



**POLYTECH<sup>®</sup>**  
**2011-2012**

# Mise en place d'un service SMS

## Rapport Technique



Guillaume Leroy  
Emmanuel Yagapen  
Polytech' Montpellier  
pour Synox représenté par  
Jérôme Fenwick & Joffrey Verdier

# Remerciements

---

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

- M. Jérôme Fenwick et M. Joffrey Verdier, de l'entreprise demandeuse Synox, pour 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.....	Error! Bookmark not defined.
2	Présentation de l'environnement .....	Error! Bookmark not defined.
2.1	Le groupe SYNOX.....	Error! Bookmark not defined.
2.2	La plateforme Machine-to-Machine de gestion d'objets communicants .....	Error! Bookmark not defined.
	<b>Bookmark not defined.</b>	
3	Présentation du projet .....	Error! Bookmark not defined.
3.1	Le problème initial.....	Error! Bookmark not defined.
3.2	Les besoins fonctionnels.....	Error! Bookmark not defined.
3.3	La mission .....	Error! Bookmark not defined.
3.4	Contraintes .....	Error! Bookmark not defined.
3.4.1	Contraintes techniques .....	Error! Bookmark not defined.
3.4.2	Contraintes temporelles.....	Error! Bookmark not defined.
3.5	Ressources.....	Error! Bookmark not defined.
4	Déroulement du projet.....	Error! Bookmark not defined.
4.1	Gestion du projet.....	Error! Bookmark not defined.
4.2	Démarche .....	Error! Bookmark not defined.
4.2.1	Méthodes utilisées .....	Error! Bookmark not defined.
4.2.2	Choix technologiques .....	Error! Bookmark not defined.
4.3	Communication et synchronisation .....	Error! Bookmark not defined.
5	Travail réalisé.....	12
5.1	Architecture du système d'envoi et de réception de SMS.....	12
5.1.1	Le service SMS .....	13
5.1.2	La base de données .....	13
5.1.3	Interface utilisateur.....	13
5.2	Conception .....	13
5.2.1	Diagrammes des cas d'utilisation.....	13
5.2.2	Dictionnaire de données .....	15
5.2.3	Diagramme de classe.....	15
5.2.4	Maquettage .....	17
5.2.5	La boîte de réception.....	19
5.3	Environnement de travail.....	19
5.3.1	Le Framework .Net .....	19

5.3.2	Installation d'un serveur.....	25
5.3.3	Mise en place du modem GSM .....	27
5.3.4	GitHub.....	28
5.3.5	RememberTheMilk.....	29
5.3.6	Cacoo .....	30
5.4	Développement.....	31
5.4.1	Les commandes AT.....	31
5.4.2	Envoyer et recevoir des données .....	31
5.4.3	Le mode PDU .....	34
5.4.4	La librairie ATSMS.....	36
5.4.5	Intégration de la base de données.....	38
5.4.6	Le service SMS .....	40
5.4.7	L'interface graphique .....	48
5.5	Tests.....	52
5.6	Limites .....	52
6	Améliorations possibles.....	54
6.1	Permettre une utilisation multi-utilisateurs.....	54
6.2	Sécuriser l'accès à la base de données.....	54
6.3	Implémenter une gestion des contacts.....	54
7	Difficultés rencontrées .....	55
7.1	Apprentissage des technologies.....	55
7.2	Problèmes de compatibilité entre le Framework .Net et le JavaScript.....	55
7.3	Les accusés de réception.....	55
8	Conclusion .....	56
9	Annexes .....	57
9.1	Diagramme de Gantt .....	57

## Table des images

---

Figure 1 - Schéma d'architecture .....	12
Figure 2 - Diagramme des cas d'utilisation : Application .....	14
Figure 3 - Diagramme des cas d'utilisation : Utilisateur.....	14
Figure 4 - Diagramme des classes .....	16
Figure 5 - Maquette de l'interface d'envoi.....	17
Figure 6 - Maquette de la boîte d'envoi.....	18
Figure 7 - Maquette de la boîte de réception .....	19
Figure 8 - Extrait du code source du lancement du service SMS .....	20
Figure 9 - Modèle client-serveur .....	21
Figure 10 - Exemple de script ASP .....	22
Figure 11 - Résultat de l'interprétation de l'ASP par le serveur.....	22
Figure 12 - Exemple d'événement au niveau du client .....	23
Figure 14 - Schéma du fonctionnement général ASP.Net.....	24
Figure 15 - Schéma d'utilisation du serveur de base de données.....	25
Figure 16 - Fenêtre de connexion de VNC Viewer .....	26
Figure 17 - L'écran de connexion du serveur dans VNC Viewer .....	27
Figure 18 - Le modem GSM dans le gestionnaire des périphériques .....	27
Figure 19 - La console de Git .....	28
Figure 20 - Les priorités dans RememberTheMilk .....	29
Figure 21 - Les informations sur les tâches dans RememberTheMilk .....	29
Figure 22 - Aperçu de l'outil de dessin Cacco.....	30
Figure 23 - Extrait de l'algorithme de connexion au modem GSM .....	32
Figure 24 - Extrait de l'algorithme d'envoi de commandes au modem GSM .....	32
Figure 25 - Algorithme de réception des messages dyu modem GSM .....	33
Figure 26 - Protocole de communication entre le service SMS et le modem GSM .....	34
Figure 27 - Exemple de commande AT d'envoi de SMS en mode texte .....	34
Figure 28 - Structure de base d'une commande AT en mode PDU.....	35
Figure 29 - Exemple de commande AT d'envoi de SMS en mode PDU.....	35
Figure 30 - Les membres de la classe SMS .....	36
Figure 31 - Exemple d'instanciation de la librairie ATSMS.....	37
Figure 32 - Exemple d'utilisation des membres de la librairie ATSMS.....	37
Figure 33 - Exemple d'utilisation de la fonction d'encodage de ATSMS.....	38
Figure 34 - Exemple d'utilisation de la fontion de décodage de ATSMS.....	38
Figure 35 - Création des classes LINQ to SQL.....	39
Figure 36 - La base de données dans l'Explorateur de serveurs de Visual Studio .....	39
Figure 37 - Modèle logique de données dans Visual Studio .....	40
Figure 38 - Récupération d'objets en LINQ .....	40
Figure 39 - Modification d'un objet en LINQ.....	40
Figure 40 - Validation des modifications en LINQ.....	40
Figure 41 - Principales variables du service SMS .....	41
Figure 42 - Algorithme d'initialisation, de démarrage et d'arrêt du service SMS.....	42
Figure 43 - Schéma descriptif de l'algorithme du timer de lecture .....	43

Figure 44 - Fragment de la fonction EnvoyerSMS (message texte et gestion des erreurs) .....	44
Figure 45 - Schéma descriptif de l'algorithme du timer de lecture .....	45
Figure 46 - Enregistrement d'un nouveau message reçu.....	46
Figure 47 - Traitement des accusés de réceptions.....	46
Figure 48 - Fonction de vidage de la mémoire.....	47
Figure 49 - Projet d'installation du service.....	47
Figure 50 - Lancement du service .....	47
Figure 51 - Formulaire d'envoi de SMS .....	48
Figure 52 - Extrait de la fonction de traitement du formulaire d'écriture .....	49
Figure 53 - Capture d'écran de la boîte d'envoi .....	50
Figure 54 - Extrait du code applicatif qui génère le tableau des SMS.....	50
Figure 55 - Capture d'écran de la boîte de réception .....	51
Figure 56 - Extrait du code applicatif permettant d'extraire les messages à lire.....	51
Figure 57 - Extrait du code applicatif permettant de répondre à l'émetteur .....	52
Figure 58 - Diagramme de Gantt (tableau) .....	57
Figure 59 - Diagramme de Gantt (schéma) .....	57

# 1 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*<sup>i</sup> à travers le réseau *GPRS*<sup>ii</sup>. Cependant ce réseau présente des problèmes de fiabilité compromettant le bon fonctionnement de la plateforme. Le réseau *GSM*<sup>iii</sup> sur lequel s'appuie la technologie *SMS*<sup>iv</sup> 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 souhaitons 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.

## 2 Présentation de l'environnement

### 2.1 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. Il optimise pour ses clients leur organisation interne, les relations de travail, la transmission de l'information et la productivité globale.

Créée en 2003, SYNOX compte aujourd'hui 53 employés à travers quatre sites en France.

En 2011 son chiffre d'affaires a atteint plus 2,5 millions d'euros porté par la bonne santé du **cloud computing**. L'objectif du groupe en 2012 est d'atteindre un chiffre d'affaires de 5,5 millions d'euros.

### 2.2 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 aux clients accès à un suivi de leurs objets distants.

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.

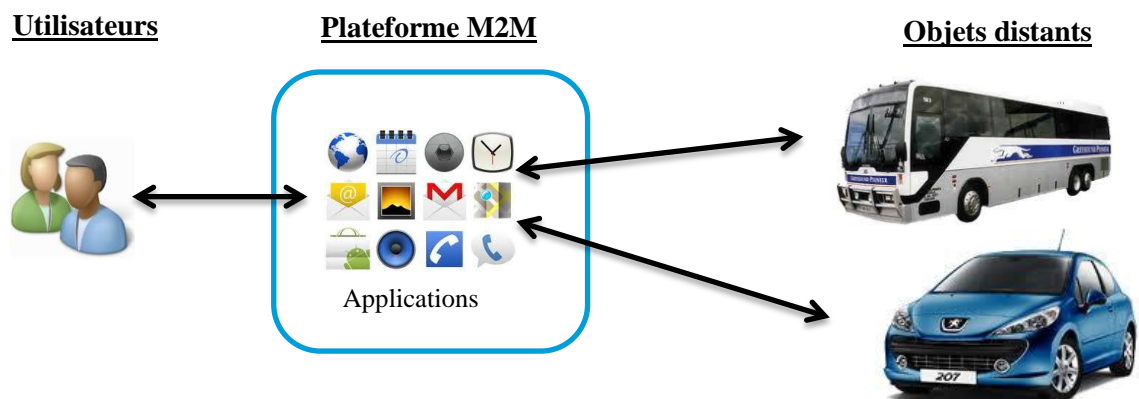


Figure 1 - Fonctionnement de la plateforme M2M

De même, on peut imaginer une aide à la conduite grâce à une technologie qui capte les variations de consommation de carburant selon la manière de conduire (accélérations ou freinages brusques) et les envoie à une application de la plateforme qui conseille ensuite le conducteur.



## 3 Présentation du projet

---

### 3.1 Le problème initial

Les objets de la plateforme M2M du groupe Synox utilisent 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.

### 3.2 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 et de réceptionner 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*<sup>v</sup> 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.

### 3.3 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*<sup>vi</sup>
- 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.

### 3.4 Contraintes

#### 3.4.1 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*<sup>vii</sup> 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.

### **3.4.2 Contraintes temporelles**

Nous disposons 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.

## **3.5 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 d'un ordinateur 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.

## 4 Déroulement du projet

---

### 4.1 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 pris en compte le temps de prise en main du modem GSM et des commandes AT, ainsi qu'une productivité plus faible durant la période de Noël.

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

### 4.2 Démarche

#### 4.2.1 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.

#### 4.2.2 Choix technologiques

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

Afin de mettre en place la liaison entre le service, l'interface graphique et la base données, nous avons 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
Performance	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 avons déjà utilisé cette technologie dans un précédent projet.

