

**2011-2012**

Guillaume Leroy Emmanuel Yagapen

Polytech’Montpellier pour Synox représenté par Jérome Fenwick & Joffrey Verdier

Rapport Technique



Mise en place d’un service SMS

Remerciements

Nous tenons à remercier toutes les personnes qui nous ont permis de mener à bien ce projet pendant ces deux mois :

* M. Jérome Fenwick et M. Joffrey Verdier, de l’entreprise demandeuse Synox, pour leur disponibilité et leur aide au bon déroulement du projet
* Mme Michelle Cart, notre responsable pédagogique, qui nous a suivis régulièrement et conseillé tout au long de ce projet
* L’administration de Polytech’Montpellier pour nous avoir fourni le matériel nécessaire à la bonne réalisation de ce projet

Sommaire

[1 Introduction 5](#_Toc315792588)

[2 Présentation de l’environnement 6](#_Toc315792589)

[2.1 Le groupe SYNOX 6](#_Toc315792590)

[2.2 La plateforme Machine-to-Machine de gestion d’objets communicants 6](#_Toc315792591)

[3 Présentation du projet 7](#_Toc315792592)

[3.1 Le problème initial 7](#_Toc315792593)

[3.2 Les besoins fonctionnels 7](#_Toc315792594)

[3.3 La mission 7](#_Toc315792595)

[3.4 Contraintes 7](#_Toc315792596)

[3.4.1 Contraintes techniques 7](#_Toc315792597)

[3.4.2 Contraintes temporelles 8](#_Toc315792598)

[3.5 Ressources 8](#_Toc315792599)

[4 Déroulement du projet 9](#_Toc315792600)

[4.1 Gestion du projet 9](#_Toc315792601)

[4.2 Démarche 9](#_Toc315792602)

[4.2.1 Méthodes utilisées 9](#_Toc315792603)

[4.2.2 Choix technologiques 9](#_Toc315792604)

[4.3 Communication et synchronisation 10](#_Toc315792605)

[5 Travail réalisé 11](#_Toc315792606)

[5.1 Architecture du système d’envoi et de réception de SMS 11](#_Toc315792607)

[5.1.1 Le service SMS 12](#_Toc315792608)

[5.1.2 La base de données 12](#_Toc315792609)

[5.1.3 Interface utilisateur 12](#_Toc315792610)

[5.2 Conception 12](#_Toc315792611)

[5.2.1 Diagrammes des cas d’utilisation 12](#_Toc315792612)

[5.2.2 Diagramme de classe 13](#_Toc315792613)

[5.2.3 Maquettage 15](#_Toc315792614)

[5.3 Développement 15](#_Toc315792615)

[5.3.1 Les commandes AT 15](#_Toc315792616)

[5.3.2 Le mode PDU et la librairie ATSMS 15](#_Toc315792617)

[5.3.3 Le service SMS 17](#_Toc315792618)

[5.3.4 La base de données 18](#_Toc315792619)

[5.3.5 L’interface graphique 19](#_Toc315792620)

[5.4 Tests 21](#_Toc315792621)

[5.5 Limites 22](#_Toc315792622)

[6 Améliorations possibles 23](#_Toc315792623)

[6.1 Permettre une utilisation multi-utilisateurs 23](#_Toc315792624)

[6.2 Sécuriser l’accès à la base de données 23](#_Toc315792625)

[6.3 Implémenter une gestion des contacts 23](#_Toc315792626)

[7 Difficultés rencontrées 24](#_Toc315792627)

[7.1 Apprentissage des technologies 24](#_Toc315792628)

[7.2 Problèmes de compatibilité entre le Framework .Net et le JavaScript 24](#_Toc315792629)

[7.3 Les accusés de réception 24](#_Toc315792630)

[8 Conclusion 25](#_Toc315792631)

[9 Annexes 26](#_Toc315792632)

[9.1 Les commandes AT 26](#_Toc315792633)

[9.1.1 Choix du format des messages 26](#_Toc315792634)

[9.1.2 Choix de la mémoire 26](#_Toc315792635)

[9.1.3 Affichage des messages 26](#_Toc315792636)

[9.1.4 Envoi d’un message 26](#_Toc315792637)

[9.2 Envoyer et recevoir des données 26](#_Toc315792638)

[9.2.1 Connexion au modem 27](#_Toc315792639)

[9.2.2 Envoi de données au modem 27](#_Toc315792640)

[9.2.3 Réception de données 28](#_Toc315792641)

[9.2.4 Déconnexion du modem 28](#_Toc315792642)

[9.2.5 Exemple d’utilisation du protocole de communication 28](#_Toc315792643)

[9.3 Diagramme de Gantt 30](#_Toc315792644)

Table des images

[Figure 1 - Schéma d'architecture 11](#_Toc315791749)

[Figure 2 - Diagramme des cas d'utilisation : Application 13](#_Toc315791750)

[Figure 3 - Diagramme des cas d'utilisation : Utilisateur 13](#_Toc315791751)

[Figure 4 - Diagramme des classes 14](#_Toc315791752)

[Figure 5 - Informations nécessaires pour l’envoi d’un SMS en mode texte 15](#_Toc315791753)

[Figure 6 – Informations nécessaires pour l’envoi d’un SMS en mode PDU 16](#_Toc315791754)

[Figure 7 - SQL Server Management Studio : Structure de la table Message 19](#_Toc315791755)

[Figure 8 - Requête LINQ : SMS en attente d'accusé 19](#_Toc315791756)

[Figure 9 - Capture d'écran de l'interface d'envoi de SMS 20](#_Toc315791757)

[Figure 10 - Capture d'écran de la boîte d'envoi 20](#_Toc315791758)

[Figure 11 - Capture d'écran de la boîte de réception 21](#_Toc315791759)

[Figure 12 - Extrait de l'algorithme de connexion au modem GSM 27](#_Toc315791760)

[Figure 13 - Extrait de l'algorithme d'envoi de commandes au modem GSM 27](#_Toc315791761)

[Figure 14 - Algorithme de réception des messages dyu modem GSM 28](#_Toc315791762)

[Figure 15 - Protocole de communication entre le service SMS et le modem GSM 29](#_Toc315791763)

[Figure 12 - Diagramme de Gantt (tableau) 30](#_Toc315791764)

[Figure 13 - Diagramme de Gantt (schéma) 30](#_Toc315791765)

# Introduction

Dans le cadre de notre formation de 5ème année en école d’ingénieurs à Polytech’Montpellier, nous devons effectuer un projet industriel en collaboration avec une entreprise. Ce projet a pour but de nous placer dans les conditions du monde professionnel.

Dans cette optique, nous avons choisi de réaliser le projet industriel proposé par M. Fenwick, représentant de l’entreprise SYNOX, société de prestation de services informatiques aux entreprises.

Synox possède une plateforme permettant à des applications d’échanger des informations avec des *objets distants*[[1]](#endnote-1)à travers le réseau *GPRS*[[2]](#endnote-2). Cependant ce réseau présente des problèmes de fiabilité compromettant le bon fonctionnement de la plateforme. Le réseau *GSM*[[3]](#endnote-3)sur lequel s’appuie la technologie *SMS*[[4]](#endnote-4) étant plus fiable, il est alors nécessaire de s’appuyer dessus afin d’assurer une continuité des échanges.

La mission qui nous a été confiée consiste donc à développer un mécanisme **d’envoi et de réception de SMS qui s’intégrera à la plateforme existante**.

Notre choix s’est porté sur ce projet car ce travail porte sur des technologies en fort développement que sont les objets mobiles et le Web. De plus, cette mission comprend des aspects dans le domaine du réseau que nous souhaitions approfondir.

Le projet a débuté le 5 décembre pour prendre fin le 9 février, soit une durée de 10 semaines.

Ce rapport vise à expliquer de manière synthétique le travail accompli durant ce projet. Pour plus de détails, nous vous invitons à vous reporter au rapport technique.

Dans un premier temps nous présenterons l’entreprise et le contexte dans lequel s’inscrit notre mission. Puis nous aborderons les objectifs attendus et le travail réalisé pour les atteindre. Nous décrirons ensuite la démarche suivie et les problèmes que nous avons rencontrés. Enfin, nous ferons un bilan sur le travail que nous avons effectué.

# Présentation de l’environnement

## Le groupe SYNOX

Le groupe SYNOX est une société informatique spécialisée dans les solutions mobiles et collaborative. Son activité porte essentiellement sur le développement spécifique et la mise en place d’infrastructures mobiles.

En 2010 son chiffre d’affaires a atteint plus 2,5 millions d’euros porté par la bonne santé du **cloud computing.**

## La plateforme Machine-to-Machine de gestion d’objets communicants

Synox développe pour ses clients des applications communiquant avec des objets distants grâce au réseau GPRS. Cette communication est gérée par une plateforme Machine-to-Machine (**M2M**) centralisée qui donne accès aux clients à un suivi de leurs objets distants.

Schéma

On peut ainsi prendre l’exemple d’un autobus qui est équipé d’un capteur qui envoie un signal à la plateforme à chaque fois qu’il arrive à un arrêt. La plateforme se charge alors de retransmettre l’information à d’autres services comme par exemple une application mobile.

# Présentation du projet

## Le problème initial

Les objets de la plateforme M2M du groupe Synox utilise actuellement le réseau GPRS pour communiquer. Cependant, il se peut que ce réseau ne soit pas disponible à certains endroits et à certains moments, rendant impossible toute communication entre les objets distants et la plateforme.

On cherche donc **à résoudre un problème de** **fiabilité** lié au système actuel.

## Les besoins fonctionnels

Pour pallier ce problème, Synox a choisi d’utiliser le réseau GSM, beaucoup plus fiable, en permettant aux objets distants de communiquer avec la plateforme par SMS.

Synox a donc besoin de fournir à ces programmes la possibilité d’envoyer des SMS afin d’assurer une continuité du service en cas d’une panne du réseau GPRS.

