Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
rue Grandgagnage, 21
B-5000 Namur

Un logiciel d'illustration des protocoles GSM et GPRS sur la voie radio

Martin De Wulf

Mémoire présenté pour l'obtention du grade
de maître en informatique
Promoteur : le Professeur Olivier Bonaventure

Année académique 2000-2001

Je tiens à remercier les nombreuses personnes qui m'ont encouragé et aidé tout au long de ce travail.

Tout d'abord, je m'acquitte volontiers d'un devoir de gratitude envers le professeur Olivier Bonaventure, promoteur de ce mémoire de fin d'études. Ses remarques pertinentes m'ont été d'une aide précieuse.

Un tout grand merci aussi à Monsieur Xavier Lagrange, Maître de Conférences à l'ENST-Bretagne, pour m'avoir encadré aussi attentivement et pour sa formidable disponibilité lors de mon stage à Rennes.

Un grand merci à tous les relecteurs de ce mémoire : Gregory, Isabelle, Maman, Pierre, Stéphane et Thibaut qui m'ont aidé à rendre ce texte à peu près compréhensible.

L'aboutissement de ce mémoire doit beaucoup à ma famille et mes amis qui m'ont soutenu et encouragé durant ces longs mois de labeur. Un remerciement tout particulier à Charlotte qui m'a supporté et réconforté durant les moments difficiles.

Résumé

L'originalité principale de GSM et GPRS réside dans la façon dont ces systèmes gèrent les ressources radio qui leur ont été allouées. Mais comme souvent, l'optimisation a débouché sur une plus grande complexité. Par conséquent, l'immense succès de ces réseaux nécessite de plus en plus de personnes qualifiées.

Un logiciel permettant l'observation en direct des protocoles de communication utilisés sur la voie radio faciliterait la formation de ces personnes. Il permettrait un enseignement très vivant et concret et pourrait aussi servir à la mise en évidence des principes à la base de tout protocole de communication.

Ce mémoire a pour objectif la conception et l'implémentation en Java d'un tel logiciel. Il décrit en se concentrant sur la voie radio les principes de base de GSM et GPRS et inclut la spécification architecturale et fonctionnelle du programme. En outre, il explique comment on peut créer automatiquement un décodeur pour les messages échangés sur la voie radio dans GPRS à partir de la spécification officielle de ces messages.

Abstract

The main originality of the GSM and GPRS sytems rests in the way they manage the radio ressources which were allocated to them. As often, optimization led to a greater complexity. So the huge success of those networks requires more and more qualified people.

A software allowing the live observation of the protocols used on the radio way would ease the training of those people. It would allow a very animated and tangible teaching and could also be useful to put prominently the base principles of every communication protocol.

This thesis is aimed at the design and implementation in Java of such a software. It describes the base principles of GSM and GPRS, especially on the radio way, and includes the architectural and functional specification of the application. Moreover, it explains how a decoder of the messages exchanged on the radio way in GPRS can be created from the official specification of those messages.

Sommaire

Introduction

Le système GSM (*Global System for Mobile communications* représente un des succès industriels les plus marquants de ces dernières années. Ce système de téléphonie mobile a été standardisé par l'ETSI (*European Telecommunication Standards Institute*).

Vu la rareté de la ressource radio, ses concepteurs se sont concentrés sur une utilisation optimale de la bande de fréquence qui leur serait allouée. Et de fait, l'interface radio du système en constitue la principale orginalité. Mais comme souvent, l'optimisation a débouché sur une complexité accrue.

Pour enseigner le fonctionnement de GSM sur la voie radio, Xavier Lagrange, Maître de Conférences à l'ENST-Bretagne (*Ecole Nationale Supérieure des Télécommunications de Bretagne*), a développé un logiciel d'illustration des protocoles du système. Cette application, nommée GSM-Show, se base sur l'utilisation d'un mobile GSM de trace. Ce type d'appareil transmet à un ordinateur, via une liaison série, une trace des communications qu'il entretient avec le réseau. A partir de ces informations, GSM-Show représente de façon très lisible et quasiment en direct, le fonctionnement de l'interface radio GSM.

Xavier Lagrange a écrit son programme en Visual Basic et trouvait que son architecture manquait de souplesse. Il regrettait notamment que son programme ne puisse facilement intégrer l'utilisation d'autres mobiles de trace que celui dont il disposait. De plus, il souhaitait à moyen terme réaliser une application permettant d'illustrer le fonctionnement d'une nouvelle extension de GSM : GPRS. Enfin, il désirait porter son application sur d'autres plate-formes que Windows afin notamment de l'utiliser dans des salles de travaux pratiques dont les ordinateurs utilisent un système d'exploitation de la famille UNIX.

Ce mémoire va décrire la réalisation de ce double projet : porter le programme en améliorant son architecture et y ajouter une partie GPRS.

Structure de ce mémoire

Au chapitre 1, nous nous intéresserons aux différents concepts au coeur des systèmes GSM et GPRS. Cette étude a été nécessaire pour permettre le portage de GSM-Show et

le développement de nouvelles fonctionnalités relatives à GPRS.

Le chapitre 2 évoquera la réalisation du portage de GSM-Show en Java, langage choisi pour sa grande portabilité. Nous nous concentrerons principalement sur l'architecture du nouveau programme, conçue pour satisfaire aux exigences citées précédemment.

Le format binaire des messages échangés sur la voie radio dans GPRS a été spécifié au moyen d'une notation ad hoc nommée CSN.1 (*Concrete Syntax Notation 1*). Cette notation est compilable : il est possible de créer un programme appelé compilateur qui à partir d'un texte écrit dans cette notation génère un décodeur. Ce dernier est capable de déchiffrer les messages au format décrit dans le texte d'origine. Le chapitre 3 se penchera sur la réalisation d'un tel compilateur dont on décrira l'utilisation au chapitre 4.

En effet, au chapitre 4, nous nous pencherons sur la réalisation de la partie GPRS du programme : la façon dont nous l'avons spécifiée et en avons implémenté une partie.

Nous conclurons, finalement, sur les perspectives d'avenir du programme, ses éventuelles extensions futures et tenterons de donner au lecteur une estimation du travail déjà réalisé et restant à réaliser.

Chapitre 1

Introduction à GSM et GPRS

Ce premier chapitre introduit au fonctionnement des réseaux GSM et GPRS en insistant sur les techniques employées sur la voie radio. Nous présenterons tout d'abord quelques concepts utiles, puis l'architecture d'un réseau GSM classique pour finalement introduire celle d'un réseau GPRS.

Ce chapitre emprunte beaucoup à [BW97], [CG97], [LGT00] et [Zeg00].

1.1 Téléphonie cellulaire et autres concepts utiles

Un système de *radiotéléphonie* a pour but de permettre à un terminal d'accéder au réseau téléphonique sur un territoire d'une assez grande étendue (par exemple, un pays, voire un continent). Ce service utilise une liaison radioélectrique entre le terminal et le réseau.

La *téléphonie cellulaire* est un cas particulier de la radiotéléphonie. Un réseau est dit cellulaire s'il comprend une série de stations de base (*Base Transceiver Station, BTS*) qui chacune offre le service sur un petit territoire appelé *cellule*. Cette architecture se justifie de deux façons.

Elle permet premièrement de limiter la consommation électrique des stations mobiles (*MS, Mobile Station*) en leur évitant de devoir déployer une grande puissance d'émission. En effet, avec une telle architecture, un terminal est toujours assez proche du point d'accès au réseau avec lequel il dialogue.

Et elle permet deuxièmement d'économiser le spectre hertzien, c.-à-d. permettre un maximun de communications en parallèle dans les bandes de fréquence allouées au système. En effet, s'il n'y avait qu'une seule BTS pour un certain territoire, il n'y aurait moyen d'écouler simultanément qu'un nombre de communications étant le résultat de la division de la bande passante disponible par la bande requise pour une communication. On peut augmenter ce

nombre de communication possibles en réutilisant la même fréquence à plusieurs endroits sur le territoire. A cette fin, au lieu de placer une BTS émettant très fort au milieu du territoire, on va en placer une multitude émettant moins fort à intervalles réguliers. Les fréquences utilisées par deux BTS aux cellules contingentes seront différentes pour éviter les interférences.

Plusieurs problèmes émergent à cause de la mobilité des utilisateurs :

Tout d'abord, un utilisateur doit pouvoir passer d'une partie du territoire à un autre. Sa communication ne doit pas être interrompue parce qu'il quitte une cellule pour en gagner une autre. Une autre BTS doit prendre le relai. Cette procédure de changement de cellule par laquelle un terminal va se caler sur une autre fréqence porteuse pendant une conversation, de façon transparente à l'utilisateur, s'appelle le *handover*.

Ensuite, un autre problème posé par la mobilité est le volume de signalisation. En effet, garder en permanence une trace de la position de chaque MS allumée requiert un énorme flux de signalisation. Pour diminuer ce volume, le réseau ne cherchera pas à savoir à tout moment dans quelle cellule exactement se situe un terminal mobile. Il définit des sous-ensembles de l'ensemble des cellules, appelée *zones de localisation*, et se contente de savoir dans laquelle se trouve chaque utilisateur actif. Lorsque le mobile quitte ce sous-ensemble, c'est lui qui va provoquer une mise à jour de localisation dans les registres du réseau.

Le réseau ne connaît donc souvent que la localisation approximative d'un mobile et lorsqu'il doit l'atteindre, il fait un appel en diffusion, c'est-à-dire qu'il émet un appel contenant un identifiant du mobile sur toutes les cellules de la zone de localisation. Le mobile qui reconnaît son identifiant peut alors signaler sa position. Cette procédure d'appel en diffusion est appelé *paging*.

Mais pour qu'un appel en diffusion atteigne sa cible, celle-ci doit être à l'écoute. Tous les mobiles présents dans une cellule sont à l'écoute en état de veille sur une fréquence appelée *voie balise*. Cette écoute leur permet de s'informer d'éventuels appels qui leur seraient destinés. Cette voie sert aussi à transmettre des messages de synchronisation et à diffuser régulièrement la zone de localisation à laquelle appartient la BTS, ce qui permet au mobile de savoir quand il la quitte. Chaque BTS a sa propre voie balise.

La voie balise transmet des messages de synchronisation parce que le mobile doit pouvoir situer où commence chaque message dans le flux de données qu'il reçoit.

Un autre problème posé aux réseaux de téléphonie cellulaire est celui l'*itinérance* ou *roaming*. Un utilisateur abonné dans un certain réseau devrait pouvoir se rendre dans un territoire couvert par un autre réseau utilisant le même système et communiquer. Ce problème requiert la définition des modes d'interaction entre réseaux. Par exemple, on doit définir comment un réseau A signale à un réseau B la présence d'un de ses utilisateurs. Cette information est nécessaire pour pouvoir faire transiter les appels vers cet utilisateur.

1.2 GSM

1.2.1 Introduction

Le système de téléphonie cellulaire GSM (*Global System for Mobile communications*) est largement utilisé à travers toute l'Europe et est devenu peu à peu la référence pour la téléphonie cellulaire digitale à travers le monde. Il est passé par un long processus de normalisation géré par l'ETSI *European Telecommunication Standards Institute* pour arriver aujourd'hui à une certaine maturité. Ses limitations commencent à se faire sentir. Ses performances en téléphonie sont tout à fait satisfaisantes. Par contre en terme de transfert de données, en comparaison avec les réseaux fixes, il reste beaucoup de chemin à parcourir. Les deux défauts principaux sont les suivants :

- Les débits sont limités à 14,4 kbps (voire 9,6kbps dans la plupart des réseaux opérationnels).
- La transmission de données exige l'établissement d'une connexion et la réservation de ressources qui sont en général loin d'être utilisées pleinement tout au long de la connexion.

Mais la norme GSM ne cesse d'évoluer. Après la phase 1 qui en 1992 n'offrait que la téléphonie, la phase 2 a apporté les messages courts (SMS *Short Message Service*) et le transfert de données. A l'heure actuelle, la phase 2 introduit entre autres des terminaux bibandes (bandes de 900MHz et de 1800MHz), l'utilisation des concepts de réseau intelligent, et le routage optimal[1]. Plusieurs évolutions sont en cours d'élaboration :

Tout d'abord, GPRS, qui permettra d'augmenter les débits, de faire disparaître les temps de connexion et de facturer en fonction du volume transmis (et non plus de la durée de connexion). Cette évolution nous intéressera plus particulièrement dans ce mémoire.

Ensuite, EDGE, qui grâce à des techniques de modulation plus efficaces permettra l'augmentation des débits.

Enfin, ce qui est appelé la troisième génération (3G) et qu'on pourrait qualifier d'évolution "plus" : plus de débit, plus de services, plus de souplesse. La troisième génération est emblématiquement représentée par l'UMTS (*Universal Mobile Telecommunication System*). Plus qu'une évolution, il s'agit là d'un système concurrent utilisant d'autres bandes de fréquence et offrant des services plus attractifs. Cependant, l'avènement de l'UMTS ne signifiera pas la fin de GPRS. En effet, la solution pratique qui semble se dessiner est celle d'abonnements et de mobiles multi-plateformes permettant l'accès sur plusieurs types de réseaux. Etant donné la portée limitée et le coût important des installations UMTS, les

[1] A l'heure actuelle, un appel émanant d'un mobile A et dirigé vers un mobile B doit forcément passer par le réseau GSM ou le possesseur du mobile B a souscrit son abonnement , même si aussi bien lui que son correspondant se trouvent dans une zone géographique tout à fait différente. Le routage optimal vise à ne plus faire passer que du traffic de signalisation par le réseau où a été souscrit l'abonnement.

opérateurs n'assureront en effet pas une couverture complète des territoires actuellement couverts par GSM. Et là où UMTS sera absent, GPRS prendra sans doute le relai. UMTS est encore en cours de spécification. Néanmoins, on a déjà concédé dans de nombreux pays une partie de la bande hertzienne à ces services, souvent à prix d'or.

1.2.2 Caractéristiques

Un réseau de téléphonie mobile a plusieurs contraintes propres et la manière dont sont gérées ces contraintes est très caractéristique des réseaux GSM.

On va surtout insister ici sur l'interface radio qui occupe le centre de ce mémoire.

Tout d'abord, quelles sont les contraintes principales de la liaison radio :

- il est possible pour un tiers de pratiquer des écoutes indiscrètes
- il y a beaucoup d'interférences
- comme sur un vieux réseau Ethernet, le médium de transmission est partagé
- les bandes de fréquences sont limitées et doivent donc être utilisée au mieux

Pour gérer ces contraintes, GSM a opté pour un système TDMA à saut de fréquence et à duplexage fréquentiel.

La technique de multiplexage temporel, dite TDMA (*Time Division Multiple Access*), est fréquement utilisée dans le domaine des réseaux. Nous n'en expliquerons le fonctionnement que dans le cas particulier de la voie radio GSM : étant donné qu'une bande de fréquence dans GSM peut véhiculer huit fois le débit d'une conversation téléphonique, on va segmenter le temps d'émission/réception en 8 intervalles de temps répétés à l'infini, qu'on appelera *slots*. Chacun sera alloué à une conversation différente où à de la signalisation. Une de ces séquences de 8 slots est appelé une *trame*. On optimise ainsi l'utilisation de la voie radio.

Le saut de fréquence consiste pour un mobile à changer de fréquence pour chaque slot temporel qu'il utilise. La suite de fréquences peut-être cyclique ou pseudo-aléatoire. On utilise cette technique parce que les interférences ne sont pas réparties équitablement entre les fréquences et que de cette façon, les erreurs sont réparties entre un maximum de flux utilisateurs. On améliore ainsi l'efficacité des codes correcteurs d'erreur. La figure 1.1 illustre les concepts de trames et de slot. Les flèches représentent le saut de fréquence.

Le duplexage fréquentiel consiste à séparer en fréquence la voie montante et la voie descendante d'une connexion. Pour GSM dans la bande 9OOMHz, l'écart entre la voie montante et la voie descendante est fixé à 45MHz. Comme expliqué plus haut, 8 communications peuvent être multiplexées sur chaque couple de fréquence (voire 16, si on utilise des algorithmes de compression de la voix plus performant).

Nous n'insistons pas sur les notions de saut de fréquence et de duplexage fréquentiel qui

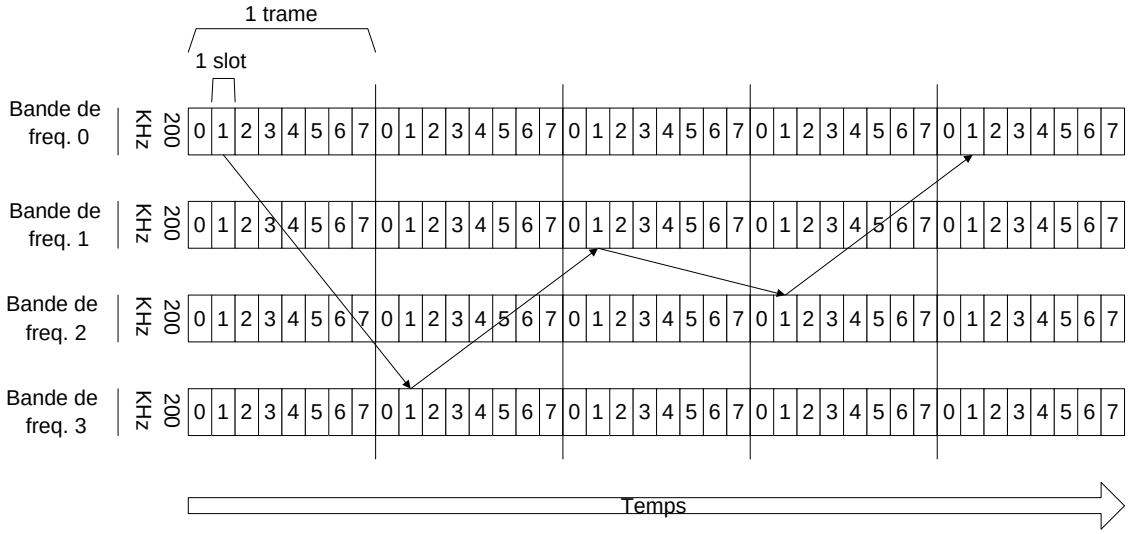


FIG. 1.1 – TDMA avec saut de fréquence

ne sont pas indispensables à la compréhension de la suite. Nous considèrerons d'ailleurs par la suite qu'il n'y a pas de saut de fréquence et que donc un mobile reste sur une seule fréquence pendant une communication, ce qui facilitera grandement la compréhension sans nuire à la généralité du propos.