### 4.3 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.

## 5 Travail réalisé

### 5.1 Architecture du système d'envoi et de réception de SMS

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

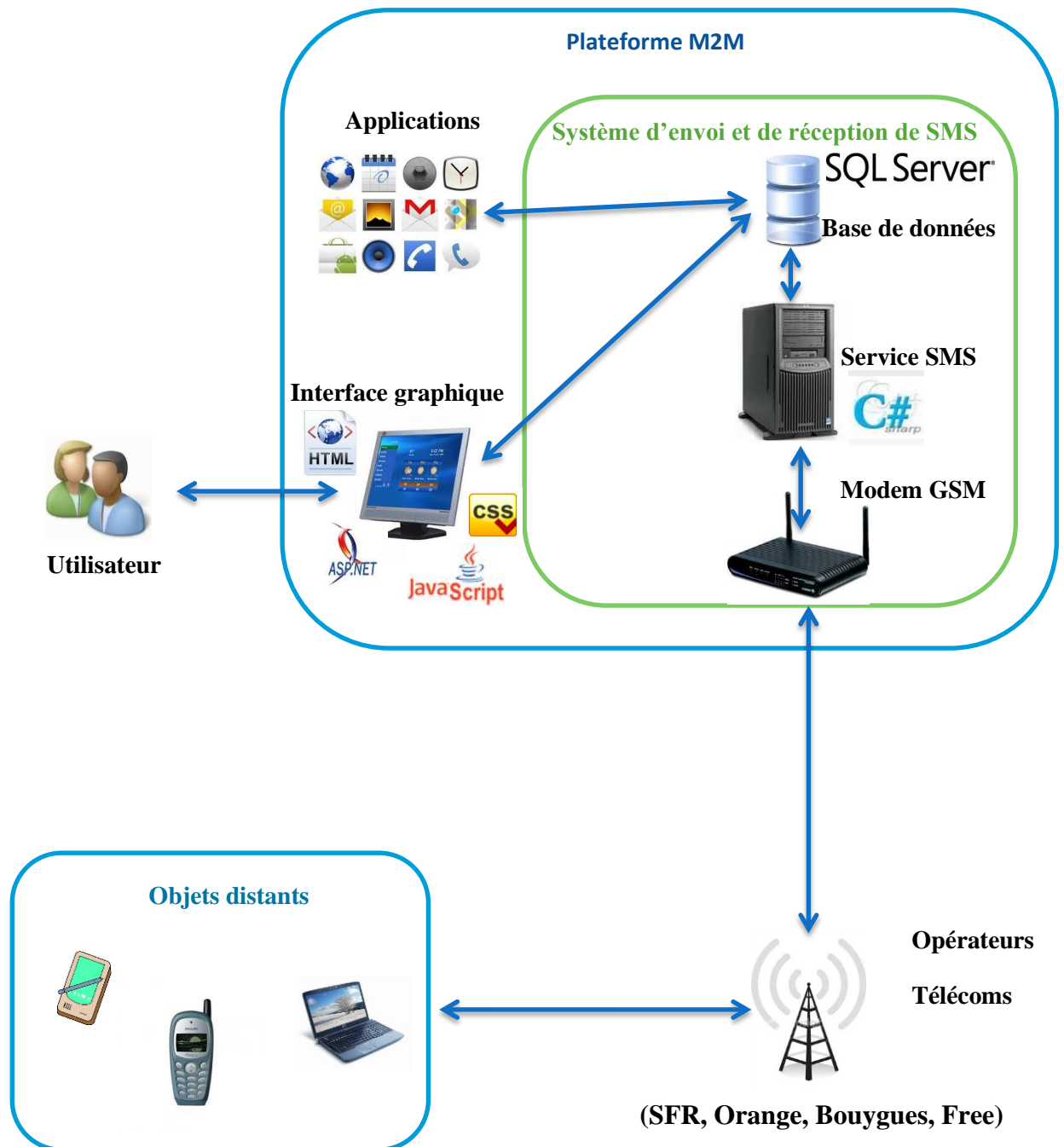


Figure 2 - Schéma d'architecture

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

### 5.1.1 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 Windows qui 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).

Le service SMS est écrit en C# tout en utilisant les apports du Framework .Net.

### 5.1.2 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.

La base de données est basée sur la technologie SQL Server qui offre des outils facilitant sa gestion.

### 5.1.3 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.

L'interface utilisateur repose sur des technologies Web telles que l'ASP.Net, du HTML, du JavaScript ou encore du CSS.

## 5.2 Conception

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

### 5.2.1 Diagrammes des cas d'utilisation

#### 5.2.1.1 Application

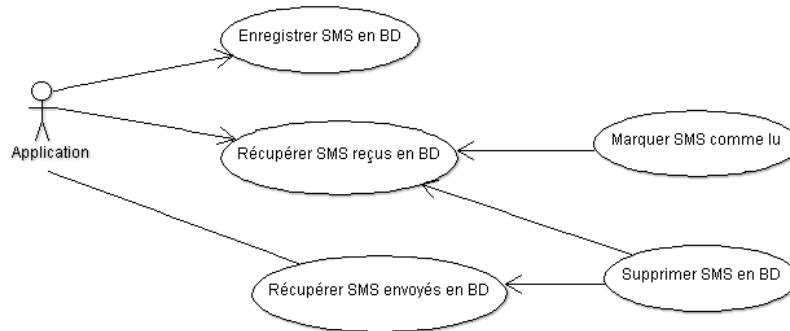


Figure 3 - Diagramme des cas d'utilisation : Application

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.

#### 5.2.1.2 Utilisateur

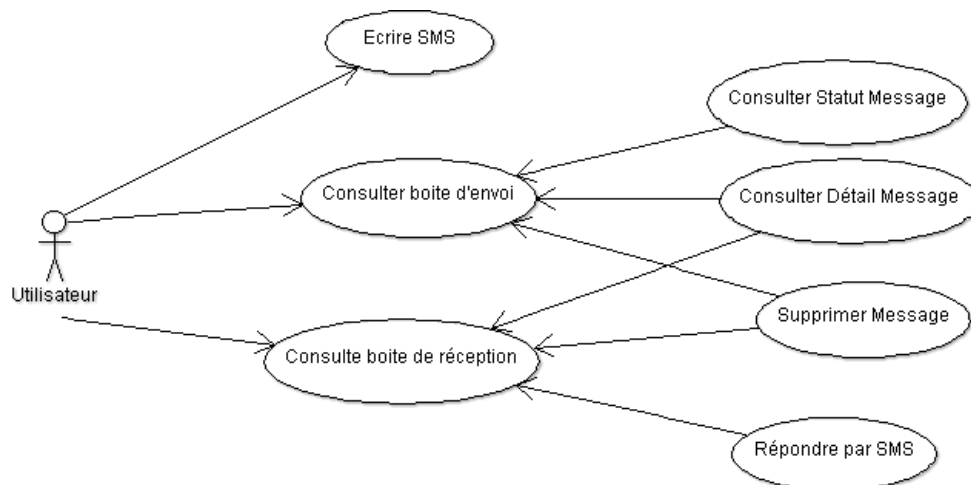


Figure 4 - Diagramme des cas d'utilisation : Utilisateur

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.

### 5.2.2 Dictionnaire de données

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

Le dictionnaire de données suivant présente l'ensemble de ces informations, regroupées par thème.

Thème	Nom	Type	Commentaires
<b>Message</b>	idMessage	Entier	L'identifiant d'un message
	messageTexte	Texte	Le contenu du message, s'il s'agit d'un message au format texte
	messagePDU	Texte	La trame PDU envoyé
	accuseReceptionDemande	Booléen	VRAI si un accusé est requis, FAUX sinon
	noDestinataire	Texte	Le numéro du destinataire
	noEmetteur	Texte	Le numéro de l'émetteur
<b>MessageEnvoyé</b>	dureeValidité	Entier	La durée de validité, convertie selon la norme PDU
	dateDemande	Date	La date de soumission (insertion dans la BD) du message
	dateEnvoi	Date	La date d'envoi du message
	dateAccusé	Date	La date de réception de l'accusé
	refModem	Entier	Le numéro de référence que le modem a associé au message
<b>MessageReçu</b>	dateRéception	Date	La date de réception du message
	dateLecture	Date	La date de lecture du message
<b>Encodage</b>	idEncodage	Entier	L'identifiant de l'encodage
	libelleEncodage	Texte	Le libellé de l'encodage (7bits, 8bits, 16bits)
<b>Statut</b>	idStatut	Entier	L'identifiant du statut de message
	libelleStatut	Texte	Le libellé du statut de message (En attente, Envoyé, Accusé, Erreur)

### 5.2.3 Diagramme de classe

A partir du dictionnaire de données, nous avons élaboré le diagramme de classe. Les thèmes deviennent alors des classes, et chaque information d'un thème devient un attribut de la classe.



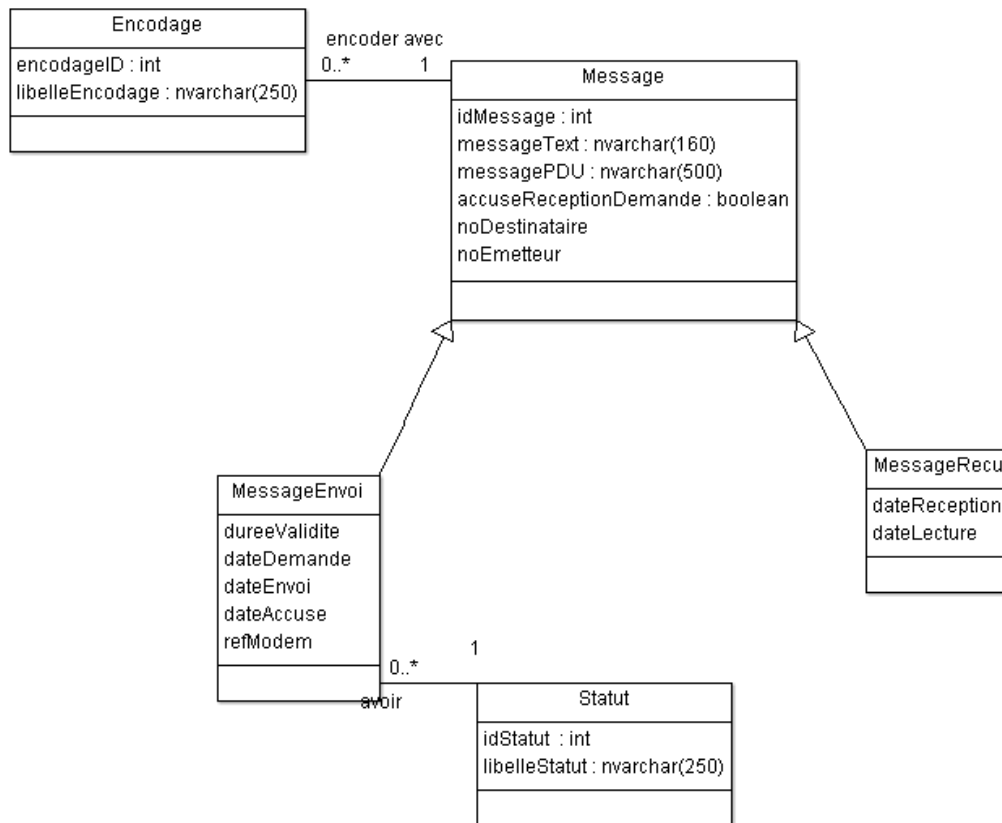


Figure 5 - Diagramme des classes

La classe **Message** regroupe les informations partagées par les classes **MessageEnvoi** et **MessageRecu** qui en héritent. Chaque message est caractérisé par un identifiant unique, un destinataire, un émetteur et un message, qu'il soit au format texte ou PDU. Les numéros de téléphone sont enregistrés sous forme de chaînes de caractères. Il n'était pas possible de les enregistrer au format numérique, car ils peuvent contenir le signe « + » au début pour l'indicateur géographique. L'attribut **accuseReceptionDemande** indique si le destinataire ou l'émetteur a demandé un accusé de réception du message au correspondant. Cet attribut devrait être un booléen, mais la base de données SQL Server ne propose pas ce type de données. Nous avons donc choisi d'utiliser un entier, 0 pour faux, 1 pour vrai. Toutes ces informations sont obligatoires (**messagePDU** ou **messageText** peut être nul, si l'autre ne l'est pas).