Il faut donc que les objets distants, dotés en conséquence d’un *modem[[5]](#endnote-5)* GSM, puissent envoyer et recevoir des SMS. De même, la plateforme M2M doit être dotée des mêmes capacités.

La solution sera associée à une base de données enregistrant tous les messages, ainsi qu’à une interface graphique de gestion en ligne pour permettre aux utilisateurs de communiquer par SMS avec les objets distants.

## La mission

Notre mission consiste donc à concevoir et à développer les fonctionnalités suivantes du système d’envoi et de réception de SMS :

* Envoi et réception de SMS à travers un *service Windows[[6]](#endnote-6)*
* Stockage des messages envoyés et reçus dans une base de données
* Associer un statut aux SMS envoyés afin de suivre leur état
* Ecriture et consultation des messages depuis une interface graphique

L’interface graphique a pour objectif de permettre à un utilisateur d’avoir un accès à l’ensemble des fonctionnalités proposées par le système.

## Contraintes

### Contraintes techniques

Pour mener à bien ce projet, nous devons respecter plusieurs contraintes techniques. Tout d’abord, le projet doit être développé à l’aide du *framework .Net*[[7]](#endnote-7) en version 4.0 ou 3.5.

La base de données, afin de s’intégrer au framework .Net, doit être implémentée avec le système de gestion de base de données SQL Server 2008. Ces contraintes sur le framework sont dues au fait que la plateforme M2M déjà existante est développée avec cette technologie. Le Groupe Synox maintient ainsi une cohérence entre tous les éléments de cette plateforme.

Enfin, la liaison des données entre la base de données et le service ou l’interface web doit être effectuée en LINQ to SQL ou en ADO.NET.

### Contraintes temporelles

Nous disposions pour réaliser ce projet d’un délai de 9 semaines à compter du 5 Décembre 2011. Le planning prévisionnel a été établi comme suit dans la lettre de mission :

Le projet débute le lundi 5 Décembre 2011 pour prendre fin le vendredi 3 Février 2012 avec la livraison de la solution.

De plus, une soutenance de projet est prévue entre le 8 et le 10 Février 2012.

## Ressources

Pour mener à bien ce projet, nous avons constitué une équipe de deux élèves-ingénieur avec un planning de 30 jours.

Nous avons pu disposer d'une salle de projet fournie par Polytech’ Montpellier et équipée de d’un ordinateurs et de trois écrans. Nous avons également utilisé nos ordinateurs personnels.

M. Fenwick, notre demandeur, nous a également fourni un modem SMS muni d’une carte SIM.

# Déroulement du projet

## Gestion du projet

Le projet a commencé avec le premier entretien que nous avons eu avec le demandeur. À partir de cette discussion et des réponses qu’il nous a fournies, nous avons élaboré notre lettre de mission qui définit le champ de l’étude, le point d’arrivée et les modalités du projet.

Au cours de cet entretien, nous avons également découpé le projet en plusieurs phases.

La première phase consiste à mettre en œuvre l’envoi de SMS à l’aide du modem GSM. La communication avec le modem se faisant à l’aide d’instructions particulières (les **commandes AT**, aussi appelées **commandes Hayes**) via un port **COM**, il nous faut dans un premier temps étudier les possibilités de ce système et le prendre en main.

La deuxième phase consiste à traiter la réception des SMS et des accusés de réception.

Enfin, la troisième phase, la plus longue, consiste à concevoir et à réaliser le service Windows et l’interface graphique en ligne de gestion des messages.

Nous avons utilisé comme support de planification le logiciel Microsoft Project qui permet de planifier automatiquement les tâches en fonction des contraintes de début et de fin, qui prend en compte les jours fériés et les week-ends. Cet outil permet également de suivre le projet notamment grâce aux outils graphiques (diagramme de Gantt par exemple) qu’il met à disposition des utilisateurs.

Lors de l’évaluation des durées des tâches, nous avons dû prendre en compte le temps de prise en main du modem GSM et des commandes AT, ainsi que d’une productivité plus faible durant la période de Noël.

Vous trouverez en annexe le **diagramme de Gantt** du projet.

## Démarche

### Méthodes utilisées

Nous avons utilisé UML pour modéliser la base de données et écrire les cas d’utilisation. Nous n’avons pas utilisé d’autres méthodes particulières, le projet étant principalement concentré sur la conception et le développement.

### Choix technologiques

Les choix technologiques que nous avons à effectuer sont relativement restreints pas les contraintes du projet.

Afin de mettre en place la liaison entre le service, l’interface graphique et la base données, nous avions le choix entre deux bibliothèques logicielles, **LINQ to SQL** et **ADO.Net**. Nous avons synthétisé les avantages et inconvénients dans le tableau suivant :

|  |  |  |
| --- | --- | --- |
| Bibliothèque Logicielle | LINQ to SQL | ADO.Net |
| Mise en place | Rapide | Plus longue |
| Utilisation | Relativement aisée avec le langage LINQ | Nécessite un temps d’apprentissage |
| Perfomance | Problème avec des bases de données complexes | Très bonne |

Notre base de données n’étant pas complexe, nous avons décidé d’utiliser LINQ to SQL, plus rapide à mettre en place et plus facile à manipuler. De plus, nous avions déjà utilisé cette technologie dans un précédent projet.

## Communication et synchronisation

Afin de pouvoir partager rapidement le résultat de nos travaux, nous avons décidé **de réaliser ce projet de la façon la plus collaborative possible**.

Nous avons ainsi utilisé les outils **Google Documents** pour le partage de la documentation, **Github** pour le partage du code et de documents, et **Remember The Milk** pour le partage des tâches.

De plus la possibilité de disposer d’une salle de projet a été une excellente occasion pour travailler ensemble et communiquer directement pendant ces six semaines.

Concernant notre propre communication et synchronisation au sein du groupe, elle a pu se faire facilement puisque nous étions tous deux présents aux mêmes horaires dans une même salle. De plus, nous nous entendons bien.

Ainsi, nous ne nous sommes pas attribués de rôles fixes mais avons alterné. Cela dans le but de manipuler l’ensemble des technologies et de pouvoir prendre du recul.

# Travail réalisé

## Architecture du système d’envoi et de réception de SMS

**Guillaume**

**Placer les technologies, explications plus techniques**

L’architecture du système d’envoi et de réception de SMS peut être représentée par le schéma suivant :

**Plateforme M2M**

**Système d’envoi et de réception de SMS**

**Objets distants**

**Opérateurs**

**Télécoms**



**Utilisateur**

**Applications**

**Interface graphique**

**Base de données**

**Service SMS**

**Modem GSM**

**(SFR, Orange, Bouygues, Free)**

Figure - Schéma d'architecture

Le système d’envoi et réception de SMS est composé de trois parties : le service SMS, la base de données et interface graphique.

### Le service SMS

Le service a pour but d’envoyer et de réceptionner des messages afin de satisfaire les demandes des clients. Pour cela, il va contrôler le modem SMS à travers des commandes AT afin que ce dernier puisse envoyer et recevoir correctement des messages.

De plus, pour des questions de compatibilité avec la plateforme M2M de Synox, il a été décidé que le service SMS prendra la forme d’un service Windowsqui sera à l’écoute de la base de données afin de traiter les demandes de réception et d’envoi à travers un traitement par lot (batch).

### La base de données

La communication entre les clients et le service SMS s’effectue par l’intermédiaire d’une base de données.

En effet, lorsqu’un utilisateur de la plateforme veut envoyer un SMS, les données du nouveau message sont insérées dans la base de données. Ainsi, le service, qui est configuré pour vérifier régulièrement la base de données, va lire les informations relatives au message afin de transmettre les bonnes commandes d’envoi au modem.

Au niveau de la réception des SMS, le service interroge régulièrement le modem pour y récupérer des nouveaux messages afin de les insérer dans la base de données.

### Interface utilisateur

Cette interface permet aux clients d’envoyer et de recevoir des SMS en alimentant ou en consultant la base de données. Elle sera sous forme d’un site web d’administration qui permettra la visualisation des messages envoyés/reçus et elle inclura une interface pour envoyer le SMS avec les options possibles par le format PDU.

Il peut également être envisagé de créer un web Service pour que les applications Synox puissent s’y connecter simplement.

## Conception

Afin de procéder à la phase de conception, nous avons réalisé des **diagrammes de cas d’utilisation** et un **diagramme de classe.**

### Diagrammes des cas d’utilisation

#### Application

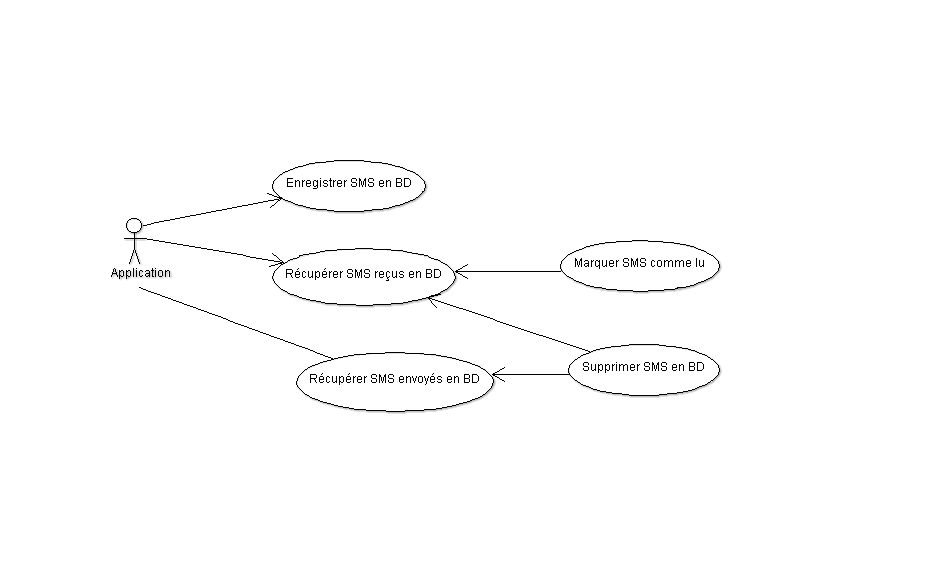


Figure - Diagramme des cas d'utilisation : Application

La communication entre le service chargé de l’envoi et la réception des messages avec les différentes applications se font par l’intermédiaire d’une base de données.

