POLYTECH LILLE

Université Lille1 Sciences et Technologies

## Rapport de projet de seconde année du cycle ingénieur

**Département Informatique, Microélectronique, Automatique**
**Polytech'Lille, Villeneuve d'Ascq**

# Réalisation de SMSMail


**Source :**
**http://projets-imasc.plil.net/mediawiki/index.php?title=Projets_IMA4_SC_%26_SA_2012/2013**

| Elèves | Enseignants | Année Scolaire |
|---|---|---|
| Bastien Chalaux<br>Vincenti Jean-Marie | Alexandre Boé<br>Thomas Vantroys | 2012/2013 |

# Sommaire

# Introduction

Ce rapport présente notre travail effectué dans le cadre du projet en seconde année dans le département système communicant de l'école d'ingénieur Polytech'Lille. Ce projet a été proposé par M Alexandre Boé et M Thomas Vantroy et est s'inscrit à la fois dans le domaine de l'informatique et de l'électronique. Le travail ayant été fait en binôme, nous commencerons par détailler le sujet et les objectifs puis nous présenterons la répartition du travail sous deux sous-systèmes.

L'ensemble du travail demandé est la réalisation d'un système technique ainsi qu'un ensemble de rapports sous plusieurs formes. Une vidéo de présentation, l'historique de l'avancé du travail ainsi que les diapositives d'une présentation orale est disponible à l'adresse suivante :

Source : *http://projets-imasc.plil.net/mediawiki/index.php?title=Site_web_par_SMS*

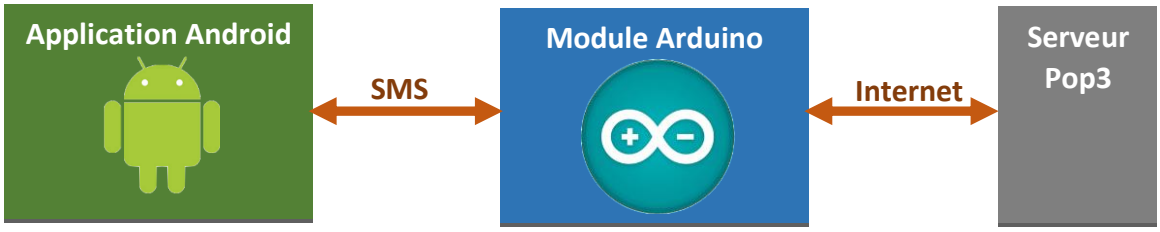
# Présentation du sujet

L'objectif de ce projet est la réalisation d'un système permettant de lire ses emails sur un téléphone Android sans utiliser la communication de données GSM ou 3G mais uniquement par SMS. Ainsi le système est composé de deux parties :

- *Un module GSM compatible Arduino qui recevra et transmettra les SMS de et vers un téléphone mobile,*
- *Une application client sous Android.*

Pour rajouter une difficulté supplémentaire et d'apporter une solution plus autonome nous avons proposé d'effectuer la récupération d'email sans passer par l'utilisation d'un ordinateur. Ainsi le module récupère lui-même les emails et se connecte seul à internet.

Voici le schéma théorique du système :




L'objectif du module Arduino est d'effectuer une lecture d'emails sur un serveur distant en se connectant de manière autonome au réseau internet. De plus il devra transmettre les données récoltées à un téléphone Android uniquement par l'envoi et la réception de SMS.

L'objectif de l'application Android est de proposer une interface permettant de demander et lire ses emails sur le téléphone tout en cachant la communication par SMS à l'utilisateur. Le fonctionnement devra également limiter l'envoi de SMS.

# Partie Arduino

Comme nous l'avons vu précédemment ce projet comporte deux parties bien distinctes. L'une d'entre elle servant à la récupération et au traitement des emails ainsi que leur envoi (fragmenté ou non) vers un téléphone cible (qui lui-même fera tourner une application Android). Cette partie propose de détailler la conception du système basé sur un Arduino et d'en décrire les différentes parties ainsi que les fonctionnalités.

## Spécifications techniques

Les points suivants représentent les problématiques techniques induites par le cahier des charges concernant la partie traitement.

- Le système doit être capable de récupérer (Connexion POP3) et stocker (Carte SD) de façon autonome (Timer) les emails situés sur des serveurs POP3 distants
- Le système doit pouvoir extraire les informations intéressantes de l'entête et du corps d'un mail qui n'auront pas toujours exactement la même syntaxe.
- Le système devra être capable, pour respecter les normes de taille des SMS, de découper les données extraites selon un protocole propre au projet tout en gardant leur intégrité.
- Le système devra pouvoir envoyer des SMS.
- Pour le debugge du système on devra pourvoir visualiser son évolution sur le port série de l'Arduino

## Choix techniques

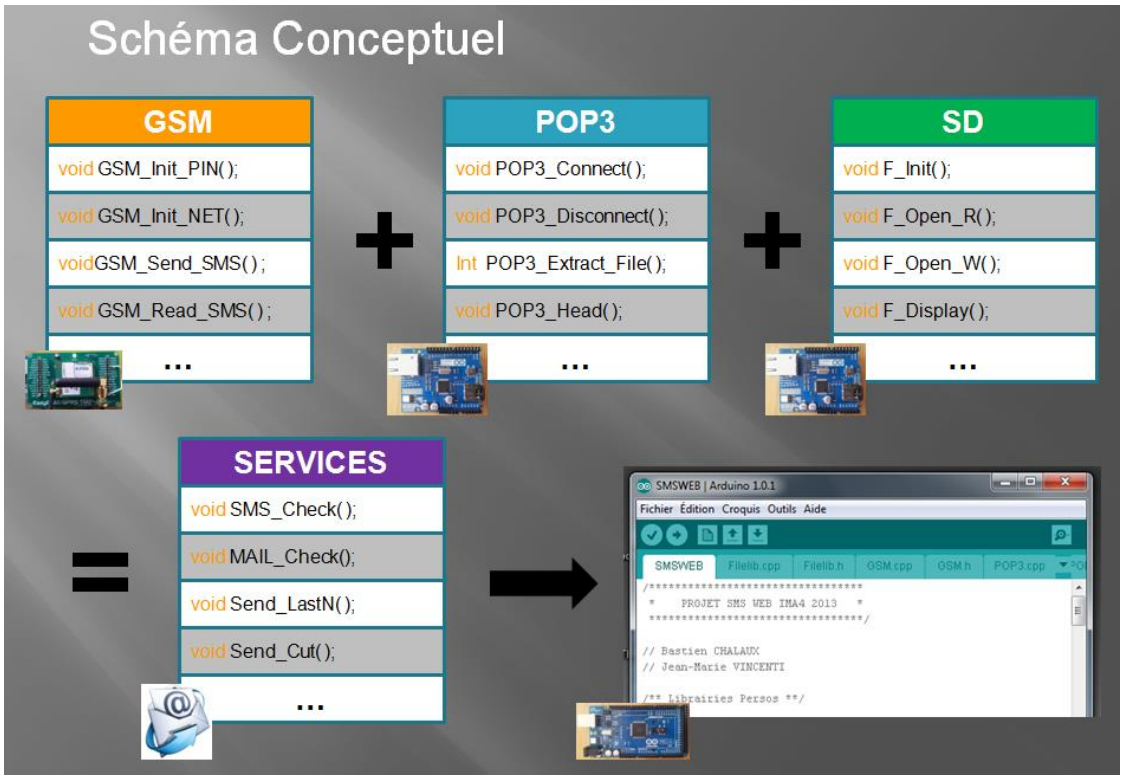
Pour répondre au cahier des charges ainsi qu'aux problèmes posées précédemment voici les solutions techniques retenues :

