivement, bien que la norme permette d'avertir des clients GPRS depuis le réseau, cette fonction n'est pas encore disponible sur le réseau GPRS.

15 Comment envoyer des SMS

L'envoie d'un SMS peut se faire de plusieurs façons.

15.1 Par Email

L'une d'entre elle est d'utiliser l'Email, cette idée nous est venue, car Swisscom possède une passerelle permettant de réaliser une telle fonction. Pour envoyer un SMS au client ayant par exemple, le numéro 0792814219, il suffit de lui envoyer un Email contenant le texte représentant le SMS à l'adresse 0792814219@sms.swisscom.com

Malheureusement pour nous et heureusement pour Swisscom, ce service n'est accessible qu'aux utilisateurs se trouvant sur le réseau Intranet de Swisscom, pour des raisons évidentes de sécurité. Il est bien évident que notre application n'aura jamais accès à ce service.

Surtout lorsque l'on sait que dans sa version finale, notre serveur devra être connecté au réseau Gprs. Ceci implique que notre application relirait deux réseaux entre-eux, d'un côté le réseau Gprs et de l'autre le réseau Intranet. Il est indéniable que de mettre en relation deux réseaux d'une telle importance et devant répondre à des niveaux de sécurité élevés est plus que suicidaire.

On peut imaginer réaliser le même service par nos propres moyens, en créant et en mettant en place toute l'infrastructure permettant d'envoyer un SMS à l'aide d'un Email en y insérant l'adresse 0792814219@sms.vogprs.ch par exemple. Voyons les ressources qu'une telle solution demande.

Il faut créer un Email, le convertir en SMS et l'envoyer, on constate déjà facilement que cette solution ne nous simplifie pas la tâche, au contraire elle nous la complique, puisque nous ne voulons que créer un SMS et l'envoyer. Cette solution nous oblige à convertir un Email en SMS, étape qui dans l'opération que nous voulons créer nous est superflue. De plus l'Email demande l'utilisation d'un serveur d'Email, donc un rajout de logiciel et évidemment un surplus de ressources pour sa mise en place et la gestion qui est inévitable.

15.2 Par téléphone mobile

Une autre solution qui ne demande pas de grandes ressources et celle d'utiliser un module GSM que l'on connecte par un simple câble sur l'un des ports de communication de notre serveur. Swisscom mobile, a mis à notre disposition un téléphone mobile, l'Ericsson R520m dédié à cette effet.

Le R520m supporte les commandes AT, il peut donc être utilisé en mode modem. En recherchant dans la documentation du téléphone, on s'aperçoit qu'il existe une commande permettant d'envoyer des SMS. La commande est AT+CMGS. Il suffit par conséquent d'envoyer cette commande avec les paramètres indiquant le contenu du SMS, le destinataire sur le port série et le SMS sera envoyé. Telle fut notre réaction à la découverte de cette commande, mais l'envoie d'un SMS est bien plus complexe.

Il existe deux types de SMS, le type **Text** et le type **PDU**, l'un utilise un codage contrairement à l'autre. Le type **Text,** comme son nom l'indique permet, d'envoyer des SMS sans codage, il

suffit d'exécuter quelques commandes AT, pour réaliser l'envoi d'un SMS. Ces quelques commandes sont décrites en figure 10.

Figure 10 *Commande AT réalisant l'envoi d'un SMS au format Text*

AT+CMGF=1	définit le format du SMS, ici format Text
AT+CMGS=+41792814219	indique le destinataire du SMS
>Taper le SMS a envoyer et finir par un CtrlZ	définit le SMS à envoyer.

En mode Text, l'envoi d'un SMS est une chose simple lorsque. Malheureusement notre Ericsson R520m ne supporte que le mode PDU, ce qui va compliquer la tâche.

15.3 La trame PDU

Le format d'une trame PDU diffère lorsque il s'agit de l'envoi d'un SMS ou de sa réception.

Pour l'envoi d'un SMS la trame PDU à le format montré en figure 11.

Figure 11 *Format de la trame pour l'envoi d'un SMS*

0011000A8170291824910000AA04D4E2940A []

Décortiquons la trame, et commençons par la gauche

00	Taille des informations du SMSC. 00 indique que celles stockées dans le téléphone sont utilisées.
11	Indique le première octet du SMS envoyé.
00	TP référence du message. 00 indique que c'est le téléphone lui-même à définir la référence.
0A	Définit la taille du numéro de téléphone. Ici la taille est de 10.
81	Définit le format international du numéro de téléphone.
7029182491	Indique le numéro de téléphone du destinataire, pour le décodage voir figure 12.
00	Identificateur du protocole TP-ID.
00	Définit le type de codage des données définissant le SMS.
AA	Définit la période de validité du SMS.
04	Indique la taille des données en octets
D4E2940A	Représente le message contenu dans le SMS

	codées sur 7 bit, voir 13.
[]	Indique le Ctrlz de fin de message.

Le numéro de téléphone du destinataire est codé. Son codage est assez simple, puisque il ne s'agit que d'inverser les chiffres, comme l'illustre la figure 12.

Figure 12 *Décodage des numéros de téléphone dans la trame PDU*

7	0	2	9	1	8	2	4	9	1
0	7	9	2	8	1	4	2	1	9

Quant au codage des données définissant le SMS il consiste à convertir les lettres qui sont définies en code Ascii sur 7 bits en octets (8 bits).

L'exemple montré en figure 13 code le texte TEST au format SMS. La première chose à faire est de coder chaque lettre du mot TEST en code Ascii sur sept bits.

Figure 13 *Codage des données représentant le contenu du SMS au format PDU*

T = 1010100 E = 1000101 S = 1010011 T = 1010100

Ensuite on met à la suite les codes Ascii de chaque lettre et ceci nous donne :

T	S	E	T
1010100	1010011	1000101	1010100

On regroupe ensuite les bits par paquet de huit, en commençant depuis la droite, on complète par des 0, ceci nous donne :

0A	94	E2	D4
00001010	10010100	11100010	11010100

Pour l'envoi d'un SMS au format PDU, il suffit d'exécuter la commandes AT montrées en figure 14.

Figure 14 *Commande AT réalisant l'envoi d'un SMS au format PDU*

AT+CMGS=17	Définit la taille de la trame
>0011000A8170291824910000AA04D4E2940A[]	Définit la trame PDU

Nous avons réalisé un logiciel qui permet d'envoyer des SMS à n'importe quel utilisateur, par contre le message du SMS qui est "**you have got a new VoGprs mail**", n'est pas modifiable il est codé en dur dans l'application. Bien que nous soyons conscients que cette solution n'est pas optimale, nous l'avons tout de même adopté, et ceci pour plusieurs raisons : Elle répond au besoins de l'application et ne pénalise nullement les clients. Si pour des raisons quelconques, le contenu du SMS doit être modifier et bien notre application pourra facilement s'adapter à cette réalité, puisqu'il suffira de changer le téléphone mobile par un appareil qui supporte le mode Text.