Un **message envoyé** est caractérisé par une durée de validité. La durée de validité est enregistrée sous forme d'entier, car c'est le format qu'il faut spécifier dans la trame PDU. La conversion est toujours possible en appliquant l'algorithme de conversion. 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 de réception.

Un **statut** est obligatoirement 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. La présence ou non de la date de lecture permet également de connaître le statut du message reçu, lu ou non lu.

## 5.2.4 Maquettage

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

### 5.2.4.1 L'interface d'envoi

Banniere

Ecrire SMS Boite Envoi Boite reception

No Destinataire Encodage

Duree Validite Type Message

Message Texte ou Trame PDU

Envoyer

Figure 6 - 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.

#### 5.2.4.2 La boîte d'envoi

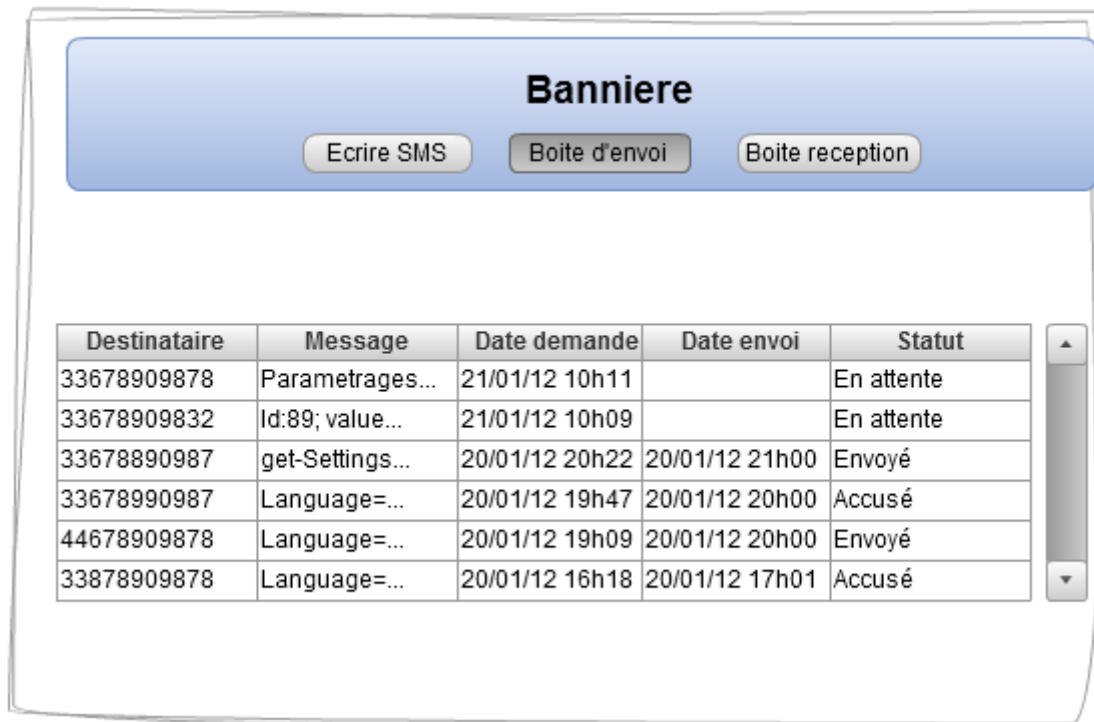


Figure 7 - Maquette de la boîte d'envoi

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 trouvé 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.

### 5.2.5 La boîte de réception

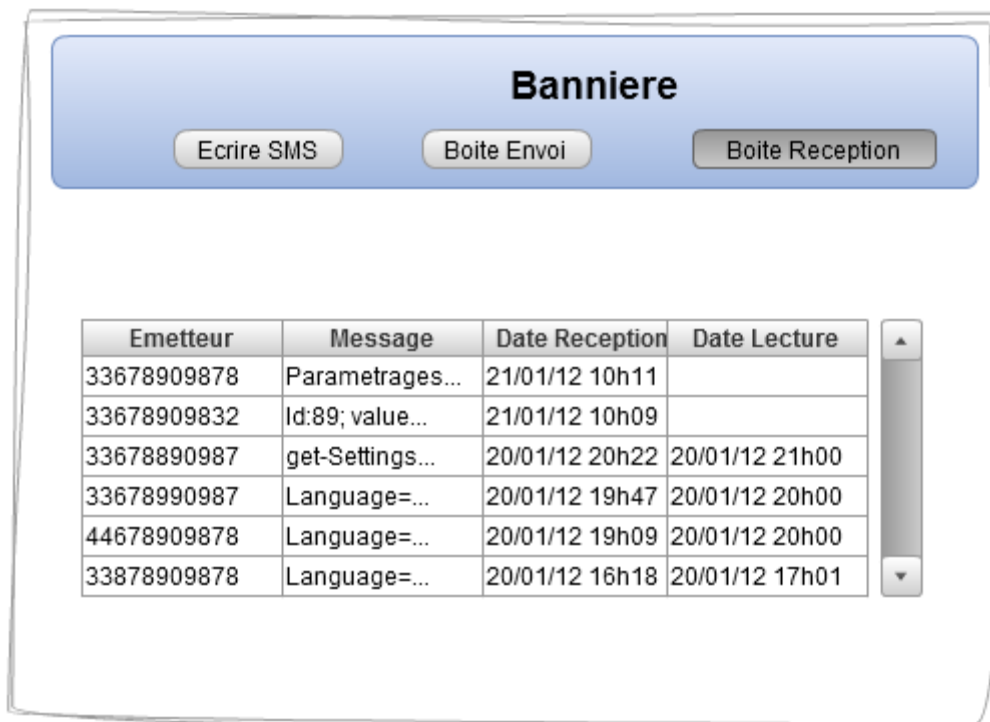


Figure 8 - Maquette de la boîte de réception

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.

## 5.3 Environnement de travail

### 5.3.1 Le Framework .Net

#### 5.3.1.1 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. Ce Framework 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.

### 5.3.1.2 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.

```

//si mode debug on passe un argument a l'application
#if DEBUG
    args = new string[1] { "/cons" };
#endif

if (args.Length > 0) //si un argument est passé à l'application
{
    //on lance l'application en mode console
    ServiceManager serverSync = new ServiceManager();
    serverSync.Start();

    //boucle qui permet de passer des commandes a l'application pendant l'exécution
    Attend une saisie de la commande 'quit' pour sortir

    serverSync.Stop();
    serverSync.Dispose();
}
else //on lance l'application en tant que service windows
{
    //on instancie le service
    ServiceBase[] ServicesToRun;
    ServicesToRun = new ServiceBase[]
    {
        new ServiceSMS()
    };

    //on lance le service
    ServiceBase.Run(ServicesToRun);
}
}
}
}

```

Figure 9 - 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, des arguments peuvent être passés à l'application pendant son exécution ce qui facilite la mise en place de tests unitaires.

### 5.3.1.3 L'interface graphique

#### 5.3.1.3.1 Le modèle client-serveur

Les interfaces utilisateurs du projet ont été développées avec la technologie **ASP.Net** incluse 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.

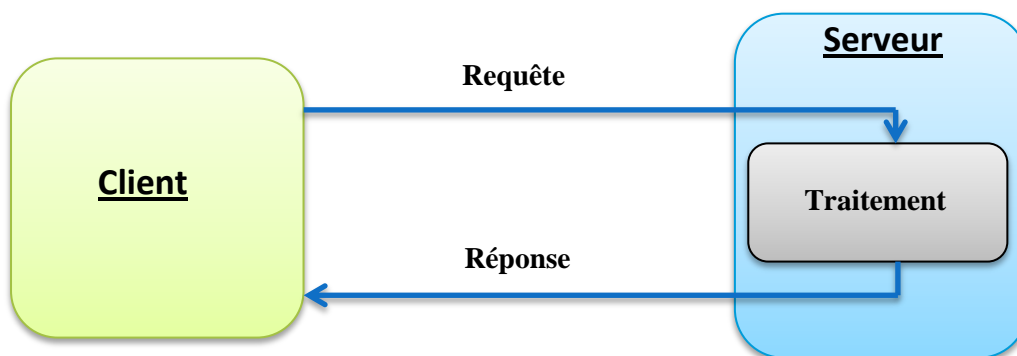


Figure 10 - Modèle client-serveur

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.

#### 5.3.1.3.2 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 :

```
<html>
<head>
  <title>Demonstration fonctionnement ASP</title>
</head>
<body>
  <!-- EXEMPLE SCRIPT ASP -->
  <%
    //Boucle qui va de 0 a 9
    for (int i = 0; i < 10; i++)
    {
      //Affichage du numero actuel
      Response.Write("Numero actuel :" + i + "<br>");
    }
  %>
  <!-- FIN EXEMPLE -->
</body>
</html>
```

Figure 11 - 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.

```
▼ <html>
  ► <head>...</head>
  ▼ <body>
    <!-- EXEMPLE SCRIPT ASP -->
    "
      Numero actuel :0"
    <br>
    "Numero actuel :1"
    <br>
    "Numero actuel :2"
    <br>
    "Numero actuel :3"
    <br>
    "Numero actuel :4"
    <br>
    "Numero actuel :5"
    <br>
    "Numero actuel :6"
    <br>
    "Numero actuel :7"
    <br>
    "Numero actuel :8"
    <br>
    "Numero actuel :9"
    <br>
    <!-- FIN EXEMPLE -->
  </body>
```

Figure 12 - Résultat de l'interprétation de l'ASP par le serveur

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

#### 5.3.1.3.3 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** ».

#### 5.3.1.3.4 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 traitements 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'évènement **onselectedindexchanged** est déclenché à chaque fois qu'un élément de la liste déroulante dont l'identifiant est « **DropDownEncodage** » est sélectionnée.