- Microcontrôleur : Arduino MEGA 2530 basé sur un ATMega2530 cœur de l'application contrôlant les autres périphériques.
- Connexion POP3 : Utilisation d'un Shield Ethernet Arduino officiel avec la librairie open source disponible sur le site de la communauté. Envoi des requêtes standard du protocole POP3 pour manipuler les emails.
- Stockage : Utilisation du port micro SD disponible en natif sur le Shield Ethernet et la librairie associée.
- Autonomie : Utilisation de la librairie « SimpleTimer » permettant d'effectuer des tâches périodiques comme la consultation des emails ou la lecture de la mémoire SMS de la carte SIM
- Traitement : Développement des fonctions de « Parsing » basées sur la librairie standard C « String »

- **Fragmentation** : Développement de fonctions de découpage des données à envoyer.

- **Envoi /Réception de SMS** : Utilisation du module GSM TM2 de chez MikroElectronika et développement d'une librairie cachant l'utilisation des commandes AT.

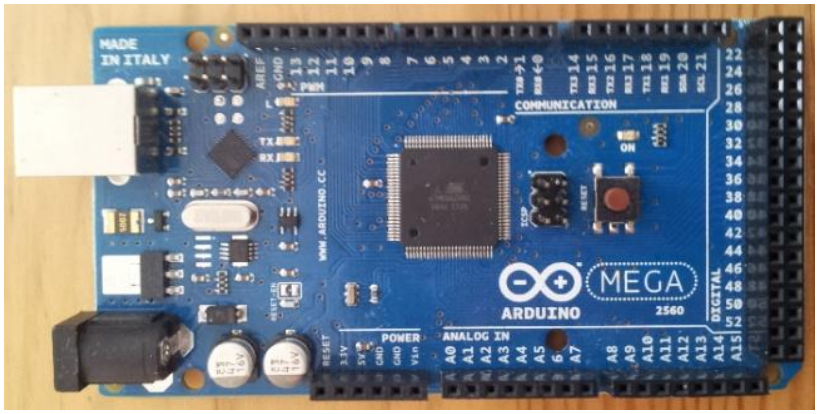
Ce qui d'un point de vu conceptuel peut être représenté sous la forme suivante qui garantie une grande modularité :



# Développement

Cette partie rend compte du développement des différentes solutions techniques citées précédemment. Le but n'est pas de rendre compte de façon exhaustive du code qui a été écrit mais plutôt de dégager la structure des différents interne et des possibilités des éléments composant ce sous système du projet et ceci que ce soit au niveau hardware ou software.

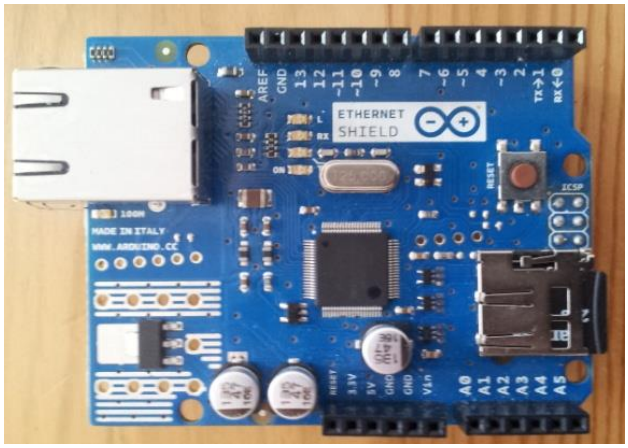
## Microcontrôleur




Le sujet restant relativement vague au sujet de l'unité de calcul qui devrait faire le traitement des emails et commander les différents périphériques nous avons fait le du « tout embarqué » en utilisant un Arduino comme cœur de l'application et non un PC. Notre système peut tout à fait fonctionner sur un Arduino UNO, le choix d'utiliser un Arduino MEGA est simplement justifié par le besoin d'utiliser des liaisons séries physiques supplémentaires, non disponibles sur le UNO, afin de correctement visualiser sur un terminal le bon fonctionnement du système en temps réel. La librairie « Software Serial » n'est pas utilisable ici car elle ne comporte pas de buffer de lecture/écriture on risque donc de perdre des données.

C'est donc autour d'un Arduino MEGA 2530 basé sur un ATMega2530 que nous avons choisis de développer notre système.

## Connexion POP3




La connexion au serveur POP3 pour la récupération des emails s'ai fait grâce au « Shield Ethernet » officiel Arduino. C'est une carte d'extension qui joue le rôle de carte réseau et permet de dialoguer sur un réseau Ethernet. Selon le code chargé dans l'Arduino l'ensemble peut se comporter comme un serveur ou dans notre cas comme un client. En effet la librairie « Ethernet Librairy » fourni en série avec l'IDE Arduino permet à l'utilisateur de réaliser des requêtes sur le réseau à travers des commandes simplifiées telles que :

| Ethernet Librairy | |
|---|---|
| Ethernet.begin(mac, ip, dns, gateway, subnet); | Permet d'initialiser le module |
| EthernetClient.connect(ip, port); | Etablie une connexion vers un serveur |
| EthernetClient.print(data) | Envoi une requête au serveur |

Source : http://arduino.cc/en/Reference/Ethernet

Ces fonctions ne sont pas propres aux serveurs POP3 mais servent simplement à établir une connexion. Une fois la connexion établie on utilise les commandes propres au protocole POP3

| Commandes POP3 | |
|---|---|
| NOOP | Permet de garder la connexion active en ne faisant rien |
| USER | Commande pour rentrer le login |
| PASS | Commande pour rentrer le password |
| UIDL | Affiche le tableau de correspondance Num/Id |
| RETR n | Affiche le message complet de numéro n |
| TOP n x | Affiche l'entête et les x premières lignes du mail n |
| QUIT | Ferme la connexion |