De plus il est bien évident que la solution qui nécessite un téléphone mobile pour générer des SMS n'est utilisée que dans la phase de développement, puisque lorsque notre serveur sera relié au réseau GSM/Gprs nous aurons à disposition le SMSC *SMS Center* pour envoyer les SMS. En sachant cela, il était inutile à nos yeux d'améliorer notre solution en lui permettant de définir le contenu du SMS puisque son emploi est limitée dans le temps et qu'elle n'apporte pas de réels avantages.

Il faut savoir que le codage des SMS pour le SMSC répondent aux normes UCP et SMPP, il est donc différent du format PDU utilisé dans l'envoi depuis un téléphone mobile. Ce qui implique que le logiciel développé pour notre application ne sera pas utilisable par la suite. On peut facilement pallier à cette inconvénient, puisque il existe un logiciel permettant de créer des SMS au format du SMSC. Celui-ci a été développé par Mr Christian Wolf et Antoine Maître, lors d'un travail de diplôme à l'EIVD en l'année 2000.

16 Configuration du service

Pour pouvoir utiliser notre application, il faut que le client s'identifie, c'est-à-dire qu'il spécifie un moyen qui permette de le reconnaître de façon univoque. La manière la plus simple est d'utiliser le numéro de téléphone du mobile, le MSISDN qui lui est unique pour tous les réseaux GSM. Cette identification est obligatoire car il faut s'assurer que le client qui consulte ou qui envoie des messages le fassent en indiquant son identité et non pas celle d'un autre utilisateur.

16.1 Identification du client

La meilleure façon de s'assurer que la configuration de l'Ipaq soit correcte est qu'elle ne soit pas élaborée par l'utilisateur mais par le réseaux GSM/Gprs lui-même.

Idéalement, l'Ipaq devrait envoyer un SMS de configuration au serveur et celui-ci lui répondre en configurant l'Ipaq. De cette façon la configuration est faite indépendamment de l'utilisateur et on assure une correcte configuration. Dans notre situation, un tel niveau d'optimisation n'est pas possible car nous avons deux modules indépendant qui constitue le client, nous devons donc insérer une communication entre-eux.

En pratique cela se traduit par la réalisation des opérations suivantes: L'utilisateur entre le MSISDN dans l'Ipaq, et demande au mobile d'envoyer un SMS contenant le MSISDN inséré au serveur, lorsque celui-ci reçoit le SMS, il vérifie que son contenu (le MSISDN) corresponde à l'émetteur du SMS qu'il a reçu. Si tel est le cas, l'utilisateur a entré son numéro de téléphone. A ce moment, le serveur renvoie un SMS indiquant que la configuration est acceptée, l'Ipaq peut par conséquent mémoriser le numéro de téléphone et réaliser la configuration.

Ce scénario bien que simple demande beaucoup de ressources logiciels pour le mettre en œuvre, la première est de permettre à l'Ipaq de coder et décoder les trames PDU (on ne peut pas utiliser le code déjà écrit, car on n'utilise pas le langage Java), la seconde est de permettre à l'Ipaq d'envoyer des SMS en utilisant le téléphone mobile et d'aller lire les SMS reçus par ce dernier.

On constate que l'utilisation de l'Ipaq pour configurer le service rend cette tâche lourde à réaliser, nous avons donc décidé de ne pas l'utiliser du moins dans la phase de génération du SMS de configuration. Ce choix nous permet de ne plus nous préoccuper du codage et de l'envoi du SMS puisque le téléphone mobile s'en charge. Cette modification nous oblige à changer notre démarche de configuration du service.

Le nouveau scénario demande de suivre les opérations suivantes: Le client envoie un SMS contenant le message suivant : «configure 1234 » ou configure indique que c'est un SMS de configuration et 1234 représente le mot de passe défini par l'utilisateur, lorsque le serveur reçoit un tel SMS il récupère alors le numéro de téléphone du destinataire et crée un fichier portant le nom MSISDNPassword.txt contenant le MSISDN et le mot de passe dans le répertoire Configuration sur le serveur FTP. Imaginons que l'utilisateur envoyant le SMS ait le numéro de téléphone 0792814219, on retrouverait sur le serveur FTP dans le répertoire /Configuation/ un message de nom 07928142191234.txt contenant le MSISDN du client et son mot de passe.

Une fois le SMS envoyé, l'utilisateur entre dans son Ipaq, son MSISDN et son mot de passe. Là il effectue une requête FTP dans le but d'aller rechercher son fichier de configuration, s'il existe il le télécharge et l'Ipaq est configuré. Si il n'existe pas c'est que l'utilisateur n'entre pas des valeurs correctes dans son Ipaq et il n'est donc pas configuré.

Seul un Ipaq configuré permet d'utiliser le service, d'envoyer ou de consulter des messages.

16.2 Décodage du SMS

Comme nous l'avons déjà vu, il existe deux types de formats pour les SMS : Le format Text et PDU. La lecture des SMS reçu par le serveur se fait sur l'Ericsson R520m qui est connecté sur le port série de notre serveur. Ce téléphone ne supporte que le mode PDU, il faut donc décoder la trame. Comme pour l'envoi de SMS, la réception se fait à l'aide des commandes AT, les quelques commandes décrites en figure 15 permettent de lire les SMS.

Figure 15 *Commande AT réalisant la lecture d'un SMS*

AT+CPMS="me"	Indique qu'il faut lire sur le téléphone et non sur la carte SIM
AT+CMGL= 0	Indique que l'on ne veut lire que les nouveaux SMS

La figure 16 représente la trame d'un SMS reçu

Figure 16 *Format de la trame pour un SMS reçu*

07911497949900F0040B911497160973F0000010117201801000EC3A7D3983C56A545100C068301

Décortiquons la trame, en commençant par la gauche

07	Taille des informations du SMSC.
91	Définit le format du numéro de téléphone du SMSC.
1497949900F0	Indique le numéro de téléphone du SMSC. Le numéro de téléphone est 41794999000. Pour le décodage voir figure 12.
04	Indique le première octet du SMS reçu.
0B	Définit la taille du numéro de téléphone du SMSC. Ici la taille est de 11.
91	Définit le format du numéro de téléphone de l'expéditeur.
1497160973F0	Indique le numéro de téléphone de l'expéditeur. Le numéro de téléphone est 41796190370. Pour le décodage voir figure 12.
00	Identificateur du protocole TP-ID.
00	Définit le type de codage des données

	définissant le SMS.
10117201801000	Définit le timbre horaire.
0E	Indique la taille des données en octets.
C3A7D3983C56A545100C068301	Représente le message contenu dans le SMS codées sur 7 bit, voir figure 13.