<u>Client</u>
<pre> &lt;!-- ENCODAGE --&gt;     &lt;td&gt;Encodage :&lt;/td&gt;     &lt;td&gt;         &lt;asp:DropDownList ID="DropDownEncodage"/&gt;         onselectedindexchanged="DropDownEncodage_SelectedIndexChanged"/&gt;     &lt;/td&gt; &lt;!-- FIN ENCODAGE --&gt; </pre>

Figure 13 - Exemple d'évènement au niveau du client

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



### 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":
            // encodage 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;
    }
}
```

Figure 14 - Exemple d'événement au niveau du serveur

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

#### 5.3.1.3.5 Schéma du fonctionnement général de l'ASP.Net

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

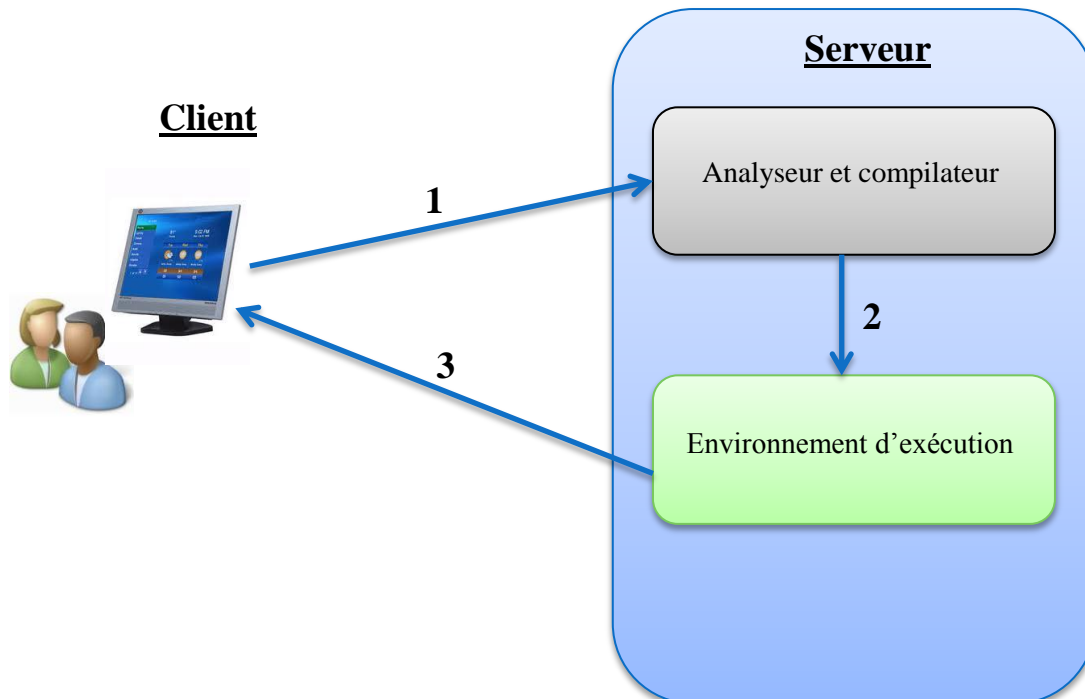


Figure 15 - Schéma du fonctionnement général ASP.Net

- 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.

### 5.3.2 Installation d'un serveur

#### 5.3.2.1 SQL Server

SQL Server est un système de gestion de base de données (**SGBD**) développé par Microsoft. C'est un outil indispensable afin de 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.

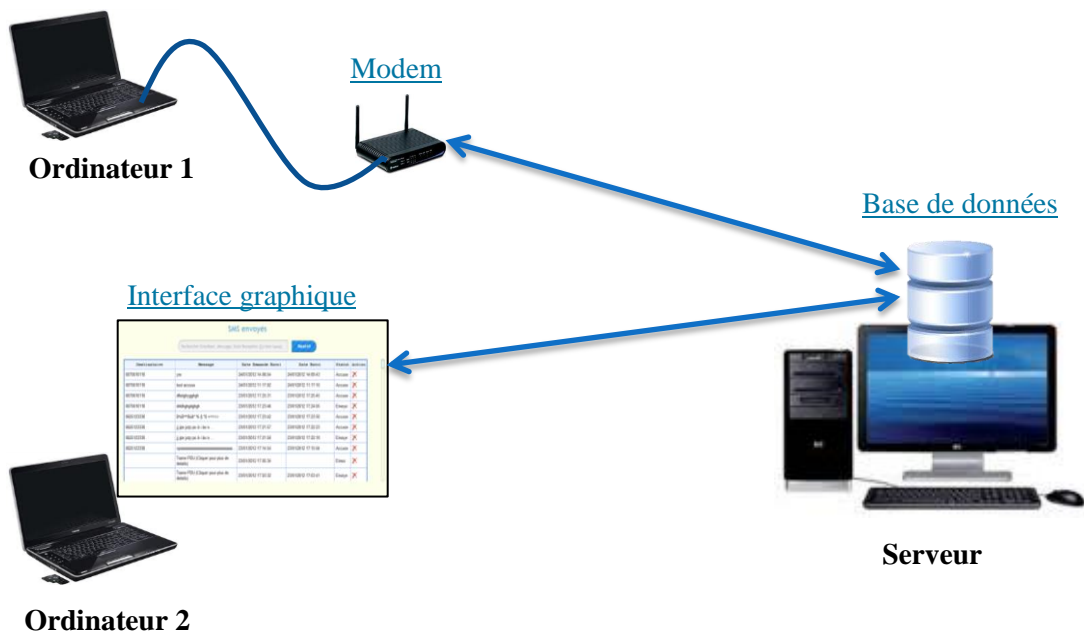


Figure 16 - Schéma d'utilisation du serveur de base de données

Le serveur est un Dell Optiplex 760 équipé d'un processeur double cœur à 3Ghz et de 2Go de RAM. Au niveau 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 au serveur de base de données il a fallu installer SQL Server Management Studio.

Afin de permettre un accès à distance à la base de données, 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.

### 5.3.2.2 VNC

Puisque nous ne disposons que d'un seul modem, il nous fallait trouver une solution pour pouvoir travailler à distance chacun de notre côté sans devoir emporter le modem (à cause des vacances de Décembre, notamment). Nous avons donc décidé d'installer sur le serveur une solution de prise de contrôle à distance, et avons choisi VNC.

Virtual Network Computing (VNC) est un système d'accès à un bureau distant qui permet de prendre le contrôle d'un ordinateur distant. Il permet de transmettre les saisies au clavier ainsi que les clics de souris d'un ordinateur à l'autre. VNC se compose de deux parties, le client et le serveur. Le serveur est l'ordinateur auquel on veut accéder à distance, alors que le client est l'ordinateur avec lequel on accède au serveur.

Nous avons donc installé RealVNC, une implémentation libre de VNC, sur le serveur dans la salle de projet. L'installation en elle-même est assez classique. Il a ensuite fallu ouvrir le port 5900, qu'utilise le protocole de VNC, sur le pare-feu du serveur.

Ensuite, il faut installer VNCViewer, le client VNC, sur nos propres ordinateurs. Après l'installation, la fenêtre suivante apparait pour la connexion au serveur :

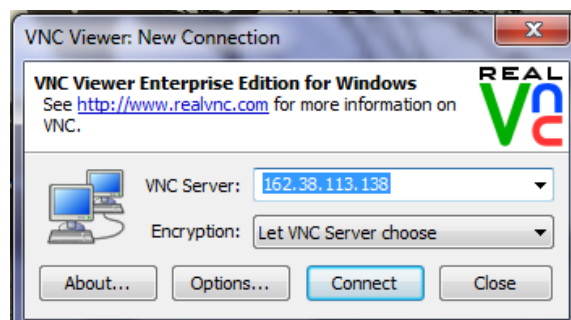


Figure 17 - Fenêtre de connexion de VNC Viewer

Il faut alors renseigner l'adresse IP du serveur et valider. On accède alors au bureau du serveur.



Figure 18 - L'écran de connexion du serveur dans VNC Viewer

### 5.3.3 Mise en place du modem GSM

La société Synox nous a fourni un modem GSM, modèle AirLink GL6100, pour toute la durée du projet. Un modem GSM se branche habituellement sur le port COM d'un ordinateur. Cependant, cette connectique est rarement présente sur les ordinateurs portables, et nous n'en disposons malheureusement pas sur nos propres machines. Nous avons donc utilisé un adaptateur USB (modèle ATEN UC232A), lui aussi fourni par Synox, qui nous permet de brancher le modem sur un port USB. Cet adaptateur nécessite un driver, que nous avons trouvé sur le site du constructeur (ATEN).

Une fois tout ceci installé, le modem apparaît dans la liste des périphériques, dans les périphériques branchés sur un port COM.

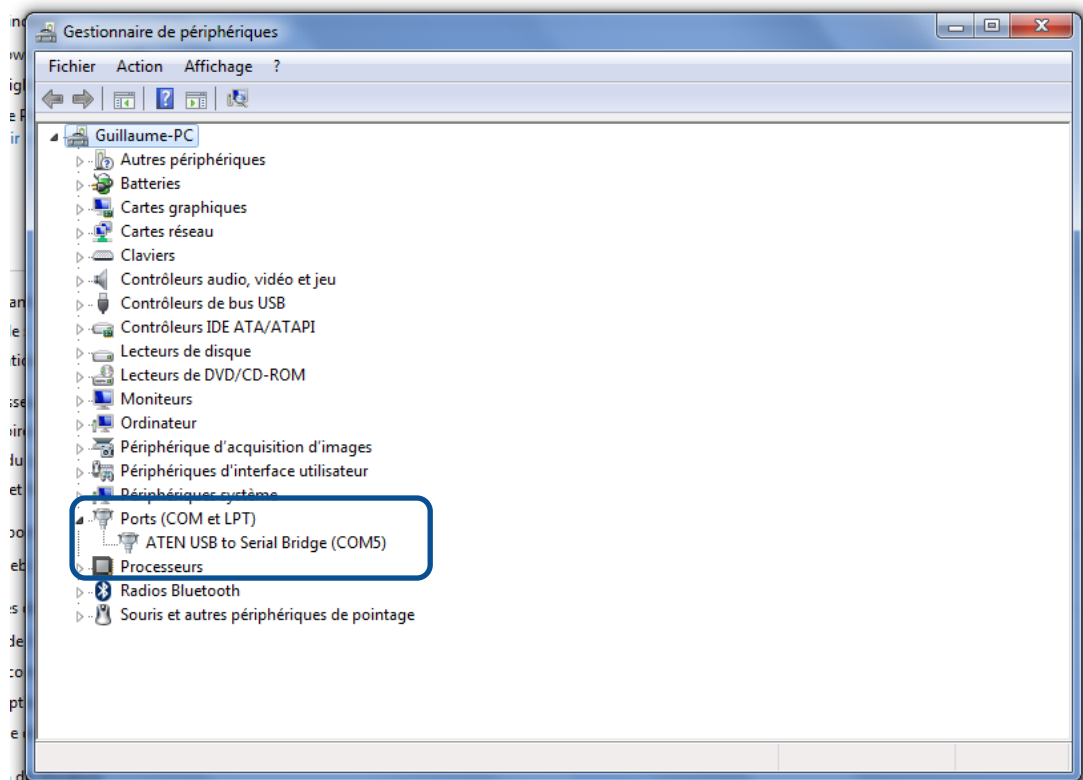
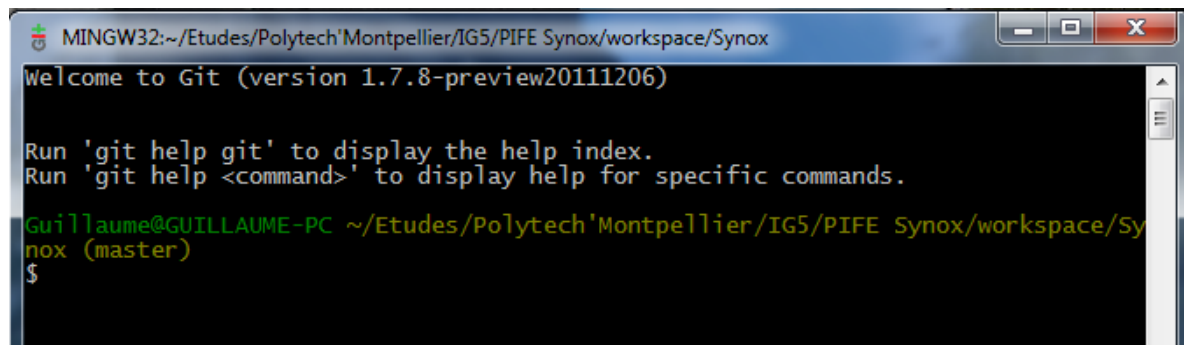


Figure 19 - Le modem GSM dans le gestionnaire des périphériques

Il apparait le nom de l'adaptateur, mais il s'agit bien du modem GSM.

### 5.3.4 GitHub

Afin de synchroniser notre travail, il nous fallait une solution de partage et d'échange de fichiers. Nous avons opté pour le logiciel de gestion de versions Git, réputé pour sa performance, et que nous souhaitions utilisé.