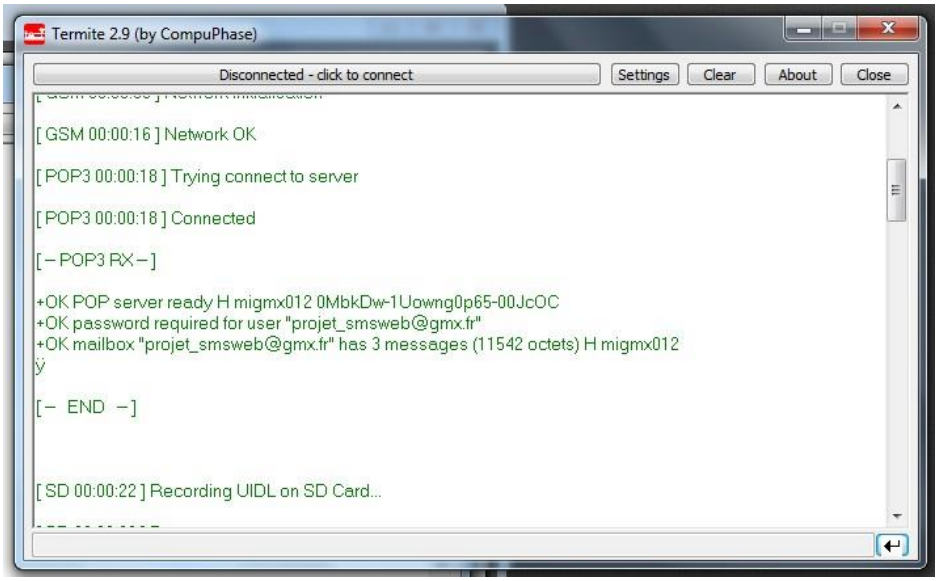
Source : http://irp.nain-t.net/doku.php/180pop3:020_commandes

A partir de ces deux jeux d'instructions nous avons écris une librairie permettant d'automatiser les requêtes et de cacher à l'utilisateur l'utilisation de ces commandes.

| Fonctions POP3 | |
|---|---|
| void ETH_Init(int config); | Initialisation Carte Ethernet |
| void POP3_Connect(char * login, char * pass, int config); | Connexion |
| void POP3_Disconnect(); | Déconnexion |
| void POP3_Rx(int disp, int rec); | Lecture de la réponse du Serveur POP3 |

*Voir POP3.h*

Ici on peut voir le résultat de la connexion au serveur dans le terminal série.



## Stockage

Comme nous avons pu le voir sur la photo du « Shield Ethernet » ce dernier comporte un port pour carte microSD. En effet la RAM de l'Arduino MEGA 2530 ne faisant que 8 Ko cela est trop peu pour pouvoir manipuler de grandes chaînes de caractères correspondant aux entêtes et au corps des emails non filtrées. La base de travail sur la carte SD a été la librairie « SD Librairy » Arduino qui permet de manipuler la carte facilement  en faisant abstraction de la gestion du system de fichier.

| SD Librairy | |
|---|---|
| SD.begin(cspin) | Initialise la carte SD en sélectionnant une PIN |
| SD.open(filepath, mode) | Ouvre un canal d'écriture ou de lecture selon le mode |
| SD.close() | Ferme le canal |

Source : http://arduino.cc/de/Reference/SD

A partir de ces fonctions nous avons écrit une librairie de plus haut niveau pour faciliter l'écriture du code son découpage en module dans le cas où nous voudrions faire évoluer le system et utiliser un autre support de stockage.

| Fonctions Carte SD | |
|---|---|
| void F_Init(); | Initialise la carte SD |
| void F_Open_R(int fc,char *name); | Ouvre un fichier en lecture |
| void F_Open_W(int fc,char *name); | Ouvre un fichier en écriture |
| void F_Close(int fc); | Ferme un fichier |
| void F_Display(int fc,char * name); | Affiche sur la console le contenu d'un fichier |

*Voir FileLib.h*

## Autonomie

Au regard de l'utilisation du système décrite dans le cahier des charge il peut être utile voir indispensable que le système soit autonome quand à la scrutation des nouveaux SMS dans sa mémoire (pas d'interruption matérielle disponible sur le module GSM) et à la récupération automatique des nouveaux emails sur le serveur POP3. Pour implémenter cette fonctionnalité nous avons cherché à utiliser les « Timers » de l'arduino. Un « Timer » est un dispositif matériel inclus dans le microcontrôleur permettant de générer une interruption logicielle, c'est-à-dire interrompre le déroulement du programme principale sauvegarder son état et exécuter un autre programme stocké en mémoire puis de restaurer le contexte du programme principal lors de son interruption. Concrètement en C cela revient à appeler une fonction ne comportant aucune boucle infinie à intervalles réguliers pour cela on utilise la librairie « SimpleTimer » (non incluse par défaut dans l'IDE arduino) qui permet de les utiliser très facilement en choisissant la fonction à appeler et la période d'appel.

Source : http://playground.arduino.cc/Code/SimpleTimer

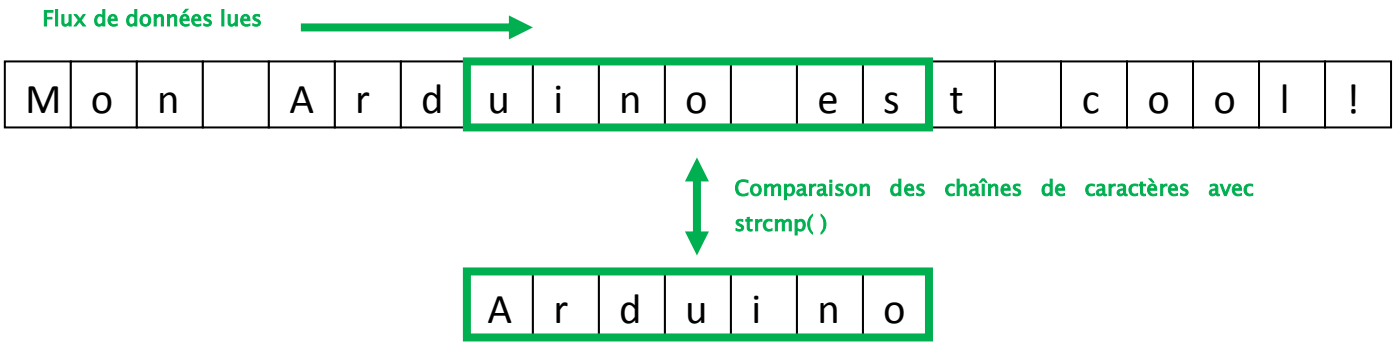
| Fonctions Périodiques | |
|---|---|
| void SMS_Check(); | Scrute la mémoire de la carte SIM |
| void MAIL_Check(); | Scrute le serveur POP3 pour les nouveaux emails |

*Voir Service.h*

## Traitement

Le traitement des données suite à la récupération des emails constitue le cœur du sous système Arduino. En effet il s'appuie sur tous les autres modules pour récupérer les données et les stocker sur microSD mais l'ensemble des fonctions décrites ci-après permettent d'extraire les informations intéressantes des entêtes et du corps des emails. La librairie de manipulations de chaînes de caractères est la librairie standard C « String » car elle offre un panel de primitives à la fois légères et utiles pour notre projet.