Dans cette situation, nous sommes obligés de décoder le contenu du SMS car il varie en fonction de l'utilisateur. Nous avons réalisé un logiciel qui permette de récupérer le numéro de téléphone de l'utilisateur et les données qu'il a envoyées. Pour les décoder, il s'agit uniquement du contenu du SMS, nous utilisons un logiciel qui a été développé par notre collègue Pierre Daccord, réalisant également un travail de diplôme pour Swisscom Mobile.

Le logiciel que Pierre nous a gentiment mis à disposition ne permet que de décoder le contenu du SMS, tout le reste c'est à dire définir les données représentant le SMS ainsi que l'expéditeur sont réalisée par nos soins.

17 Les fonctions du client

Lorsque nous parlons des fonctions du client, nous nous référons aux opérations que l'Ipaq doit permettre de réaliser. Ces opérations sont principalement aux nombre de sept et sont les suivantes :

- configurer le service
- l'envoi d'un message
- consultation des nouveaux messages
- la réception d'un message
- le transfert d'un message
- l'effacement d'un message
- l'écoute d'un message

Ces sept fonctions ne sont évidemment pas exhaustives, elles peuvent être complétées par d'autres fonctions que les exploitants du service jugent utiles. En ce qui nous concerne et dans le cadre de notre développement, nous n'avons réalisé que ces fonctions-ci. Il faut souligner que pour tout ce qui concerne les fonctions relatives au message, elles ne peuvent traiter qu'un message et un utilisateur à la fois. C'est-à-dire que l'envoi, la réception et le transfère de plusieurs messages à plusieurs destinataires n'est pas possible, tout comme la réception de plusieurs messages à la fois. Nous sommes bien conscients qu'une telle situation est un frein à un éventuel déploiement commercial de ce service. Cette solution a tout de même été privilégiée dans le cadre du développement principalement pour une question de temps, nous avons préféré nous concentrer à réaliser les fonctions de bases de sorte à ce qu'elles puissent être facilement reprises dans le développement de nouvelles fonctions.

17.1 Configurer le service

Comme mentionné, la configuration du service se fait à l'aide de l'envoi d'un SMS au serveur de la part de l'utilisateur. Si le serveur admet que le SMS est bien un SMS de configuration, il créer le fichier correspondant dans répertoire /Configuration/ et met à jour la table Utilisateurs de la base de données.

Ici nous allons nous intéresser à ce qu'implique cette phase de configuration pour l'Ipaq. Imaginons que le SMS ait déjà été envoyé et que le fichier de configuration correspondant ait déjà été crée par notre serveur dans le répertoire /Configuration/.

L'utilisateur entre ces données, (numéro de téléphone et mot de passe) dans l'Ipaq, en pressant sur le bouton OK, il effectue une commande FTP demandant d'aller chercher dans le répertoire /Configuration/ son fichier de configuration. S'il existe il le charge sur l'Ipaq en lui donnant le nom mySettingFile et l'Ipaq est configuré, s'il ne le trouve pas, il indique que les informations qu'il a entré sont fausses et la configuration n'est pas réalisée.

A chaque démarrage de l'application sur l'Ipaq, la première chose qu'elle réalise est de tester si le fichier mySettingFile existe (se trouve sur l'Ipaq), si oui elle utilise les informations qu'il contient afin de configurer l'Ipaq, si le fichier n'existe pas l'application demande à l'utilisateur de configurer l'application. Tant que celle-ci n'est pas configurée l'exploitation du service n'est pas possible puisque l'Ipaq est inutilisable.

L'utilisation d'un fichier permet de garder la configuration intact même lorsque l'Ipaq n'est plus alimenté. Les fichiers contrairement aux données contenues dans la mémoire Ram ne sont pas volatils.

La figure 17 ci-dessous montre l'interface graphique de l'Ipaq réalisant la configuration de celui-ci.

Figure 17 *Configuration du client*

Dans le but d'éviter des erreurs de la part de l'utilisateur, nous avons programmé le champ permettant d'insérer le numéro de téléphone pour qu'il n'accepte que dix chiffres et commençant par 07 alors que les champs concernant le mot de passe n'acceptent que quatre chiffres.

17.2 L'envoi d'un message

Pour envoyer un message depuis l'Ipaq, il faut évidemment qu'il existe. La création d'un message se fait à l'aide Voice Recorder Tools. Il faut donc d'abord créer le message et ensuite lancer l'application. Notre application elle-même ne fournit pas d'outils pour la création de messages vocaux.

L'envoi d'un message vocal nécessite en fait l'envoi de deux messages, l'un étant le message vocal lui-même et l'autre étant le fichier de commande informant le serveur des opérations qu'il doit réaliser. Il renseigne également sur quel message est envoyé, qui en est le destinataire et qui en est l'émetteur. Toutes ces informations sont indispensables pour assurer une bonne gestion des messages.

L'envoi d'un message nécessite que l'utilisateur définisse le message qu'il souhaite envoyer, ce choix se fait à l'aide d'une liste déroulante. Cette liste contient tous les messages vocaux se trouvant sur le répertoire /MyDocuments/ de l'Ipaq. Le choix de se répertoire n'est pas innocent, en effet c'est le répertoire où sont stockés tous les messages crées par le Voice

Recorder Tools. La liste déroulante affiche donc tous les fichiers portant l'extension *.wav correspondant aux fichiers vocaux.

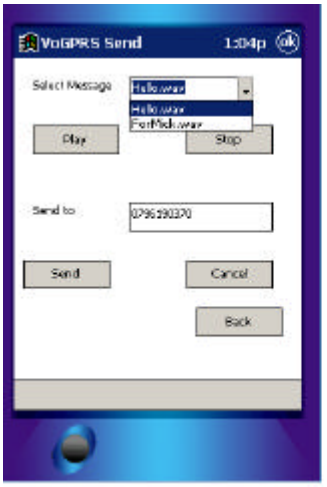
Une fois le message choisi, il faut définir le destinataire. Ceci se fait en insérant son numéro de téléphone. Dans le but d'éviter des erreurs, en effet nous n'avons aucun moyen de savoir si le numéro de téléphone entré est valide ou pas puisque nous n'avons aucun accès au ressources du réseau particulièrement le HLR, nous avons programmé le champ permettant d'insérer le numéro de téléphone du destinataire pour qu'il n'accepte que des dix chiffres et commençant par 07.

Lorsque toutes les informations consentant d'envoyer le message ont été fournies, on presse sur le bouton Send. Ceci permet de créer un fichier de commande portant le nom FileCmdMSISDN, ce fichier de commande contient toutes les informations nécessaires pour l'envoi du message, une fois le fichier de commande crée, l'application envoie le message vocale et le fichier de commande au serveur FTP à l'aide des commande FTP.

Toutes les données concernant l'émetteur sont fournies par le fichier de configuration mySettingFile contenu dans l'Ipaq.