Pour gérer la complexité, les différents flux de données qui peuvent transiter par la voie radio sont classés en fonction du type de données transportées. En fonction de ce type, il seront transmis dans différents *canaux logiques*.

Un canal logique est une suite de slots qui véhiculent un flux particulier. La caractéristique qui les désigne est à la fois physique, correspondant à une séquence de slots bien définie, et logique, relative aux fonctions qu'ils servent. Par exemple, le côté logique du canal logique BCCH (*Broadcast Control CHannel*) est qu'il sert à transmettre aux mobiles situés à portée de l'émission d'une BTS des informations sur la cellule et le côté physqiue est sa position dans les trames, dans le slot 0 des trames de la voie balise. Les canaux logiques sont séparés en deux types : les canaux de données et les canaux de signalisation. On distingue aussi les canaux communs, utilisés par plusieurs mobiles simultanément, des canaux dédiés, réservés à un seul mobile.

Afin de repérer physiquement les canaux logiques dans les slots et les trames, la norme utilise le notions de *multitrame*, *supertrame* et *hypertrame*.

Une multitrame à 51 regroupe 51 slots situés à la même place dans 51 trames consécutives. Les multitrames se succèdent à l'infini. La norme utilise cette structure pour définir où doivent se situer les différents canaux logiques dans le flux de trames. Par exemple, elle précise que le canal logique SCH (*Synchronization CHannel*) occupe toujours les slots 0, 10, 20, 30, 40 et 50 d'une multitrame à 51. Attention, il faut bien comprendre qu'il y a

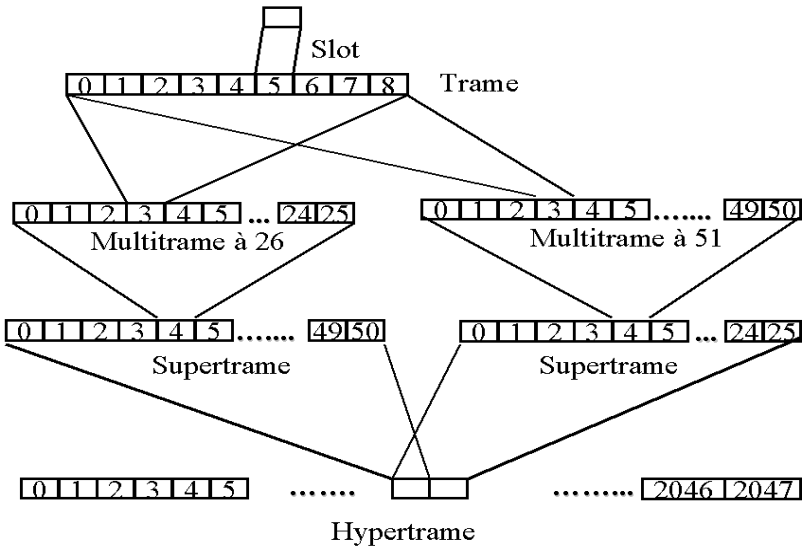


FIG. 1.2 – Structuration logique des slots

une double numérotation, une dans la structure de trame et une autre dans la structure de multitrame. Autrement dit, un slot porte un numéro compris entre 0 et 7 qui le situe dans une trame et un autre numéro compris entre 0 et 50 qui le situe dans une multitrame. Le contexte indiquera en général lequel de ces deux numéros on utilise. Quand on parlait des slots 0, 10..., on citait leur numérotation dans la structure de multitrame.

Pour certains canaux dédiés, on utilise une structure de *multitrame à 26*. Les deux structures de multitrames peuvent être utilisées pour se repérer dans la même bande de fréquence mais pour des slots différents. On peut utiliser par exemple la multitrame à 51 pour se repérer les slots numérotés 0 dans chaque trame et la multitrame à 26 pour les autres.

La structure supérieure est la supertrame. Elle se constitue de soit 26 multitrames à 51 soit 51 multitrames à 26. Cette structure de supertrame sert juste à agréger les différents types de multitrames. Ici aussi, les supertrames se succèdent à l'infini.

Enfin, une hypertrame se constitue de 2048 supertrames. Un structure d'hypertrame est donc constituée de $2048 * 56 * 21$ trames . C'est en référence à cette structure que la norme définit le Frame Number de chaque trame : c'est sa position dans la structure d'hypertrame courante. Une hypertrame dure 3h 28min 53s et 760ms. Après l'écoulement de ce temps, toute une série de valeurs évoluant cycliquement dans le réseau vont reprendre leur valeur initiale.

1.2.3 Application concrète

Voici une application concrète des différents principes présentés jusqu'ici ; intuitivement, comment fait un mobile pour se "caler" sur une BTS :

Tout d'abord, il scrute toutes les fréquences situées dans le spectre de GSM. Il va repérer un certain signal émis par chaque BTS qu'il perçoit et sélectionner celui qu'il reçoit le mieux. Ce signal est présent sur le slot numéroté 0 dans une trame de la voie balise, ce qui permet au mobile d'à la fois repérer la fréquence de la voie balise et de se repérer dans la structure de trames. Ensuite, le mobile va se synchroniser plus finement en écoutant le canal SCH (*Synchronization Channel*) qui est le premier de la trame TDMA suivante. Dans les informations du SCH, il va trouver le Frame Number et un code de couleur (BSIC *Base Station Identification Code*) qui sert à différencier les BTS peu éloignées qui émettent sur la même fréquence balise. A partir de ce moment, il est synchronisé avec la BTS.

Maintenant, il va écouter sur certains slots bien définis de la voie balise les informations dont il a besoin. Il les trouve dans le canal logique BCCH dont l'emplacement est fixe dans la structure de multitrames à 51, ce qui lui permet de s'y retrouver. Ces informations sont entre autres :

- les paramètres de sélection de la cellule qui permettent à un mobile de déterminer s'il peut se mettre en veille sur la cellule
- le numéro de zone de localisation qui permet à un mobile de savoir s'il doit procéder à une inscription, comme il doit le faire lorsqu'il change de zone de localisation
- les paramètres RACH (*Random Access Channel*) qui contiennent les règles d'accès aléatoire que le mobile devra utiliser s'il veut se signaler à la BTS
- la description de l'organisation des canaux de contrôle communs qui indique au mobile les slots à écouter pour détecter les pagings.

Le mobile va alors se signaler au réseau, pour faire connaître sa localisation . Il le fait par un accès aléatoire sur le canal logique RACH (*Random Access CHannel*) dont l'emplacement physique dans les structures de multitrame lui a été précisé dans les informations du BCCH. Si cet accès réussit, le réseau va diffuser sur le canal logique AGCH (*Access Grant CHannel)* un message d'allocation de canaux logiques dédiés au mobile à la fois sur la voie descendante et la voie montante. Sur ces canaux logiques, le mobile va négocier des paramètres de cryptage et ensuite transmettre un de ses identifiants. Dès lors, le réseau sait quel mobile exactement l'a contacté et peut enregistrer sa zone de localisation. Ensuite, le mobile émettra régulièrement un message pour signaler au réseau qu'il est toujours à l'écoute. Si ce message n'est pas reçu pendant une certaine période, assez longue, ce mobile sera considéré comme inaccessible par le réseau.

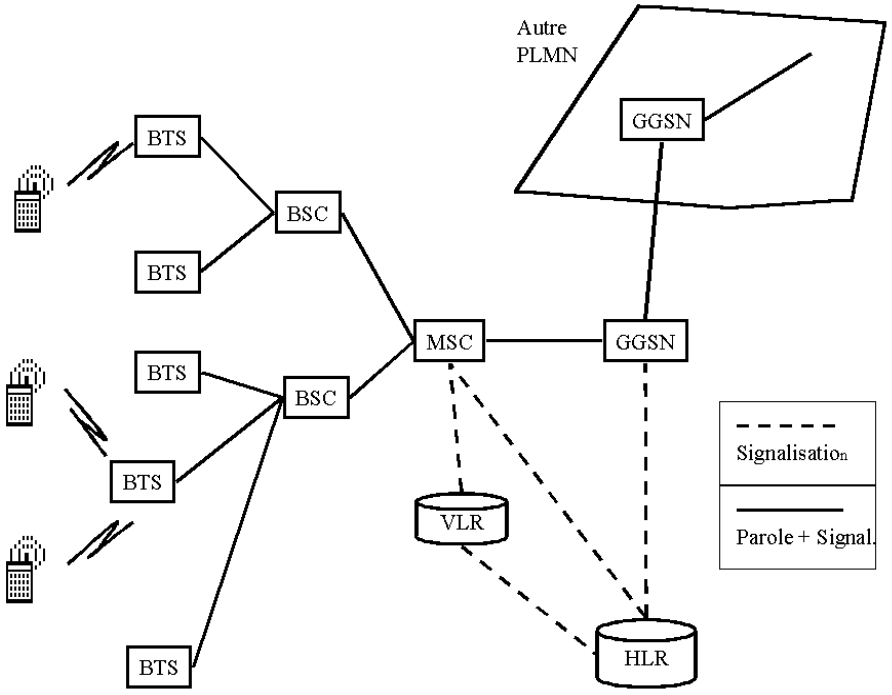


FIG. 1.3 – Architecture d'un réseau GSM

1.2.4 Architecture

L' architecture d'un réseau GSM est spécifiée dans la norme de l'ETSI. Ceci permet l'interaction, nécessaire à l'itinérance, de plusieurs réseaux installés par des opérateurs différents mais aussi l'achat des différents composants d'un réseau auprès de différents fournisseurs. Plusieurs entités sont définies dans la norme. Ce qu'on entend par entité, c'est un équipement physique, doté d'une certaine intelligence, d'une capacité à traiter de l'information. On trouvera à la figure 1.3 une représentation de cette architecture. Voici un cours descriptif fonctionnel de ces entités [2] :

les BTS(*Base Transceiver Station*) émetteur-récepteur gérant une cellule. Gère la couche physique sur la voie radio : modulation, démodulation, CRC's, multiplexage, saut de fréquence, chiffrage. Il gère également la couche liaison de donnée avec le mobile (protocole LAPDm).

les BSC(*Base Station Controller*) commutateur qui réalise un première concentration de circuit. S'occupe de la gestion de la ressource radio (allocation des canaux, décision du handover...).

les MSC(*Mobile-services Switching Center*) commutateur pour des entités mobiles. Gère l'établissement de circuits à travers le réseau, les SMS et l'exécution du handover.

[2] On a omis certaines fonctions qu'il n'est pas nécessaire de comprendre pour la suite.

Certains MSC peuvent être des GMSC (*Gateway MSC*), et servent alors de passerelle avec un autre réseau de téléphonie.

le HLR(*Home Location Register*) base de donnée centralisant les informations d'identification et de localisation de chaque abonné du réseau.

les VLR(*Visitor Location Register*) base de données agissant à la fois comme un tampon évitant des accès trop fréquent au HLR et comme élément d'une base de donnée distribuée dans tous les VLR et HLR GSM. Généralement placé à proximité d'un MSC (souvent dans le même équipement). Contiennent les données d'abonnement des abonnés présent dans une zone géographique.

1.2.5 Identifiants

Enfin, il sera utile de savoir qu'un mobile a plusieurs identifiants :

IMSI (*International Mobile Subscriber Identity*) Cet identifiant est celui d'un abonnement, il est stocké dans la carte SIM du mobile. Il est transmis aussi rarement que possible sur la voie radio pour préserver l'anonymat de l'utilisateur

TMSI(*Temporary Mobile Subscriber Identity*) Cet identifiant est utilisé sur la voie radio autant que possible en lieu et place de l'IMSI pour compliquer la tâche d'éventuelles écoutes indiscrètes. Il est défini au début de l'interaction du mobile avec le réseau et régulièrement mis à jour pour compliquer toute tentative de repérage de l'utilisateur.

IMEI(*International Mobile Equipment Identity*) Cet identifiant est celui de l'appareil(sans carte SIM). Il est peu utilisé à l'heure actuelle mais devrait permettre d'interdire l'accès au réseau à du matériel volé ou fonctionnant mal.

MSISDN(*Mobile Station ISDN Number*) Cet identifiant est le numéro de téléphone correspondant à l'abonnement. Une table de correspondance IMSI/MSISDN est stockée dans le HLR et le MSISDN n'est jamais transmis sur la voie radio.

Cette profusion d'identifiants sert principalement à éviter le repérage d'un utilisateur par des écoutes indiscrètes.

1.2.6 La pile de protocoles gérée par la MS(*Mobile Station*)

La complexité des fonctionnalités que l'on désire obtenir au niveau de la MS(*Mobile Station*) est classiquement gérée par une pile de protocoles.

On trouvera à la figure 1.4 une représentation de la pile de protocole gérée par la MS.

La couche liaison de données est gérée par le protocole LAPDm qui est de la famille de HDLC. Ses caractéristiques principales sont : taille fixée des paquets, codes détecteurs d'erreur, numérotation des trames, correction par retransmission sélective (mécanisme ARQ, *Automatic Repeat Query*). Le protocole LAPDm permet aussi bien un mode avec connexion

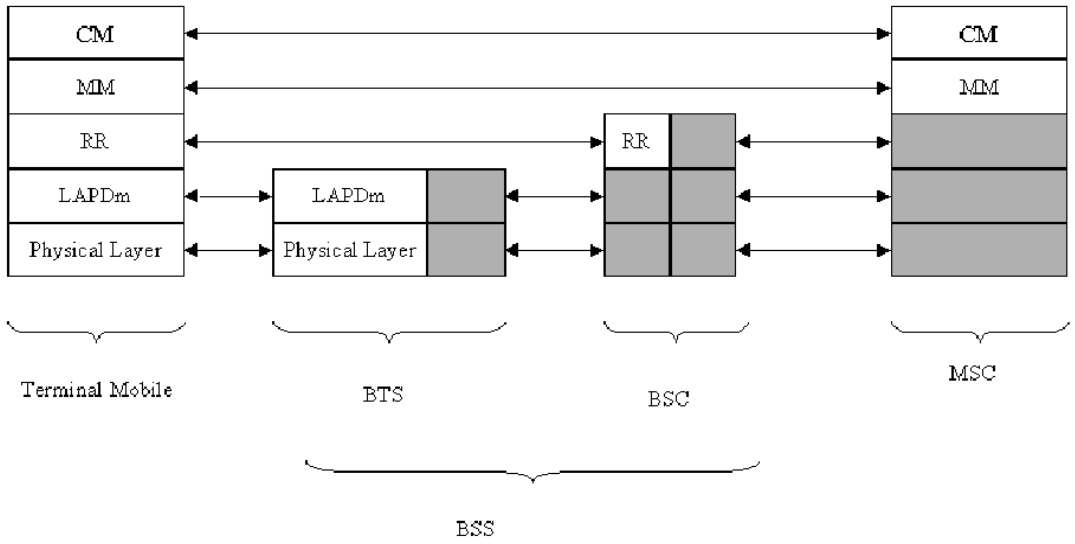


FIG. 1.4 – La pile de protocoles gérées par la MS

qu'un mode sans connexion pour le transfert de données. Il est possible de multiplexer différents flux sur le même canal physique. La distinction des flux se fait grâce au SAPI(*Service Access Point Identifier*).

La couche 3 comprend trois sous-couches. Si la norme les présente comme une pile, dans les fait, ces trois couches agissent en parallèle. Autrement dit, il n'y a pas d'encapsulation entre les couches. Ces trois couches sont :

RR (*Radio Ressource*) Cette couche gère la connexion radio et principalement l'établissement (ou le réétablissement) d'un canal dédié lors de la procédure de handover. L'entité RR de la MS dialogue avec un BSC.

MM (*Mobility Management*) Cette couche gère la gestion de la mobilité, principalement, les mises à jours de localisation, et assure les fonctions de sécurité.

CM (*Connection Management*) Cette couche est très semblable à ce qui existe sur les réseaux téléphoniques fixes pour gérer les appels entre abonnés. Les problèmes liés à la mobilité en ont en effet été autant que possible supprimés pour être déplacés vers la couche MM.

1.2.7 Canaux logiques

Les différents canaux logique GSM sont séparés en deux classes :

- Les canaux dédiés à un mobile :

  TCH (*Traffic CHannel*) **:** Réservé au transfert de la voix (ou des données en mode

circuit).

SDCCH (*Stand-alone Dedicated Control CHannel*) **:** Permet au mobile de transférer de la signalisation sur la voie montante quand une conversation n'est pas en cours. C'est sur cette voie que transite les SMS.

SACCH (*Slow Associated Control CHannel*) **:** Durant une conversation, c'est cette voie qui est utilisée pour remonter au réseau les mesures effectuées par le mobile ainsi que d'autres éléments de signalisation. Sert à assurer le bon déroulement de la conversation.

FACCH (*Fast Associated Control CHannel*) **:** Lorsqu'en cours de conversation, en phase de handover, le besoin se fait sentir d'un débit élevé pour la signalisation, on crée un FACCH. Les ressources radio sont "volées" au TCH, pour transmettre ce surplus de signalisation.

– Les canaux communs à plusieurs mobiles :

BCCH (*Broadcast Control CHannel*) **:** diffuse les informations systèmes(voir supra)

PCH (*Paging CHannel*) **:** diffuse les recherches d'utilisateurs par paging

RACH (*Random Access CHannel*) **:** utilisé pour les accès aléatoires que réalise un mobile pour demander l'allocation de canaux dédiés. C'est le seul canal commun sur la voie montante.

AGCH (*Access Grant CHannel*) **:** canal de la voie descendante par lequel se réalise l'allocation de canaux dédiés

1.3 GPRS

1.3.1 Introduction : les services offerts

Tout d'abord, nous allons procéder à une petite mise au point terminologique. Etant donné que GPRS (*General Packet Radio Service*) est partie intégrante de GSM, pour différencier les services offerts par GPRS de ceux offerts par la norme GSM "classique" on parlera non pas de GPRS et GSM, mais de GPRS et GSM-Circuit (ou GSM-C). De plus, il faut noter qu'on désigne le protocole réseau utilisé au dessus de GPRS par le terme générique PDP(*Packet Data Protocol*). Et par extension, un réseau interagissant avec un mobile à travers l'architecture GPRS est appelé réseau PDP. Dans la plupart des cas, il s'agira d'un réseau IP.