Le problème principal du projet réside dans le fait que l'on ne peut pas manipuler et à fortiori chercher un motif particulier dans un texte comme on le ferait dans un simple tableau de caractères. L'astuce réside dans la technique dite de « fenêtrage » c'est-à-dire que l'on alloue un tableau de la taille du mot que l'on cherche dans le texte, on effectue un remplissage par la queue et à chaque lecture d'un caractère provenant du flux de données (ex : fichier texte) on compare l'état du tableau avec le mot cherché. En comptant le nombre de caractères absorbé on connaitra la position du mot dans le texte d'entrée.

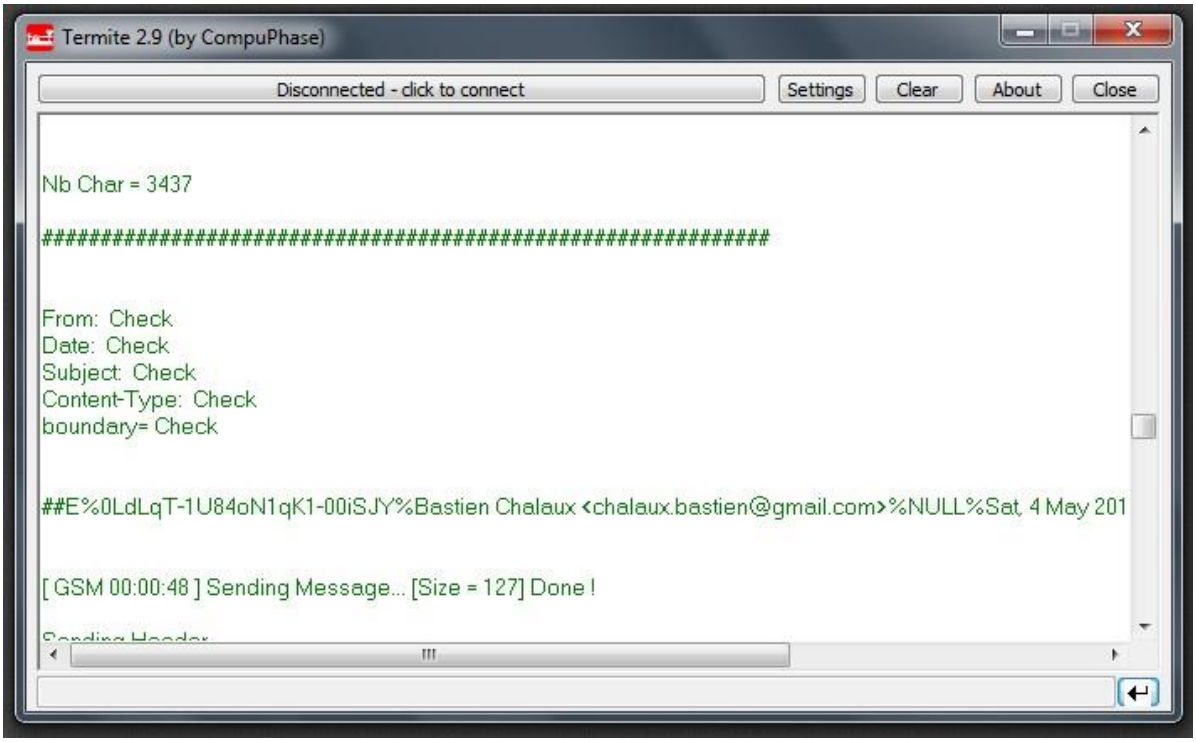



Une fois que l'on sait détecter un motif dans un texte il faut traiter un maximum de cas pour être sûr de ne pas rater d'informations des les entêtes des mails (majuscules, minuscules) que les informations sont lisibles directement (détection multipart, ISO, UTF8,…) et ainsi lancer le traitement adéquat.

| Fonctions de « Parsing » des emails | |
|---|---|
| int POP3_Extract_File(char *cmp,char fin); | Fonction qui cherche un motif dans un fichier |
| void POP3_Read_Top(int n); | Extraction Header (From,Date, Subject,Content,…) |
| void POP3_Read_Mess(int n); | Extraction Corps d'un mail |

*Voir POP3.h*

Ici on peut voir la détection des informations importantes dans l'entête d'un mail sur le terminal série




## Fragmentation

En format texte simple les SMS ne doivent pas comporter plus de 160 caractères. Nous avons donc développé une fonction capable de prendre en entrée un fichier texte long et d'envoyé sms par sms en ajoutant des balises de fragmentations (voir protocole mis en place en commun) pour qu'à l'arrivée l'application Android puisse facilement reconstituer le message même si l'opérateur les restitues dans le désordre. (*Voir Service.h)*

| Fonctions de Fragmentation | |
|---|---|
| void Send_Cut(int fc, char * name, char * head); | Fonction de fragmentation SMS |

## Envoi/Réception de SMS


Le système ne serait pas complet sans le module permettant d'envoyer des SMS. C'est un module TM2 monté sur une platine de connectique EasyGSM de Mikroelectronika. Il peut réaliser toutes les fonctions de base d'un téléphone à savoir SMS, Voix et Internet. Il dispose de plusieurs dispositifs de communications. Nous nous intéresserons ici uniquement à se liaison série qui permet le commander via les commandes AT.

Les commandes AT sont des commandes standard généralement utilisées pour la commandes des modems elles sont disponibles sur tout les téléphones et permettent d'en prendre quasiment le contrôle complet. Dans la liste ci-dessous on retrouve les commandes utilisées lors du projet pour envoyer et recevoir des SMS

| Commandes AT | |
|---|---|
| AT | Renvoie simplement OK |
| AT+CPIN | Permet fournir le Code PIN de la carte SIM au module |
| AT+COPS | Permet de sélectionner un operateur |
| AT+CMGF | Permet de choisir le mode d'envoie des SMS (ici mode TEXT) |
| AT+CMGS | Permet d'envoyer un SMS |
| AT+CMGR | Permet de lire un SMS dans la mémoire de la SIM en précisant son Id |
| AT+CMGD | Permet de supprimer un SMS dans la mémoire |