La figure 18 montre l'interface permettant d'envoyer des messages vocaux.

Figure 18 *Envoi des messages vocaux*



17.3 consultation des nouveaux messages

Avant de pouvoir télécharger un message qui nous est destiné ou de pouvoir le transférer à une tierce personne, nous devons connaître les messages qui nous ont été adressés et se trouvant sur le serveur.

Ce point est réalisé en grande partie par le serveur. Chaque fois que celui-ci reçoit un message, il met à jour un fichier de résultat pour chaque destinataire. Ce fichier porte le nom "MSISDN.res" et contient la liste de tous les messages qu'un utilisateur a reçus avec les

différentes informations relatives aux messages : L'émetteur et la date de réception. Le serveur stocke ce fichier dans le répertoire /Resultat/.

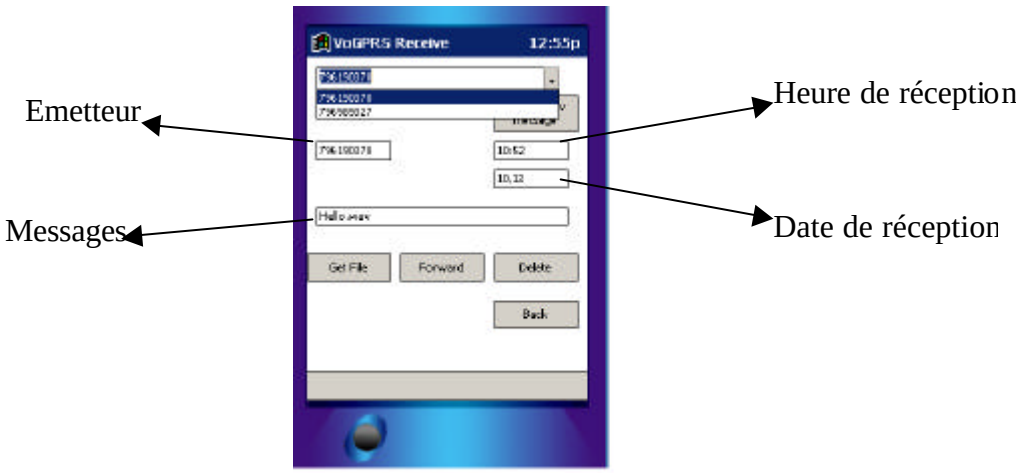
Le choix de créer un fichier de résultat s'est imposée car elle offre la possibilité aux différents utilisateurs d'avoir connaissance des messages qui leurs sont destinés avant de les télécharger physiquement sur leur Ipaq et ainsi pouvoir choisir s'ils désirent en prendre connaissance ou non. Cette solution permet également de transférer un message à une tierce personne sans l'avoir téléchargé au préalable.

La consultation des messages reçus se fait généralement à la suite de la réception du SMS indiquant que l'utilisateur à reçu un nouveau message. Sur l'Ipaq cette consultation se fait en pressant sur le bouton **Check New Messages**. Ceci lance une commande FTP demandant d'aller récupérer le fichier correspondant au numéro de téléphone de l'utilisateur, MSISDN.res se trouvant sur le répertoire /Resultat/ du serveur.

Une fois ce fichier téléchargé, l'application l'ouvre et met à disposition de l'utilisateur les informations relatives aux différents messages dans des boîtes de dialogues, elles définissent le nom du message, l'émetteur et la date de réception. Avec toutes ces informations l'utilisateur peut décider s'il veut télécharger le message, le transférer à quelqu'un d'autre ou l'effacer du serveur.

La figure 19 montre l'interface permettant de télécharger le fichier de résultat.

Figure 19 *Consultation des nouveaux messages reçus*



17.4 La réception d'un message

Une fois avoir pris connaissance des différents messages qui nous sont destinés, nous pouvons décider si nous voulons les télécharger sur notre Ipaq pour les écouter ou non.

Cette opération se traduit sur l'Ipaq en pressant sur le bouton **Get File**, cela lance une requête FTP qui va aller chercher le fichier mentionné dans la boîte de dialogue **Hello.wav** sur la figure ci-dessus, et se trouvant sur le serveur dans le répertoire /Messages/. Le message est

téléchargé sur l'Ipaq dans le répertoire /MyDocuments/. Nous avons programmé notre Ipaq de sorte que lorsque le téléchargement du message est fini, l'application joue directement le message.

17.5 Le transfert d'un message

Comme pour la réception d'un message, le transfert ne peut s'effectuer que lorsque l'utilisateur a pris connaissances des nouveaux messages qui lui sont destinés à l'aide du fichier de résultat se trouvant sur le serveur.

Le transfert d'un message vocal à une tierce personne, n'implique pas forcément l'envoi de deux fichiers comme c'était le cas lors de l'envoi d'un message mais ne génère que l'envoi d'un seul fichier. Dans ce cas, uniquement le fichier de commande est envoyé, puisque le fichier représentant le message se trouve déjà sur le serveur, on n'a donc pas besoin de le renvoyer.

Le transfert d'un message nécessite que l'utilisateur définisse le message qu'il souhaite envoyer, ce choix se fait à l'aide du fichier de résultat. Une fois le message choisi, il clique sur le bouton **Forward,** là il faut définir le destinataire, en insérant son numéro de téléphone.

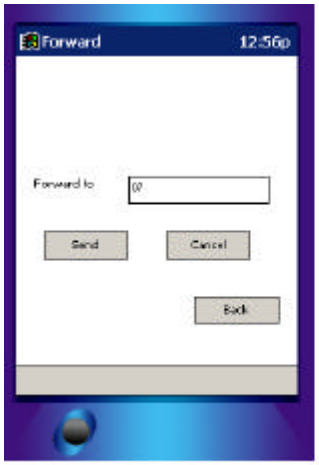
Dans le but d'éviter des erreurs, comme c'était le cas pour l'envoi d'un message, nous avons programmé le champ permettant d'insérer le numéro de téléphone du destinataire pour qu'il n'accepte que des dix chiffres et commençant par 07.

Lorsque toutes les informations permettant le transfert du message ont été fournies, on presse sur le bouton **Send**, afin de créer un fichier de commande portant le nom FileCmdMSISDN, ce fichier de commande contient toutes les informations nécessaires pour le transfert du message, une fois le fichier de commande créer, l'application l'envoie à l'aide des commande FTP au serveur FTP à sa racine .

Toutes les données concernant l'émetteur sont fournies par le fichier de configuration mySettingFile contenu dans l'Ipaq.

La figure 20 montre l'interface permettant de transférer les messages vocaux à un tierce.