GPRS n'apporte pas à vrai dire de nouveaux services à l'utilisateur, puisque le transfert de données est déjà disponible au moyen d'un terminal à la norme GSM. Ce que GPRS apporte, c'est une augmentation des débits et une plus grande souplesse d'utilisation : connexion quasi-instantanée et facturation au temps de transmission réel, voire au volume réellement transmis. Pour bien comprendre, il faut voir qu'aussi bien GSM-C que GPRS ne fournissent pour les données que les deux premières couches du modèle OSI : les couches

physique et liaison de données. Les différences se trouvent dans la façon dont sont assurées les fonctions de ces deux couches :

- Dans GSM-C pour réaliser un transfert de données, on établit un circuit physique virtuel entre deux points, au moyen duquel seront transmises les données. Les ressources sont réservées pour toute la durée de l'échange, même si elles ne sont utilisées en fait que pendant une infime portion du temps de connexion. Ceci est très courant pour de nombreuses applications classiques, par exemple, la consultation de documents html. Ensuite, on utilise un protocole point à point pour fiabiliser le transfert le long de ce circuit. Par exemple pour un utilisateur voulant accéder à un réseau datagrammes comme Internet, GSM-C établit un circuit entre la MS et une passerelle appartenant à un fournisseur d'accès. On utilise ensuite le protocole PPP (ou SLIP) pour faire communiquer la MS et cette passerelle. Le tout (circuit + PPP) fournit les couches 1 et 2 du modèle OSI.
- Dans GPRS ce n'est plus un circuit qui permet la communication entre le terminal mobile et la passerelle mais bien un réseau à commutation de paquets avec concurrence d'accès aux ressources et possibilité d'associer des qualités de services par flux. Sur la voie radio, l'allocation de ressources se fait à la demande du mobile pour transmettre un volume de données fixé au préalable. Les ressources allouées sont donc adaptées au volume à transmettre.

Pour conclure, s'il est juste de dire que GPRS n'est qu'une amélioration de services déjà disponibles, il faut se rendre compte de l'importance de l'évolution qui doit être faite au sein du réseau GSM pour assurer ces améliorations. C'est une refonte très complexe du réseau.

1.3.2 Caractéristiques

GPRS utilise les mêmes bandes de fréquence que GSM-C. La cohabitation des deux services se fait sur base des structures de slots et de trames décrites au point 1.2.2 et à la figure 1.1.

Pour augmenter les débits disponibles pour l'utilisateur, on utilise principalement deux techniques finalement très simples :

- la diminution de la protection des données utilisateurs (moins d'overhead)
- l'utilisation pour transporter des données de plusieurs slots sur une trame TDMA, au lieu d'un seul dans GSM-C

La deuxième de ces techniques repose sur le principe du multiplexage statistique : par multiplexage statistique, on entend permettre à un utilisateur de se voir allouer ponctuellement bien plus que

$$\frac{bande\ passante}{nombre\ d'utilisateurs}$$

si les autres utilisateurs n'utilisent pas leur portion de bande passante. Dans ce cas, notre

utilisateur récupère les ressources inutilisées par les autres.

La norme définit de nouveaux canaux logiques GPRS dont l'allocation est flexible : sur une fréquence, le réseau peut choisir de leur allouer la totalité ou seulement une partie des 8 slots de chaque trame. Autrement dit, l'utilisation d'une fréquence peut être partagé entre la parole et les données. La répartition de ces slots consacré à GPRS entre les utilisateurs actifs se fait séparément sur les voies montante et descendante. Le chapitre 4 se concentrera plus en détail sur les mécanismes d'allocation de la ressource radio pour GPRS.

Plusieurs schémas de codage sont définis pour permettre des taux de transfert de 8 Kbps à plus de 150 Kbps (inaccessibles dans les faits). Chaque schéma de codage offre une protection des données plus ou moins importante.

La norme définit l'interaction entre réseaux GPRS et réseaux IP, et entre réseaux GPRS et réseaux X.25. Les données utilisateurs sont transférées de manière transparente entre le terminal mobile et les réseaux de données externes par une technique de "tunneling". Le tunneling consiste à encapsuler les messages transmis par la MS dans des paquets d'un autre protocole. Le réseau n'utilise pas les informations du paquet utilisateur pour le router jusqu'à une passerelle. Cette dernière va, en réception du message, décapsuler le message de la MS et le transmettre sur un réseau PDP correspondant. Ceci permet au réseau GPRS de ne gérer qu'un seul protocole dans son backbone et facilite l'utilisation du réseau GPRS avec de nouveaux protocoles de données dans le futur.

Pour gérer les problèmes de mobilité, GPRS utilise le même système de paging que GSM-C mais définit des sous-ensembles de cellules plus petits que ceux de GSM-C. On appelle ces sous-ensembles des *zones de routage*. Une zone de routage est toujours totalement incluse dans une zone de localisation.

Les intérêts principaux de GPRS sont les temps d'accès, de l'ordre d'une seconde pour commencer un transfert de données, et la possibilité de facturer en fonction du volume de données transféré, plutôt qu'en fonction du temps de connexion.

1.3.3 Architecture

La mise en place du service GPRS sur le réseau GSM actuel nécessite le rajout à l'architecture envisagée au point 1.1.4 de nouvelles entités de réseau dédiées à l'acheminement des paquets :

SGSN (*Serving GPRS Support Node*) **:** Entités associées à des zones de routage. Les SGSN communiquent d'un côté avec les terminaux mobiles et d'un autre côté avec le "backbone" du réseau GPRS. Les SGSN gardent trace de la localisation des MS et assurent les fonctions de sécurité et de contrôle d'accès sur la voie radio.

GGSN (*Gateway GPRS Support Node*) **:** Passerelles entre le backbone du réseau GPRS et un réseau fixe de transmissions de données. (PDN *Public Data Network*). Elles sont les points d'accès principaux au réseau GPRS.

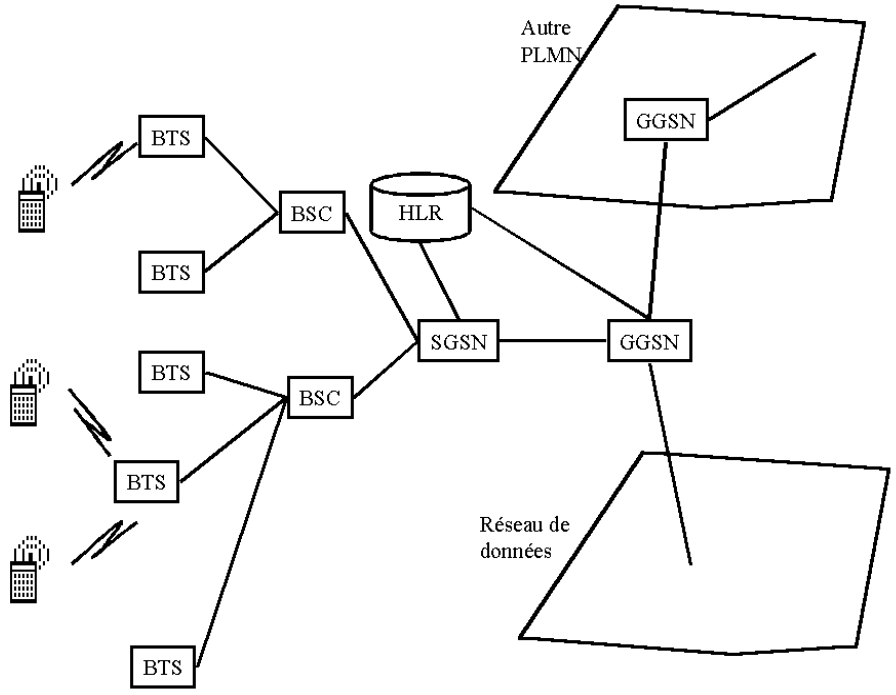


FIG. 1.5 – Architecture d'un réseau GPRS

On trouvera une illustration de l'architecture d'un réseau GPRS à la figure 1.5.

La norme spécifie que le réseau qui relie tous les SGSN et GGSN doit être un réseau IP.

Comme on peut le voir à la figure 1.5, GSM-C et GPRS partagent l'infrastructure radio, c.-à-d. l'ensemble des BTS et des BSC. Pour le reste de leur parcours, les flux GSM-C et GPRS sont séparés.

Dans GPRS, à partir du SGSN, les messages PDP envoyés par une MS sont encapsulés dans des paquet IP et transférés à un GGSN. C'est la technique de tunneling déjà évoquée. Le GGSN choisi dépend du type de message encapsulé. En effet, un GGSN ne gère l'accès qu'à un seul type de réseau PDP. Le deuxième critère de choix du GGSN est sa proximité avec la zone de routage de la MS.

Le GGSN qui reçoit le paquet IP va décapsuler le message PDP de la MS qui y est contenu et l'envoyer sur le réseau PDP.

En plus de ces deux entités, on va rajouter de l'information dans le HLR pour gérer GPRS. On rajoute au HLR les informations d'abonnement GPRS, la zone de routage des abonnés, quand on la connaît, et une table de correspondance entre adresses PDP et identifiants de MS.

1.3.4 Deux scénarios d'illustration

Nous allons envisager deux scénarios pour faciliter la compréhension du fonctionnement d'un réseau GPRS. Tout d'abord, l'émission d'un message par une MS à direction d'un réseau PDP et ensuite, l'émission d'un message par une entité d'un réseau PDP à destination d'une MS. On supposera par la suite que le réseau PDP en question est Internet et que le message émanant ou à direction de la MS est un paquet IP.

MS -> Internet

Pour transmettre un message à destination d'une adresse IP, la MS va d'abord essayer d'obtenir des ressources sur la voie radio. Pour ce faire, elle va se signaler par un accès aléatoire au réseau. Si cet accès réussit, le réseau va lui réserver un canal logique sur lequel ils vont pouvoir dialoguer.

Après s'être identifié, avoir exprimé sa requête et reçu une allocation de ressources, la MS va transmettre son paquet IP qui va parvenir jusqu'au SGSN. Celui-ci va demander au HLR l'adresse d'un GGSN offrant l'accès à Internet. Quand il disposera de cette information, le SGSN va encapsuler le paquet IP de la MS dans un autre paquet IP portant l'adresse du GGSN. Le paquet IP de la MS est donc encapsulé dans un autre paquet IP.

Lorsqu'il reçoit ce paquet, le GGSN en question décapsule le paquet IP de la MS et l'envoie sur Internet.

Internet -> MS

Quand un ordinateur connecté à Internet souhaite envoyer un paquet IP à un abonné GPRS, il l'envoie comme n'importe quel paquet. Ce paquet est routé vers un GGSN du réseau GPRS parce qu'il contient une adresse IP qui le destine à ce réseau.

Quand le GGSN reçoit le paquet en question, il peut s'adresser au HLR pour connaître la zone de routage actuelle de l'abonné qui possède l'adresse IP. Quand le GGSN dispose de cette information, il va envoyer le message PDP encapsulé dans un paquet IP au SGSN qui gère cette zone de routage.

Ce SGSN peut alors contacter la MS en faisant du paging pour savoir exactement où il se trouve. Il va ensuite lui signaler quand et sur quelle fréquence écouter. Au moment convenu, le mobile va se mettre à l'écoute et le SGSN va lui transmettre le paquet IP.

1.3.5 Identifiants

De nouveaux identifiants ont du être définis pour être utilisés dans les différents protocoles GPRS :

P-TMSI(*Packet TMSI*) **:** C'est l'équivalent du TMSI pour GSM-C. Il est nécessaire de disposer d'un identifiant supplémentaire du mobile dans le réseau parce que le mobile peut être à la fois actif en GPRS et en GSM-C. C'est pour cela qu'on n'utilise pas le TMSI pour les deux services et qu'on a créé le P-TMSI.

TLLI (*Temporary Logical Link Identity*) **:** Identité temporaire qui identifie un mobile particulier pour le SGSN. Choisi aléatoirement par le mobile à l'initialisation d'un flux de donnée s'il ne s'est pas encore vu allouer de P-TMSI. A chaque fois que le mobile possède un P-TMSI valable, le TLLI est égal au P-TMSI.

1.3.6 La pile de protocoles gérée par la MS

En fait, dans GPRS, une MS ne gère pas une mais des piles de protocoles situés dans deux plans différents : le plan de signalisation et le plan de transmission. Le plan de transmission sert à transférer toutes les données utilisateurs. Le plan de signalisation sert quant à lui à assurer la gestion de la mobilité (MM : *Mobility Management*), mais aussi la transmission de messages courts.

Mais en fait, seul les sommets de ces deux piles diffèrent. Dans le plan de signalisation on trouve au sommet la couche GMM (*GPRS Mobility Management*) surmontée des couches SM (*Session Management*) et GSMS (*GPRS SMS*). Dans le plan de transmission, on trouve au sommet de la pile le protocole SNDCP surmonté de différents PDP (*Packet Data Protocol*). On trouvera une illustration de ces deux piles aux figures 1.6 et 1.7.

couche physique

La structuration du temps sur une fréquence en slots et en trames est reprise à GSM-C, ce qui permet de faire cohabiter les deux types de flux sur une même bande de fréquence. Par contre, une nouvelle structure du niveau des multitrames de GSM-C est défini : la multitrame à 52 slots[3]. On approfondira cette notion dans le chapitre 4.

Couche RLC

Le protocole RLC assure la segmentation des paquets LLC en blocs RLC/MAC et effectue les retransmissions nécessaire quand un paquet s'est perdu (mécanisme de type ARQ *Automatic Repeat Query*). Ce protocole est étroitement lié à l'interface radio utilisée.

[3]Pour rappel dans GSM-C on utilise des multitrames à 26 et 51

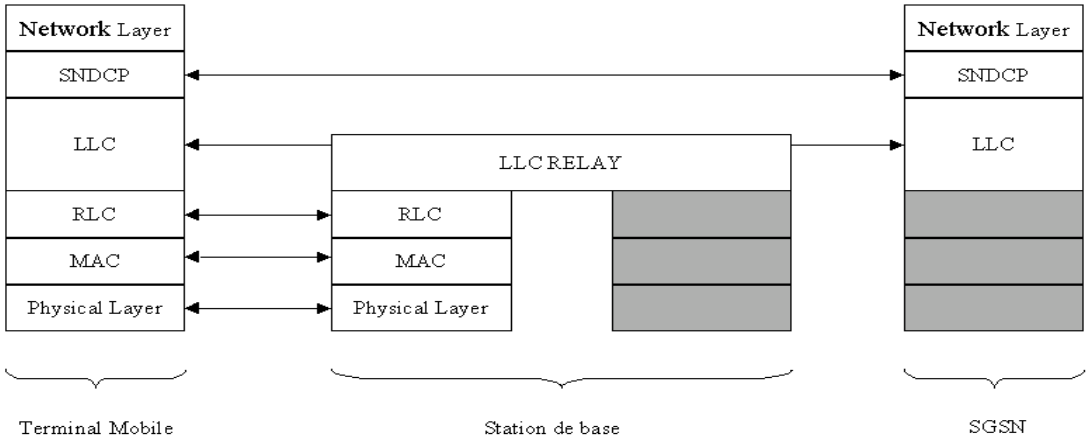


FIG. 1.6 – Pile de protocoles dans le plan de données

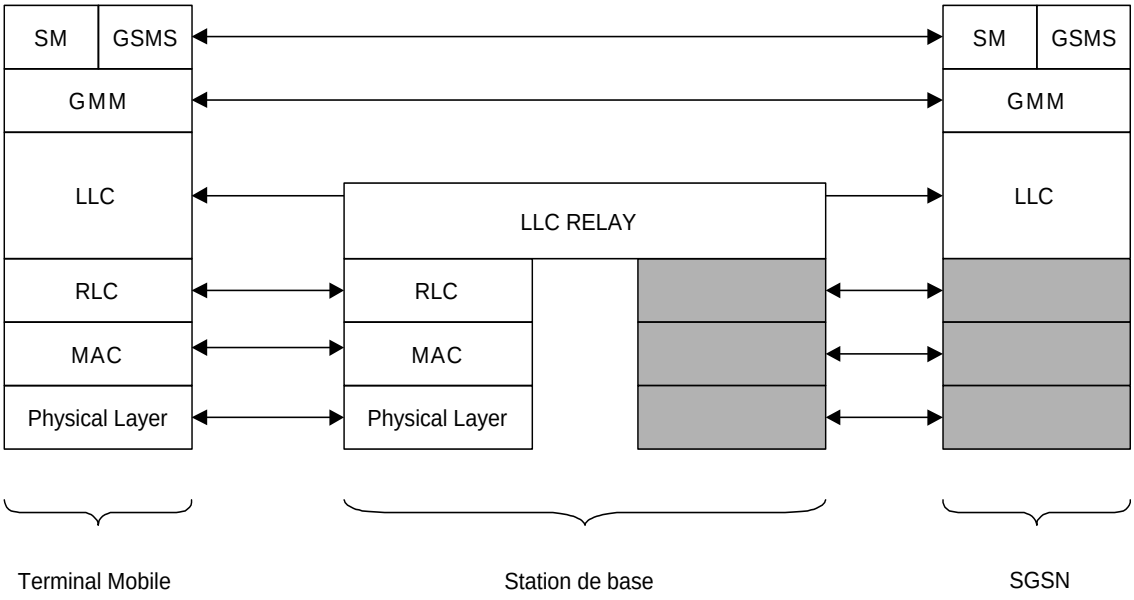


FIG. 1.7 – Pile de protocoles dans le plan de signalisation

couche MAC

Le protocole MAC gère l'accès concurrent aux différents canaux physiques. Un mobile peut se voir allouer plusieurs intervalles de temps d'une trame TDMA s'il est capable de les gérer (terminaux multislots).

Les en-têtes des protocoles RLC et MAC sont mélangées. La séparation est donc assez factice d'autant plus que les deux protocoles sont définis dans le même document. On parlera donc souvent de la couche RLC/MAC.