Ainsi, lorsqu’une application veut envoyer un SMS, elle insère toutes les informations concernant le message en base de données. De même, lorsqu’elle a besoin de récupérer les SMS reçus ou envoyés, elle consulte la base de données. L’application peut alors supprimer ou marquer comme lu les messages reçus.

#### F:\Internet\public_space\Polytech\Synox\GitHub\Synox\Docs\Cas utilisation\images\usecase_user.pngUtilisateur

Figure - Diagramme des cas d'utilisation : Utilisateur

Les utilisateurs auront la possibilité d’envoyer et de réceptionner des SMS par l’intermédiaire d’une interface graphique. La consultation des messages se compose de deux parties : les SMS envoyés et ceux reçus par le modem. L’utilisateur pourra les supprimer et marquer comme lu les messages réceptionnés.

### Dictionnaire de données

**Guillinar**

### Diagramme de classe

**Guillinar**

**Explications plus technique**

Après l’étude des besoins fonctionnels et des cas d’utilisation, nous avons pu déterminer de quelles informations nous avions réellement besoin pour le bon fonctionnement de notre application.

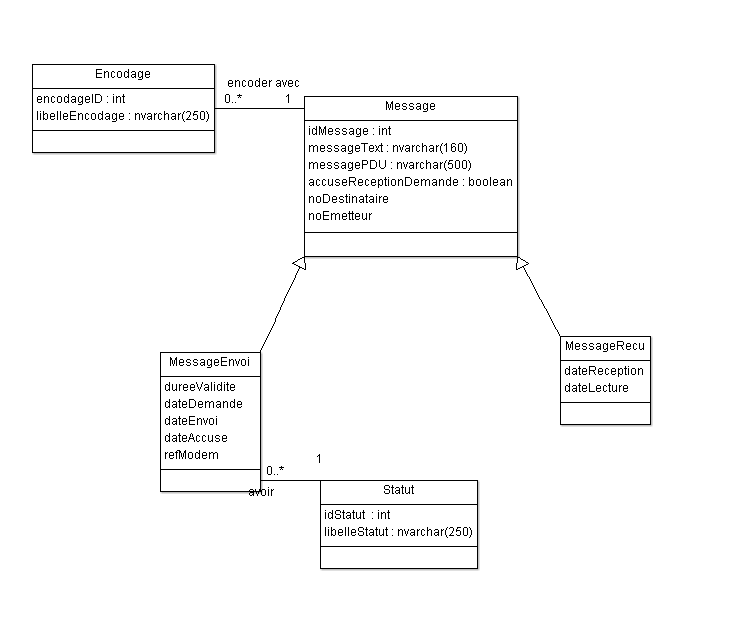


Figure - Diagramme des classes

La classe **Message** regroupe les informations partagées par les classes **MessageEnvoi** et **MessageRecu** qui en héritent. En effet, chaque message est caractérisé par un identifiant unique, un destinataire, un émetteur et un message. L’attribut **accuseReceptionDemande** indique si le destinataire ou l’émetteur a demandé un accusé de réception du message au correspondant.

Le modem peut envoyer et recevoir des SMS au format **PDU**. Ceux-ci sont alors stockés dans le champ **messagePDU**. Si c’est un SMS au format texte, ce champ garde une valeur nulle.

Un **message envoyé** est caractérisé par une durée de validité. Un destinataire ne reçoit pas un SMS dont la période de validité est expirée, ce contrôle étant effectué par l’opérateur de télécommunication. De plus, le fonctionnement du service SMS étant basé sur du traitement par lot, les messages ne sont pas envoyés immédiatement. L’attribut **dateDemande** renseigne donc le moment où l’utilisateur a validé le SMS à envoyer et **dateEnvoi** correspond alors au moment où le message est réellement envoyé par le service. Lorsqu’un SMS est envoyé par le service, une **référence** lui est assignée par le modem. Cette identifiant est le seul moyen de faire correspondre le message envoyé avec un potentiel accusé réception.

Un **statut** est associé à chaque message envoyé. Cela permet de connaître le statut d’un SMS à tout moment, s’il est en attente d’envoi, s’il a été envoyé, s’il a été accusé ou s’il y a eu des erreurs.

Un **message reçu** est caractérisé par une date de réception et une date de lecture qui correspond au moment où le message est lu pour la première fois.

### Maquettage

Afin de s’assurer de la bonne compréhension de la demande, nous avons élaboré des maquettes des différentes interfaces graphiques prévus.

#### L’interface d’envoi



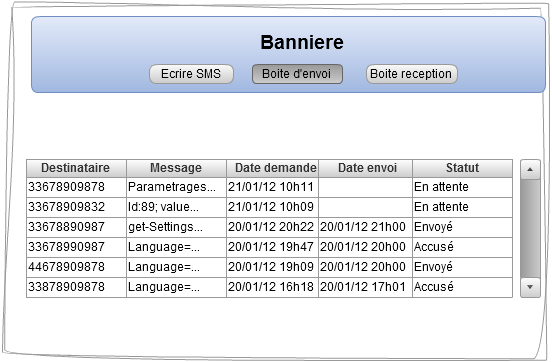
Figure - Maquette de l'interface d'envoi

Cette interface doit permettre aux utilisateurs d’envoyer simplement un SMS.

Le champ « **type Message** » peut prendre la valeur **texte** ou **PDU**.

Avec le type **texte**, il est obligatoire de spécifier un numéro de destinataire et un encodage. Par contre avec le type **PDU**, seul le contenu du message doit être rempli, les autres champs ne sont plus d’aucune utilité. En effet, avec le type PDU, toutes les informations concernant le SMS sont présentes dans le contenu du message.

#### La boîte d’envoi

Figure - Maquette de la boîte d'envoi

Les utilisateurs doivent pouvoir consulter tous les SMS envoyés à travers cette interface. Sur les conseils de notre demandeur, nous avons associé à chaque message une **date d’envoi** et un **statut** afin de pouvoir suivre en temps réel l’état des messages qui ont été envoyés par les utilisateurs.

Lors du développement de cette interface nous avons pertinent d’ajouter **une barre de recherche** au-dessus du tableau contenant les messages afin de filtrer les SMS selon le destinataire, le contenu des messages ou le statut.

De même, afin de permettre aux utilisateurs de supprimer des messages de cette interface, nous avons associé un bouton de **suppression** à chaque ligne du tableau.

### La boîte de réception

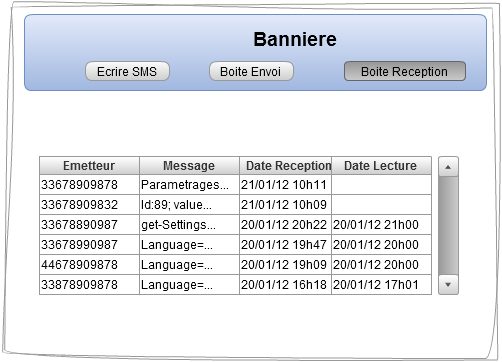


Figure - Maquette de la boîte de réception

Cette interface doit permettre aux utilisateurs de consulter simplement les SMS reçus par le modem GSM. L’implémentation de la colonne **date lecture** nous a été conseillé par notre demandeur, car cela permet de rapidement identifier les messages non lus.

Lors du développement de cette interface, nous avons mis en place une colonne **action** qui permet de **supprimer** un SMS ou de **répondre** à l’émetteur.

## Environnement de travail

### Le Framework .Net

#### Présentation

Le **Framework .Net** a pour but de faciliter la tâche des développeurs en proposant une approche unifiée à la conception d'applications Windows ou Web, tout en introduisant des facilités pour le développement, le déploiement et la maintenance d'applications. Il a besoin d'être installé sur la machine de l'utilisateur final, rendant les applications créées sous cet environnement impropres à un usage portable

Le Framework .NET permet de développer plusieurs types d’applications :

* Les applications consoles
* Les applications graphiques
* Les sites Web
* Les Webservices
* Les services Windows

Une **machine virtuelle,** semblable à celle de Java, est fournie à l’installation du Framework. Elle permet de faire fonctionner les applications développées sous ce Framework. Cependant, cette machine virtuelle ne fonctionne que sous Windows.

#### Le service SMS

Le service SMS est un service Windows qui tourne en tâche de fond afin d’effectuer des opérations d’envoi et de lecture de SMS à des intervalles de temps réguliers. Cela permet d’exécuter ces actions dès le démarrage sans avoir besoin d’interface graphique.

Pour le développement du service, nous nous sommes appuyés sur une application exemple de Synox. En effet, cet exemple comprend deux modes de fonctionnement différents, dépendant si l’application est lancée en mode d’exploitation ou en mode débogage.



Figure - Extrait du code source du lancement du service SMS

Lors d’un lancement en mode exploitation, le service SMS se comporte comme un service Windows. C’est-à-dire qu’une fois installée, son exécution est totalement transparente pour l’utilisateur. Cependant, en cas d’erreurs non traités par le code applicatif, le service s’arrêtera et il faudra une intervention humaine pour le relancer.

Dans le cas d’une exécution en mode débogage, le service SMS se lance comme une application console. Cela permet de voir les exceptions levées sur la console ce qui facilite le débogage. De plus, on peut passer des commandes à l’application pendant son exécution ce qui facilite la mise en place de tests unitaires.

#### L’interface graphique

##### Le modèle client-serveur

Les interfaces utilisateurs du projet ont été développées avec la technologie **ASP.Net** inclue dans le Framework .Net. ASP.Net permet de construire des applications sur le **modèle client-serveur**.

Un **serveur** peut être défini comme un programme offrant un service sur un réseau. Un **client** peut être représenté par un programme qui émet des requêtes (ou demande de service) vers un serveur.