Figure 20 *Transfert d'un message vocal à un tierce utilisateur*



17.6 L'effacement d'un message

Lorsqu'on parle d'effacement d'un message, il s'agit évidemment de pouvoir détruire les messages qui nous sont destinés se trouvant sur le serveur. Ce qui était vrai pour la réception et le transfert d'un message, l'est également pour l'effacement d'un message, c'est-à-dire qu'il faut connaître les messages que l'on est susceptible de vouloir effacer et qui se trouvent sur le serveur. Pour réaliser cela on utilise le fichier de résultat.

L'effacement d'un message ne se traduit pas forcément par l'effacement physique du message sur le serveur, en effet il se peut que plusieurs utilisateurs ait reçu le même message, grâce notamment à la fonction de transfert. Ceci ne permet donc pas à un utilisateur d'effacer lui même le message. Tout ce que l'utilisateur peut faire c'est d'effacer de la base de données les informations lui appropriant le message. L'effacement physique du message est gérée par le serveur, voir chapitre Les fonctions du serveur.

L'opération d'effacement demande de créer un fichier de commande contenant toutes les informations qui définissent le tuple dans la base de données, principalement dans la table Communications. Ces informations sont l'émetteur, le destinataire et la date de réception.

Cette opération se traduit sur l'Ipaq en pressant sur le bouton **Delete**, ceci sert à créer un fichier de commande portant le nom FileCmdMSISDN, ce fichier de commande contient toutes les informations nécessaires pour l'effacement du message, une fois le fichier de commande créé, l'application l'envoie à l'aide des commande FTP au serveur FTP à sa racine .

Toutes les données relatives au message sont données par le fichier de résultat.

17.7 Ecouter le message

Cette fonction permet de jouer les messages vocaux afin que l'utilisateur puissent les écouter. Il existe deux mode pour l'écoute du son, le synchrone et l'asynchrone. Nous utilisons le mode asynchrone car il permet de stopper le son. Les sons joués en mode synchrone ne peuvent être interrompus par aucune application.

18 Les fonctions du serveur

Le serveur doit répondre aux demandes émissent par les différents clients. Comme nous l'avons déjà vu, ces demandes se réalisent à l'aide de l'envoi d'un fichier de commande et regroupent trois types d'opérations, qui sont :

- l'envoi d'un nouveau message
- le transfert d'un message
- l'effacement d'un message

Le serveur exécute les demandes des clients en traitant les fichiers de commande.
Nous décrivons à présent toutes les opérations que le serveur réalise pour répondre aux trois types d'opérations définis par le fichier de commandes. L'allure des trois types de fichier de commandes est donnée en figure 8. Rappelons que les fichiers de commandes et les messages vocaux envoyés par les clients à l'aide de FTP sont reçu dans le répertoire Commande du serveur FTP.

Le traitement des fichiers de commandes à une partie commune aux trois types d'opérations qui est évidemment l'identification de la commande contenu dans le fichier. Celle-ci se réalise en lisant la première ligne, si elle contient **Insert** cela indique un envoi, si elle contient **Forward** cela indique un transfert et si elle contient **Delete**, cela indique évidemment un effacement.

18.1 Envoi d'un nouveau message

L'envoi d'un nouveau message, demande au serveur d'insérer dans la base de données toutes les informations relatives au message, à l'émetteur et au destinataire. De plus lorsque le serveur reçoit un message à l'intention d'un utilisateur, il doit lui envoyer un SMS l'informant de la présence d'un nouveau message lui étant destiné.

Voyons en détail comment le serveur réalisent ces opérations, dans le cas ou la première ligne du fichier de commande est Insert.

1. Le serveur récupère les informations contenues dans le fichier de commande, c'est à dire l'émetteur, le destinataire, le nom du message
2. Le message vocal se trouvant sur le serveur dont le nom à été récupéré par le serveur est renommé et déplacé dans le répertoire /Messages/. Puisque les messages émis par les clients peuvent porter n'importe quel nom, il est fort probable qu'on retrouve deux messages différents mais ayant le même nom. Une telle situation posent le problème que le message le plus ancien se fait écraser par le plus récent. Dans le but d'éviter cela, il est obligatoire de renommer le message de façon à ce qu'il ne puissent exister deux messages portant le même nom. Ceci est réaliser en ajoutant au nom du message, la date et l'heure de réception fournit par le serveur. Le message portent désormais le nom Date&HeureMessage.wav. Le serveur définit également la taille du message.
3. Connexion et insertion dans la base de données des informations nécessaires pour indiquer l'envoi d'un nouveau message. Les contraintes d'intégrités définies par notre base de données Access imposent qu'un utilisateur ne se trouvant pas dans la table Utilisateur ne puisse pas apparaître dans la table Communication. Ceci nous impose que avant d'insérer les données dans la table Communication, il faut s'assurer que l'Utilisateur existe dans la table Utilisateur, si ce n'est pas le cas il faut l'insérer. Ceci permet qu'un Utilisateur ne

s'étant pas encore signaler au serveur par la procédure de configuration puisse tout de même recevoir des messages. Une fois l'utilisateur présent dans la table Utilisateur, il faut insérer les données relatives à la table Messages et enfin les données relatives à la table Communication.
4. Le fichier de résultat du destinataire est mis à jour en insérant les informations relatives à la table Communication. Le fichier de résultat se trouve dans le répertoire /Resultat/.
5. Le fichier de commande est effacé.
6. Retrouve dans la base de données, le numéro de téléphone du destinataire pour lui envoyer le SMS indiquant la réception d'un nouveau message, réalise l'envoie et met à jour le champ SMS envoyé de la table Communication.
7. Déconnexion de la base de données.

18.2 Transfert d'un message

Le traitement effectué par le serveur dans le cas d'un transfert de message est très similaire à celui réalisé lors de l'envoi d'un nouveau message. Celui consiste à réaliser les points suivants :

1. Le serveur récupère les informations contenus dans le fichier de commande, c'est à dire l'émetteur, le destinataire, **le numéro de message.**
2. Connexion et insertion dans la base de données des informations nécessaires pour indiquer le transfert d'un message. Les contraintes d'intégrité définies par notre base de données Access imposent qu'un utilisateur ne se trouvant pas dans la table Utilisateur ne puisse pas apparaître dans la table Communication. Ceci nous impose que avant d'insérer les données dans la table Communication, il faut s'assurer que l'Utilisateur existe dans la table Utilisateur, si ce n'est pas le cas il faut l'insérer. Ceci permet qu'un Utilisateur n'étant pas encore signaler au serveur par la procédure de configuration puisse tout de même recevoir des messages. Une fois l'utilisateur présent dans la table Utilisateur, il faut insérer les données relatives à la table Messages et enfin les données relatives à la table Communication.
3. Le fichier de résultat du destinataire est mis à jour en insérant les informations relatives à la table Communication. Le fichier de résultat se trouve dans le répertoire /Resultat/.
4. Le fichier de commande est effacé.
5. Retrouve dans la base de données, le numéro de téléphone du destinataire pour lui envoyer le SMS indiquant la réception d'un nouveau message, réalise l'envoie et met à jour le champ SMS envoyé de la table Communication.
6. Déconnexion de la base de données.