```
MINGW32:~/Etudes/Polytech'Montpellier/IG5/PIFE Synox/workspace/Synox
Welcome to Git (version 1.7.8-preview20111206)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Guillaume@GUILLAUME-PC ~/Etudes/Polytech'Montpellier/IG5/PIFE Synox/workspace/Synox (master)
$
```

Figure 20 - La console de Git

4 commandes permettent de synchroniser le travail, chacune précédée par le mot-clé « git ».

#### 5.3.4.1 La commande add

Cette commande prend comme paramètre un chemin (relatif ou absolu). Elle marque les nouveaux fichiers à synchroniser, dans l'arborescence issue du nœud pointé par le chemin passé en paramètre.

La commande est couramment appelée comme suit, la console Git étant configuré pour pointer par défaut sur le répertoire à synchroniser.

**Git add .**

#### 5.3.4.2 La commande pull

Cette commande permet de récupérer en local la dernière version sur le serveur. Si des modifications ont été faites depuis le dernier pull, il faut au préalable le signifier avec la commande commit, puis régler les éventuels conflits.

#### 5.3.4.3 La commande commit

Cette commande crée une nouvelle version de l'espace de travail en prenant en compte les modifications faites depuis la dernière version. Cette commande prend en paramètre un message expliquant les modifications apportées.

**git commit -m « message »**

Un commit permet également de signifier que les éventuels conflits survenus lors d'un pull sont réglés. Il est donc important de s'assurer que ce travail est fait avant de faire un commit.

#### 5.3.4.4 La commande push

Cette commande envoie sur le serveur la dernière version validée par le commit. Il est impossible de faire un push tant que des conflits ne sont pas résolus.

### 5.3.5 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 21 - Les priorités dans RememberTheMilk

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



Figure 22 - 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 complétée 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.

### 5.3.6 Cacao

Cacao 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://cacao.com/>.

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

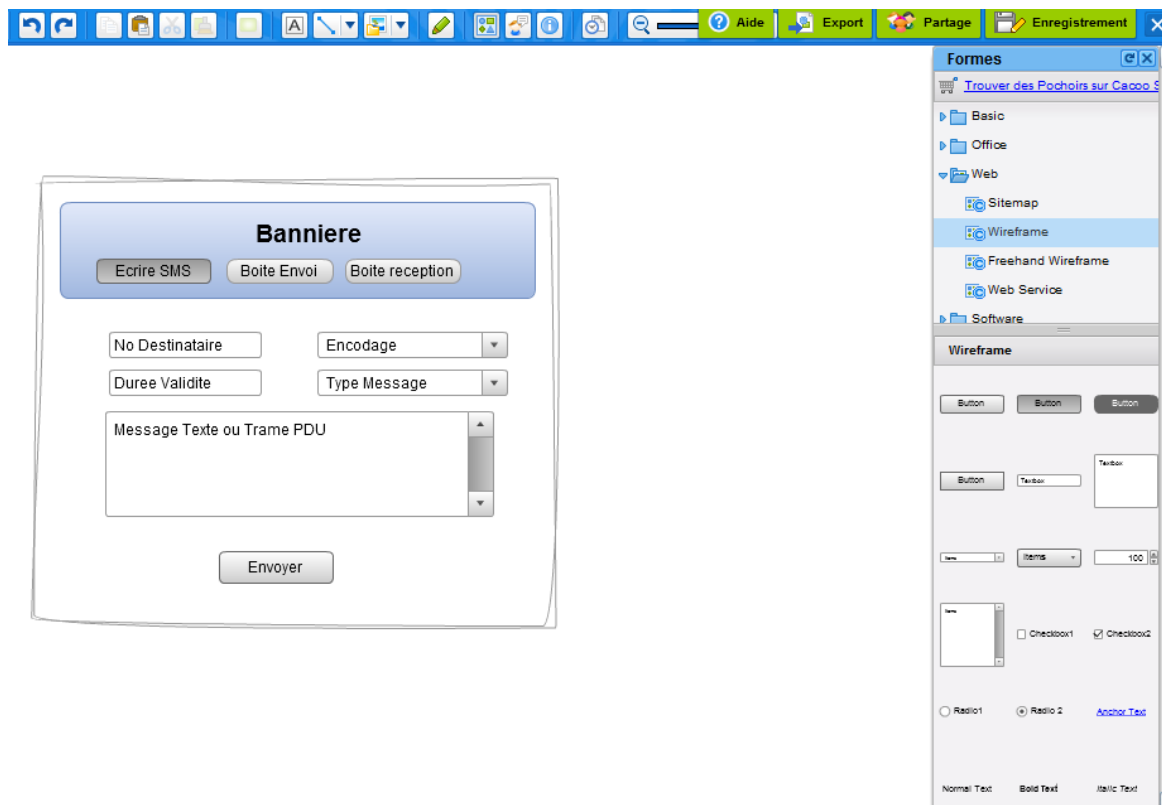


Figure 23 - Aperçu de l'outil de dessin Cacao

L'interface utilisateur de Cacao 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 au format « jpeg » à partir de la maquette afin de l'exporter.

## 5.4 Développement

### 5.4.1 Les commandes AT

#### 5.4.1.1 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.

#### 5.4.1.2 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

#### 5.4.1.3 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).

#### 5.4.1.4 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.

### 5.4.2 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.

#### 5.4.2.1 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 .Net qui nous permet d'instancier un objet représentant la liaison avec le modem.

```
PortCom = new SerialPort()  
{  
    PortName = serialPortName, //no de port, ex : COM11  
    BaudRate = 115200, //vitesse a laquelle on envoie/reçoit les donnees (bps)  
    DataBits = 8, //longueur standard des bits de données par octet  
};
```

Figure 24 - 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 cet objet pour établir une connexion avec le modem.

#### 5.4.2.2 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**.

```
//buffer : Tableau d'octets qui contient les données à écrire sur le port.  
//offset : 0, index du buffer partir duquel commencer la copie des octets vers le port  
//buffer.length : Nombre d'octets à écrire sur le port  
PortCom.Write(buffer, 0, buffer.Length);
```

Figure 25 - 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.

### 5.4.2.3 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.

```
//si le signal comme quoi on a reçu un message est active dans les 10 prochaines secondes
if (receiveNow.WaitOne(10000, false))
{
    //on recupere les donnees presentes sur le port serie
    string message = PortCom.ReadExisting();

    //on l'ajoute a la reponse car on n'est pas sur que le message soit complet
    response += message;
}
else
{
    //message incomplet ...
}
}
while (!response.EndsWith("\r\nOK\r\n") && !response.EndsWith("\r\nERROR\r\n"));
```

Figure 26 - 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.

### 5.4.2.4 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.

### 5.4.2.5 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.

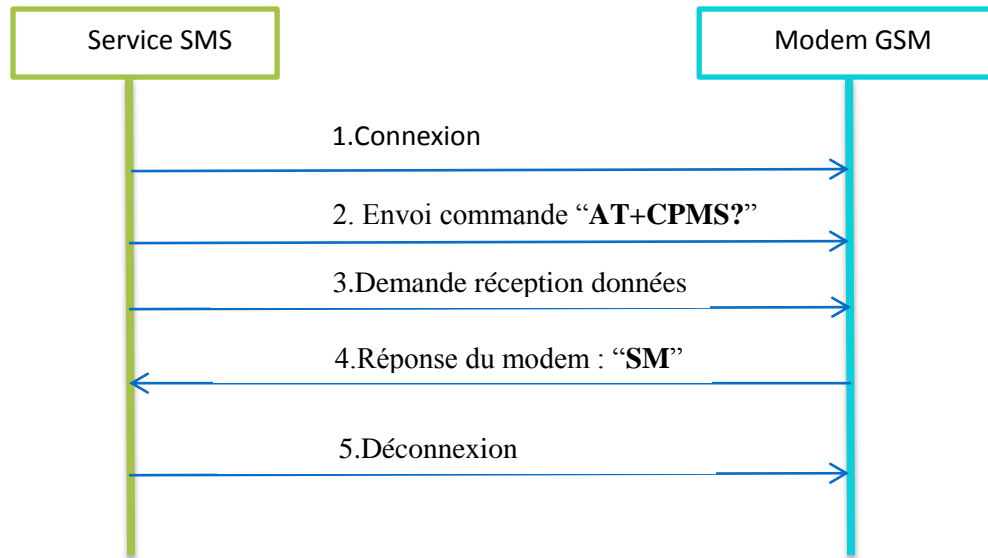


Figure 27 - Protocole de communication entre le service SMS et le modem GSM

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

### 5.4.3 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 28 - 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 ». « **Ctrl+Z** » correspond au code ASCII 26.

Le mode **PDU** est un peu plus complexe car il utilise des commandes AT contenant de l'*hexadécimal*<sup>viii</sup>. L'envoi de commandes AT dans ce mode respecte une structure de base.

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

Figure 29 - 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 30 - 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**.

#### 5.4.4 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> 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 librairie 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.

##### 5.4.4.1 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 31 - 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 fonctionnalité.
- ✓ **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

#### 5.4.4.2 Utilisation

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

```
SMS sms = new SMS();
```

Figure 32 - Exemple d'instanciation de la librairie 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 reception  
sms.StatusReportIndication = true;
```

Figure 33 - 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 34 - 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 35 - Exemple d'utilisation de la fonction 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** ».

#### 5.4.5 Intégration de la base de données

Une fois la base de données SQL Server créée, il faut la faire communiquer avec le service SMS et l'interface graphique. Le Framework .Net permet d'intégrer rapidement une base de données dans un projet en ajoutant des classes LINQ to SQL, comme le montre la capture d'écran suivante.