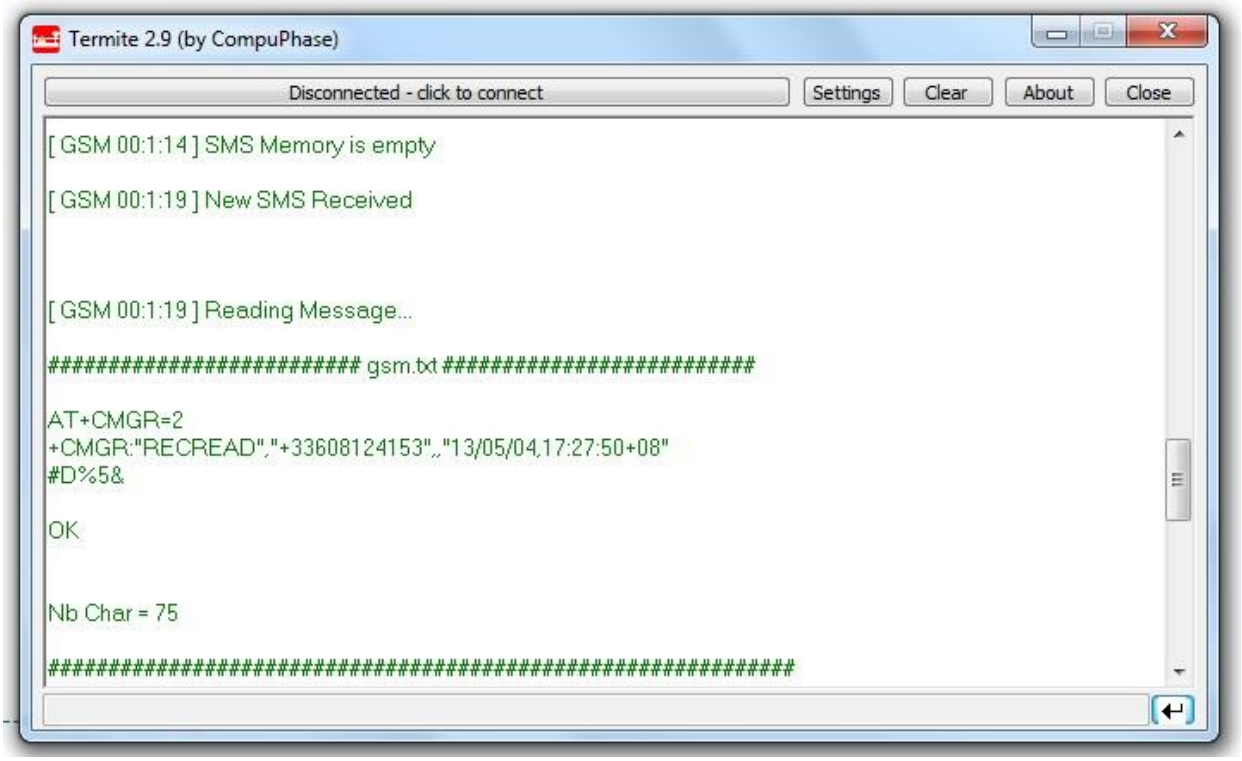
Source : http://projets-imasc.plil.net/mediawiki/index.php?title=Fichier:TM2_AT_Command.pdf

Comme pour les commandes POP3 nous avons développé une librairie permettant de cacher à l'utilisateur l'utilisation des commandes AT et ainsi simplifier la commande du module.

| Fonctions pour le module GSM | |
|---|---|
| void GSM_Rx(int disp); | Lit la réponse du module |
| void GSM_Init_PIN(char * pin); | Déverrouille la carte SIM avec le code PIN |
| void GSM_Init_NET(); | Enregistre le module sur le réseau |
| void GSM_Send_SMS(char * tel,char * mess); | Envoi un SMS à un numéro passé en paramètre |
| void GSM_Read_SMS(int num, int disp); | Lit un SMS en mémoire |
| void GSM_Terminal_Command(); | Fait le lien entre le Serial1 et le Serial0 |
| void GSM_Del_SMS(int num); | Supprime un SMS en mémoire |
| void GSM_Clear_SIM(); | Supprimme tout les SMS en mémoire |
| void GSM_Com_Execute(); | Exécute une commande en fonction du SMS reçu |

*Voir GSM.h*

Ici on peut voir la réception d'un SMS demandant de récupérer les Id des 5 derniers emails reçus (voir protocole) sur le terminal série



# Partie Android

Afin de visualiser les mails sur le téléphone. Nous avons développé une application en utilisant le langage natif Android. Cette partie détaille le travail effectué ainsi que les choix des élèves. L'objectif est de fournir un sous-système capable de recevoir les SMS, de les traiter et d'afficher les données contenues sous forme d'email. L'application sera également capable de générer des SMS afin de communiquer avec la platine Arduino.

## Spécifications techniques

Les points suivant représentent les problématiques techniques induites par le cahier des charges.

- L'application doit pouvoir stocker les mails. En effet il est primordial que toutes informations reçues ne soient pas redemandées à l'Arduino. Nous étudierons la méthode la plus rapide d'accès et de stockage de ces données.

- L'application doit limiter le nombre d'envois de SMS, principalement pour des raisons économiques (le coût pour l'utilisateur de l'envoi de SMS) mais également de temps de réponse.

- L'interface et le design principal doit cacher le fonctionnement par SMS. On cherchera les solutions qui permettent de masquer à l'utilisateur et autres applications cette communication par SMS.

## Choix techniques

Pour répondre au cahier des charges ainsi qu'aux spécifications techniques voici les solutions techniques retenues :

- Langage android

Pour une plus grande flexibilité et une expérience pédagogique intéressante le développement se fera sans Framework, on utilisera donc uniquement me langage Android. Les logiciels utilisés sont Eclipse et les librairies Android IDE. L'interface est décrite en langage XML spécifique et le cœur est en Java-Android.

En effet l'application devant effectuer de nombreuses interactions avec le système (comme l'envoi de SMS ou le stockage de données) il est bien plus performant et plus rapide de travailler directement en langage natif. De plus cette solution s'inscrit dans notre formation, puisqu'elle utilise le langage Java.

- Base de données

Toutes les informations relatives aux mails sont dynamiques. On utilisera donc une base de données SQLiteDatabase. Cette base de données sera optimisée et complète pour pouvoir traiter un nombre de mail important. L'objectif sera de limiter le temps de calcul lors de l'accès aux données car il conditionne le temps d'affichage et de rafraichissement des vues, et donc l'expérience utilisateur.