18.3 Effacement d'un message

La suppression d'un message, demande au serveur d'effacer des informations dans la base de données et ci nécessaire le message lui se trouvant dans le répertoire /Messages/.

Le serveur réalise cette opération de la manière suivante :

1. Le serveur récupère les informations contenus dans le fichier de commande, c'est à dire l'émetteur, le destinataire, le numéro de message et les informations relatives à la date et l'heure de réception du message par le serveur. Toutes ces informations sont indispensables pour effacer le bon tuple de la table Communication.
2. Connexion et effacement dans la base de données des informations relatives au message appartenant à un utilisateur. La suppression dans la table Communication demande de

sélectionner le tuple définit par le fichier de commande, une fois celui-ci trouvé on le supprime. Afin de savoir si l'on peut effacer physiquement le message sur le serveur, il faut savoir si ce message appartient également à un autre utilisateur. Si ce n'est pas le cas, on peut effacer les données relatives au message contenues dans la table Messages et physiquement le message se trouvant sur le serveur dans le répertoire /Messages/. Si le message appartient également à un ou plusieurs autres utilisateurs, le message ainsi que les informations le concernant contenues dans la table Messages ne sont pas modifiées.

3. Déconnexion de la base de données.
4. Le fichier de commande est effacé.

Le traitement des fichiers de commandes de la part du serveur demande outre la mise à jour d'une base de données et la gestion de fichiers de pouvoir traiter des SMS. Le serveur doit pouvoir envoyer un SMS à un utilisateur lui indiquant la réception d'un message et doit être capable de traiter les SMS de configurations lui étant envoyés par les clients.

18.4 Envoi du SMS par le serveur

Dans le cadre des opérations effectuées lors du traitement d'un fichier de commande indiquant l'envoi d'un nouveau message ou le transfert d'un message, l'une d'entre elle est d'envoyer un SMS au destinataire ayant reçu un message.

Cette section ne traite pas de la façon de créer un SMS, ce point à déjà été traité au paragraphe 15.2 page 29, mais indique les opérations réalisées par le serveur pour envoyer le SMS à l'aide du téléphone mobile Ericsson R520m. Ces opérations sont les suivantes:

1. Création de la trame PDU représentant le SMS à envoyer à l'utilisateur ayant reçu un nouveau message vocale.
2. Connexion sur le port série Com2, c'est le port sur lequel est connecté le téléphone mobile.
3. Envoi du SMS à l'aide des commandes AT, voir figure 14.
4. Déconnexion du port série Com2.

18.5 Réception du SMS de configuration par le serveur

Le serveur doit pouvoir recevoir et traiter des SMS de configuration qui lui sont envoyés par les différents clients. Cette fonction est nécessaire afin d'identifier de manière sure les clients. Pour plus d'informations à ce sujet voir chapitre 16 page 33. La réception d'un SMS de configuration demande au serveur de mettre à jour la table Utilisateur de la base de données.

Comme c'est le cas pour le paragraphe *Envoi du SMS par le serveur*, cette section ne traite pas de la façon par laquelle le SMS est décodé, ce point à déjà été traité au paragraphe 16.2 page 34 , mais indique les opérations réalisées par le serveur afin de lire et traiter les SMS reçu à l'aide du téléphone mobile Ericsson R520m. Ces opérations sont les suivantes.

1. Connexion sur le port série Com2, c'est le port sur lequel est connecté le téléphone mobile.
2. Lecture des nouveaux SMS à l'aide des commandes AT, voir figure 15.
3. Si le serveur n'a pas de reçu de SMS, on va au point 7, par contre si il à reçu un SMS, on vérifie qu'il s'agit bien d'un SMS de configuration, pour cela on test si le SMS commence

par le mot **Configure**, si c'est le cas on récupère le mot de passe contenu dans le SMS et le numéro de téléphone de l'émetteur du SMS.

4. Connexion à la base de données et test si l'utilisateur existe déjà. Si ce n'est pas le cas insère dans la table Utilisateur le numéro de téléphone et le mot de passe, si l'utilisateur est déjà présent, on vérifie si le mot de passe est le même que celui contenu dans le SMS, si c'est le cas on ne fait rien l'utilisateur est déjà configuré si ce n'est pas le cas, cela indique que l'utilisateur à déjà reçu des messages mais il ne s'est pas encore enregistré auprès du serveur, on ne modifie donc que son mot de passe.
5. Effacer le SMS du téléphone mobile, ceci afin de assurer qu'il reste toujours de l'espace disponible pour la réception d'autres SMS.
6. Déconnexion de la base de données.
7. Déconnexion du port série Com2.

Afin d'être continuellement informé de l'arrivé d'un nouvel SMS, nous devons réaliser cette opération de lecture à intervalle régulier. Nous avons donc défini un Thread qui test la présence d'un SMS et le traite le cas échéant.

19 Améliorations

A nos yeux et en toute sincérité, notre application possède deux points faibles qui doivent être améliorés. Ces points n'ont pas été améliorés pour une question de temps

Le premier est que l'utilisateur ne puisse envoyer ou transférer pas plus d'un message à un utilisateur à la fois. Il faut que notre application mette à disposition des clients la possibilité de réaliser des envois ou des transferts multiples à différents utilisateurs.

Le second est l'esthétique de l'interface graphique du client. La réalisation d'une interface graphique qui soit ergonomique, intuitive et de bel aspect est extrêmement complexe à réaliser. Celle-ci s'obtient après plusieurs versions demandant un dialogue continu entre les utilisateurs et les concepteurs. Dans notre cas, nous en sommes à la première version. A notre décharge nous tenons à dire que l'esthétique n'est pas notre plus grande qualité.

20 Installation de l'application

20.1 Installation du serveur

Le fonctionnement de notre application demande que certain logiciels soient installés sur le serveur. Ces logiciels sont une base de données Access et un serveur FTP plus bien évidemment un système d'exploitation. La machine sur laquelle tourne notre application est un pentium cadencé à 166Mhz avec 100Mo de Ram et fonctionnant sous Windows NT.