Couche LLC

Le protocole LLC doit :

- fournir un lien logique fiable entre la MS et le SGSN.
- être indépendant des protocoles sous-jacents à l'interface radio pour permettre l'introduction de nouvelles solutions radio pour GPRS avec un minimum de changement au NSS (*Network Sub-System*).
- supporter des paquets d'information de taille variable
- supporter un mode avec acquittement et un autre sans
- permettre de faire une distinction de qualité de service entre les différents types d'utilisateur
- préserver la confidentialité des identités des utilisateurs

Couche SNDCP

SNDCP gère le multiplexage de plusieurs connexions PDP, la compression/décompression des en-têtes[4] et la compression/décompression des données. Ce protocole assure le respect de la séquence des messages. Le protocole SNDCP assure aussi la segmentation (et la reconstitution) des paquets de données pour fournir des blocs de données de taille acceptable pour le protocole LLC.

Couche GMM

La couche GMM (*GPRS Mobility Management*) va gérer, comme son nom l'indique, l'itinérance du terminal dans le réseau. Cette couche n'est pas vraiment en-dessous de SM étant donné qu'il n'y a pas d'encapsulation entre les couches. GPRS reprend le principe de groupement en zone des cellules de GSM-C, mais ces zones sont plus petites : elles sont appelées zone de routage (ou *routing area*, RA)[5].

[4] pour le moment, seule la compression des en-têtes IP est définie

[5] Si GSM-C et GPRS cohabitent, une RA est forcément un sous-ensemble d'une zone de localisation

Couche SM

La couche SM permet au mobile de demander au réseau la mémorisation d'un contexte, appelé contexte PDP, dans le SGSN et le GGSN. Ce contexte va servir à ces deux entités à router les paquets en provenance du mobile ou destinés à celui-ci sans devoir consulter les bases de données de localisation. Il comprend principalement :

- le type de réseau utilisé (X.25, IP...)
- l'adresse PDP du mobile(adresse IP par exemple)
- l'adresse IP du SGSN gérant la cellule où se trouve l'abonné
- le point d'accès au service réseau utilisé (NSAPI, *Network Service Access Point*)
- la qualité de service négociée

Un abonnement GPRS contient la souscription d'une ou plusieurs adresses PDP. Chaque adresse PDP, s'il elle est actuellement en usage, est décrite par un contexte PDP individuel dans la MS, dans le SGSN et dans le GGSN . Chaque contexte PDP peut être individuellement actif ou inactif.

Les adresses PDP peuvent être allouées dynamiquement ou statiquement. Dans le cas dynamique, ce peut-être le réseau où l'abonné a souscrit l'abonnement qui va attribuer l'adresse ou ce peut être un autre réseau auquel le mobile est attaché pour le moment.

1.3.7 Canaux logiques

La notion de canal logique perd beaucoup de son sens dans GPRS, comme on l'expliquera au chapitre 4. Seuls les canaux communs gardent un certain sens. Ils sont très semblables à ceux de GSM-C. D'ailleurs l'utilisation des canaux communs spécifiques à GPRS est facultative. Il est possible de n'utiliser que les canaux GSM-C pour la signalisation et les accès aléatoires.

1.4 Conclusion

La profusion d'exception aux règles générales représente la principale difficulté de la présentation de GSM/GPRS. Ce chapitre n'a pu tout présenter, il sert simplement à introduire ou rappeler au lecteur des principes nécessaires à la compréhension de ce mémoire.

Si certains de ces principes ne serviront pas directement dans les chapitres qui suivent, ils ont néanmoins dû être compris pour avancer dans ce mémoire. Et à terme, ils devraient tous intervenir dans le projet J-GSM-Show.

Chapitre 2

Portage et amélioration d'un logiciel d'illustration des protocoles de GSM sur la voie radio

Dans ce chapitre, nous décrirons la partie GSM du projet : J-GSM Show(*Java-GSM Show*). Il s'agit d'un logiciel d'illustration du fonctionnement du système GSM et notamment de la pile de protocoles gérée par un terminal mobile (que l'on désignera souvent par la suite par l'acronyme MS, *Mobile Station*). J-GSM Show est un portage/amélioration en Java du logiciel GSM-Show, à l'origine conçu par Xavier Lagrange et programmé en Visual Basic.

Ce chapitre se structure de la façon suivante :

- Au point 2.1 nous décrirons les fonctions du programme.
- Au point 2.2 nous décrirons l'architecture du programme, où l'on mettra en évidence les améliorations apportées par rapport à GSM-Show.
- Au point 2.3 nous donnerons un exemple d'utilisation de J-GSM Show.

Pour des suppléments d'information sur le programme on se réfèrera aux annexes électroniques, où l'on trouvera non seulement la documentation du programme générée au moyen de l'outil *Javadoc* de Sun mais aussi un petit guide d'installation.

2.1 Fonctions du programme

J-GSM Show est destiné à être utilisé avec un mobile de trace GSM, qui retransmet sur une ligne série connectée à un ordinateur tous les échanges de signalisation qu'il entretient avec le réseau. On dit qu'il trace ses échanges avec le réseau. Ces traces, dont le format dépend du mobile de trace utilisé, sont ensuite décodées par J-GSM-Show et le résultat

de ce décodage est affiché sous des formes nettement plus lisibles pour l'utilisateur qu'un flux d'octets bruts. Par exemple, le programme va reconstituer à partir des traces les messages de niveau 3 et les afficher sous la forme des primitives de services auxquels ils correspondent. Ainsi, il est possible d'observer, quasiment en temps réel les dialogues dans le plan de signalisation entre le réseau et le mobile aux différents niveaux de protocole.

Toutes les fonctionnalités de GSM-Show et de son clone J-GSM Show :

- Visualisation des traces brutes, telles que reçues sur la liaison série en provenance du mobile de trace.
- Visualisation des échanges de primitives dans le plan de signalisation GSM, aussi bien au niveau 2 qu'au niveau 3.
- Visualisation des fréquences, niveaux de réception et codes de couleur d'au plus 6 stations de base situées à proximité du mobile.
- Visualisation de la configuration utilisée sur le moment pour les différents canaux physiques : quels slots de quelle trame sont occupés et par quel canal logique.
- Visualisation de variables d'états de la station de base courante sur laquelle le mobile de trace est à l'écoute. Par exemple : identifiant de cette station dans le réseau, ou niveau maximal d'émission permis à un mobile communiquant avec cette station.
- Visualisation de l'évolution du niveau de réception sur la station de base courante ou sur une station de base adjacente , du niveau d'émission en cas de communication avec le réseau, etc..
- visualisation aussi, dès que possible, des différents identifiants du mobile

Nous n'avons pas encore implémenté les 3 derniers éléments de cette liste dans J-GSM Show.

Le programme peut fonctionner dans trois modes différents :

- Le mode série : dans ce mode, un mobile de trace doit être branché à l'ordinateur via une liaison série. Le programme va alors décoder "à la volée" les messages émis par le mobile sur cette liaison et afficher le résultat de son travail dans les différents écrans que l'utilisateur aura choisi d'afficher. Il est possible de sauvegarder les messages du mobile dans un fichier.
- Le mode temporisé : dans ce mode, on utilise un fichier sauvegardé dans le mode série. Le programme va "rejouer" la trace à raison d'un message toutes les 0,75 secondes. Ce mode permet d'utiliser le programme à des fins d'illustrations à partir de scénarii connus à l'avance et sans disposer d'un mobile de trace.
- Le mode pas-à-pas : très semblable au précédent, sauf qu'ici c'est l'utilisateur qui commande la lecture de chaque message via l'interface, ce qui permet de rejouer un scénario étape par étape, au cours d'un exposé par exemple.

2.2 Architecture

2.2.1 Préambule

Dans la suite de ce chapitre, nous parlerons par abus de langage de "trame série" quand il s'agira d'un message transmis sur la liaison série et contenant une en-tête spécifique au protocole utilisé par le mobile pour cette liaison ; et nous parlerons de trame (tout court) lorsqu'il s'agira d'un message au format utilisé sur la voie radio dans GSM. Et donc, souvent, une trame série contient une trame précédée d'une en-tête spécifique au mobile utilisé.

Mais parfois, une trame série ne contient pas de trame, mais plutôt un rapport qui est généré par le mobile à partir des mesures qu'il a réalisées. Un rapport n'est pas transmis sur le réseau GSM, contrairement à une trame. La figure2.1 illustre ces différents concepts.

2.2.2 Objectifs et solutions

Les objectifs à atteindre lors de la conception de ce programme étaient triples :

- Faciliter autant que possible l'intégration de nouveaux composants permettant l'utilisation de mobiles de trace non prévus à l'origine. Etant donné que plusieurs marques proposent des mobiles de trace aux capacités très semblables, il a semblé judicieux de prévoir dès le début la possibilité de l'utilisation d'autres mobiles que l'Orbitel dont on disposait à l'origine.
- Faciliter autant que possible l'ajout de nouveaux écrans illustrant d'autres parties des protocoles GSM(par exemple GPRS).
- S'assurer de la totale indépendance des différents écrans, de façons à ce qu'on puisse toujours en utiliser un indépendamment d'un autre.

Pour gérer plusieurs mobiles de façon transparente, on a utilisé une classe abstraite : *MobileManager*. Chaque mobile sera représenté par une instance d'une classe héritant de *MobileManager*. Cette structure permet d'abstraire tout ce qui, dans la gestion du dialogue avec le mobile de trame, est commun à tous les mobiles, quelle que soit leur modèle. Cela garantit une relative indépendance du programme vis-à-vis du mobile utilisé (relative parce que par exemple, le contenu des rapports diffère d'un mobile à l'autre).

Pour faciliter l'ajout de nouveaux écrans, on a utilisé un modèle de conception dénommé Observer et tiré du livre Design Patterns[GHJV97]. Ce modèle utilise la communication par événement. Un composant se charge de créer des événements. En toute généralité, on ne sait pas à quel composant sont destinés ces événements. Pour se voir adresser un exemplaire d'un événement, un composant doit s'abonner, c.-à-d. s'enregistrer auprès du créateur d'événements. Ces composant constituent les observateurs qui donnent son nom

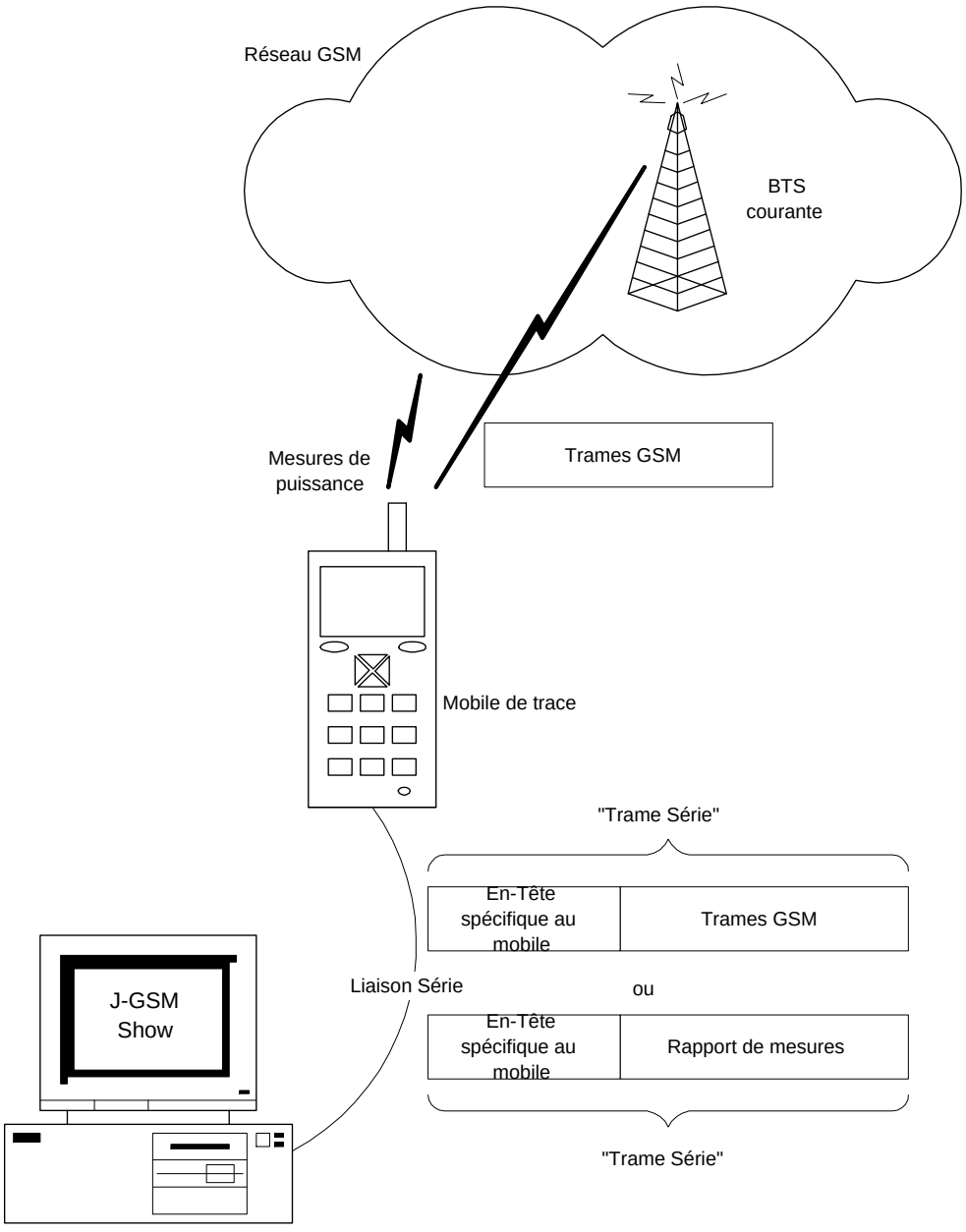


FIG. 2.1 – Trames, trames série et rapports

au modèle. Dans notre cas, un événement sera la mise à disposition d'une nouvelle trame ou d'un nouveau rapport et les différentes fenêtres du programme observeront ces événements pour les traiter et afficher leur résultat. Le distributeur sera une instance de la classe *Dispatcher*, les événements seront des instances des classes *TrameEvent* et *RapportEvent* et les observateurs implémenteront, selon les événements souhaités, les interfaces *TrameListener* ou *RapportListener*. Pour abonner un observateur, on utilisera les méthodes *addRapportListener(RapportListener rl)* et *addTrameListener(TrameListener tl)* du *Dispatcher*.

Le fait d'implémenter les interfaces TrameListner ou *RapportListener* impose aux observateurs d'implémenter des méthodes de traitement de ces évènements, *processTrameEvent(TrameEvent te)* ou *processRapportEvent(RapportEvent re)*.

Le diagramme de classe de la figure 2.2 illustre cette architecture.

Ce modèle présente l'avantage de rendre aussi indépendant que possible la source des événements et leur destination, permettant ainsi d'ajouter très facilement des observateurs ou de dissocier sources et récepteurs sur des machines différentes.

Enfin, pour assurer l'indépendance entre les écrans on n'a partagé aucune action de décodage entre différents écrans. Cela conduit bien sûr parfois à procéder deux fois au même décodage, mais les cas sont suffisamment peu nombreux pour être négligés.

2.2.3 Présentation Générale

Le logiciel s'articule principalement autour d'une classe : *IHM*. C'est cette classe qu'il faut exécuter pour lancer le programme et c'est à partir d'elle que toutes les autres sont utilisées. Elle comprend la définition de tous les éléments interactifs de l'interface et, dans la méthode *ActionPerformed(...)*, le coeur du programme : pour chaque action de l'utilisateur (clic sur un menu en général), c'est là qu'est enclenchée la réaction du programme.

La liaison série est gérée par la classe *PortManager*. Via les méthodes de cette classe, on peut ouvrir le port, fixer ses paramètres (baudrate, bits de parité...) et le fermer. La gestion du port série utilise la package *commapi* que l'on peut trouver sur le site de Sun [SM01].

Une instance de la classe *FlowParser* segmente le flux d'octets qu'on capte ensuite sur le port. Cette instance gère la partie du protocole propre au mobile relative aux limites d'une trame série. La classe *FlowParser* est donc abstraite et peut être instanciée par plusieurs classes différentes dont l'implémentation dépend du mobile utilisé. On a décrit un mécanisme identique, utilisé pour la classe *MobileManager*, au point 2.2.2.

La classe *MobileManager* est abstraite. Les classes concrètes qui l'implémentent doivent permettre de lancer la communication avec le mobile, de l'arrêter et de traiter les trames séries qui lui sont transmises pour décodage. Après traitement, une instance de *MobileManager* provoquera la création d'un évènement par le *Dispatcher* via les méthodes *fireRap-*

portEvent(Rapport r) et *fireTrameEvent(Trame t)* de ce dernier. Les différentes instances possible de *MobileManager* vont toujours remplir la même fonction : décoder l'en-tête propre au mobile des trames série. Pour les trames séries contenant une trame, elles vont principalement pouvoir en déduire le sens de la communication, la fréquence de transmission et le canal logique.

Une trame série peut contenir en plus de l'en-tête une trame, c.-à-d. dans notre terminologie, un champs de données au format GSM. Ce format est bien sûr identique pour tous les mobiles. Il est décrit dans la norme GSM. Ces données sont donc analysées par une classe unique, quel que soit le mobile : *Interpreter*. Cette classe va décoder certains renseignements supplémentaires tels que type de couche de niveau 3 (RR,CM, ou MM) et l'emplacement du début du message de niveau 2 et des données de niveau 3 qu'il peut contenir.

Pour stocker les informations contenues dans un rapport ou dans une trame on a créé deux structure de données spécifiques, les classes *Trame* et *Rapport*. On trouvera dans les tableaux 2.1 et 2.2 la liste des champs de ces classes ainsi que la définition de leur sémantique. Ces classes ne sont pas représentées sur le diagramme de classe de la figure 2.2 pour ne pas surcharger.

Le reste du traitement des trames GSM est effectué pour chacun des observateurs dans les méthodes *processTrame* et *processRapport()*, comme expliqué dans la section "Objectifs et solutions".

La classe *tempFileWriter* constitue un de ces observateurs et implémente donc ces deux interfaces. Elle est destinée à stocker sur le disque les séquences de structures décodées. Le but en stockant le résultat du décodage plutôt que sa matière brute est d'éviter de devoir reprocéder à l'interprétation à chaque fois et d'avoir un format de fichier indépendant du mobile de trace utilisé.