- Fonctionnement en fond

Suite à l'étude des possibilités et aux libertés qu'offre le système Android, nous avons constaté qu'il est possible d'envoyer dynamiquement dans le code des SMS, ainsi que d'en recevoir et les traiter. De plus la réception de SMS par les applications est gérée par niveau de priorité. Tous ces éléments permettent donc de mettre en place un système de communication par SMS invisible pour l'utilisateur. Cependant, dans tous

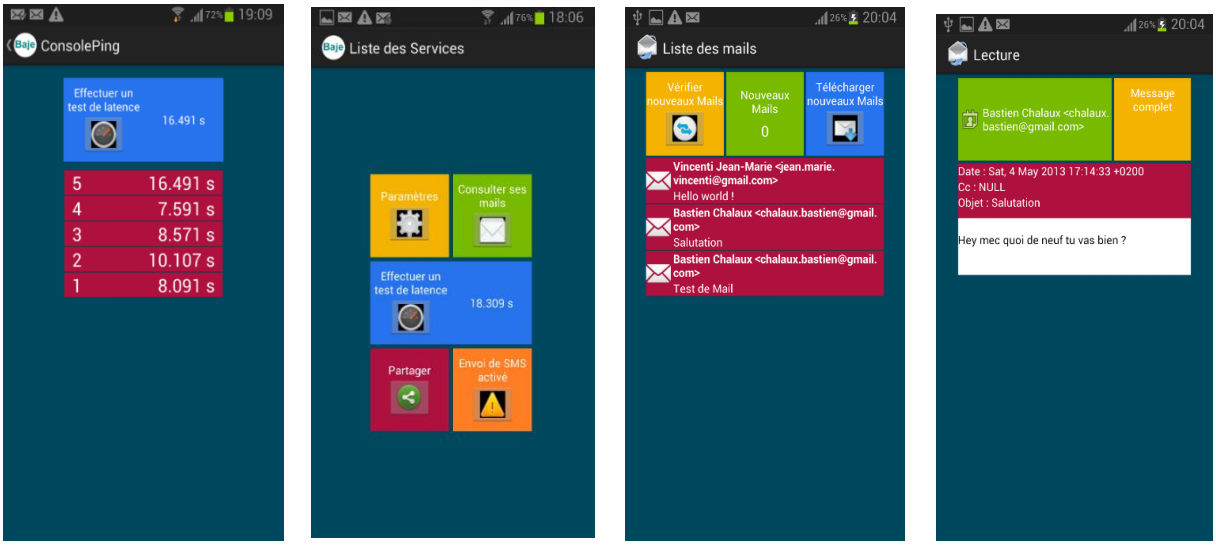
les cas les traitements et calculs sont itératifs il faut donc construire une architecture qui favorise la fluidité et l'expérience de l'utilisateur.

## Développement

L'application a été pensée comme une application concept servant principalement à tester la faisabilité et les performances possibles du système. Elle intègre donc des fonctionnalités permettant de visualiser ces performances et propose une interface type qui permet à l'utilisateur de visualiser ses emails.

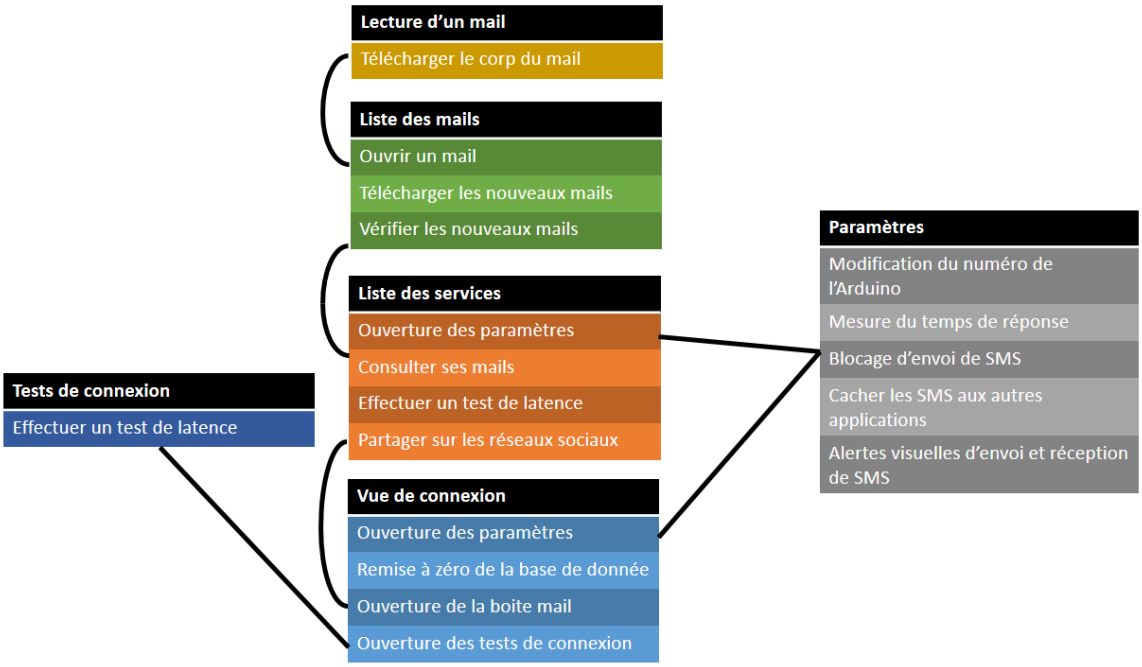
La présentation du développement se fera en trois étapes. Nous commencerons par présenter les vues ou activités, puis l'organisation de la base de données, enfin nous détaillerons l'architecture de l'application et du système d'envoi et de réception de SMS.

Voici quelques images du résultat de l'application :




### La partie visible de l'application : les vues

Les vues ou activités sont géré par le système Android comme une pile d'exécution où la création et destruction d'instance va respectivement empilé et dépilé des vues. C'est dans ce principe là qu'a été construite la navigation de l'application. Le schéma suivant illustre les choix possibles à chaque instant dans l'application. Les éléments représentent les vues avec les différents éléments d'interraction comme les boutons.




- Tests de connexion :

Cette apllication a été pensé comme une application de concept et de test, ainsi nous avons décidé de rajouter une vue permettant de visualiser le résultat de tout les tests de latence effectués depuis la création de la base de donnée. Les résultats ont montrés que le temps qu'il faut pour envoyer le SMS, que l'Arduino le reçoive, le traite et renvoi le signal au téléphone est de 11 sec au minimum, 20 secondes en moyenne. La grande partie de ce temps dépend du réseau et n'est pas compressible par l'amélioration du système.