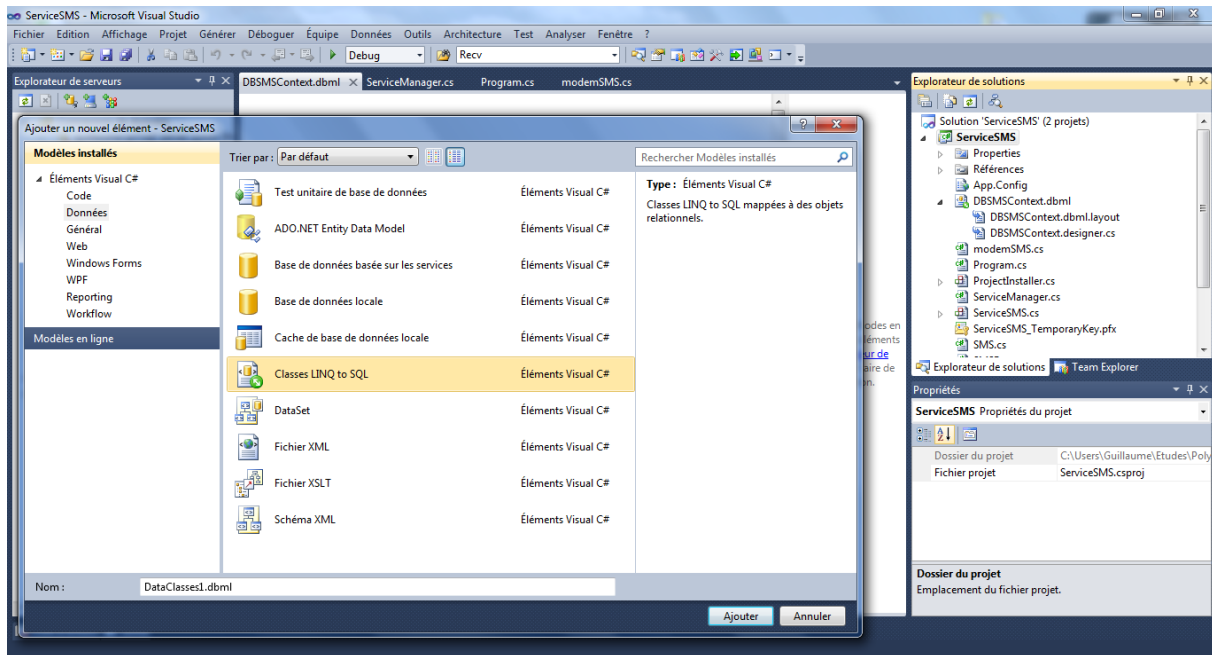


Figure 36 - Création des classes LINQ to SQL

Une fenêtre demande ensuite de spécifier l'adresse du serveur et les identifiants de connexion. Une fois cela fait, la base de données apparaît dans les ressources serveur, comme le montre la capture d'écran suivante.

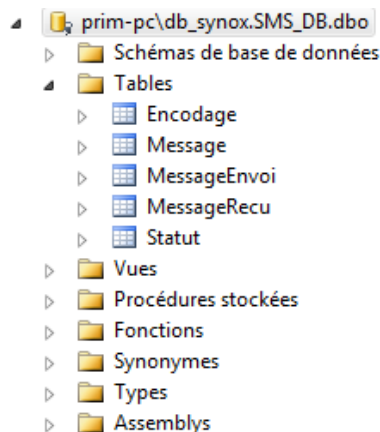


Figure 37 - La base de données dans l'Explorateur de serveurs de Visual Studio

La liaison effectuée, on indique quelles tables doivent être prise en compte pour les classes métiers en faisant glisser les tables dans l'espace de travail. On crée ainsi un modèle, assez proche d'un modèle logique de données (MLD).



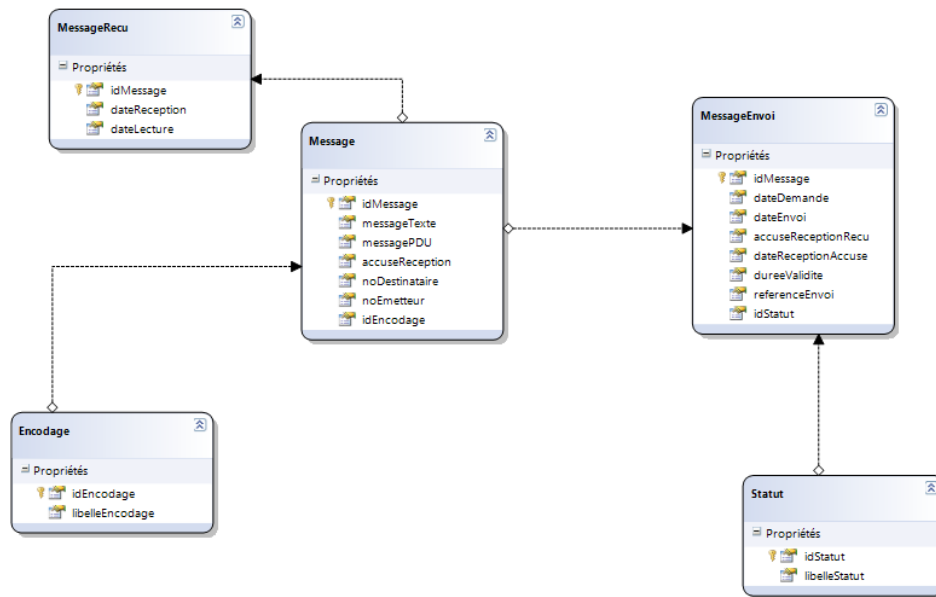


Figure 38 - Modèle logique de données dans Visual Studio

Les classes métiers correspondantes sont automatiquement créées, et directement utilisables dans le code de l'application. L'accès et la modification des données se fait alors par le biais de requêtes LINQ.

Voici par exemple la requête LINQ récupérant tous les messages en attente d'envoi.

```
//on recupere tous les message en attente d'envoi dans la base de donnees
MessageEnvoi[] lesSMSAEnvoyer = (from msg in dbContext.MessageEnvoi
    where msg.Statut.libelleStatut == "En attente"
    select msg).ToArray();
```

Figure 39 - Récupération d'objets en LINQ

On remarque que la syntaxe se rapproche de celle du SQL. On peut la traduire ainsi en français : « Sélectionner tous les messages à envoyer dont le statut est en attente ». La requête, intégrée dans le code de l'application retourne directement un tableau d'objets de type MessageEnvoi. On peut donc modifier directement les attributs des objets ainsi retournés, comme par exemple pour spécifier la date d'envoi.

```
lesSMSAEnvoyer[0].dateEnvoi = DateTime.Now;
```

Figure 40 - Modification d'un objet en LINQ

Une fois les modifications terminées, on effectue les changements dans la base de données.

```
dbContext.SubmitChanges();
```

Figure 41 - Validation des modifications en LINQ

L'objet dbContext est la référence de la base de données.

## 5.4.6 Le service SMS

Un service Windows n'est pas un programme classique qui s'exécute à la demande au premier plan, mais est lancé au démarrage de l'ordinateur et s'exécute de manière invisible. Il effectue des traitements automatiquement, sans que l'utilisateur n'ait besoin d'intervenir.

Le service a donc accès au modem via le port COM. On a réutilisé ici la classe que nous avons créée pour gérer le modem SMS ainsi que la librairie ATSMS. Lorsque le service est démarré, deux *timers*<sup>ix</sup> sont lancés, l'un pour la lecture des messages et les accusés reçus, l'autre pour l'envoi des messages en attente.

```

/// <summary>
/// Flag pour savoir si les services sont démarrés
/// </summary>
public bool IsStarted = false;
/// <summary>
/// Timer Général
/// </summary>
private System.Timers.Timer _timerService;
/// <summary>
/// Flag qui indique si la tâche de fond est en cours de travail
/// </summary>
private bool _busy;
/// <summary>
/// Date de la dernière execution du timer
/// </summary>
private DateTime _dateDerniereLecture;
/// <summary>
/// Initialisé à 15 secondes
/// </summary>
private TimeSpan _dureeEntre2Lectures = new TimeSpan(0, 0, 15);
/// <summary>
/// Date de la dernière execution du timer
/// </summary>
private DateTime _dateDerniereEnvoi;
/// <summary>
/// Initialisé à 15 secondes
/// </summary>
private TimeSpan _dureeEntre2Envois = new TimeSpan(0, 0, 15);

```

Figure 42 - Principales variables du service SMS

Cet extrait du service SMS présente les principales variables. On remarquera plusieurs booléens, indiquant si le service est démarré ou s'il est en train de recevoir ou d'envoyer. Sont aussi initialisés ici la durée entre 2 envois ou 2 réceptions, ainsi que les dates de dernière réception et d'envoi.

```

/// <summary>
/// Constructeur
/// </summary>
public ServiceManager()
{
    // Taches de fonds toutes les 2 secondes
    _timerService = new System.Timers.Timer();
    _timerService.Interval = 2000;
    _timerService.Elapsed += new ElapsedEventHandler(timerService_Elapsed);

    _dateDerniereLecture = DateTime.Now;
}

/// <summary>
/// Lancement du Timer
/// </summary>
public void Start()
{
    // Démarrage des taches de fond
    IsStarted = true;
    this._timerService.Start();

    //on initialise le modem
    modem = new modemSMS(noPortModem);
}

/// <summary>
/// Arrêt des services
/// </summary>
public void Stop()
{
    // Arrêt des taches de fond
    this._timerService.Stop();
    modem = null;
    IsStarted = false;
}

```

Figure 43 - Algorithme d'initialisation, de démarrage et d'arrêt du service SMS

Comme le montre cet extrait du code portant sur l'initialisation, le démarrage et l'arrêt du timer, il n'y a en réalité qu'un seul timer, au sens informatique du terme. Toutes les 2 secondes, le timer déclenche un évènement, qui va ensuite effectuer les opérations de lecture ou d'écriture, si le service n'est pas déjà occupé et si les durées respectives sont écoulées.

On peut donc assimiler ce mécanisme à 2 timers, assimilation que nous conserverons dans le reste de ce rapport.

#### 5.4.6.1 Envoi de nouveaux messages

Quand le timer d'envoi lance un évènement, le service récupère les messages en attente en base de données, convertit les messages en PDU pour ceux qui ne sont pas des trames PDU, et effectue l'envoi de ces messages via le modem.

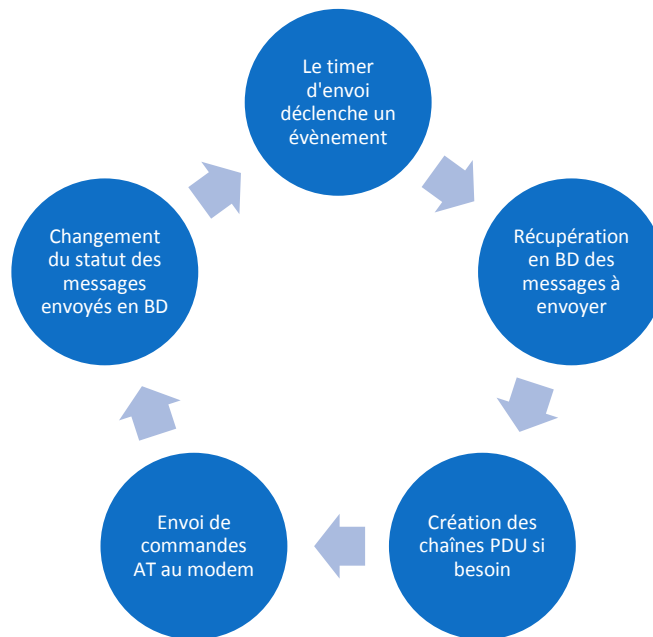


Figure 44 - Schéma descriptif de l'algorithme du timer de lecture

```

// tache d'envoi des sms
if (_dateDerniereEnvoi.Add(_dureeEntre2Envois) < DateTime.Now)
{
    //on recupere tous les message en attente d'envoi dans la base de donnees
    MessageEnvoi[] lesSMSAEnvoyer = (from msg in dbContext.MessageEnvoi
                                     where msg.Statut.libelleStatut == "En attente"
                                     select msg).ToArray();

    // S'il y a des messages à envoyer
    if (lesSMSAEnvoyer.Length > 0)
    {
        //connexion au modem
        modem.connectToModem();

        //pour chaque message a envoyer
        foreach (MessageEnvoi sms in lesSMSAEnvoyer)
        {
            //on envoie le sms
            Console.WriteLine("Envoi d'un message"); // Pour le débbugage
            envoyerSMS(sms);
        }

        modem.disconnectToModem();
    }

    // to do
    _dateDerniereEnvoi = DateTime.Now;
}
  
```

Nous reviendrons dans la partie 5.4.6 sur les requêtes LINQ.