Figure - Modèle client-serveur

**Client**

**Serveur**

**Traitement**

**Requête**

**Réponse**

Le **modèle client-serveur** désigne un mode de communication à travers un réseau où un client envoie des requêtes à un serveur qui est chargé de renvoyer une réponse après un traitement de la demande.

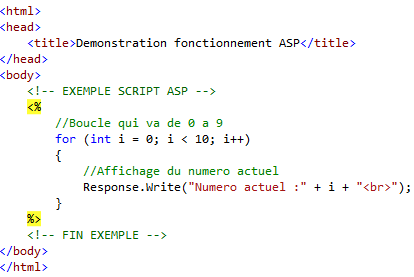
##### La technologie ASP.Net

ASP.Net est à la fois un serveur et un client. En effet, cette technologie permet de gérer dans une seule application les interfaces utilisateur ainsi que leurs interactions avec le serveur.

Pour programmer le côté client, le langage de script **ASP** (Active Page Server) est utilisé. Ce langage permet d’implémenter des scripts très variés, comme par exemple des accès à des bases de données, la génération de contenu HTML ou encore la gestion de fichiers.

L’ASP s’intègre entre les balise « <% %> » d’une page respectant la norme **HTML** et ayant pour extension « **asp** ». Lorsqu’une page ASP est demandée, le serveur interprète les scripts ASP de la page puis génère une réponse au format HTML.

Un exemple de script ASP est présenté ci-dessous :

  
Figure - Exemple de script ASP

Ce script ASP contient une boucle qui affiche successivement des chiffres allant de 0 à 9. Ci-dessous nous pouvons avoir un aperçu du résultat de l’interprétation de ce script par le serveur.

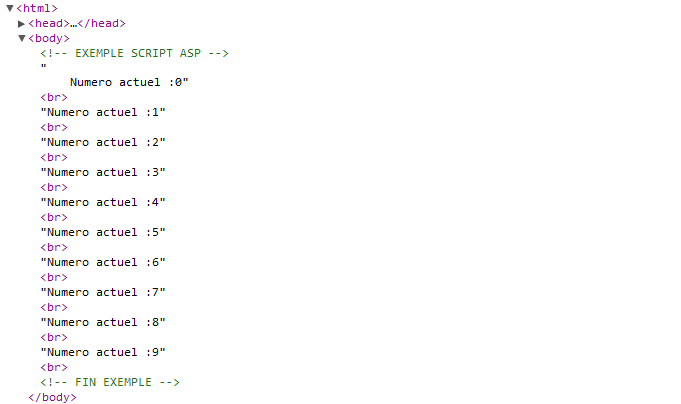


Figure - Résultat de l'interprétation de l'ASP par le serveur

Nous constatons bien que c’est du HTML qui est retourné.

##### Le JavaScript

Le JavaScript est, comme son nom l’indique, un langage de script basé sur le JAVA. Contrairement à l’ASP.Net, le JavaScript s’exécute uniquement au niveau du client.

Ce langage de script est souvent utilisé pour interagir avec les éléments d’une page comme par exemple les champs de saisie ou les boutons. On peut ainsi utiliser du JavaScript pour afficher des boîtes de dialogues ou encore faire disparaître un élément de la page.

Le JavaScript peut s’intégrer dans une page Web respectant le format HTML ou être placé dans un fichier dédié ayant l’extension « **js**».

##### Les évènements

Un évènement est déclenché à la suite d’une action d’un utilisateur. Par exemple, un clic sur un bouton est considéré comme un évènement.

Avec ASP.Net, les évènements sont traités par le serveur. Le langage utilisé pour le traitement est le **C#**. Pour chaque évènement d’une page web il existe une fonction du côté serveur qui définit les fonctions à effectuer.

Ainsi lorsqu’un évènement est déclenché, une fonction spécifique à l’évènement est appelée au niveau du serveur. Ce dernier, effectue alors le traitement définit dans la fonction et retourne une réponse HTML au client.

Dans l’exemple ci-dessous, l’événement onselectedindexchanged est déclenché à chaque fois qu’un élément de la liste déroulante dont l’identifiant est « DropdownEncodage » est sélectionnée.

**Client**

<!-- ENCODAGE -->

<td>Encodage :</td>

<td>

<asp:DropDownList ID="DropDownEncodage"/> onselectedindexchanged="DropDownEncodage\_SelectedIndexChanged"/>

</td>

<!-- FIN ENCODAGE -->

Figure - Exemple d'événement au niveau du client

Lors du déclenchement de l’événement onselectedindexchanged la fonction DropDownEncodage\_SelectedIndexChanged est appelée. Un extrait de cette fonction qui se situe au niveau du serveur est présenté ci-dessous.

Figure - Exemple d'événement au niveau du serveur

**Serveur**

//lorsqu'on choisit un encodage

protected void DropDownEncodage\_SelectedIndexChanged(object sender, EventArgs e)

{

switch (DropDownEncodage.SelectedValue)

{

case "1":

// encodage 7 bits

contenuSMS.MaxLength = 160;

break;

case "2":

// ecodage 8 bits

contenuSMS.MaxLength = 140;

break;

case "3":

// encodage 16 bits

contenuSMS.MaxLength = 70;

break;

case "4":

// Mode PDU, on ne fixe pas de limite

//contenuSMS.MaxLength = 0;

break;

}

contenuSMS.MaxLength = 5;

}

La fonction « **DropDownEncodage\_SelectedIndexChanged** » permet d’ajuster la longueur du contenu du SMS à envoyer en fonction de l’encodage choisi.

##### Schéma du fonctionnement général de l’ASP.Net

Le schéma ci-dessous présente le fonctionnement de l’ASP.Net :

Figure - Schéma du fonctionnement général ASP.Net



**Serveur**

**Client**

Analyseur et compilateur

Environnement d’exécution

**1**

**2**

**3**

1. L’utilisateur demande une nouvelle page web avec l’extension « asp » ou un événement est déclenché.
2. La requête est analysée afin de déterminer les fonctions du serveur ou les scripts ASP à compiler et à exécuter. La compilation s’effectue uniquement lors de la première exécution.
3. Les scripts ASP ou les fonctions C# s’exécutent dans un environnement appelé le runtime. Puis une réponse HTML est renvoyée au client.

### Installation d’un serveur

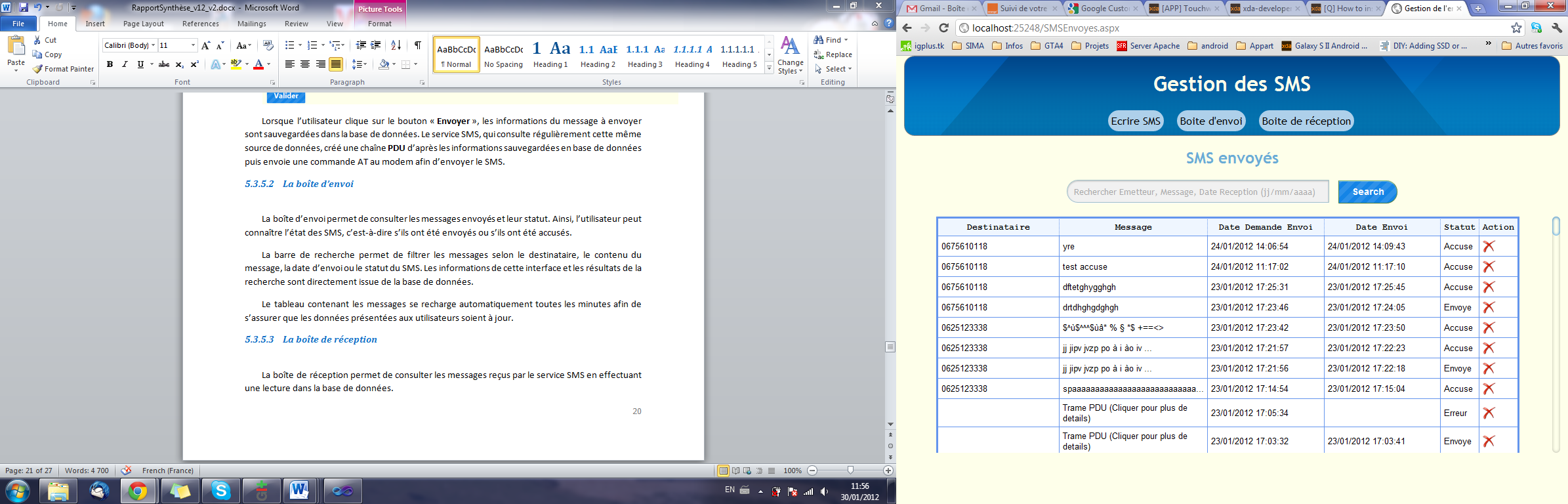
#### SQL Server

SQL Server est un système de gestion de base de données (**SGBD)** développé par Microsoft. C’est un outil indispensable créer et de maintenir une base de données sous un environnement Windows. Ce SGBD figure dans les contraintes imposées par notre demandeur afin de garder une cohérence avec la plateforme M2M déjà existante.

Nous avons trouvé pertinent de mettre en place un serveur de base de données afin que nous puissions travailler sur les mêmes données. En effet, comme le service SMS et l’interface graphique communiquent à travers la base de données, il faudrait que ces deux applications puissent avoir un accès à la base de données quel que soit l’ordinateur sur lequel ils sont exécutés.

De plus, la création de ce serveur nous a beaucoup aidés dans la mesure où nous ne travaillions pas souvent pas sur la même application. Ainsi pour les tests, l’un de nous pouvait insérer les messages à travers l’interface tandis que l’autre binôme exécutait le service SMS sur son ordinateur.

Figure - Schéma d'utilisation du serveur de base de données



**Ordinateur 1**

**Ordinateur 2**

**Serveur**

Modem

Interface graphique

Base de données

Le serveur est un Dell Optiplex 760 équipé d’un processeur double cœur à 3Ghz et de 2Go de RAM. Au du système d’exploitation, nous avons opté pour Windows 7 édition Professionnel car il offre une bonne stabilité et il s’adapte parfaitement à la configuration du serveur.