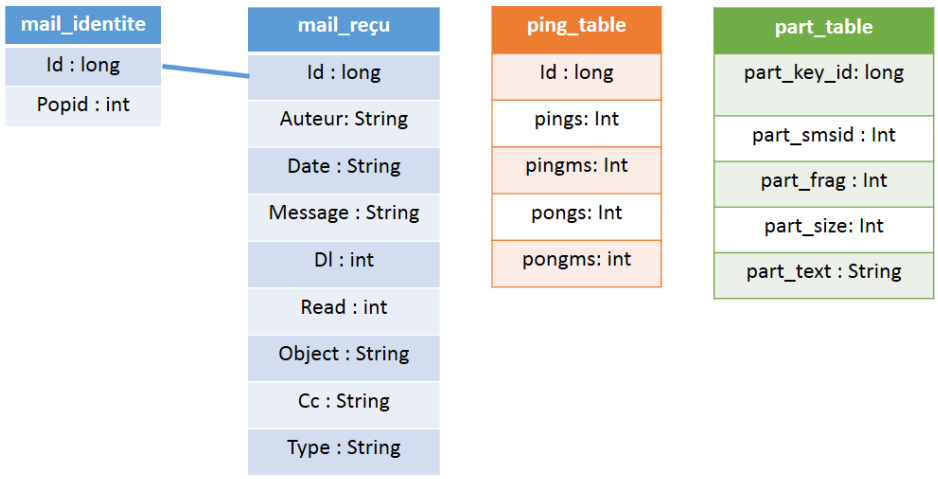
- Liste des mails :

Pour limitter l'envoi et la réception de SMS nous avons découpé la reception des SMS par étapes. Ainsi l'utilisateur peut commencer par connaître le nombre de mail et seulement le nombre. Il est alors invité à télécharger, s'il le veut, les entêtes des mails c'est-à-dire l'emmeteur, l'objet et la date de réception du mail. Finalement l'utilisateur peut demander de télécharger le corp sous forme de texte.

## Le stockage des données : la base de données

L'application a besoin de traiter différents types de données. Dans le système Android, les tables ont comme clés primaires un unique champ de type long. Ce principe est obligatoire et permet d'être auto-

incrémentale, il est très utile pour traiter et afficher des listes rapidement dans des vues.




Ainsi nous avons donc décidé de créer une table de correspondance entre l'ID interne (de type long) d'un mail, et l'ID externe de ces mails qui correspond à celui sur le serveur Pop3. D'un point de vue théorique on associe l'ID interne à une ou plusieurs informations qui identifient de manière unique l'élément « mail ». Dans notre cas on a uniquement l'ID du serveur Pop3 mais cette astuce a le mérite de s'adapter aux évolutions du système. De plus c choix nous permis de tester un fonctionnement de la base de donnée plus complexe qu'une simple table.

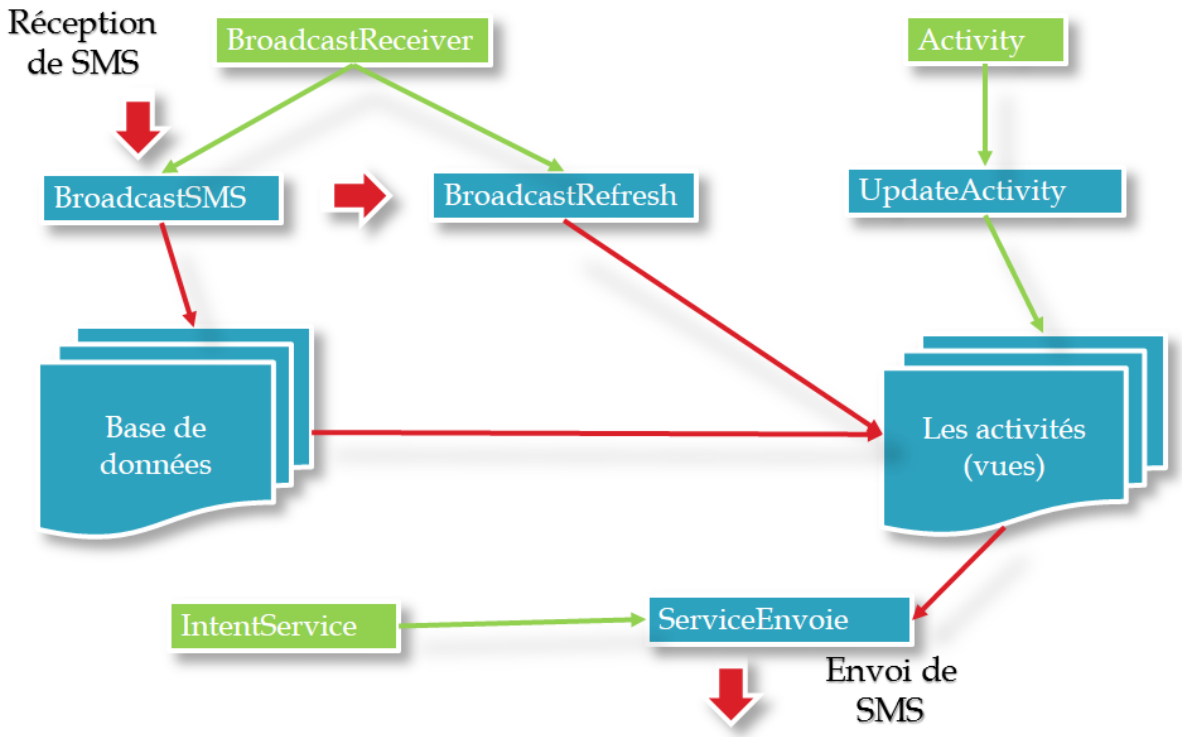
Les emails sont donc représentés par leurs IDs, l'auteur, la date, le corp, l'objet ainsi que le type de mail. Ce type permet notamment de différencier les emails rédigés en HTML ou en texte simple. A ces données sont rajoutés deux identifiants qui servent à l'affichage et à la disponibilité des boutons de téléchargement.

Pour proposer le service de calcul de latence nous avons également implémenté une table avec les requêtes et les réponses des tests.

Finalement, une dernière table sert à la communication. Comme présenté dans la partie Arduino, les communications sont hachés en plusieurs SMS si besoin. Cette table permet donc de stocker les parties incomplètes avant la fusion et le traitement.

### La partie invisible de l'application : l'envoi et la réception de SMS.

L'architecture de l'application a été pensée en fonction des possibilités techniques liées à l'envoi et la réception de SMS. Le premier défi, qui est une des spécificités du cahier des charges, est le fonctionnement caché de la communication.




Pour y répondre, nous avons utilisé l'objet de BroadcastReceiver de l'API Android. Ces « BroadCaster » sont appelé par le système lors de la réception de SMS. Ils ont un niveau de priorité qui permet de bloquer ou non la diffusion des SMS aux autres applications. Pour proposer un fonctionnement en tache de fond, c'est-à-dire même si l'application n'est pas ouverte, nous avons déclaré le BroadcastSMS séparément des vues. Ainsi les données peuvent être reçus et traiter à tout moment. Cependant cet avantage induit un inconvénient, celui de ne proposer aucun moyen efficace pour interagir avec les activités. Ainsi chaque activité instancie pendant qu'elle est visible sur l'écran un BroadcastRefresh qui force la réactualisation des données affichés. On a donc une architecture où la partie qui enrichie la base de données est indépendante de celle qui les affiches.