Dans les cas du mode pas-à-pas ou du mode temporisé, ce n'est pas une instance de la classe *MobileManager* qui va utiliser les méthodes *fireRapportEvent* et *fireTrameEvent*. En effet dans ces modes, il n'y a pas de mobile à gérer. A la place, une instance de la classe *TrameFileSender* va lire un fichier qui lui est spécifié par *IHM* et déclencher les événements que le *Dispatcher* va distribuer. Si on est en mode temporisé, cette instance va provoquer un événement toutes les 0,75 secondes. Si on est en mode pas-à-pas, c'est sur appel de sa méthode *readNext()* par *IHM* qu'elle va provoquer un événement.

Pour faciliter la compréhension du code, il n'y a pas d'autres variables globales au programme que les constantes qui sont toutes situées dans la classe *Constantes*. De plus, toutes les interactions entre classes se font par les méthodes disponibles et seuls les paramètres de l'en-tête de chaque méthode sont modifiés par la méthode.

Structure de trame :		
Type	Nom du Champ	Commentaire
String	heure	Temps en ms depuis le début du processus de trace
int	direction	Sens de transmission de la trame : UPLINK ou DOWNLINK
int	frequence	Numéro de fréquence ARFCN sur laquelle le message est reçu(de 0 à 1023)
int	channel	Canal logique
boolean	incomplet	Valeur vraie si le message de niveau 3 n'est pas complet
boolean	repetition	Valeur vraie si la trame est répétée
int	ns	Les QUATRE bits de poids faible du champ *control* (voir norme GSM) qui contiennent le numéro de la trame de niveau 2 (permet d'éviter d'afficher les répétitions)
int	decompteur	Valeur du décompteur uniquement sur SACCH en mode dédié (à 0=> fin de com)
int	couche_L3	Type de couche (RR,CM,MM)
int	transaction_ID	Référence de la transaction(pour couche CC)
int	longueur	Longueur en nombre d'octets du contenu
int	pt_debut_L2	Pointeur indiquant (pour tous canaux logiques) le début de la trame niveau 2
int	pt_debut_L3	Pointeur indiquant (pour tous canaux logiques) le début du message de niveau 3
int	pt_contenu_L3	Pointeur indquant le début du contenu du niveau 3
int[]	contenu	Contenu octet par octet ou champ par champ
String	message	Contenu du message brut
String	nom_L3	Nom du message L3 (cf norme)

TAB. 2.1 – Les champs de la classe Trame

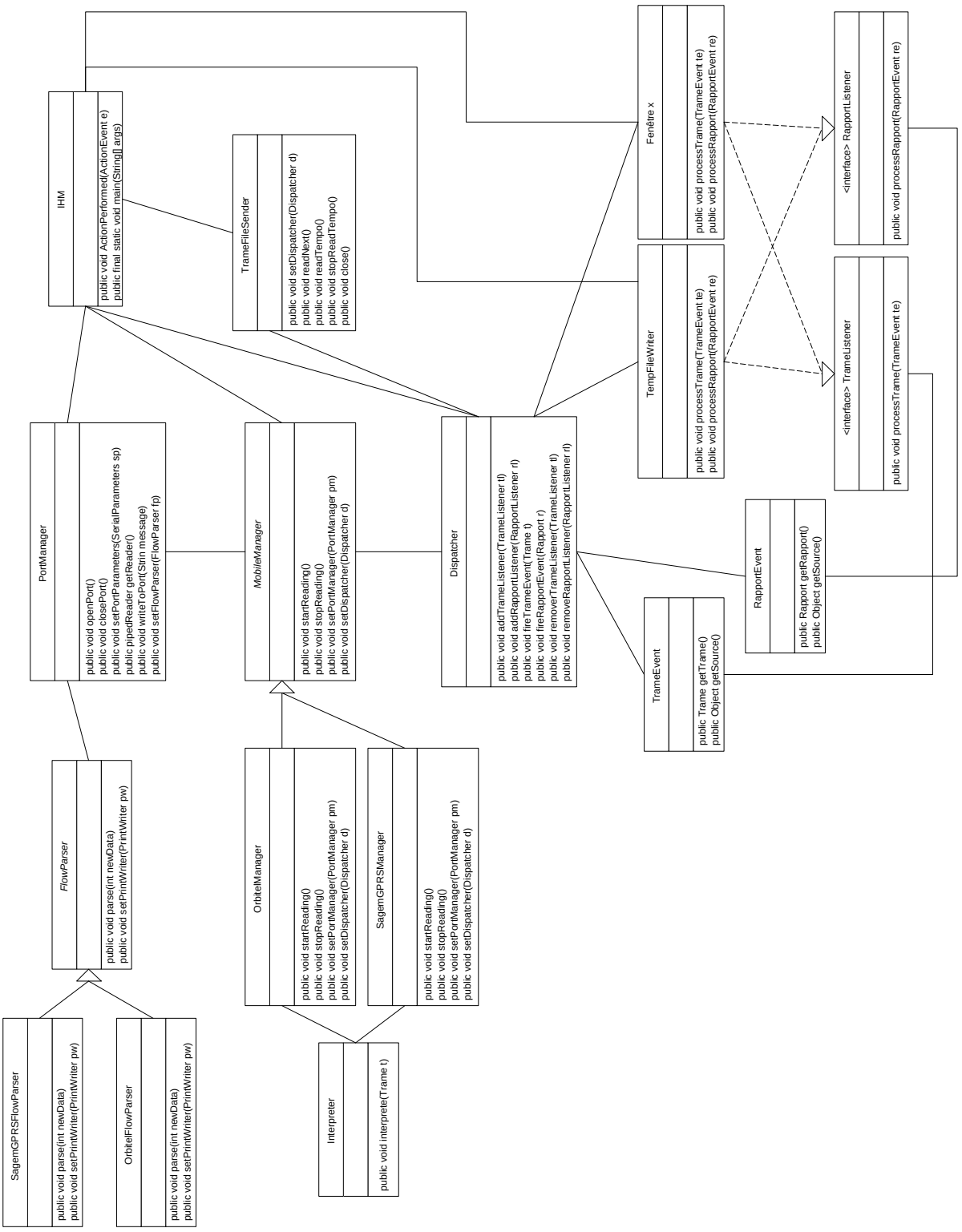


FIG. 2.2 – Diagramme UML des classes du noyau du programme

Structure de Rapport :		
Type	Nom du Champ	Commentaire
int	bande	Bande de fréquence actuellement utilisée
int[]	BSIC	Codes de couleur des BTS les plus proches
int	etat	Mode dedié ou idle
int[]	freq_vois)	Fréquences des BTS les plus proches
int	frequence	Numéro ARFCN de fréquence courante
int	Niv_Puiss	Niveau de puissance codé façon gsm
int[]	RX_Level	Niveau de réception sur les BTS les plus proches
int	RX_Level_cour	Niveau de réception sur la BTS courante
int	RX_Qual_cour	Qualité de la réception sur la BTS courante
int	TA_Reel	Avance en temps

TAB. 2.2 – Les champs de la classe Rapport

2.2.4 Diagramme de classes

Le schéma de la figure 2.2 n'est pas exhaustif, il est simplement destiné à comprendre l'architecture générale du programme[1].

Le rôle de chaque classe :

- **IHM**
  La classe dont la méthode *main()* est appelée au lancement. Cette classe gère toute l'interaction avec l'utilisateur.
- **PortManager**
  Permet de configurer la liaison série et de lancer la réception du flux d'octets sur le port choisi. Ce flux est ensuite segmenté en messages par une instance de la classe abstraite *FlowParser*. Ensuite, ces messages sont mis à disposition dans un tampon que l'on peut obtenir en utilisant la méthode *getReader()*. De plus, on peut émettre des chaînes de caractère sur la liaison série via la méthode *writeToPort()* [2].
- **FlowParser**
  Classe abstraite dont les instances concrètes doivent être capables de segmenter un flux d'octets en trames série. La classe va accumuler les octets dans un tampon jusqu'à obtenir une trame série complète et va alors écrire cette trame dans un tampon où un autre composant pourra la récupérer.
- **SagemGPRSFlowParser**
  Une des implémentations particulières de *FlowParser*. Elle segmente le flux de données transmis par un mobile Sagem de modèle OT96MGPRS. Le protocole utilisé utilise des flags et un champs longueur pour délimiter les trames et permet un contrôle de validité par un CRC sur les trames séries.
- **OrbitelFlowParser**

[1] Pour une description exhaustive, se référer à la documentation html générée avec JavaDoc.

[2] Ces messages sont émis sans délai sur la ligne et servent soit à lancer le processus de trace, soit à l'arrêter, soit à acquitter les trames séries si le protocole utilisé par le mobile comporte un tel mécanisme

Une des implémentations particulières de *FlowParser*. Elle segmente le flux de données transmis par un mobile Orbitel de trace. Le mobile en question sépare les trames séries par des CR/LF's. La segmentation est donc particulièrement simple.

- **MobileManager**
  C'est une implémentation de cette classe qui doit décoder les messages que le *MobileManager* lit dans le tampon que lui fournit un *Portmanager*, garnir une *Trame* ou un *Rapport* et provoquer un événement via la méthode idoine du *Dispatcher*. Cette classe abstraite permet de masquer l'utilisation de mobiles de modèles différents. Elle est prévue pour permettre une extension plus aisée du programme.
- **Interpreter**
  Une classe contenant principalement la méthode *Interprete()* qui va décoder une trame telle que transmise sur la voie radio en s'aidant de renseignements fournis par le protocole spécifique au mobile , tels que le sens de la transmission (sur la voie montante ou sur la voie descendante) et la canal logique utilisé. Cette classe peut-être utilisée par toutes les instances de *MobileManager* vu qu'elle ne dépend pas d'un protocole utilisé par un mobile particulier mais uniquement des spécifications du système GSM.
- **OrbitelManager**
  Une des implémentations particulières de *MobileManager*.
- **SagemGPRSManager**
  Une des implémentations particulières de *MobileManager*.
- **Dispatcher**
  Permet d'ajouter ou d'enlever dynamiquement des *TrameListener* ou des *RapportListener*. Isole la source (*MobileManager* ou *TrameFileSender*) des receveurs. Devrait augmenter la réutilisabilité du code.
- **TrameFileSender**
  En cas d'utilisation des modes pas-à-pas et temporisé, devient la source des instances de la classe *Trame*. Deux modes d'utilisations :
  - une pour le mode temporisé (méthode *StartTempo()*) qui démarre un thread qui toutes les 0,75 s émet une trame en provenance du fichier
  - une pour le mode pas à pas (méthode *readNext()*) où c'est l'IHM qui déclenche l'émission de la trame suivante
- **TrameListener**
  Une interface à implémenter par toutes les classes qui veulent s'abonner à la distribution de *TrameEvent* par un *Dispatcher*.
- **RapportListener**
  Une interface à implémenter par toutes les classes qui veulent s'abonner à la distribution de *RapportEvent* par un *Dispatcher* .
- **TempFileWriter**
  En mode série, on garde une trace de toutes les trames dans un fichier. Le *FileWriter* est abonné au *Dispatcher* et enregistre la trace dans un fichier temporaire qui ne sera sauvegardé que si l'utilisateur le souhaite.
- **Fenêtre X**
  Une des nombreuses fenêtres qui peut traiter les *TrameEvent*.

2.3 Exemple d'utilisation

La figure 2.3 représente une saisie d'écran de JGSM-Show en cours de fonctionnement. L'utilisateur y a activé quatre écrans : les traces brutes dans le quadrant supérieur gauche, les primitives de niveau 2 dans le quadrant supérieur droit, les primitives de niveau 3 dans le quadrant inférieur gauche et les fréquences, niveaux de réception et codes de couleur dans le quadrant inférieur droit.

Dans le quadrant supérieur gauche, l'utilisateur peut visualiser directement les trames séries, en ce compris l'en-tête ajoutée par le mobile. A titre d'exemple, la figure 2.3 représente des traces émises par le mobile Orbitel.

Dans le quadrant supérieur droit, on peut visualiser un échange de primitives de niveau 2 entre le mobile et le réseau. Cette fenêtre permet d'observer en direct le fonctionnement du protocole LAPDm. La représentation d'une primitive s'articule autour d'une flèche indiquant le sens de transmission, qui peut être montant, c.-à-d. de la MS à la BTS ou descendant, de la BTS à la MS. Le nom et les différents paramètres de la primitive sont inscrits au dessus de la primitive. Si la primitive sert à transporter des données, celles-ci s'affichent en dessous de la flèche. Dans la figure 2.3, on peut par exemple observer le mécanisme de fenêtre glissante utilisée par le protocole pour les acquittements multiples. Le numéro N(R) de la première primitive indique quelle trame le mobile attend. Sa valeur est de 3. Ensuite, la BTS transmet une trame portant ce numéro. Et par conséquent, comme cette transmission s'est effectuée sans problème, on peut voir que la primitive suivante transmise par la MS transporte un numéro N(R) de valeur 4, c.-à-d. augmenté d'une unité.

Dans le quadrant inférieur gauche, l'utilisateur peut observer un échange de primitives de niveau 3. Le programme représente ces primitives de la même façon que celles de niveau 2. Dans la figure 2.3 on peut observer le mécanisme de handover. Le réseau ordonne au mobile par la deuxième primitive affichée à l'écran, HANDOVER_COMMAND, de se caler sur une autre fréquence. Les paramètres exacts de cette opération sont transmis dans les données inscrites en dessous de la primitive. Lorsque le mobile a fini, il transmet à la BTS le message HANDOVER_COMPLETE.

Dans le quadrant inférieur droit, on peut observer les fréquences de la BTS courante et de maximum 6 autres BTS, les mieux reçues par le mobile parmi celles appartenant à un réseau GSM qu'il a l'autorisation d'utiliser. Pour chaque BTS on peut observer la qualité de réception correspondante. En général, la meilleure réception correspond à la BTS courante. Dans le cas du HANDOVER évoqué au paragraphe précédent on pourra observer dans cet écran un changement de fréquence pour la BTS courante. A droite de cette liste, le programme affiche quelques paramètres de la communication en cours : puissance de transmission du mobile, avance en temps ...

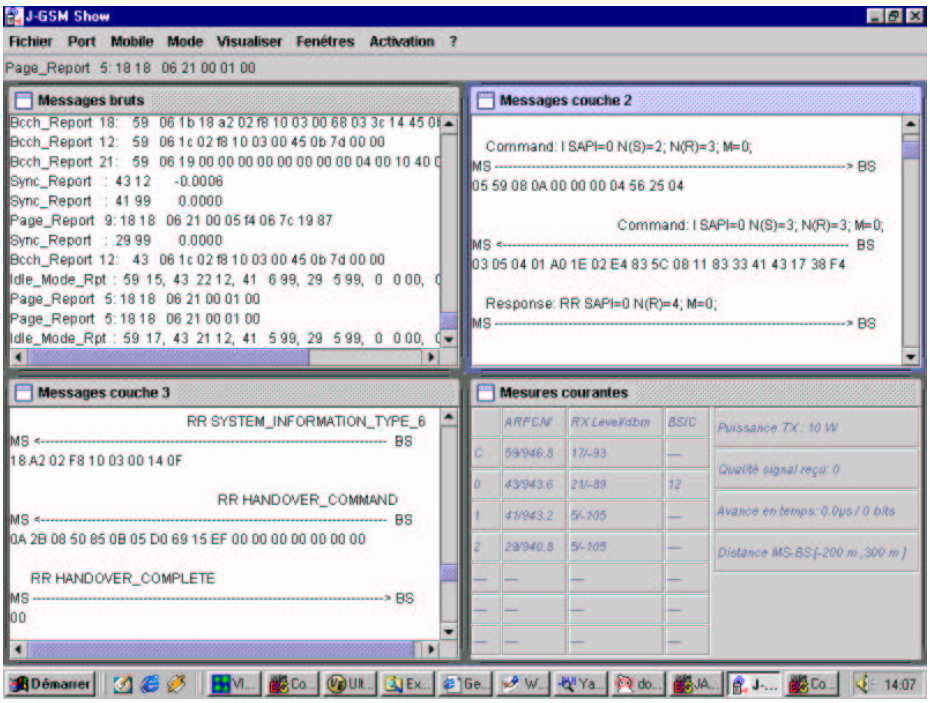


FIG. 2.3 – JGSM-Show en plein fonctionnement

2.4 Conclusion

Il reste encore à réaliser dans J-GSM-Show trois écrans de GSM-Show. Ce travail est en cours. Xavier Lagrange l'a donné comme exercice à des élèves de l'ENST-Paris. De plus, le programme pourrait encore être amélioré :

Premièrement, il devrait informer l'utilisateur en cas d'échec dans le décodage d'une trame envoyée par le mobile de trace.

Deuxièmement, il devrait proposer des raccourcis claviers (mnémoniques).

Troisièmement, il devrait offrir une aide en ligne

Et dernièrement, il devrait rajouter une en-tête spécifique au fichier résultant du décodage. Elle pourrait contenir la date de l'enregistrement et un champs contenant un descriptif du scénario enregistré.

Ce ne sont là que les améliorations auxquelles nous avons pensé. Quoi qu'il en soit, si le programme ne présente pas encore toutes les facilités d'utilisation d'un logiciel professionel, il est parfaitement fonctionnel à l'heure actuelle. De plus, nous avons incorporés très facilement l'utilisation d'un autre mobile que celui utilisé par GSM-Show, ce qui tend à prouver la souplesse de l'architecture choisie. Enfin, la réalisation de fonctionnalités GPRS et leur incorporation au programme existant nous a encore confortés dans notre conviction du bien-fondé de cette architecture.

Chapitre 3

Le compilateur CSN.1

3.1 Introduction

A l'heure actuelle, les nouvelles normes de téléphonie mobile ne cessent de paraître. Et les constructeurs d'équipements cherchent à produire des appareils implémentant ces normes et leurs nouvelles fonctionnalités dans des délais de plus en plus courts. Or la complexité des protocoles gérant ces fonctionnalités est de plus en importante. Pour résoudre ce problème, on a pensé à exprimer le format des différents messages sous une forme directement compréhensible par un ordinateur. Michel Mouly a défini un telle forme avec CSN.1 (*Concrete Syntax Notation number 1*) dans son livre [Mou00].

Si on écrit le format des messages sous cette forme, on peut générer automatiquement un programme de décodage correspondant. Ensuite, il est relativement facile d'implémenter ce programme dans un téléphone mobile ou tout autre appareil.