Au niveau de l’implémentation de SQL Server sur le serveur, nous avons téléchargé un installateur sur le site web de Microsoft que nous avons exécuté avec les droits administrateurs. Cependant, afin d’associer une interface graphique serveur de base de données il a fallu installer SQL Server Management Studio.

Afin de permettre un accès à distance à SQL Server, nous avons configuré le pare feu pour qu’il ne bloque pas les connexions sortantes. De plus, au niveau de SQL Server, il a fallu créer un compte utilisateur ayant des droits spécifique.

#### VNC

**Guillinar**

### Mise en place du modem GSM

**Guillinar**

Parler de l’extension USB

### GitHub

**Guillinar**

Commandes pull/push/ add/ commit

Captures d’écran

### RememberTheMilk

RememberTheMilk est une application web accessible depuis un simple navigateur permettant de maintenir des tâches tant professionnelles que personnelles. Cet outil est réputé pour sa simplicité d’utilisation tout en offrant une multitude fonctionnalités. L’accès à RememberTheMilk est gratuit, il suffit de s’enregistrer.

RememberTheMilk offre une vue synoptique des tâches à effectuer. L’interface utilisateur de cette application est simple et très intuitive. Pour entrer une tâche il suffit de saisir son libelle et d’appuyer sur la touche « Entrée ».

Au niveau des fonctionnalités, il est possible de donner trois niveaux de priorités aux différentes tâches tout en utilisant les touches du clavier



Figure - Les priorités dans RememberTheMilk

Il est également possible de paramétrer différentes informations au niveau de chaque tâche.



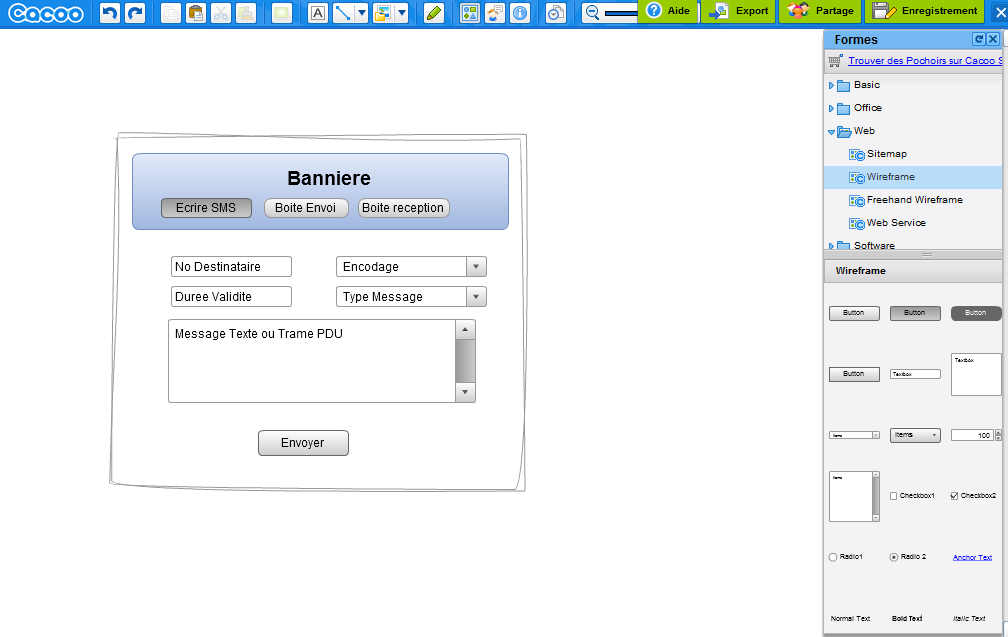
Figure - Les informations sur les tâches dans RememberTheMilk

Ainsi, la date et l’heure à laquelle la tâche doit être effectuée est paramétrable via une date. Il est également possible de spécifier que la tâche doit être compléter pour vendredi prochain. Dans ce cas-là, il suffit de saisir “vendredi prochain” et la date affichée sera celle de vendredi. On peut aussi donner la durée de la tâche, donner des mots clés, ou encore définir un lieu, qui est géré avec l’API de Google Map.

### Cacoo

Cacoo est un outil de dessin en ligne qui permet de créer des diagrammes très variés comme des par exemple, des chartes graphiques, des maquettes, des cartographies réseaux ou encore des diagrammes UML. Cet outil est gratuit, il suffit de s’inscrire sur le site <https://cacoo.com/>.

Nous avons utilisé cet outil afin d’élaborer nos maquettes de l’interface graphique.

Figure - Aperçu de l'outil de dessin Cacco

L’interface utilisateur de Cacoo est très intuitive, l’outil permet de créer des dessins en faisant des glisser-déposer de composants depuis la boîte à outil. De plus, il existe la possibilité de générer une image de la maquette afin de l’exporter.

## Développement

### Les commandes AT

**Guillinar**

Annexe synthèse

Plus en détails

### Le mode PDU

Un modem GSM peut fonctionner selon le mode texte ou le mode **PDU** (Protocol Data Unit). Les commandes AT ci-dessous s’utilisent d’une manière différente dépendant du mode utilisé.

* **AT+CMGS** : Envoie un message
* **AT+CMSS**: Envoie un message depuis la mémoire
* **AT+CMGR** : Lecture d’un message
* **AT+CMGL** : Liste tous les messages du modem
* **AT+CMGW** : Ecrit le message en mémoire

Cependant, pour ce projet, seule la commande permettant l’écriture d’un SMS nous intéresse. Nous avons donc étudié la structure de cette commande en fonction du mode choisi.

Le mode **texte** permet d’envoyer plus facilement des messages car il utilise des commandes AT simples. En effet, il faut juste fournir au modem GSM le numéro du destinataire et le contenu du message à travers une commande AT.

**AT+CMGS="+85291234567"<CR>C’est simple d’envoyer un SMS en mode texte <Ctrl+z>**

Figure - Exemple de commande AT d'envoi de SMS en mode texte

La commande AT ci-dessus permet d’envoyer un SMS dont le contenu est « **C’est simple d’envoyer un SMS en mode texte** » au destinataire « +**85291234567** ». « **CR** » représente le caractère ASCII de la touche « Entrée ». « **Ctr+Z** » correspond au code ASCII 26.

Le mode **PDU** est un peu plus complexe car il utilise des commandes AT contenant de l’*hexadécimal*[[8]](#endnote-8). L’envoi de commandes AT dans ce mode respecte une structure de base.

**AT+CMGS="<taille>"<CR><Chaîne PDU><Ctrl+z>**

Figure - Structure de base d'une commande AT en mode PDU

Dans la commande ci-dessus, « **taille** » correspond à la taille en hexadécimal de la chaîne PDU qui suit. « **Chaîne PDU** » représente le contenu et les paramètres du SMS encodés au format PDU.

Lors de l’envoi d’un SMS en mode PDU il est possible de spécifier les paramètres du message figurant dans le tableau ci-dessous :

|  |  |  |
| --- | --- | --- |
| No du champ | Taille (octet) | Description |
| 1 | 1 | Taille de la chaîne |
| 2 | 1 | Information sur l’élément identificateur |
| 3 | 1 | Taille de l’entête |
| 4 | 1 | Numéro du centre de messagerie |
| 5 | 1 | Le nombre de parties, si le message est trop long |
| 6 | 2 | Numéro de partie, si le message est trop long |
| 7 | 4 | Contenu du message |

Ci-dessous, un exemple d’une commande AT permettant d’envoyer un SMS en mode PDU.

**AT+CMGS="23 "<CR>0011000B916407281553F80000AA0AE8329BFD4697D9EC37<Ctrl+z>**

Figure - Exemple de commande AT d'envoi de SMS en mode PDU

Le tableau suivant permet de décrire la chaîne PDU contenu dans l’exemple :

|  |  |
| --- | --- |
| Octets | Description |
| 00 | Taille du numéro de centre de messagerie. 00 signifie que c’est celui par défaut qui doit être utilisé |
| 11 | Indique si une période de validité est associée au message |
| 00 | Indique que le modem GSM est libre d’attribuer une référence au message |
| 0B | Taille du numéro du destinataire |
| 91 | Indique que le numéro du destinataire est au format international |
| 6407281553F8 | Le numéro du destinataire en hexadécimal |
| 00 | Le type du message (data, texte, flash…) |
| 00 | L’encodage du message, ici 7 bits |
| AA | Période de validité de 4 jours |
| 0A | Longueur du message texte |
| E8329BFD4697D9EC37 | Le message à envoyer, ici « hellohello » |

Comme le système d’envoi et de réception de SMS doit permettre aux utilisateurs de choisir un encodage et une durée de validité et également gérer des accusés réception, nous avons décidé d’envoyer des messages uniquement en mode **PDU**.

### La librairie ATSMS

Lors de l’analyse de la structure des commandes **PDU** nous nous sommes aperçus que la mise en place d’algorithmes de création de commandes PDU allaient nous prendre beaucoup de temps. En effet, pour créer de telles fonctions de conversion, il faut d’une part maîtriser parfaitement le protocole **TPDU** (Transport Protocol Data Unit) et d’autre part mettre en place plusieurs calculs de conversion. Nous nous sommes mis à la recherche d’une librairie permettant une création simple de trames **PDU**.

Ainsi, nous avons intégré la librairie **ATSMS** à notre projet. Cette librairie issue d’un projet open source trouvé sur le site [http://www.csharpfr.com](http://www.csharpfr.com/) permet d’encoder rapidement des messages en trames PDU tout en spécifiant un encodage, une durée de validité et des accusés réception. De plus, étant donné que le code source est accessible, la maintenance et les évolutions sont rendus plus faciles.

Cette libraire est composée d’une classe **SMS** dont les **membres** représentent les différents paramètres présents dans le mode PDU et dont les **fonctions** assurent l’encodage et le décodage des chaînes PDU.