La fonction `envoyerSMS`, prenant en paramètre un objet `MessageEnvoi`, s'occupe de la conversion du message en PDU si nécessaire et de son envoi.

```

//si trame PDU est nulle alors c'est un message Texte (si pas nul aussi)
if (sms.Message.messageTexte != null && sms.Message.messagePDU == null)
    //si trame PDU est nulle alors c'est un message Texte (si non nulle aussi)
    {
        messageAEnvoyer = sms.Message.messageTexte;

        //on recupere la duree de validite

        int dureeValidite = -1;

        if (sms.dureeValidite.HasValue)
            dureeValidite = sms.dureeValidite.Value;

        //on recupere la reference du message envoye

        reference = modem.sendSMS(PDU(sms.Message.noDestinataire, messageAEnvoyer,
            demandeAccuse, sms.Message.Encodage.libelleEncodage, dureeValidite));
    }

//verifie s'il y a une erreur
if (reference.Contains("ERROR"))
{
    //on passe le statut du sms a erreur
    sms.Statut = (from stat in dbContext.Statut where stat.libelleStatut == "Erreur" select stat).First();
}
else //on sauvegarde la reference en BD
{
    sms.referenceEnvoi = reference;
}

```

```

//on sauvegarde le sms comme envoye s'il y a une reference d'envoi
if (reference != null)
{
    //on complete les informations sur l'envoi des SMS
    sms.Message.noEmetteur = numeroModem;
    sms.accuseReceptionRecu = 0; //pas encore reçu

    //on passe le sms en statut envoye
    sms.Statut = statutEnvoye;
    sms.dateEnvoi = DateTime.Now; //date du jour
}
else // erreur
{
    //on passe le statut du sms a erreur
    sms.Statut = (from stat in dbContext.Statut where stat.libelleStatut == "Erreur" select stat).First();
}

//on valide les changements dans la BD
dbContext.SubmitChanges();

```

Figure 45 - Fragment de la fonction EnvoyerSMS (message texte et gestion des erreurs)

Cet extrait traite de l'envoi d'un message au format texte par le modem, puis du traitement de la référence retournée pour définir le statut du message en base de données. Lorsque le modem envoie un message, il retourne une référence que l'on enregistre en base de données. Si elle contient une erreur, ou qu'il n'y a pas du tout de référence, le message est inséré en base de données avec le statut erreur.

#### 5.4.6.2 Lecture de nouveaux messages

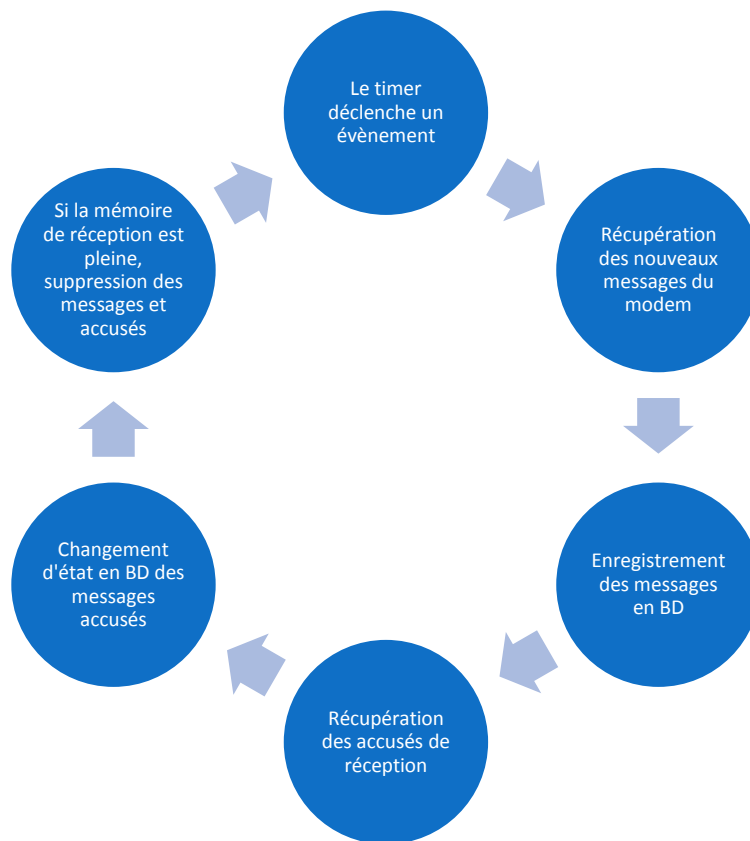


Figure 46 - Schéma descriptif de l'algorithme du timer de lecture

Quand le timer de lecture déclenche un évènement, on commence par récupérer les messages reçus par le modem. S'il y en a, on les insère dans la base de données.

```

//on recupere les messages sur la sim
SMS[] lesMessagesSurSIM = modem.readPDUMessage();

//pour chaque message
foreach (SMS sms in lesMessagesSurSIM)
{
    //on fait le mapping avec un nouvel objet de la BD
    //initialisation
    MessageRecu msg = new MessageRecu();
    msg.Message = new Message();

    //remplissage
    msg.dateReception = sms.ServiceCenterTimeStamp;
    msg.Message.messageTexte = sms.Message;
    msg.Message.noEmetteur = sms.PhoneNumber;
    msg.Message.noDestinataire = numeroModem;
    msg.Message.accuseReception = 0; //faux par default

    if (sms.StatusReportIndication)
        msg.Message.accuseReception = 1;

    //encodage
    msg.Message.Encodage = (from enc in dbContext.Encodage where enc.libelleEncodage == "PDU" select enc).First();
    //enregistrement du message
    dbContext.Message.InsertOnSubmit(msg.Message);
    dbContext.MessageRecu.InsertOnSubmit(msg);
}

//sauvegarde des changements
dbContext.SubmitChanges();

```

Figure 47 - Enregistrement d'un nouveau message reçu

Ensuite, on vérifie dans la mémoire des accusés de réception si de nouveaux accusés sont arrivés. Si c'est le cas, on cherche dans la base de données les messages non accusés en ayant demandé un, et on compare les numéros de référence pour trouver le bon message et le mettre dans l'état « accusé reçu ».

```

MessageEnvoi[] lesSMSVerifier = (from msg in dbContext.MessageEnvoi
    where msg.Statut.libelleStatut == "Envoye"
    && msg.Message.accuseReception == 1
    select msg).ToArray();

String[][] accuses = null;

if (lesSMSVerifier.Count() > 0)
{
    accuses = modem.readDeliveryReport(); // on recupere les accuses du modem
}

//pour chaque message
foreach (MessageEnvoi sms in lesSMSVerifier)
{
    //si on a reçu l'accuse
    if (verifierAccusePresentDansModem(sms, accuses))
    {
        //on change le statut
        sms.Statut = (from stat in dbContext.Statut where stat.libelleStatut == "Accuse" select stat).First();
        sms.accuseReceptionRecu = 1;
    }
}

dbContext.SubmitChanges();

```

Figure 48 - Traitement des accusés de réceptions

La fonction readDeliveryReport retourne sous formes de tableau à double entrée tous les accusés de réception reçus. Pour chaque message en attente d'accusé, on cherche si l'accusé est arrivé avec la fonction verifierAccusePresentDansModem. Si c'est le cas, on le notifie dans la base de données.

Enfin, le service vérifie le nombre de messages sur chaque mémoire, et vide celles qui sont presque pleines. En effet, si la mémoire est pleine, les nouveaux messages entrant ne seront pas enregistrés et seront donc perdus.

```

/// <summary>
/// Supprime les messages lus si la mémoire est pleine ainsi que les accusés de réception si la mémoire est pleine
/// </summary>
public void deleteAllReadSMS()
{
    // Si on est proche de la saturation mémoire des accusés (limitée à 50 accusés)
    if (countReceiptOnSIM() >= 45)
    {
        Console.Out.WriteLine("Deleting all RECEIPTS (" + countReceiptOnSIM().ToString() + ")");
        Send("AT+CPMS=\"SR\"");
        Send("AT+CMGD=1,2");
    }

    // Si il y a de nombreux messages lus sur la SIM
    if (countSMSOnSim() >= 25)
    {
        Console.Out.WriteLine("Deleting all READ messages (" + countSMSOnSim().ToString() + ")");
        Send("AT+CPMS=\"SM\"");
        Send("AT+CMGD=1,2");
    }
}
}

```

Figure 49 - Fonction de vidage de la mémoire

#### 5.4.6.3 Lancement et débogage du service SMS

Un service ne peut pas se lancer comme un simple programme et doit être installé. Pour le déploiement, il faut donc ajouter au service un programme d'installation. Visual Studio 2008 permet de faire cela assez rapidement, en ajoutant un projet Setup à la solution.

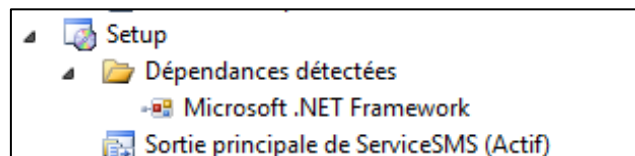


Figure 50 - Projet d'installation du service

Cependant, le débogage n'est pas pratique de cette façon. Nous avons donc fait en sorte d'ouvrir en parallèle du service une console qui nous permettra de visualiser (en mode débogage uniquement) le déroulement du service.

```

static void Main(string[] args)
{
    //si mode debug on passe un argument a l'application
    #if DEBUG
        args = new string[1] { "/cons" };
    #endif

    if (args.Length > 0) //si un argument est passé à l'application
    {
        //on lance l'application en mode console
        ServiceManager serverSync = new ServiceManager();
        serverSync.Start();

        //boucle qui permet de passer des commandes a l'application pendant l'exécution
        //Attend une saisie de la commande 'quit' pour sortir

        serverSync.Stop();
        serverSync.Dispose();
    }
    else //on lance l'application en tant que service windows
    {
        //on instancie le service
        ServiceBase[] ServicesToRun;
        ServicesToRun = new ServiceBase[]
        {
            new ServiceSMS()
        };
        //on lance le service
        ServiceBase.Run(ServicesToRun);
    }
}

```

Figure 51 - Lancement du service



### 5.4.7 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.

#### 5.4.7.1 L'envoi de SMS

The screenshot shows a web interface titled 'Gestion des SMS' with a blue header. Below the header are three buttons: 'Ecrire SMS', 'Boîte d'envoi', and 'Boîte de réception'. The main section is titled 'Envoyer des SMS' and contains a form with the following fields:

- Choix du mode SMS:** A dropdown menu with 'Texte' selected.
- Ecriture du SMS:**
  - Numéro du destinataire:** A text input field.
  - Encodage:** A dropdown menu with '7bits' selected.
  - Date de validité:** A checkbox labeled 'Demander un accusé de réception' followed by input fields for 'jours', 'heures', and 'minutes'.
  - Message à envoyer:** A large text area.
- Valider:** A blue button at the bottom left of the form.

Figure 52 - Formulaire d'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.

```

Message msg = new Message();

if (ListeMode.SelectedValue == "Texte")
{
    // Il s'agit d'un message texte

    //insertion d'un message
    msg.messageTexte = contenuSMS.Text;
    msg.noDestinataire = numDestinataire.Text;
    //on recupere l'encodage
    msg.Encodage = (from enc in dbContext.Encodage where enc.idEncodage == int.Parse(DropDownEncodage.SelectedValue) select enc)

    //demande accuse reception
    if (CheckBoxAccuse.Checked) // on a demande un accuse
    {
        msg.accuseReception = 1;
    }
    else
    {
        msg.accuseReception = 0;
    }
}
else
{
    // Message PDU
    msg.messagePDU = contenuSMS.Text;

    // Encodage PDU
    msg.Encodage = (from enc in dbContext.Encodage where enc.idEncodage == 4 select enc).First();
}

dbContext.Message.InsertOnSubmit(msg);

```

Figure 53 - Extrait de la fonction 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.