20.1.1 Installer le JDK

Le JDK *Java Developpement Kit* se trouve sur le CD dans le répertoire JDK, pour l'installer lancer simplement l'exécutable et suivez les instructions. Une fois celui-ci installé il faut mettre les classpaths:

- C:\JDK1.3
- C :\JDK1.3\lib
- Et un sur la fichier comm.jar

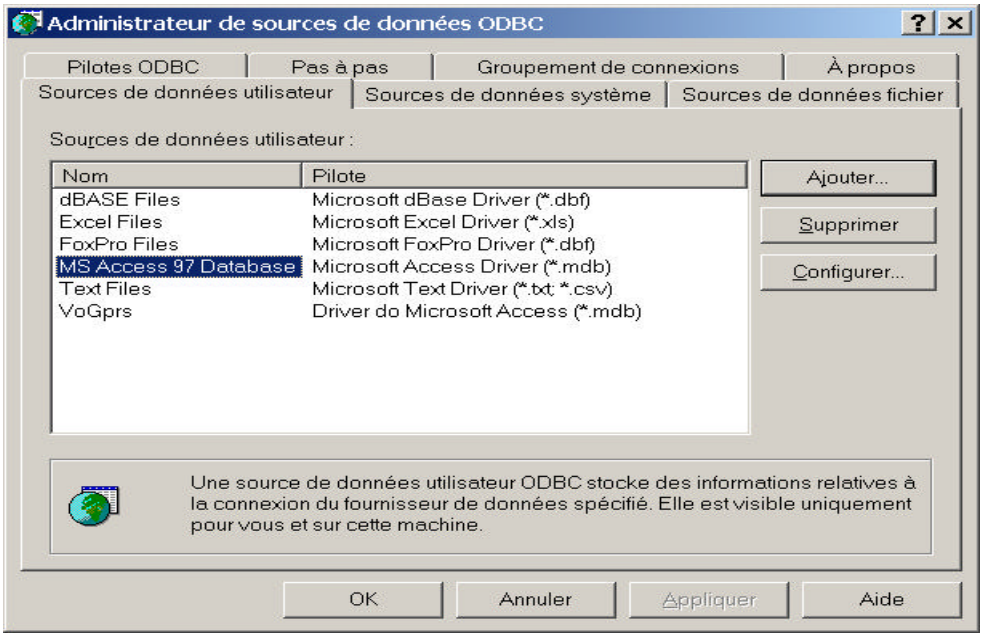
Il faut mettre les fichiers constituant l'application sur le serveur, ces fichiers se trouve sur le CD dans le répertoire Logiciels application\Serveur.

20.1.2 La base de données

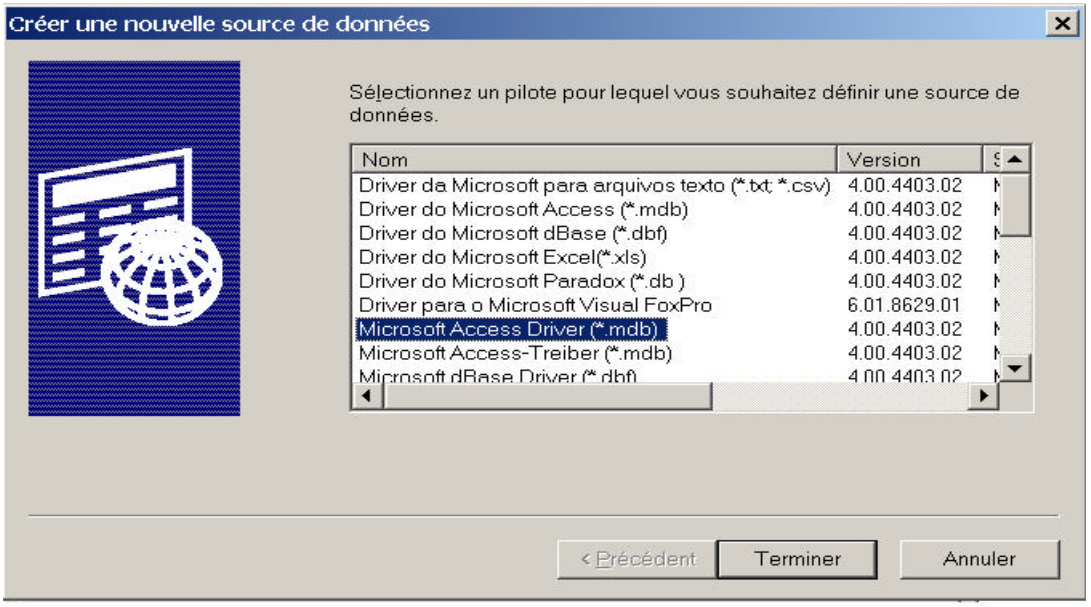
Sur le CD vous ne trouverez pas la base de données Access, tout simplement parce que c'est un produit qui n'est pas en libre utilisation. Pour son installation, il suffit de suivre les instructions fournies lors du traitement de l'exécutable.

L'emploi de la base de données Access nécessite de configurer le système d'exploitation Windows. Pour se faire suivez les instructions suivantes :

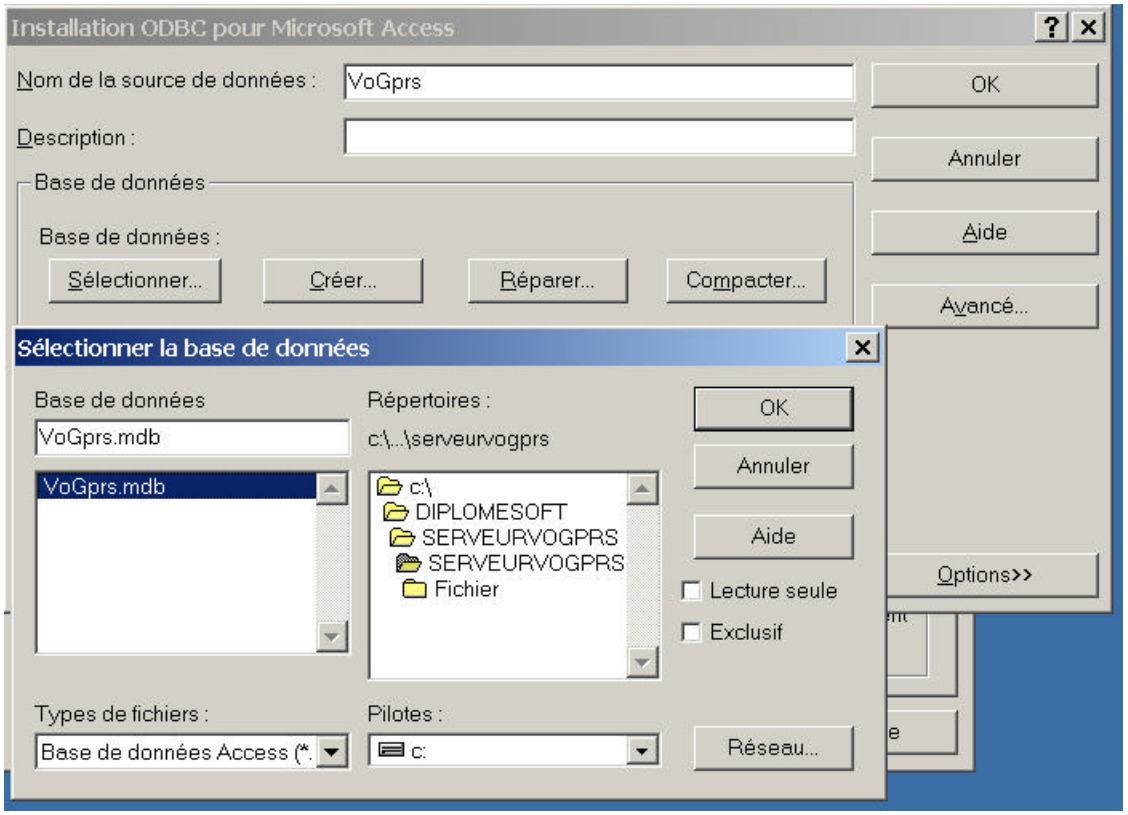
Aller dans **Paramètres**, **Panneau de configuration** sélectionner **Outils d'Administration** et choisir **Sources de Données (ODBC),** la fenêtre ci-dessous apparaît.