#### Les membres

//membres de la classe SMS issue de la librairie ATSMS

#region Members

protected bool \_moreMessagesToSend; //indique si c'est le dernier message

protected byte \_messageReference; //la reference du message sur le modem GSM

protected string \_phoneNumber; //numero du destinataire /emetteur

protected byte \_dataCodingScheme; //encodage du message

protected byte \_validityPeriod; //periode de validite

protected DateTime \_serviceCenterTimeStamp; //date ou le CMSG recoit le sms

protected string \_userData; //donnees utilisateurs

protected byte[] \_userDataHeader; //entete des donnees utilisateurs

protected string \_message; // message a envoyer

#endregion

Figure - Les membres de la classe SMS

* **moreMessagesToSend** : indique si le SMS actuellement envoyé correspond à la dernière partie d’un message. En effet, lorsqu’un message est trop long, il est possible de le couper en plusieurs parties afin de l’envoyer tout en respectant les normes PDU. Dans le projet actuel, on n’utilise pas cette fonctionalité.
* **messageReference**: lors de la réception ou de l’envoi d’un SMS, le modem GSM associe une référence au message.
* **phonenumber**: si on réceptionne un message PDU, ce membre à l’émetteur. Lors de l’envoi d’un SMS, cela correspond au destinataire.
* **dataCodingScheme**: permet de définir ou lire l’encodage du message. Il y a trois encodages possibles : 7bits, 8bits et 16bits. Dépendant de l’encodage sélectionné, la taille du contenu du SMS varie. Par exemple, avec du 16bits on est limité à 70 caractères alors qu’avec du 7bits, le contenu du message peut contenir 160 caractères. Cependant, seul l’encodage 16bits supporte les caractères spéciaux.
* **validityPeriod :** correspond à la période de validité du message. Sa valeur est obtenue grâce à un algorithme de conversion expliqué dans la partie ci-dessous.
* **serviceCenterTimeStamp**: Correspond à la date à laquelle le SMS est reçu par le centre de service de messagerie. Ce membre est uniquement consultable lors de la réception d’un message. Dans le cas contraire, il a une valeur nulle.
* **userData** : correspond à la chaîne contenant les paramètres utilisateur
* **userDataHeader**: correspond à l’entête de la chaîne contenant les paramètres utilisateur
* **message**: le contenu du SMS à envoyer

#### Utilisation

Afin d’utiliser la librairie ATSMS, il suffit d’instancier une de la classe SMS.

SMS sms = new SMS();

Figure - Exemple d'instanciation de la libraire ATSMS

Pour accéder aux membres de classe, il faut utiliser la notation pointée comme nous montre l’exemple ci-dessous.

//le no du destinaire

sms.PhoneNumber = no;

//contenu du message

sms.Message = message;

//accuse de recepetion

sms.StatusReportIndication = true;

Figure - Exemple d'utilisation des membres de la librairie ATSMS

Pour obtenir une chaîne PDU de l’objet instancié, il faut appeler la fonction « **Compose** » en précisant l’encodage voulu.

//Encodage

switch (typeEncodage)

{

case "7bits":

result = sms.Compose(SMS.SMSEncoding.\_7bit);

break;

case "8bits":

result = sms.Compose(SMS.SMSEncoding.\_8bit);

break;

case "16bits":

result = sms.Compose(SMS.SMSEncoding.UCS2);

break;

}

Figure - Exemple d'utilisation de la fonction d'encodage de ATSMS

Lors de la réception d’un message encodé en PDU, on peut en extraire les informations grâce à la fonction « **Fetch** ».

SMS.Fetch(sms, ref message);

Figure - Exemple d'utilisation de la fontion de décodage de ATSMS

« **message**» représente la chaîne PDU à décoder et « **sms** » correspond à une instance de la classe SMS. La fonction « **fetch** » remplit les membres de l’objet « **sms** » en fonction du contenu de « **message** ».

### Le service SMS

**Guillinar**

Explications et extraits de l’algorithme principal

### Intégration de la base de données

**Guillinar**

Explication Linq

### L’interface graphique

L’interface graphique doit permettre aux utilisateurs humains de la plateforme d’envoyer et de consulter des SMS. Elle communique avec le service SMS par l’intermédiaire de la base de données .Cette interface est sous forme d’un site web et se décompose en trois sous-interfaces : l’envoi des SMS, la consultation de la boîte d’envoi et la consultation des messages reçus.

#### L’envoi de SMS



L’interface d’envoi de SMS contient un formulaire permettant d’envoyer un message en remplissant simplement un formulaire.

La liste déroulante « **mode**» correspond au format dans lequel le message est inséré dans la base de données. Si le mode **texte** est sélectionné, toutes les informations saisies sur le formulaire correspondent à un format texte qui sera ensuite encodé en format **PDU**.

L’encodage correspond au format dans lequel les formations seront encodées. L’utilisateur peut choisir entre 7bits, 8bits et 16bits. En fonction de l’encodage choisi, la longueur maximale du message à envoyer change :

* 7 bits : 160 caractères maximum
* 8 bits : 140 caractères maximum
* 16bits : 70 caractères maximum

Trois champs sont réservés (jour, heure et minute) à  la **période de validité**. Ils ne sont pas obligatoires.

Une case à cocher est associée à l’accusé réception. Lorsque l’utilisateur veut un accusé réception, il coche cette case.

Lorsqu’on appuie sur le bouton « **Envoyé** », le formulaire est transmis au serveur qui se charge de vérifier les données avant de les sauvegarder dans la base de données.

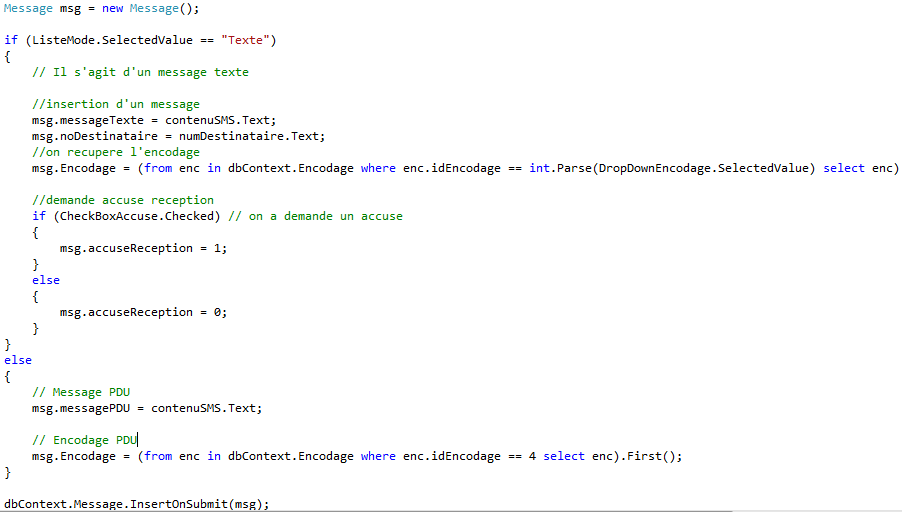


Figure - Extrait de la foction de traitement du formulaire d'écriture

Dans l’extrait ci-dessus, nous pouvons apercevoir que le traitement du formulaire diffère selon si c’est le mode texte ou le mode PDU qui est sélectionné.

Cependant le principe d’insertion dans la base de données reste le même. En effet, dans les deux cas, il faut instancier un objet **Message** et **MessageEnvoi** afin d’y mettre les informations récupérées du formulaire. Puis, à travers l’interface LINQ, on sauvegarde les deux objets précédemment créés dans la base de données.

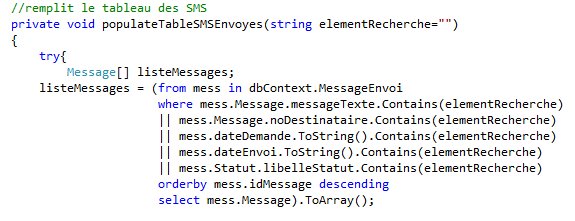
#### La boîte d’envoi

La boîte d’envoi permet de consulter les messages envoyés et leur statut. Ainsi, l’utilisateur peut connaître l’état des SMS, c’est-à-dire s’ils ont été envoyés ou s’ils ont été accusés.



Figure - Capture d'écran de la boîte d'envoi

Lorsque l’utilisateur demande le chargement de la page, le serveur va chercher dans la base de données tous les messages qui sont envoyés ou qui doivent être envoyés par le modem GSM.

Figure - Extrait du code applicatif qui génère le tableau des SMS

La variable **elementrecherche**  correspond au terme qu’on recherche dans les messages. En effet, c’est la même fonction qui remplit le tableau à l’initialisation de la page qui est appelé lors de la recherche. Lors de l’ouverture de la page, cette variable a pour une chaîne de caractère vide.

Le tableau contenant les messages se rafraîchit toutes les minutes. Pour mettre en place se rafraîchissement automatique, nous avons placé le tableau dans un **UpdatePanel** qui permet de rafraîchir tous les composants qu’il contient sans pour autant recharger toute la page.

Il a également fallu implémenter un timer qui se déclenche toutes les minutes dans ce même **UpdatePanel**. Ces deux éléments suffisent à mettre en place le rafraîchissement automatique.

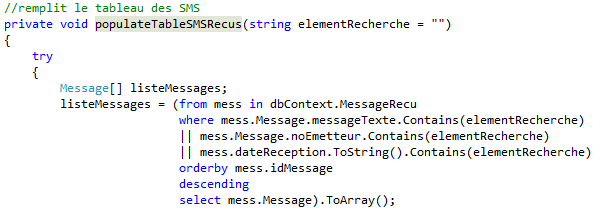
#### La boîte de réception

La boîte de réception permet de consulter les messages reçus par le service SMS en effectuant une lecture dans la base de données.

Figure - Capture d'écran de la boîte de réception

L’implémentation de l’interface de réception des SMS est similaire à celle de la consultation des messages envoyés. En effet, à part le contenu du tableau SMS qui change, nous nous sommes appuyés sur la même base que l’interface précédente.

Au niveau de la récupération des données, on sélectionne maintenant les messages qui sont reçus par le modem.