#### 5.4.7.2 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.

Destinataire	Message	Date Demande Envoi	Date Envoi	Statut	Action
0625123338	get all information	03/02/2012 15:11:22	03/02/2012 15:11:36	Envoyé	✗
0675610118	ping object	03/02/2012 15:11:00	03/02/2012 15:11:16	Envoyé	✗
0622031216	Set:"value=11; ID=668; check=...	03/02/2012 15:10:04	03/02/2012 15:10:57	Envoyé	✗
0675610118	Set:"value=09; ID=698; check=...	03/02/2012 15:09:20	03/02/2012 15:10:53	Envoyé	✗
0625123338	Get:"value=89; ID=198; check=...	03/02/2012 15:07:59	03/02/2012 15:09:00	Envoyé	✗
0622031216	Get:"value=87; ID=98; check=8...	03/02/2012 15:07:35	03/02/2012 15:08:56	Envoyé	✗
0604655154	Get:"value=87; ID=98; check=8...	03/02/2012 15:07:22	03/02/2012 15:08:39	Envoyé	✗
0604655154	Get:"value=87; ID=98; check=8...	03/02/2012 15:07:22	03/02/2012 15:08:39	Envoyé	✓

Figure 54 - 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.

```
//remplit le tableau des SMS
private void populateTableSMSEnvoyes(string elementRecherche="")
{
    try{
        Message[] listeMessages;
        listeMessages = (from mess in dbContext.MessageEnvoi
                        where mess.Message.messageTexte.Contains(elementRecherche)
                        || mess.Message.noDestinataire.Contains(elementRecherche)
                        || mess.dateDemande.ToString().Contains(elementRecherche)
                        || mess.dateEnvoi.ToString().Contains(elementRecherche)
                        || mess.Statut.libelleStatut.Contains(elementRecherche)
                        orderby mess.idMessage descending
                        select mess.Message).ToArray();
    }
```

Figure 55 - 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.

### 5.4.7.3 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.



**Gestion des SMS**

Ecrire SMS Boîte d'envoi Boîte de réception

**SMS Recus**

Rechercher Emetteur, Message, Date Reception (jj/mm/aaaa) **Search**

Emetteur	Message	Date Reception	Date Lecture	Accuse demandé	Action
+33675610118	Next message : 3m	03/02/2012 15:17:54		Non	 
+33675610118	3g network ok	03/02/2012 15:17:19		Non	 
+33675610118	3g network problem detected	03/02/2012 15:16:12		Non	 
+33675610118	Configuration ok	03/02/2012 15:15:12		Non	 
+33675610118	Data : sfr network, latency :...	03/02/2012 15:14:31		Non	 
+33625123338	Settings ok !	03/02/2012 15:13:07		Non	 
+33625123338	Check failed	03/02/2012 15:12:53		Oui	 
+33625123338	Receipt received	03/02/2012 15:12:38		Oui	 
+33675610118	Id:56 value:2556	03/02/2012 15:12:36		Non	 

Figure 56 - 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 ont été reçus par le modem.

```
//remplit le tableau des SMS
private void populateTableSMSRecus(string elementRecherche = "")
{
    try
    {
        Message[] listeMessages;
        listeMessages = (from mess in dbContext.MessageRecu
                        where mess.Message.messageTexte.Contains(elementRecherche)
                        || mess.Message.noEmetteur.Contains(elementRecherche)
                        || mess.dateReception.ToString().Contains(elementRecherche)
                        orderby mess.idMessage
                        descending
                        select mess.Message).ToArray();
    }
}
```

Figure 57 - Extrait du code applicatif permettant d'extraire 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.

```
//l'utilisateur veut repondre a l'emetteur
//noTel = numero de l'emetteur
function repondreSMS(noTel) {

    //on renvoie sur le formulaire d'ecriture
    location.href = "EcrireSMS.aspx?notel=" + noTel;
}
```

Figure 58 - Extrait du code applicatif permettant de répondre à l'émetteur

## 5.5 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.).

## 5.6 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.

## 6 Améliorations possibles

---

### 6.1 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.

### 6.2 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.

### 6.3 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.

## 7 Difficultés rencontrées

---

### 7.1 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 devons utiliser afin de communiquer avec le modem GSM.

De plus, même si nous avons 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.

### 7.2 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*<sup>x</sup>.

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 événements 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 avons 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.

### 7.3 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.



## 8 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.

## 9 Annexes

### 9.1 Diagramme de Gantt

N°	Mode Tâche	Nom de la tâche	Durée	Début	Fin	Prédécesseurs
1		<b>Projet Industriel</b>	<b>30,36 jours</b>	<b>Lun 05/12/11</b>	<b>Ven 03/02/12</b>	
2		<b>Familiarisation aux technologies</b>	<b>4 jours</b>	<b>Lun 05/12/11</b>	<b>Ven 09/12/11</b>	
3		Familiarisation avec les commandes AT	2 jours	Lun 05/12/11	Mar 06/12/11	
4		Tutoriel ASP.NET	1 jour	Mar 06/12/11	Jeu 08/12/11	3
5		Création d'une connexion entre .Net et le modem GSM	1 jour	Jeu 08/12/11	Ven 09/12/11	4
6		<b>Analyse</b>	<b>1,5 jours</b>	<b>Ven 09/12/11</b>	<b>Mar 13/12/11</b>	<b>2</b>
7		Cas d'utilisation	1 jour	Ven 09/12/11	Lun 12/12/11	
8		Réalisation de maquettes	0,5 jour	Lun 12/12/11	Mar 13/12/11	7
9		Diagramme de classe métier	0,5 jour	Lun 12/12/11	Mar 13/12/11	7
10		<b>Partie 1 : envoi de SMS</b>	<b>2 jours</b>	<b>Mar 13/12/11</b>	<b>Jeu 15/12/11</b>	<b>6</b>
11		Sélection des commandes AT nécessaires	1 jour	Mar 13/12/11	Mer 14/12/11	
12		Création d'un petit logiciel d'envoi de SMS	1 jour	Mer 14/12/11	Jeu 15/12/11	11
13		<b>Partie 2 : Réception des SMS</b>	<b>2 jours</b>	<b>Jeu 15/12/11</b>	<b>Lun 09/01/12</b>	<b>10</b>
14		Sélection des commandes AT nécessaires	1 jour	Jeu 15/12/11	Ven 16/12/11	
15		Adaptation du logiciel d'envoi afin de réceptionner des SMS	1 jour	Ven 16/12/11	Lun 09/01/12	14
16		<b>Partie 3 : Service Windows et interface graphique</b>	<b>12,93 jours</b>	<b>Mar 10/01/12</b>	<b>Mer 25/01/12</b>	<b>13</b>
17		<b>Service Windows</b>	<b>9,71 jours</b>	<b>Mar 10/01/12</b>	<b>Ven 20/01/12</b>	
18		Création et intégration de la base de données	3,29 jours	Mar 10/01/12	Jeu 12/01/12	
19		Intégration des fonctions issues du logiciel d'envoi et de réception	3,29 jours	Mar 10/01/12	Jeu 12/01/12	
20		Mise en place d'un protocole de communication	6,5 jours	Ven 13/01/12	Ven 20/01/12	
21		<b>Interface graphique</b>	<b>9,71 jours</b>	<b>Ven 13/01/12</b>	<b>Mer 25/01/12</b>	
22		Elaboration de l'interface d'envoi	3,29 jours	Lun 16/01/12	Mer 18/01/12	
23		Elaboration de la boîte d'envoi	3,29 jours	Jeu 19/01/12	Lun 23/01/12	
24		Elaboration de la boîte de réception	2,21 jours	Mar 24/01/12	Mer 25/01/12	
25		Rédaction rapports	7,57 jours	Jeu 26/01/12	Ven 03/02/12	16

Figure 59 - Diagramme de Gantt (tableau)

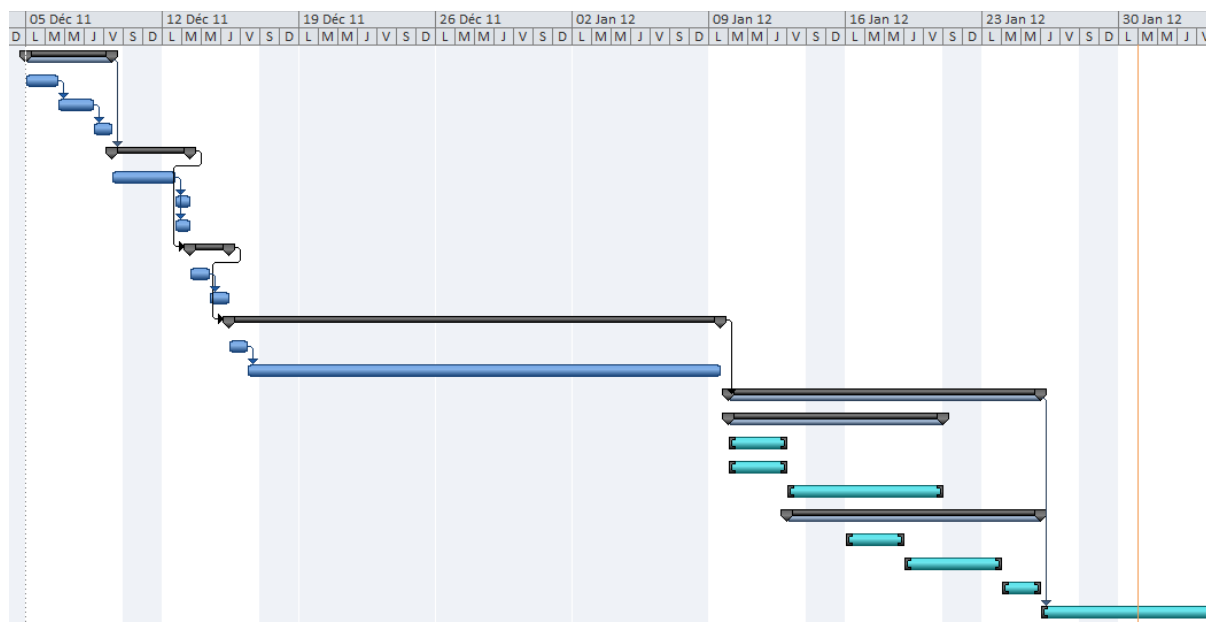


Figure 60 - Diagramme de Gantt (schéma)

# Glossaire

---

## <sup>i</sup> **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.

## <sup>iii</sup> **GSM – Global System for Mobile Communications :**

Réseau de deuxième génération pour la téléphonie mobile.

## <sup>iv</sup> **SMS – Short Message Service :**

Service permettant de transmettre de courts messages sur le réseau GSM.

## <sup>v</sup> **Modem – Modulateur-démodulateur :**

Périphérique servant à communiquer avec des utilisateurs distants.

## <sup>vi</sup> **Service Windows :**

Logiciel s'exécutant au démarrage de l'ordinateur, en arrière-plan, ne nécessitant pas d'intervention de l'utilisateur.

## <sup>vii</sup> **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 déboguer ces applications.

## <sup>viii</sup> **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.

## <sup>ix</sup> **Timer :**

Chronomètre déclenchant un évènement à chaque intervalle de temps (défini)

## <sup>x</sup> **JavaScript :**

Langage de script qui permet des interactions avec les différents objets d'une page Web