Sélectionner ce qui est indiqué sur l'image et cliquez sur le bouton **Ajouter**. La fenêtre suivante s'ouvre.



Sélectionner ce qui est indiqué sur l'image et cliquez sur le bouton **Terminer**. La fenêtre suivante s'ouvre. Là il faut définir un nom pour la base de données, dans l'exemple nous l'appelons VoGprs et il faut la sélectionner, ceci se fait en cliquant sur le bouton **Sélectionner**. Une fois ceci réalisé il ne reste plus qu'a cliquer sur le bouton **OK**.

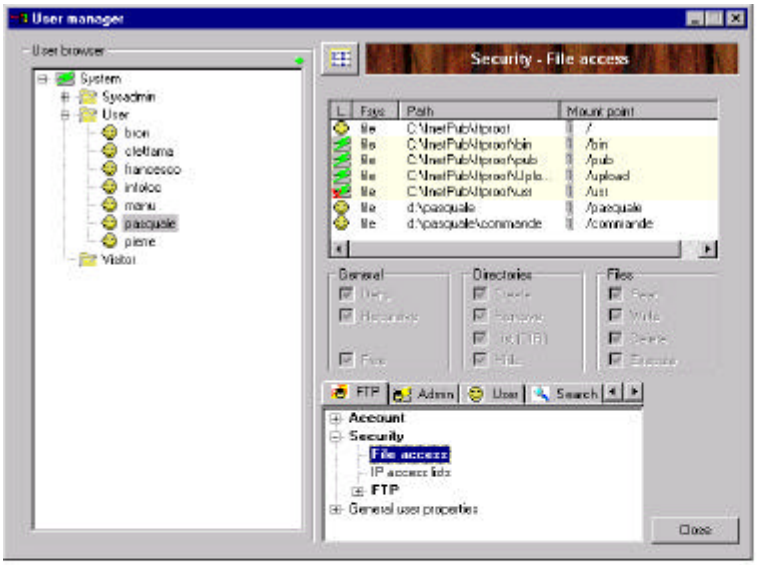


La configuration de la base de données est terminée.

20.1.3 Serveur FTP

Nous avons choisi comme serveur serveur FTP WarFTPD, principalement parce qu'il est utilisable gratuitement. L'exécutable du serveur FTP se trouve sur le CD dans le répertoire ServeurFTP, pour son installation, il suffit de suivre les instructions fournies lors du traitement de son exécutable.

Une fois le serveur installé, il faut créer un compte utilisateur, nous l'avons appelé Pasquale et les répertoires dont l'application à besoin, sont : Commande, Resultat, Message et Configuration. La figure ci-dessous montre l'interface permettant de contrôler le serveur FTP.



20.1.4 Lancement du serveur

Avant de lancer le serveur il faut s'assurer que le port COM2 soit disponible et y connecter le téléphone mobile. Attention notre serveur est configurer pour lire et traiter les fichiers sur le disque D. Si l'installation du serveur FTP ne se trouve pas sur le disque D de votre machine, il faut modifier les programmes Java, pour que l'application lise les fichiers au bon endroit.

Le lancement du serveur se fait en tapant la ligne de commande suivante :
Java ServeurVoGprs.ProgrammePrincipalThread

20.2 Installation du client

L'installation du client demande de copier le fichier VoGPRS1.0.vb qui se trouve sur le CD dans le répertoire Logiciels application\Client\Installer sur l'Ipaq dans le répertoire myDocuments.

Il faut également copier sur l'Ipaq dans le répertoire monPocketPc\Windows\ les librairies se trouvant sur le CD dans le répertoire Logiciels application \ Client \ Installer \ Librairie.

21 Conclusions

Ce travail de diplôme nous a permis de nous familiariser avec le développement d'applications destinées à la téléphonie mobile, plus particulièrement au réseau GPRS.

La réalisation de ces applications est un domaine très intéressant car il demande à la fois d'avoir de bonnes connaissances du réseau et de maîtriser les outils de développement informatique. L'étude du réseau GPRS s'est faite lors du projet de semestre, alors que pendant ce travail de diplôme l'accent à plutôt était mis sur la programmation.

Nous avons particulièrement traiter l'accès à des ordinateurs distants, le transfert de fichiers, l'utilisation de base de données et l'emploi des SMS. Ce travail nous a également donné l'occasion de nous familiariser avec la programmation d'un système embarqué l'Ipaq, notamment avec la réalisation d'une interface graphique, celle-ci demande bien plus que de simples connaissances informatiques, elle demande d'avoir des réelles compétences en ergonomie et en design.

L'expérience que nous avons acquise suite à ce travail, nous a convaincus que l'avenir de la téléphonie mobile passe par de telles applications. Ce qui à notre avis freine le développement de celles-ci est que d'une part que nous ne disposions pas encore d'un appareil regroupant les fonctionnalités d'un PDA et celles d'un téléphone mobile. Nous sommes obligé de combiner les deux. D'autre part il n'existe pas de plate forme de développement sous forme logiciel, pour l'heure, la réalisation d'une application GPRS demande donc de disposer d'équipement GPRS et d'un accès au réseau, ce qui n'est pas à la porté de tout le monde.

Yverdon, le 20 décembre 2001

Pasquale De Frino

22 Bibliographie et liens Internet

[1] Xavier Lagrange, Philippe Godlewski, Sami TAbbane
Réseaux GSM-DCS 4ème édition revue et augmentée
HERMES science publications, 1999

[2] Markus Jaton, Christian Roubaty
Réseaux de télécommunication 4ème édition
Tcom, 2000

[3] Simon Buckingham
An Introduction to the General Packet Radio Service
Mobile Lifestreams, 2000

[4] Nick Grattan, Marshall Brain
Windows CE 3.0 Application programming
Microsof technologies series, 2001

www.java.sun.com
www.javaworld.com
www.compaq.com
www.microsoft.com
www.ericsson.com