Pour l'envoi de SMS, nous utilisons un IntentService, qui est également autonome. En effet lors de l'appui d'un bouton par l'utilisateur, les activités envoient un signal à ce service qui formate le SMS et l'envoi sans ralentir l'application.

# Présentation du résultat

## Protocole

Afin que l'Arduino et le téléphone Android puissent dialoguer facilement et aussi afin d'optimiser au mieux le nombre de caractères dans les SMS nous avons mis en place un protocole de communication qui n'est pas le même dans les deux sens. Cela est du au fait que les outils de detection d'une commandes dans une chaîne sont beaucoup plus puissant en java qu'en C embarqué. Les règles de ce protocole sont résumées ci-dessous :

| Commandes SMS : Téléphone Android vers Arduino | |
|---|---|
| #P%n& | Commande de PING où n est un entier que l'arduino doit renvoyer |
| #D%n& | Demande des Id des n derniers mails reçus |
| #H%id1+id2+id3...+idn& | Demande des Headers des mails correspondant aux différents id |
| #M%id& | Demande du Corps du Mail correspondant à l'id en argument |
| #T%id& | Envoi dans l'ordre de tout les id des mails plus récents que l'id en argument. |
| #V%(0 ou 1)& | Permet à l'application Android d'indiquer si elle va envoyer des requetes ou non |

Pour Arduino vers le Téléphone Android on tient en plus compte de la fragmentation des SMS

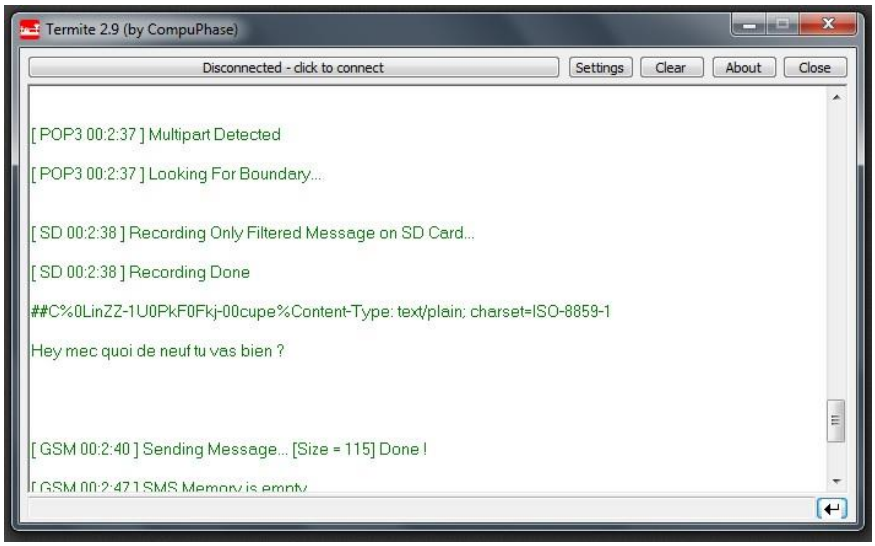
Si non fragmentation le SMS commence par : ##
Si fragmentation le SMS commence par : #id;num;size#
Où id est un entier quelconque correspondant au numéro de la transaction fragmentée,
Num est un entier de 0 à size-1 et size est un entier entre 2 et +inf

puis suivi des commandes suivantes

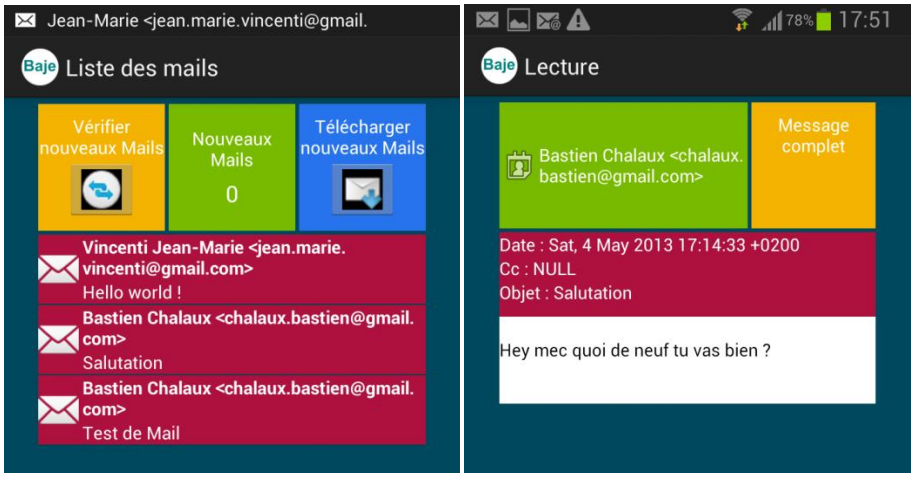
| Commandes SMS : Arduino vers Téléphone Android | |
|---|---|
| pong%num& | Réponse au PING |
| L%id1+id2+id3+...& | Renvoi des Id demandés |
| E%id%auteur%cc1;cc2;cc3;...%date%objet%type& | Envoi de l'Entête demandée |
| C%id%corp& | Envoi du Corps de mail demandé |
| NO& | Indique qu'il n'y a pas de nouveaux mails |

## Visualisation du résultat

*Montage à base d'Arduino MEGA complet avec le Shield Ethernet,*
*Le module GSM et entre les deux la carte d'adaptation réalisée pour le projet.*




*Un aperçu du terminal série permettant de suivre en temps réel l'activité de l'Arduino*




*Les vues de l'application Android correspondant à la liste des emails et au contenu de l'un d'eux*

# Conclusion

Ce projet très intéressant nous a permis de développer de nombreuses compétences.

D'un point de vue technique nous avons retranché l'Arduino dans ses limites se servant de toute la capacité de calcul disponible pour le traitement « in situ » des emails. D'un point vu de la modularité des sources nous avons atteint les limites de l'IDE Arduino et avons constaté qu'à partir d'un certain point il n'était pas l'outil le plus adapté pour le développement pour des projets de cette taille. Le défi a était de faire cohabiter des technologies comme l'accès à un serveur POP3 et l'accès au réseau GSM.

Ce projet nous a également permis de découvrir et de maitriser pleinement le langage Android. Grâce à ce projet de création d'outil, nous avons développé nos compétences en conception et réalisation d'application complexe en langage orienté objet : le Java.

Concernant la gestion de projet en elle-même nous avons constaté que cela demandait un intérêt tout particulier quand on cherche à assembler deux mondes (Android et Arduino) assez différents via le réseau GSM. Il a fallu communiquer au sein du groupe et se mettre d'accord sur un protocole facile à interpréter avec la technologie disponible de chaque côté.

Enfin même si la technologie développée n'est pas commercialisable en soi elle pose les bases d'un concept intéressant qui revient à assembler des briques existantes pour en faire naitre une nouvelle correspondant donc tout à fait au concept d'ingénierie.