Ce chapitre est structuré de la façon suivante :

Premièrement, on trouvera au point 3.2 une description de CSN.1.

Deuxièmement, au point 3.3 nous expliquerons comment nous avons réalisé un compilateur pour CSN.1, c-à-d un programme qui recevant en entrée une description de format en CSN.1 fournit en sortie une structure de données équivalente et facilement utilisable par un autre programme. Nous avons créé un compilateur pour CSN.1 afin de générer un décodeur pour les messages du protocole au coeur de la voie radio dans GPRS : RLC/MAC.

Troisièmement, au point 3.4 nous décrirons la construction du programme capable de décoder un message au moyen de la structure de données générée par le compilateur.

Et enfin, en 3.5, nous donnerons un exemple de fonctionnement du compilateur et du décodeur.

3.2 CSN.1

3.2.1 Définition

CSN.1 est une notation destinée à décrire l'encodage binaire d'un protocole. Un description écrite en CSN.1 définit un ensemble de séquences de bits considérées comme correctes. L'ensemble de séquences incorrectes en constitue donc le complémentaire. Ces séquences sont finies et ordonnées.

Concrètement une description CSN.1 est un texte utilisant l'alphabet de base IA5.

En absence de mention explicite, les espaces, tabulation et retours à la ligne n'y sont pas significatifs.

Pour indiquer un bit, on utilise les caractères '0' et '1'. Formellement, les notations '0' et '1' dénotent chacune un ensemble composé d'une seul séquence d'un seul bit. De plus le mot 'bit' dénote l'ensemble de deux séquences longues de 1 bit, '0' et '1'.

La succession de deux descriptions décrit la concaténation de ces descriptions. Formellement, elle décrit l'ensemble de séquences de bits obtenues en concaténant une séquence décrite par la première avec une séquence décrite par la deuxième .

Le caractère '|' dénote l'alternative. Formellement, il représente l'union de deux ensembles. Par exemple la description

```
011|000
```

représente l'ensemble constitué de deux séquences de bits distinctes : '011' et '001'.

Les caractère '{' et '}' délimitent une description.

Du point de vue priorité des opérateurs, la concaténation a priorité sur l'alternative. Par exemple la description :

```
00|11
```

dénote le même ensemble que :

```
{ 00 } | { 11 }
```

mais pas le même que :

```
0 {0|1} 1
```

Les caractères '<' et '>' délimitent un identifiant utilisé comme une référence à une description apparaissant ailleurs. Ce mécanisme permet la modularité. Tous les caractères compris entre ces deux caractères sont significatifs et constituent un label. Un label ne peut contenir les caractères ' :', '=', '(', ')', '<', '>', ';' ou '.'. Tous les autres caractères

sont permis, en ce compris les espaces tabulations et retour à la ligne.

Une référence suivie de la séquence de caractères ' : :=' suivi d'une description, le tout suivi d'un point-virgule dénote l'association de la référence avec la description. Par exemple,

```
<bit>::=1|0;
```

associe la référence <bit> à la description 1|0.

Un caractère '<' suivi d'un label, d'un caractère ' :' d'une description et d'un caractère '>' dénote l'association de la référence à la description. Ce mécanisme équivaut au précédent. Ce mécanisme permet de nommer une partie de message sans devoir créer une définition spécifique. Exemple :

```
<bit pair: { 1|0 } { 1|0 }>
```

La séquence de caractères 'null' dénote un ensemble vide.

Une chaîne de caractère placé entre parenthèses, '(' et ')' représente un exposant. CSN.1 permet de placer en exposant une variable dont la valeur dépendra d'une propriété d'une séquence particulière de l'ensemble décrit. Par exemple :

```
<champ>::=<longueur : bit(8)>
          <données : bit(val(longueur)>;
```

De cette façon, des données du message peuvent être utilisées pour définir la structure de ce même message. Par exemple, dans la description ci-dessus, la longueur de la chaîne de bits *champ* dépend de la valeur associée à une de ses sous-séquences : *longueur*. La valeur prise en compte est celle de la dernière instance du label avant son usage comme une variable. Le terme "avant" doit se comprendre comme "selon l'ordre des bits dans la séquence". Ce mécanisme requiert la définition de fonctions qui transforment la séquence extraite en un résultat quelconque. Deux fonctions prédéfinies sont données dans la définition de CSN.1 :

val() La fonction val() interprète une séquence de bits comme un entier représenté en binaire avec le bit de poids le plus fort précédant tous les autres dans la séquence. Exemples :

$$val(1000) = 8$$
$$val(010) = 2$$

len() La fonction len() prend en entrée une séquence de bits et en renvoie le nombre de bits. Exemples :

$$len(1000) = 4$$
$$len(010) = 3$$

CSN1. prévoit aussi la possibilité de pratiquer les opérations arithmétiques courantes +, -, *, div et mod sur les valeur de ces fonctions. Exemple :

```
<champ>::=<longueur : bit(8)>
          <données : bit(val(longueur)+1)>;
```

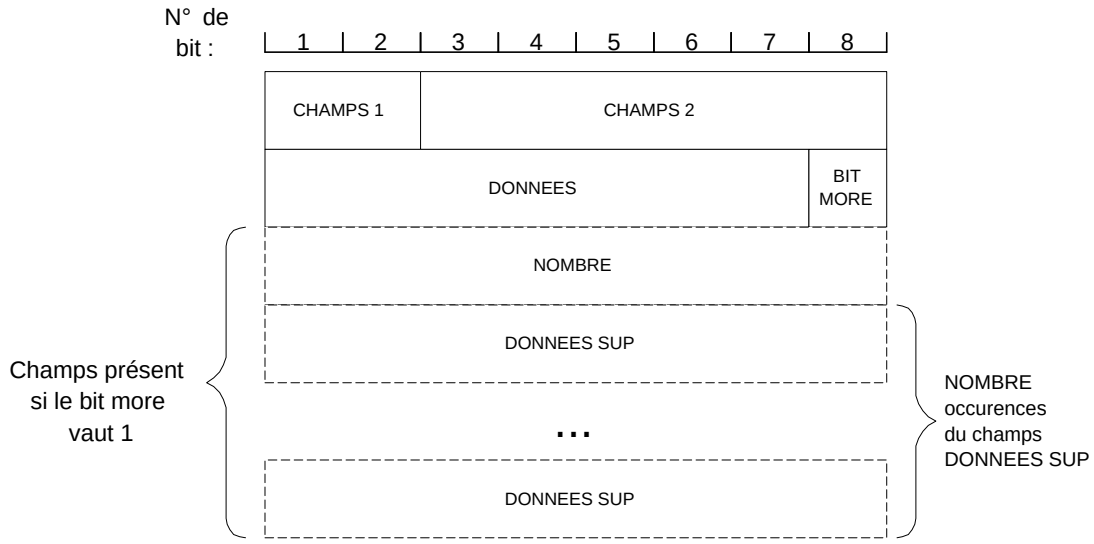


FIG. 3.1 – Une représentation classique d'un format de message binaire.

La séquence de caractères '−−' indique le début d'un commentaire. Il se termine avec le premier retour à la ligne.

Finalement le caractère '!' peut remplacer le caractère '|' quand les choix à droite du '!' représentent des erreurs possibles dans le message. Ces deux symboles s'équivalent presque. Cette construction sert principalement à permettre l'analyse d'un message partiellement reçu.

Pour finir, voici un exemple plus complexe de description en CSN.1 :

```
< exemple de description > ::=
< CHAMPS 1 : bit(2)>
< CHAMPS 2 :bit(6)>
< DONNEES  : bit(7)>
< BIT MORE : bit>
{
    {
        <NOMBRE : bit(8)>     -- commentaire
        <DONNEES SUP : octet(val(longueur))>
    }
    * (val(BIT MORE))
 };
```

Le format de message binaire défini à la figure 3.1 est exactement équivalent à cette description CSN.1.

3.2.2 Avantages et inconvénients

CSN.1 présente plusieurs avantages sur la traditionnelle représentation des messages sous forme de grilles avec une case par bit.

Tout d'abord, CSN.1 est compilable, c.-à-d. qu'il est possible de créer un programme qui à partir d'une syntaxe rédigée en CSN.1, va générer un décodeur. Par décodeur, on entend un autre programme qui pourra déterminer si une séquence de bits qu'on lui soumet appartient à l'ensemble de séquences de bits définie par la syntaxe. Le cas échéant, il devrait aussi pouvoir isoler chacune des sous-séquences nommées dans cette syntaxe.

Ensuite, CSN.1 permet de minimiser facilement le nombre de bits utilisés pour transmettre un message. Définir des champs facultatifs s'avère, en effet, d'une grande facilité. Voici la construction généralement employée :

```
0|1 <champs facultatif>
```

Elle reste très lisible contrairement aux représentations classiques qui demandaient de rajouter des commentaires en annexe aux schémas.

De plus,CSN.1 permet la description de protocoles très complexes tout en restant relativement lisible. La description d'un protocole aussi complexe que RLC/MAC n'aurait pas été possible sans une telle notation[1].

Enfin, CSN.1 permet de donner des noms à des parties de séquences de bits correctement encodées. Ceci permet de définir plus facilement la sémantique de la séquence, c.-à-d. l'interprétation du message.

Mais CSN.1 présente aussi certains inconvénients.

Tout d'abord, CSN.1 induit généralement un décodage séquentiel des messages. En effet, un programme de décodage ne peut souvent déterminer l'emplacement d'un champs qu'après avoir parcouru toute la séquence qui le précède.

Ensuite, CSN.1 permet d'écrire des descriptions difficiles à compiler. Par exemple, des descriptions récursives à gauches :

```
<liste de bits>::=<liste de bits> bit | bit;
```

[1] On peut d'ailleurs se demander si l'absence de cette notation n'aurait pas induit une protocole plus simple .

3.3 Réalisation du compilateur

3.3.1 Raisons de ce travail

Pour arriver à décoder les paquets RLC/MAC tracés par le mobile de trace, rédiger un programme "à la main" apparaissait impensable, étant donné le nombre prohibitif de cas.

Tout d'abord, nous avons pensé à utiliser un compilateur préexistant, YCSN de la société Arguin Communications. Ce programme, rédigé en C et générant du C satisfait à priori à tous les besoins de GPRS-SHOW, si ce n'est la portabilité. Mais au final, les négociations en vue d'obtenir une licence gratuite à des fins académiques en sont restées au point mort. De plus l'incorporation de code propriétaire pourrait nuire à d'éventuelles diffusions du programme.

En sus, un développement par nos soins du compilateur présentaient quelques avantages. Tout d'abord, il permettait une réalisation en Java de manière à préserver la portabilité du programme. Ensuite, il facilitait l'adaptation du décodeur aux besoins exacts de GPRS-Show et nous offrait une meilleure maîtrise de son fonctionnement.

Toutes ces raisons ont concouru à nous décider malgré la relative ampleur du projet.

3.3.2 Un peu de théorie des compilateurs

Un compilateur est un programme qui prend en entrée un programme écrit dans un langage de programmation (le langage source) et produit comme résultat un programme dans un autre langage. Fondamentalement, il s'agit d'un traducteur automatique.

Le fonctionnement d'un compilateur se découpe généralement en plusieurs phases. Une phase est une opération cohérente qui prend en entrée une représentation du programme source et produit comme résultat une autre représentation.

Nous ne nous intéresserons qu'aux deux premières phases classiques : l'analyse lexicale et l'analyse syntaxique.

L'analyse lexicale découpe la séquence de caractères qui constitue le programme source en une séquence d'unités atomiques appelées tokens. Chaque token représente une séquence de caractères qui peut être traitée comme une seule entité logique, par exemple un identifiant ou un opérateur.

L'analyse syntaxique a deux fonctions : premièrement elle vérifie que les tokens apparaissant en entrée, qui résultent de l'analyse lexicale, se présentent dans des configurations permises par la spécification du langage source. Secondement, elle impose aux tokens une structure en arbre, habituellement utilisée par les phases suivantes du compilateur. (Voir exemple à la figure 3.2)

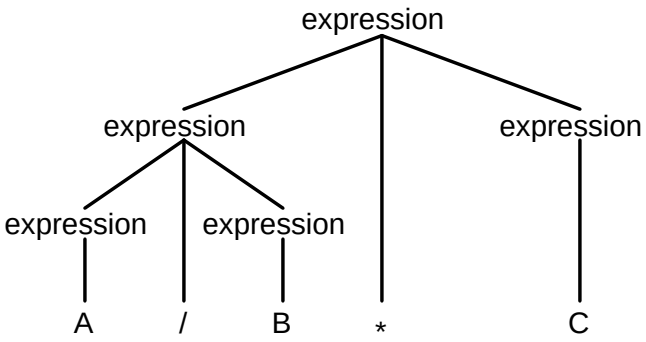


FIG. 3.2 – Exemple d'arbre syntaxique

Généralement après ces deux phases, le compilateur procède à la génération du code et à son optimisation. Nous ne parlerons pas de ces deux phases parce que le compilateur qui nous intéresse ici ne les effectue pas.

La fréquence des problèmes de compilation a incité à la création d'outils d'aide à la construction de compilateurs.

On parlera tout d'abord des générateurs d'analyseurs lexicaux, tels que LEX. A partir de la spécification de tokens sous la forme d'une liste d'expressions régulières, de tels outils génèrent l'analyseur lexical correspondant, aussi appelé "lexeur". La notation appelée "expressions régulières" permet de définir des ensembles de chaînes de caractères. Pour la théorie de cette notation, nous conseillons au lecteur de s'en référer à [AU79]. Nous pensons cette notation assez intuitive pour ne pas demander de plus longue explication. Voici un exemple d'expression régulière :

```
identifiant = lettre(lettre|chiffre)*
```

Cette expression régulière définit un ensemble de chaînes de caractères, dénommé identifiant, constitué des chaînes commençant par une lettre et suivie d'un nombre indéterminé de lettres ou chiffres.

Avec un générateur d'analyseurs lexicaux, on associe à une expression régulière un identifiant de token et une fonction qui permet d'extraire la valeur du token de la chaîne de caractères à laquelle il correspond. Exemple :

```
identifiant = lettre(lettre|chiffre)
 {return symbol(sym.IDENTIFIER,yytext());}
```

où yytext() est la méthode qui envoie une instance de la classe String représentant la chaîne de caractères correspondant à l'expression régulière.

Ensuite, les générateurs d'analyseurs syntaxiques tels que YACC (*Yet Another Compiler of Compiler*). A partir d'une spécification du langage à compiler sous la forme dune

grammaire non-contextuelle, de tels outils génèrent l'analyseur syntaxique correspondant, appelé "parseur".

La notation appelée "grammaire non-contextuelle" ou également BNF (*Backus-Naur-Form*) permet de définir des ensembles de chaînes de tokens considérées comme valides. Ici aussi, pour la théorie, nous renvoyons à [AU79]. Voici un exemple de grammaire non-contextuelle :

```
expression ::= expression PLUS terme
             | expression MOINS terme
             | terme;
terme      ::= terme FOIS ENTIER
             | terme DIV ENTIER
             | ENTIER;
```

Cette grammaire décrit un ensemble d'expression arithméthique ne comprenant que des entiers et les opérateurs d'addition, de soustraction, de multiplication, de division.

On trouve dans une grammaire deux types d'identifiants : les terminaux qui correspondent à un token renvoyé par un analyseur lexical et les non-terminaux qui représentent des constructions de plus haut niveau du langage considéré. Dans notre exemple, les terminaux s'écrivent en majuscules et les non-terminaux en minuscules. Chacun de ces identifiants se voit associé une valeur : un terminal a comme valeur celle du token correspondant. un non-terminal verra sa valeur calculée en cours d'analyse syntaxique.

Normalement, une définition doit apparaître pour chaque non-terminal. Une définition commence par le nom du non-terminal suivi de la séquence de caractères ' : :='. Elle comprend ensuite en général plusieurs productions séparées par le caractère '|'; elle se termine par le caractère ';'. Une production définit un ensemble de chaîne de tokens et la définition définit l'union de ces ensembles.

Un générateur de parseurs permet de faire correspondre à chacune de ces productions une action qui sera exécutée chaque fois que la chaîne de tokens correspondants est identifiée dans le flux en entrée. Une action peut utiliser les valeurs des différents terminaux et non-terminaux. Et elle détermine la valeur du non-terminal qui vient d'être identifié. Voici un exemple extrait de la documentation de l'outil CUP (*Creator of Useful Parser*)[HFAA99] :

```
expr      ::= expr:e1 PLUS expr:e2
          {: RESULT = new Integer(e1.intValue() + e2.intValue()); :}
          |
              expr:e1 MINUS expr:e2
          {: RESULT = new Integer(e1.intValue() - e2.intValue()); :}
          |
              expr:e1 TIMES expr:e2
          {: RESULT = new Integer(e1.intValue() * e2.intValue()); :}
          |
```

```
        expr:e1 DIVIDE expr:e2
    {: RESULT = new Integer(e1.intValue() / e2.intValue()); :}
    |
        expr:e1 MOD expr:e2
    {: RESULT = new Integer(e1.intValue() % e2.intValue()); :}
    |
        NUMBER:n
    {: RESULT = n; :}
    |
        MINUS expr:e
    {: RESULT = new Integer(0 - e.intValue()); :}
    |
        LPAREN expr:e RPAREN
    {: RESULT = e; :}
    ;
```

Cette grammaire va générer un parseur qui prend en entrée une expression arithmétique et rendre en sortie la valeur numérique de cette expression.

On appelle racine le non-terminal de la grammaire qui désigne l'ensemble auquel le parseur vérifie l'appartenance. Généralement, cette racine se trouve au début de la grammaire. Dans l'exemple qui précède, il s'agit du non-terminal expr.