Figure - Extrait du code applicatif permettant d'extrait les messages à lire

Il est à noter que dans la colonne **Action** on peut désormais répondre à l’utilisateur. Pour cela, nous avons appelé l’interface d’écriture des SMS en lui passant le numéro de l’émetteur en paramètre.

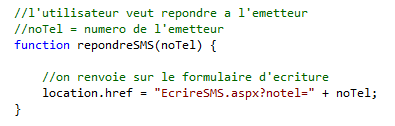


Figure - Extrait du code applicatif permettant de répondre à l'émetteur

## Tests

Tout au long du développement, nous avons effectué **plusieurs niveaux de tests**. En premier lieu, chaque fonction a été testée individuellement, afin de vérifier qu’elle retourne les résultats attendus, et qu’elle traite correctement les cas d’erreurs et les cas limites.

Nous avons également fait d’importants **tests de fiabilité et d’efficacité** sur le service SMS et le modem GSM. En effet, le service SMS est amené à devoir s’exécuter sur de longues durées sans intervention humaine possible en cas de problème. Il faut donc que le service SMS ne s’arrête pas inopinément pour une raison ou une autre, et doit redémarrer en cas de problème. Nous avons donc laissé le script s’exécuter des jours durant, ce qui nous a permis de mettre en évidence un problème récurrent : la communication entre la base de données est régulièrement rompue. Nous avons alors mis en place un **système relançant automatiquement la connexion et reprenant l’opération qui était en cours** quand le problème est survenu.

Nous avons réutilisé ce système de reprise sur erreurs pour tous les problèmes plus rares liés au réseau, comme les réponses incomplètes que renvoient parfois le modem.

Nous avons également voulu tester la capacité du modem à traiter de nombreux messages en même temps, que ce soit en réception ou en envoi, avec ou sans accusés de réception. C’est ainsi que nous avons mis en évidence les limites du modem, développées dans la partie 5.5.

L’interface graphique et les algorithmes liés ont aussi été longuement testés. Par exemple, en vérifiant le comportement de l’application lorsque l’utilisateur rentre des données aberrantes (numéro de téléphone contenant des lettres, durée de validité négative, etc.).

## Limites

Cette solution, telle que nous l’avons conçu selon les demandes du client et les contraintes matérielles, possède cependant certaines limites.

La première concerne l’interface. Celle-ci ne possède en effet aucun système de session et est donc mono-utilisateur. Si cette interface est amenée à être utilisée par plusieurs personnes, il serait alors indispensable de mettre en place un tel système, en n’affichant que les messages destinées et envoyées à ou par une certaine personne.

La seconde limite se situe au niveau du modem, qui n’est pas toujours capable de gérer correctement une importante quantité de messages. Il arrive régulièrement que certains messages soient envoyés en double, ou même jamais transmis au destinataire. Mais les problèmes les plus récurrents concernent les accusés de réception, qui n’arrivent que rarement lorsqu’ils sont nombreux à être attendus.

# Améliorations possibles

## Permettre une utilisation multi-utilisateurs

Le service SMS et la base de données ne permettent pas actuellement une utilisation avec plusieurs utilisateurs.

Pour implémenter une telle fonctionnalité, il faut mettre en place un système d’identifiants unique dans la base de données afin de pouvoir identifier les utilisateurs.

Au niveau du service SMS, il faut que les messages reçus par le modem puissent être identifiés et être redirigés vers les utilisateurs concernés. Pour cela, il faudrait que l’identifiant unique de l’utilisateur figure dans le contenu du SMS reçu.

## Sécuriser l’accès à la base de données

L’interface graphique communique directement avec la base de données pour y insérer les messages à envoyer et pour consulter les SMS. Pour rappel, l’interface créée n’est qu’un exemple afin de montrer les possibilités liées au service SMS. Cependant, il est aussi possible qu’une application puisse interagir directement avec la base de données pour envoyer ou recevoir des SMS.

Or, avec le modèle de fonctionnement actuel où les utilisateurs ont un accès total à la base de données, des problèmes peuvent survenir surtout en mode multi-utilisateurs. En effet, si les modifications de la base de données ne sont pas contrôlées cela peut créer des incohérences ou des pertes d’informations. Par exemple, actuellement, il n’y a rien qui empêche un utilisateur de supprimer tous les messages de la base de données, même ceux ne le concernant pas.

Afin d’éviter de telles problèmes, il faut mettre en place un **web service** qui garantit un accès à distance contrôlé à la base de données à travers des fonctions.

## Implémenter une gestion des contacts

Afin de pouvoir identifier rapidement l’auteur d’un message reçu, il serait intéressant d’implémenter une gestion des contacts par rapport à chaque utilisateur. Ainsi, l’interface utilisateur affichera le nom du contact à la place de son numéro de téléphone. Cette implémentation concerne surtout l’interface graphique et la base de données.

# Difficultés rencontrées

## Apprentissage des technologies

Pendant ce projet, nous avons été amenés à manipuler des technologies que nous ne connaissions pas.

Tout d’abord, il a fallu que nous comprenions l’intérêt et la syntaxe des différentes commandes AT afin de cerner celles que nous devions utiliser afin de communiquer avec le modem GSM.

De plus, même si nous avions déjà réalisé des projets avec le Framework .Net, nous n’avions jamais mis en place un protocole de communication avec cette technologie. De même, l’aspect Web de ce Framework nous était inconnu.

L’utilisation de ces différentes technologies a induit que nous devions assimiler rapidement leur fonctionnement afin d’être productif le plus tôt possible.

## Problèmes de compatibilité entre le Framework .Net et le JavaScript

Lors de la réalisation de l’interface graphique, nous avons été confrontés à plusieurs problèmes au niveau de la compatibilité entre la partie Web du Framework .Net et le *JavaScript[[9]](#endnote-9)*.

Ce problème de compatibilité vient surtout du fait que le Framework .Net possède ses propres composants Web, comme par exemple les champs de saisie ou les boutons. Leur utilisation facilite la gestion des évènements et la récupération des données depuis les champs de saisie.

Cependant, les composants du Framework .Net n’ont pas la même structure que les composants HTML. Or, le JavaScript est conçu pour fonctionner avec la structure des objets HTML.

Ainsi, dans certains cas le code JavaScript que nous avions écrit ne marchait pas. Par exemple, lorsque nous avons voulu améliorer le rendu visuel du formulaire d’envoi avec la librairie externe JQuery écrite en JavaScript, le gestionnaire d’erreurs nous indiquait que des éléments de la page étaient introuvables.

## Les accusés de réception

Pendant l’élaboration du service SMS, nous avons rencontré des difficultés avec la réception des accusés. En effet, il arrivait que modem GSM ne reçoive aucun accusés pendant quelques heures avant de reprendre un fonctionnement normal.

Après une longue série de tests nous nous sommes aperçus que le problème a deux origines distinctes. La première concerne une mauvaise configuration du modem GSM qui n’est alors pas configuré pour recevoir des accusés. La deuxième origine provient du fait que lorsque le modem reçoit un trop grand nombre d’accusés de réception, il arrive qu’il y ait une saturation et que certains accusés soient perdus.

# Conclusion

Ce projet de deux mois a été **riche en expériences** sur plusieurs niveaux. Tout d’abord, cette mission était pour nous un certain défi sur le plan technique. Nous n’avions en effet que peu d’expérience dans le domaine du réseau, et nous avons également eu l’occasion d’approfondir nos connaissances du Framework .Net.

De même, nous avons pu tester notre aptitude à être responsables de projet, c’est-à-dire estimer les charges des différentes phases, gérer nos plannings et prendre des décisions. C’est réellement la meilleure façon de comprendre un tel métier et d’apprendre de nos erreurs.

Par la suite, ce projet nous a également permis de développer notre aptitude à travailler en équipe. Nous avons donc pu **mettre en application** nos capacités d’écoute, d’ouverture d’esprit, de sens relationnel ainsi que d’organisation et d’analyse.

Nous pouvons dire que nous avons **atteint les objectifs définis** dans la lettre de mission. En outre, nous avons identifié quelques améliorations qu’il serait intéressant de mettre en œuvre afin de perfectionner le système.

Les technologies mobiles sont très demandées aujourd’hui avec la forte présence d’objets mobiles communicants comme les smartphones ou les machines embarquées. Il nous semble que ce domaine prend une place de plus en plus importante dans l’informatique, et qu’il sera inévitable pour nous, en tant que futurs ingénieurs en informatique, de s’y intéresser afin de suivre l’évolution technologique.

Finalement, que ce soit sur nos réussites ou nos difficultés, ce projet industriel reste une étape enrichissante dans cette dernière année de cycle d’ingénieur.

# Annexes

## Les commandes AT

### Choix du format des messages

La commande **AT+CMGF** permet d’indiquer au modem le format des messages. En effet, il existe 2 formats possibles : le format texte, utilisant les caractères ASCII, et le format PDU, en hexadécimal (caractères de 0 à 9 et de A à F).

En mode texte, les informations sont directement compréhensibles par l’utilisateur, alors que le mode PDU nécessite un décodage. Par ailleurs, le mode PDU offre plus de possibilités que le mode texte.

### Choix de la mémoire

La commande **AT+CPMS** permet de choisir la mémoire avec laquelle le modem communique. Il y a 3 types de mémoires :

* SM : Les messages stockés sur la carte SIM
* SR : La mémoire des accusés de réceptions, sur la carte SIM, limité à 50 accusés
* ME : La mémoire Flash

### Affichage des messages

La commande **AT+CMGL** permet d’afficher tous les messages stockés sur la mémoire choisit avec la commande AT+CPMS. Ainsi, si la mémoire SM est sélectionnée, la commande AT+CMGL affiche les messages de la carte SIM. Un paramètre permet de spécifier l’état des messages demandés (reçus, lus et/ou non lus, tous).

### Envoi d’un message

La commande **AT+CMGS** permet d’envoyer un message. La procédure change selon le mode choisit avec la commande AT+CMGF.