Pour finir, la plupart des générateurs de parseurs permettent de définir des priorités entre terminaux. Par exemple : si FOIS et prioritaire sur PLUS pour la grammaire suivante

```
expression ::= expression PLUS expression
             | expression MOINS expression
             | expression FOIS expression
             | expression DIV expression
             | ENTIER;
```

le parseur généré choisira d'utiliser plutôt la production comprenant le FOIS en premier en analysant la chaîne de token suivante :

```
ENTIER PLUS ENTIER FOIS ENTIER
```

et la réduira plutôt en

```
ENTIER PLUS expression
```

qu'en

```
expression FOIS ENTIER
```

De plus, il est souvent possible de fixer l'associativité, gauche ou droite, des terminaux. Par exemple, si l'associativité est à gauche pour +, la chaîne suivant

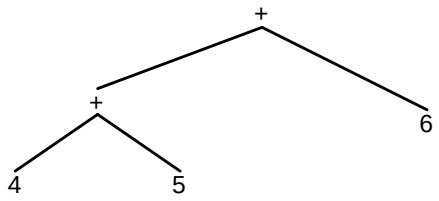


FIG. 3.3 – Premier arbre syntaxique possible

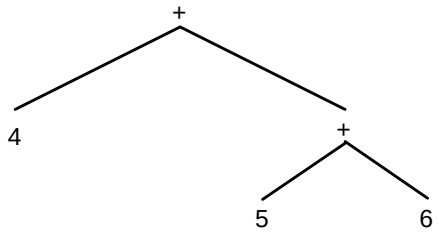


FIG. 3.4 – Second arbre syntaxique possible

```
4 + 5 + 6
```

génèrera l'arbre syntaxique de la figure 3.3 plutôt que celui de la figure 3.4.

3.3.3 Développement du compilateur

Le compilateur doit prendre en entrée une syntaxe rédigée en CSN.1 et rendre en sortie un programme capable d'identifier les séquences de bits correspondant à la syntaxe. Il doit aussi extirper toutes les sous-séquences explicitement labellisées.

Dans notre cas, au lieu de générer un programme de décodage, le compilateur va générer un arbre syntaxique qui sera ensuite passé en paramètre à un décodeur standard.

Pour créer ce compilateur, nous avons d'abord écrit la liste d'expressions régulières définissant les tokens de CSN.1. Nous ne la retranscrivons pas ici vu sa grande trivialité : elle ne se compose que de trois mots-clefs("null", "val" et "len"), d'identifiants, d'entiers et d'opérateurs d'un à trois caractères.[2]

Ensuite, nous avons écrit la grammaire non contextuelle définissant un ensemble de chaînes de tokens correctes pour CSN.1. Cet ensemble ne contient que les chaînes définissables au moyen des constructions vues au point 3.2. Il suffit pour créer un parseur des définitions des messages du protocole RLC/MAC. On trouvera cette grammaire déduite du livre de Michel Mouly [Mou00] à la figure 3.5.

Ensuite, nous avons choisi un générateur de parseurs, CUP [HFAA99], et un générateur de

[2]Pour la lire, ouvrir le fichier csn1.flex dans le sous-répertoire csn1/compiler

```
syntax      ::=   definition
                | definition syntax
                ;
definition  ::=   PP IDENTIFIER PG ALLOC description POINTVIRG
description ::=   NULL
                | INTEGER
                | IDENTIFIER
                | PP IDENTIFIER PG
                | description INFINITY
                | description PAROUV expr PARFERM
                | description MULT expr
                | PP IDENTIFIER DEUXPOINTS description PG
                | description description
                | description OF description
                | description EXCLA description
                ;
expr        ::=   expr PLUS expr
                | expr MOINS expr
                | expr MULT expr
                | expr DIV expr
                | expr MOD expr
                | MOINS
                | PAROUV expr PARFERM
                | INTEGER
                | VAL PAROUV IDENTIFIER PARFERM
                | LEN PAROUV IDENTIFIER PARFERM
                ;
```

FIG. 3.5 – BNF d'un sous-langage de CSN1

lexeurs, JFLEX (*Java Fast LEXical analysor generator*)[Kle01]. Ces deux outils présentent les avantages d'avoir été conçus pour être interfacés et de générer du code Java ce qui garantit la portabilité du parseur généré.

L'arbre syntaxique généré par CUP se constitue d'instances de la classe csn1AbstractNode. Un ensemble de classes concrètes héritent de cette classe abstraite. La classe exacte qui est instanciée pour constituer un noeud dépend de la production à laquelle il correspond. On trouvera à la figure 3.6 un représentation UML de cet ensemble de classes. Ces classes sont :

csn1Leaf Cette classe est instanciée pour représenter une chaîne de bits qui dans la syntaxe d'origine se constitue d'une séquence de '1' et de '0'.

csn1ReferenceNode Cette classe est instanciée pour représenter une référence. Elle possède un champs label.

csn1ConcatenationNode Cette classe est instanciée pour représenter une concatenation. Elle possède deux champs de type csn1AbstractNode qui représentent les deux descriptions concaténées.

csn1AlternativeNode Cette classe est instanciée pour représenter une alternative. Elle possède deux champs de type csn1AbstractNode qui représentent les deux descriptions constituant l'alternative.

csn1ExponentNode Cette classe est instanciée pour représenter une exponentiation. Elle possède un champs de type csn1ExprNode qui est la racine d'un arbre représentant une expression arithmétique.

Pour plus de détails sur l'implémentation, on se reportera aux documents générés au moyen de l'outil javadoc et placés en annexe électronique.

Les exposants sont représentés dans une autre structure d'arbre dont les noeuds instancient tous la classe abstraite csn1ExprNode. L'impossibilité de calculer les valeurs des exposant au moment de la compilation à cause des fonctions impose le stockage des expressions complètes dans la structure de données qu'utilisera le décodeur. Pour rappel, les fonctions utilisées dans les exposants en CSN.1 permettent que des données du message soient utilisées pour définir la structure de ce même message.

Pour les exposants aussi, la classe précise qu'on instancie dépend de la production à laquelle correspond le noeud. On trouvera à la figure 3.7 une représentation UML de ces classes. Nous n'insisterons pas sur cette structure, vu sa relative trivialité.

A la fin du parcours de la séquence de caractères en entrée, le compilateur a généré un nombre d'arbres égal au nombre de définitions de la syntaxe. Cette liste d'arbres est stockée dans une instance de la classe ListeDeNoeudsCSN1. Le premier arbre de cette liste est la racine. Il reste encore alors à associer les références présentes dans tous ces arbres aux définitions correspondantes. La méthode link() de la classe ListeDeNoeudsCSN1 assure cette tâche : à partir de tous les arbres générés, elle va générer un unique arbre dont la

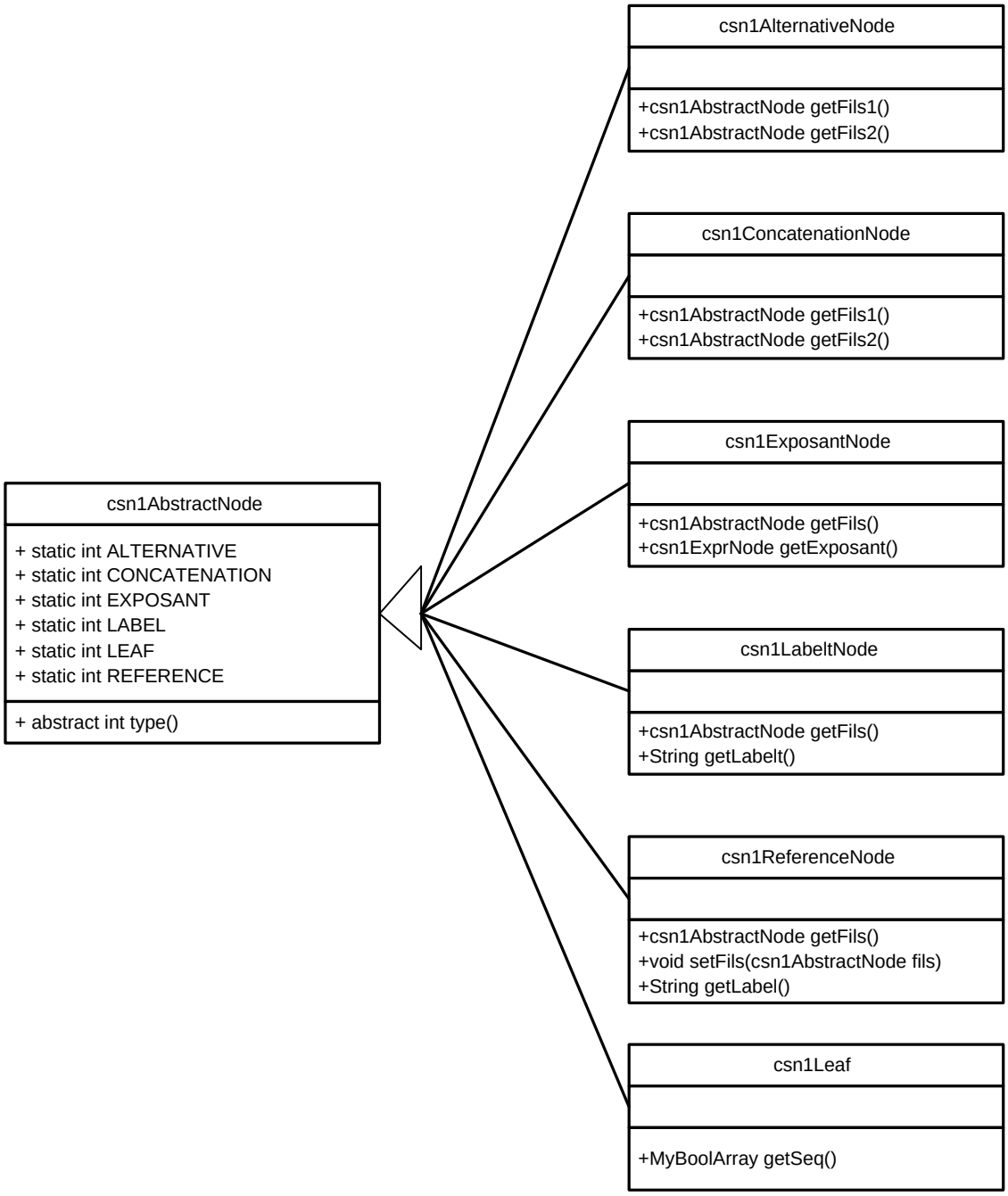


FIG. 3.6 – csn1AbstractNode et les classes concrètes qui l'implémentent

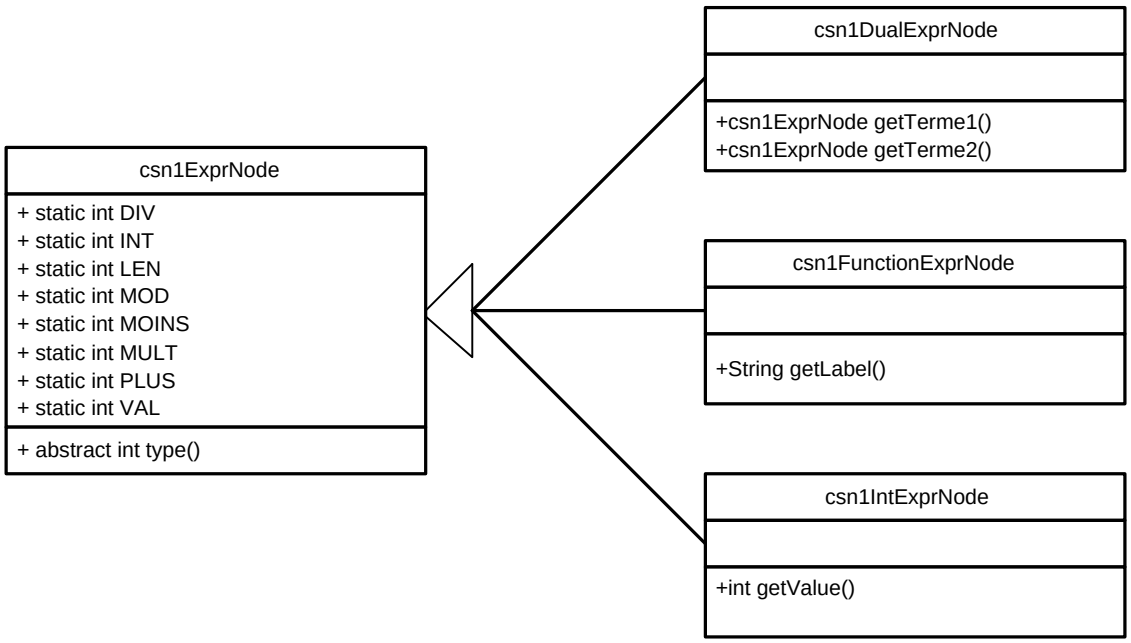


FIG. 3.7 – csn1ExprNode et les classes concrètes qui l'implémentent

racine correspondra à la première définition de la syntaxe. C'est ici que le compilateur vérifie que toutes les références ont bien été définies quelque part dans la syntaxe.

Pour finir, le compilateur va renvoyer comme résultat un pointeur vers la racine de l'arbre.

La classe jgsm.csn1.compiler.Compiler implémente le compilateur décrit ici.

3.4 Le décodeur

Le décodeur doit confronter l'arbre syntaxique d'une syntaxe CSN.1 avec un message supposé correspondre à cette syntaxe. Le cas échéant, il doit générer une structure de données représentant le décodage du message.

On va tout d'abord au point 3.4.1 décrire la structure de données utilisée pour stocker le résultat du décodage.

Ensuite, au point 3.4.2 nous décrirons les grandes lignes de la conception du décodeur, notamment pour la gestion des exposants.

3.4.1 Structure de données

CSN.1 permet de labelliser des sous-séquences d'un message binaire. Le décalage par rapport au début du message (l'offset), la longueur, le contenu et le label caractérisent ces sous-séquences. Deux relations seulement peuvent unir ces sous-séquences, la précédence stricte ou l'inclusion stricte.

Pour deux sous-séquences d'un même message, soit tous les bits de l'une précèdent tous ceux de l'autre, soit tous les bits de l'une sont inclus dans ceux de l'autres. Ces deux relations sont transitives.

La structure de données utilisées pour stocker le résultat du décodage d'un message découle de cette constatation. Elle va se composer d'instances de la classe csn1ConcreteNode qui comporte 6 champs :

offset : Décalage par rapport au début du message

data : La sous-séquence de bits proprement dite

longueur : La longueur de cette séquence

label : Le label de cette sous-séquence

fils : Un pointeur vers une structure de données représentant l'ensemble des sous-séquences labellisées de celle représentée par ce noeud-ci.

suiv : Un pointeur vers une structure de données représentant l'ensemble des sous-séquences labellisées qui suivent dans le message celle représentée par ce noeud-ci.

En fait, suiv et fils pointent vers d'autres instances de la classe csn1ConcreteNode qui représentent la racine d'un arbre de sous-séquences du message.

3.4.2 Conception

Le décodeur fonctionne récursivement sur la structure d'arbre syntaxique générée par le compilateur : il parcourt récursivement l'arbre et le confronte à un message. Si ce message appartient à l'ensemble décrit par l'arbre syntaxique, le décodeur génère une structure de données telle que vue au point 3.4.1.

L'appel récursif ne prend que deux paramètres, l'offset du prochain bit à analyser et une instance de csn1AbstractNode qui représente une partie de la syntaxe. On va essayer de vérifier l'appartenance d'une séquence de bits commençant à l'offset à l'ensemble de séquences de bits décrites par cette partie de la syntaxe.

Pour une bonne compréhension, deux cas particuliers sont à envisager :

Le noeud courant instancie la classe csn1Leaf. Le compilateur va alors vérifier s'il est possible de faire coincider la séquence de bits contenue dans ce noeud avec une séquence de bits de même longueur commençant à l'offset courant. Si c'est le cas, le décodeur génère une instance de la classe csn1ConcreteNode sans ni valeur de label, ni fils, ni suivant, mais avec

un champs data correspondant à la séquence de bits et le renvoie comme résulat. Si ce n'est pas le cas, le décodeur génère une exception InvalidMatchingException. Cette exception va remonter la pile des appels récursifs jusqu'à un appel fait sur un noeud instanciant la classe csn1AlternativeNode.

Le noeud courant instancie la classe csn1AlternativeNode. Tout n'abord, le décodeur va vérifier si la première partie de l'alternative ne correspond pas à une séquence de bits commençant à l'offset courant. Pour ce faire, il va s'appeler récursivement sur le noeud csn1AbstractNode constituant la première partie de l'alternative sans modifier la valeur d'offset. S'il y a une correspondance possible, le eécodeur va renvoyer le résultat de cet appel comme résultat de l'appel sur le noeud csn1AlternativeNode courant. S'il n'y a pas correspondance possible, cet appel récursif va provoquer la génération d'une exception InvalidMatchingException. Dans ce cas, le décodeur va essayer un appel récursif sur la deuxième partie de l'alternative. S'il fonctionnne, le décodeur va renvoyer le résultat de cet appel comme résulat de l'appel sur le noeud courant. S'il n'y a toujours pas correspondance possible, une nouvelle exception est générée et cette fois ci, le décodeur va la laisser remonter plus haut dans la pile d'appels récursifs.

Ce mécanisme d'exception permet d'essayer toutes les alternatives. Si aucune ne peut correspondre, l'exception va remonter la pile jusqu'au sommet et le décodeur va abandonner sa tentative.

Lorsque le noeud courant instancie d'autres classes, comme csn1ConcatenationNode ou csn1LabelNode, le comportement du décodeur se déduit assez facilement, nous ne l'expliquerons donc pas.

Dernier cas difficile : lorsque le noeud courant instancie la classe csn1ExposantNode. Nous avons deux problèmes :

- Tout d'abord, effectuer le calcul de la valeur entière de l'exposant représenté par l'arbre syntaxique. En effet, l'exposant peut comporter une référence vers un champ décodé précédemment. Or, comme on peut le voir à la figure 3.8 le même label peut désigner plusieurs sous-séquences différentes(en l'occurrence, le label LSA_ID).
  La règle énoncée dans la spécification de CSN.1 précise que dans un tel cas, la référence a pour objet la sous-séquence déjà identifiée à l'offset le plus élevé. Pour résoudre la valeur associée à la référence, nous créons, au fur et à mesure que les sous-séquences labellisées sont identifiées, un index de pointeurs vers chacune de ces sous-séquences. Cet index sera utilisé ensuite pour résoudre la valeur associée à la référence.
- Ensuite, comprendre la signification réelle d'un exposant. Les exposants sont des facilités syntaxiques offertes aux utilisateurs de CSN.1 pour alléger la représentation de leurs syntaxes. Mais ils recouvrent deux réalités différentes : lorsque leur valeur est entière et fixe, ils impliquent une concaténation de séquences. Lorsque leur valeur est variable (contient le caractère *), ils représentent une concaténation d'alternatives où une des deux parties de l'alternative est toujours la séquence vide. On peut trouver à la figure 3.8 une représentation de ces deux réalités.