En mode texte, il faut d’abord spécifier le numéro du destinataire, puis entrer le message en langage « humain ». En mode PDU, on spécifie d’abord la longueur de la trame PDU, puis la trame PDU elle-même. Le numéro du destinataire, entre autres informations, est déjà compris dans la trame et sera utilisé par le modem.

## Envoyer et recevoir des données

Afin d’envoyer des commandes AT et de recevoir des réponses avec le modem, il faut mettre en place un protocole de communication.

Le modem SMS communique avec le service SMS par l’intermédiaire d’un **port série**. Nous avons ainsi utilisé des fonctions du Framework .Net qui nous permettent d’envoyer et de réceptionner des données simplement sur un port de communication.

Le protocole de communication que nous avons mis en place repose sur quatre fonctions :

* Connexion au modem
* Envoi de données
* Réception de données
* Déconnexion

Chaque commande AT envoyé au modem fait appel à ces quatre fonctions.

### Connexion au modem

Avant de pouvoir envoyer des commandes AT au modem, il faut d’abord initialiser une connexion avec le modem. Pour cela, nous avons utilisé la classe **SerialPort** du Framework .Netqui nous permet d’instancier un objet représentant la liaison avec le modem.

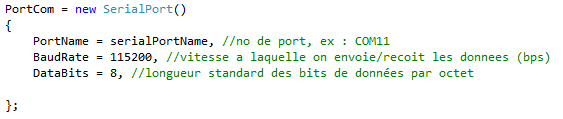


Figure - Extrait de l'algorithme de connexion au modem GSM

Lors de l’établissement de la connexion, nous devons spécifier le **numéro** du port utilisé, la **vitesse** à laquelle les données sont échangées et la **quantité** d’informations contenu dans chaque octet.

Une fois un objet de la classe **SerialPort** instancié, il suffit d’appeler la méthode «**Open()** » sur cette objet pour établir une connexion avec le modem.

### Envoi de données au modem

L’envoi de données au modem se fait exclusivement à travers des **commandes AT**. Il faut qu’une connexion soit préalablement établie auprès du modem.

Pour transmettre des commandes, il suffit d’écrire des octets sur l'objet **SerialPort** à l’aide des données d’une **mémoire tampon**.

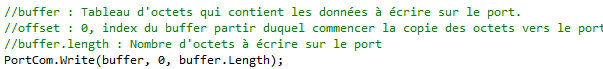


Figure - Extrait de l'algorithme d'envoi de commandes au modem GSM

Cette mémoire tampon permet d’éviter de saturer le modem si la commande de sortie est trop longue. En effet, si le nombre d'octets dans la mémoire tampon de sortie est trop important l'objet **SerialPort** va attendre que le périphérique soit prêt à accepter plus de données.

### Réception de données

Pour recevoir des données du modem, il faut préalablement avoir envoyé des données au périphérique GSM. En effet, le modem les données envoyées par le modem correspondent uniquement à des réponses de **commandes AT**.

La réception de données s’effectue en lisant continuellement les données présentes sur la mémoire tampon du modem jusqu’à que le message reçu soit complet.

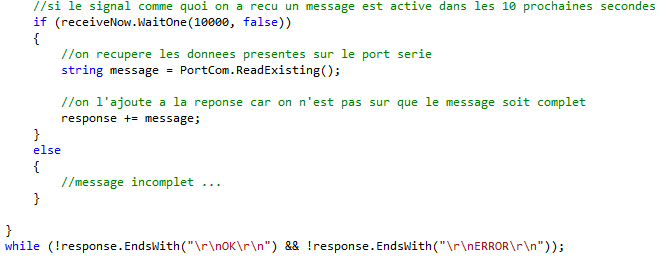


Figure - Algorithme de réception des messages dyu modem GSM

Un message est considéré comme complet lorsqu’il se termine par « **OK** » ou « **ERROR** » .

La fonction **receiveNow.Wait** retourne vrai si le modem reçoit des données avant que le **timeout** spécifié n’expire. Son utilisation a pour but d’éviter une attente active et également d’empêcher un blocage du programme en cas de réponse incomplète.

### Déconnexion du modem

Lorsque le modem n’est plus utilisé par le service, il faut le déconnecter pour éventuellement permettre à d’autre programme d’utiliser cette ressource. Pour cela, il faut tout simplement appeler la fonction « **Close()** »sur l’objet **SerialPort** instancié lors de la connexion.

### Exemple d’utilisation du protocole de communication

Le schéma ci-dessous montre un exemple de l’utilisation du protocole de communication lorsque la commande « **AT+CPMS ?**» est envoyée au modem. Cette commande permet de connaître le type de mémoire sur laquelle les messages sont stockés.

Service SMS

Modem GSM

1.Connexion

2. Envoi commande “**AT+CPMS?**”

3.Demande réception données

4.Réponse du modem : “**SM**”

5.Déconnexion

Figure - Protocole de communication entre le service SMS et le modem GSM

« **SM** » signifie que les messages sont stockés sur la carte SIM du modem.

## Diagramme de Gantt

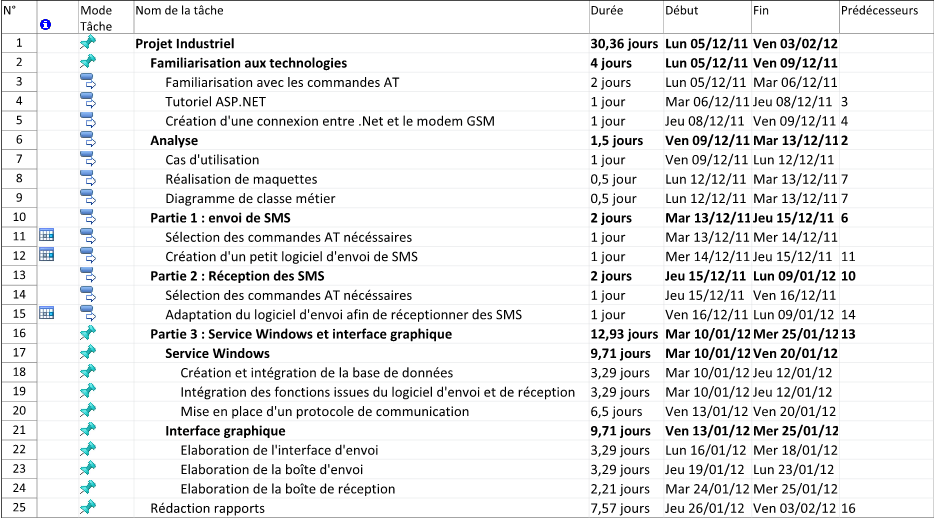


Figure - Diagramme de Gantt (tableau)

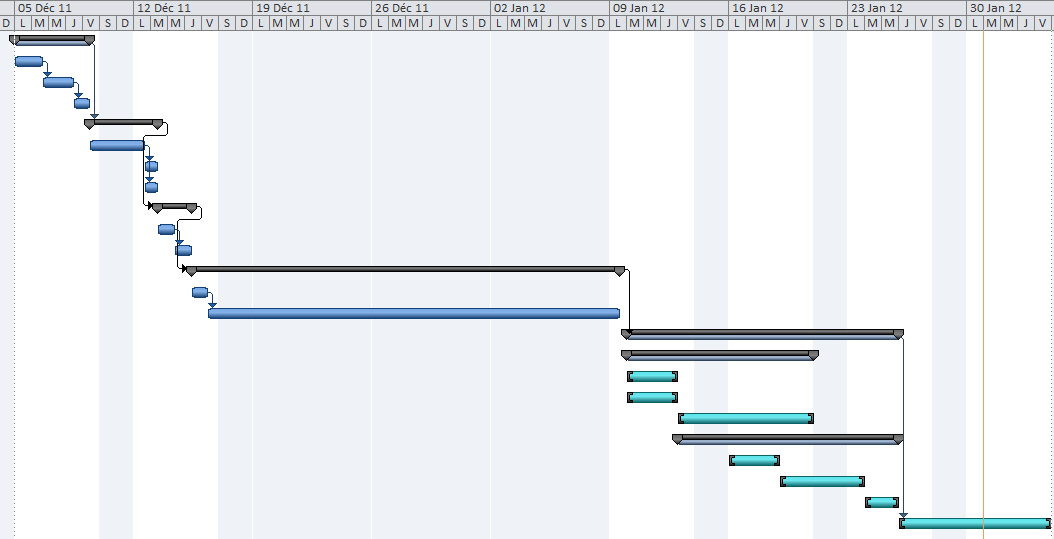


Figure - Diagramme de Gantt (schéma)

Glossaire

1. ***Objets distants :***

   Objets avec un système embarqué communiquant avec la plateforme. Ils envoient des informations qui seront interprétées par des applications Synox ou des utilisateurs. [↑](#endnote-ref-1)
2. [↑](#endnote-ref-2)
3. ***GSM – Global System for Mobile Communications :***

   Réseau de deuxième génération pour la téléphonie mobile. [↑](#endnote-ref-3)
4. ***SMS – Short Message Service :***

   Service permettant de transmettre de courts messages sur le réseau GSM. [↑](#endnote-ref-4)
5. ***Modem – Modulateur-démodulateur :***

   Périphérique servant à communiquer avec des utilisateurs distants. [↑](#endnote-ref-5)
6. ***Service Windows :***

   Logiciel s’exécutant au démarrage de l’ordinateur, en arrière-plan, ne nécessitant pas d’intervention de l’utilisateur. [↑](#endnote-ref-6)
7. ***Framework.Net :***

   Environnement de travail Microsoft, fournissant des fonctions permettant de concevoir des applications lourdes ou légères. Cet environnement possède un compilateur intégré afin de pouvoir tester et débugger ces applications. [↑](#endnote-ref-7)
8. ***Hexadécimal :***

   Système de numération en base 16. Il utilise 16 symboles : les chiffres de 0 à 9 et les 6 premières lettres de l’alphabet, A à F. [↑](#endnote-ref-8)
9. ***JavaScript :***

   Langage de script qui permet des interactions avec les différents objets d’une page Web [↑](#endnote-ref-9)