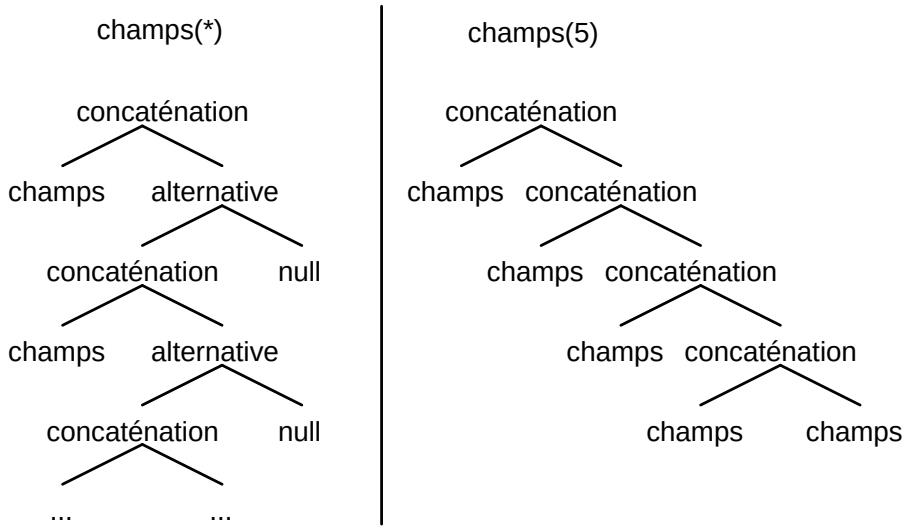


FIG. 3.8 – Deux significations différentes pour l'exposant en CSN.1

Le problème une fois énoncé se résout simplement et nous n'entrerons pas dans le détail.

La classe jgsm.csn1.decoder.Decoder implémente un décodeur résolvant ces problèmes.

3.5 Exemple de fonctionnement

Voici un exemple de fonctionnement du compilateur CSN.1. Tout d'abord, le compilateur a généré un décodeur à partir de la syntaxe CSN.1 de la figure 3.9. Ensuite, si l'on soumet au décodeur en question une des séquences de bits décrites à la figure 3.10, il va les identifier comme appartenant à l'ensemble de séquences de bits décrites par la spécification CSN.1 de la figure 3.9. Et pour par exemple la deuxième de ces séquences, il va générer la structure de données illustrée à la figure 3.11 où chacun des rectangles représente une instance de la classe *csn1ConcreteNode*.

```
< exemple de description 2> ::=      -- début de la première description
< CHAMPS 1 : bit(2)>                 -- commentaire sur la signification du champs 1
< DONNEES  : bit(7)>
{
    1 <Champs Facultatif : bit(3)>   -- Si le 10ème bit vaut 1,
                                     -- alors il est suivi de <Champs Facultatif>,
    |0                               -- sinon, on passe tout de suite à la suite.
}

< description 3>                     -- référence à une description faite ailleurs
<NOMBRE : bit(8)>
<DONNEES SUP: octet(val(longueur))>  -- le champs DONNES SUP
                                     -- sera long de val(NOMBRE) octets
;                                    -- fin de la première description
< description 3> ::=
< bloc fixe : 11>                    -- Ici,on indique directement
                                     -- la valeur des bits qu'on espère trouver
<CHAMPS 2: bit(6)>;
```

FIG. 3.9 – Exemple de spécification CSN.1

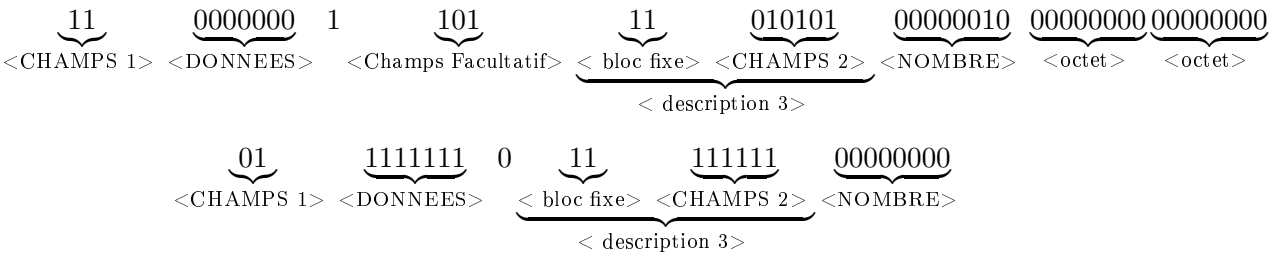


FIG. 3.10 – Deux séquences de bits appartenant à l'ensemble de séquences défini par la description de la figure 3.9

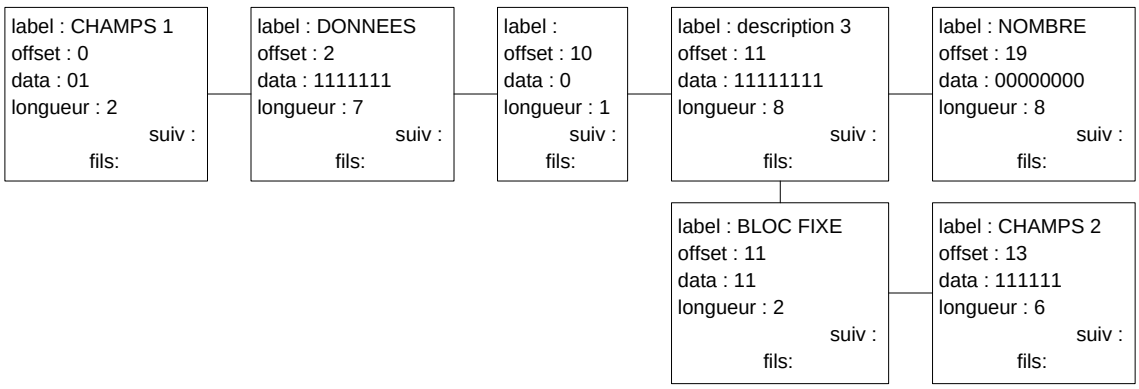


FIG. 3.11 – Structuration du résultat du décodage de la deuxième séquence de bits de la figure 3.10

3.6 Conclusion

Nous avons défini le compilateur décrit dans ce chapitre pour décoder les messages acceptés dans le cadre du protocole RLC/MAC.

En l'état actuel, ce compilateur CSN.1 pourrait être optimisé. Une des optimisations les plus évidentes, aussi bien pour le compilateur que pour les décodeurs qu'il génère, serait d'éliminer la récursivité dans toutes les méthodes qui parcourent des structures d'arbre.

De plus, à l'heure actuelle, le compilateur ne gère pas toutes les constructions possibles en CSN.1. Il ne gère par exemple pas la troncation, qui consiste en mettant le signe '//' à la fin de la description à spécifier qu'on accepte aussi des messages tronqués. Le traitement de ces constructions n'a pas été implémenté parce que nous n'en avions pas besoin pour compiler la spécification CSN.1 du format des messages RLC/MAC.

Mais ce compilateur et les décodeurs qu'il génère pourrait avoir bien d'autres utilisations que le décodage des paquets RLC/MAC

Il pourrait par exemple être utilisé pour créer des prototypes de serveur afin de tester de nouveaux protocoles. Il pourrait permettre de singulièrement réduire le temps entre la description d'un protocole et sa première implémentation.

Il pourrait aussi être utilisé dans le cadre d'un outil de trace, un "sniffer", sur un réseau local. Nous pouvons imaginer que l'utilisateur spécifie en CSN.1 le format des paquets qui l'intéresse et que le "sniffer" utilise un décodeur généré à partir de cette spécification pour identifier dans le flux qu'il surveille les paquets correspondants.

Enfin, il pourrait aussi être utilisé, dans des applications du même type que J-GSM-Show, pour aider à l'illustration du fonctionnement de divers protocoles. Pourquoi pas TCP/IP par exemple ?

Ce compilateur constitue donc un produit fini à part entière qui pourra être utilisé dans nombre d'autres projets que J-GSM-Show. Nous comptons d'ailleurs le mettre en accès libre à la disposition de la communauté d'Internet.

Chapitre 4

Illustration des protocoles de GPRS sur la voie radio

4.1 Introduction

Après avoir réalisé la partie GSM du programme, nous nous sommes penchés sur la réalisation d'une partie GPRS. Cette partie du projet a pris le nom de GPRS-Show. Ce nom ne nécessitait plus de commencer par un J, comme dans J-GSM-Show, puisqu'il ne faut plus le différencier d'un autre projet ; c'est un projet tout à fait original. Il s'agit d'une extension du programme décrit au chapitre 2 dont nous avons réutilisé une grande partie : notamment le mécanisme d'événements, la gestion de la liaison série et la gestion des fenêtres.

L'objectif est d'illustrer le fonctionnement de GPRS sur la voie radio. Pour une description des services offerts par GPRS, nous renverrons le lecteur au chapitre 1.

Le protocole au coeur de ce fonctionnement s'appelle RLC/MAC. Nous décrirons les principaux concepts y afférents au point 4.2.

Après avoir compris dans ses grandes lignes le fonctionnement de RLC/MAC, nous nous sommes attaqué à la spécification d'une interface permettant l'illustration de l'allocation des ressources sur la voie radio. Ce processus en constitue en effet la principale originalité. Nous décrirons au point 4.3 les fonctions de cette interface.

Nous décrirons ensuite la façon dont nous avons spécifié le comportement de cette interface au point 4.4. On trouvera aussi à ce point les parties les plus intéressants de la spécification en elle-même.

Mais avant de pouvoir implémenter ces spécifications, il fallait être capable d'une part de

décoder les blocs RLC/MAC[1] et d'autre part d'établir une communication avec le mobile de trace GPRS via une liaison série.

Le décodage des blocs RLC a été la motivation première du développement du compilateur décrit au chapitre 3. En l'utilisant pour générer à partir de la spécification du format binaire des blocs RLC/MAC un décodeur, nous avons rencontré divers problèmes. Ils seront évoqués au point 4.6.

Le développement des modules nécessaires à l'utilisation d'un nouveau mobile de trace sera évoqué au point 4.7.

Malheureusement, le temps a manqué pour implémenter les spécifications évoquées au point 4.4 mais nous avons cependant réalisé une interface simple utilisant tous les principes évoqués auparavant qui nous a permis de prouver la validité de la démarche. Nous nous expliquerons au point 4.8.

Enfin, nous conclurons sur les difficultés rencontrées et les perspectives d'avenir du projet.

4.2 Analyse du protocole RLC/MAC

La complexité de l'interface radio GPRS découle de plusieurs facteurs. Tout d'abord, GPRS implante un système paquet sur une interface radio ne fonctionnant jusque là qu'en circuit. Et ce système ne doit en rien perturber le fonctionnement des services GSM classiques malgré le fait qu'il fonctionne en parallèle, en utilisant les mêmes bandes de fréquence. Ensuite, vu la rareté de la ressource radio, le protocole doit utiliser de manière optimale celle dont il dispose. Et comme souvent, l'optimisation du système s'est traduite par une plus grande complexité.

Pour gérer une partie de cette complexité, les concepteurs de GPRS ont utilisé la notation CSN.1 pour spécifier le protocole au coeur de l'interface radio : RLC/MAC. Comme nous l'avons expliqué au chapitre 3, CSN.1 permet de définir les différents messages de manière très efficace en terme du nombre de bits utilisés.

GPRS partage les ressources de GSM-C sur la voie radio. Ce partage se fait sur base des trames TDMA évoquées au chapitre 1. Le services GPRS viennent occuper des slots dans ces trames.

Mais contrairement à GSM qui ne permettait l'allocation que d'un seul slot montant et d'un seul slot descendant par mobile, GPRS permet l'allocation de plusieurs slots par trame montante ou descendante. Le débit est donc multiplié par ce nombre de slots. Pour ajouter à la complexité, GPRS permet une allocation asymétrique des ressources sur les deux voies (par exemple 1 slot sur la voie montante et 4 sur la voie descendante).

[1] Petite remarque terminologique : le nom couramment employé pour désigner une trame RLC/MAC est bloc, et non message ou paquet.

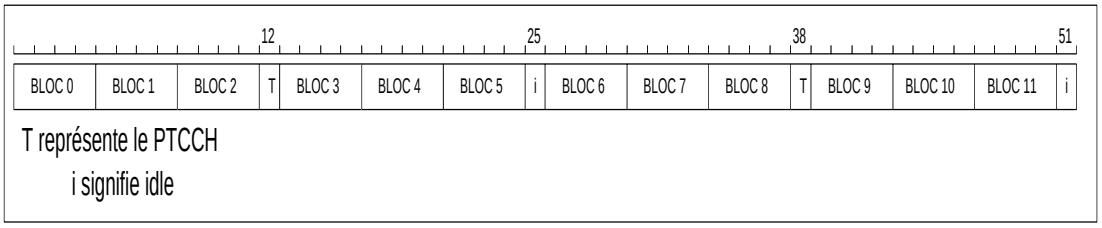


FIG. 4.1 – Multitrame à 52 (un bloc contient quatre slots)

Schéma de codage	Taille du bloc en octets
CS-1	22
CS-2	32 7/8
CS-3	38 3/8
CS-4	57 7/8

TAB. 4.1 – Les schémas de codage pour les blocs RLC/MAC

Cette allocation de slots se fait dans une structure spéciale : la multitrame à 52. Pour rappel, GSM-C utilise des multitrames à 26 et 51.

Cette allocation se fait au niveau de la couche RLC/MAC. L'allocation de slots pour GPRS se fait en général par blocs de quatre[2]. C'est donc en blocs qu'on va décomposer la multitrame à 52, en 12 blocs en fait. Les 4 slots restant seront utilisés en alternance pour le PTCCH (transmission des ajustements de l'avance en temps sur la voie descendante et de messages permettant au réseau de calculer cette avance en temps sur la voie montante) et par un slot IDLE pendant lequel le mobile peut faire des mesures de qualité et de puissance sur différentes BTS. Ces slots sont les slots 12, 25, 38 et 51 (voir figure 4.1).

L'augmentation du nombre de slots alloués par trame à un mobile est le premier moyen d'augmenter les débits. Le second moyen est la diminution de la protection des données pour diminuer l'overhead. Le volume utile de données transportées dans un bloc varie en fonction du schéma de codage utilisé. Un schéma de codage correspond à un niveau de sécurité sur les données, plus le volume utile transporté est faible, plus la redondance est forte et plus le nombre possible de corrections d'erreurs augmente. Ce paramètre permet d'augmenter le volume transporté quand les parasites sont rares.

La norme définit quatre schémas de codage : de CS-1 à CS-4, du mieux protégé à celui transportant le plus de bits "utiles". On trouvera au tableau 4.1 la liste de ces schémas et le volume de données utiles correspondant.

Enfin, comme nous l'avons déjà expliqué au chapitre 1, GPRS permet une connexion quasi instantanée et ne demande pas d'allocation de ressources pendant toute le communication de l'utilisateur. Cet objectif a été atteint sur la voie radio au moyen de TBF (*Temporary Block Flow*).

[2]Autrement dit, un PDU physique occupe quatres slots

Un TBF est un flux de données temporaire, comme son nom l'indique. Il est désigné par un identifiant, le TFI(*Temporary Flow Identifier*). Un TBF existe tant que l'émetteur a en mémoire des données à transmettre, il peut correspondre à l'émission de plusieurs paquets d'une couche supérieure. Autrement dit, un TBF est interrompu et ses ressources sont libérées pour d'autres transferts de données dès que l'émetteur n'a plus de donnée à transmettre. Il y a deux types de TBF :

Le TBF downlink Le flux de données principal va du réseau au mobile. Néanmoins, la communication n'est pas unidirectionnelle puisque le mobile renvoie des acquittements et des mesures au réseau. Le réseau envoie des messages de préallocation au mobile qui lui précisent quels blocs décoder dans les slots qui lui sont alloués. Certains de ces blocs ne seront pas forcément destinés au mobile, mais peuvent transporter des données pour un autre mobile. Le destinataire final du bloc est désigné par le champ TFI inclus dans le bloc. Et régulièrement, le mobile va trouver dans un de ces blocs une allocation pour la voie montante qui va lui préciser dans quel(s) bloc(s) il va pouvoir transmettre ses acquittements et mesures.

Le TBF Uplink Le flux de données principal va du mobile au réseau. C'est le réseau qui gère l'allocation des ressources sur la voie montante(il gère la concurrence entre les mobiles). Le mobile doit donc écouter les "ordres" du réseau sur la voie descendante pour savoir dans quels blocs et sur quels slots il peut transmettre. Ces "ordres" sont identifiés par le TFI. De plus, il doit écouter sur la voie descendante les acquittements des paquets qu'il a transmis. Il y a deux types d'allocations possibles sur la voie montante :

- L'allocation dynamique : le mobile reçoit un identifiant USF(*Uplink State Flag*) par slot qu'il gère et il écoute sur la voie descendante. Lorsqu'il repère son identifiant dans un bloc de la voie descendante, il sait qu'il peut transmettre à partir du bloc suivant sur la voie montante un certain nombre, paramétrable, de blocs(1 ou 4 en fait).
- L'allocation statique : le mobile reçoit un message désignant les blocs dans lesquels il va pouvoir émettre pendant une certaine période. Cette allocation est limitée à maximum 128 blocs mais peut être répétée pour une autre période. C'est dans les acquittements que le mobile va voir si l'allocation est reconduite. L'allocation peut aussi être reconduite partiellement, uniquement pour certains slots

On voit donc qu'un TBF implique toujours des transmissions dans les deux sens, de la MS au réseau et du réseau à la MS, que le TBF soit UPLINK ou DOWNLINK.

En toute logique, un mobile peut avoir deux TFI, un TFI UPLINK et un TFI DOWNLINK, ce qui montre bien que ces deux aspects sont indépendants. A noter aussi que les acquittements d'un TBF sont identifiés par le TFI de ce TBF alors qu'ils traversent la voie radio dans le sens opposé à celui du flux principal du TBF.

Il peut y avoir quatre états :
- Aucun TBF n'est en cours.
- Un TBF UPLINK est en cours.

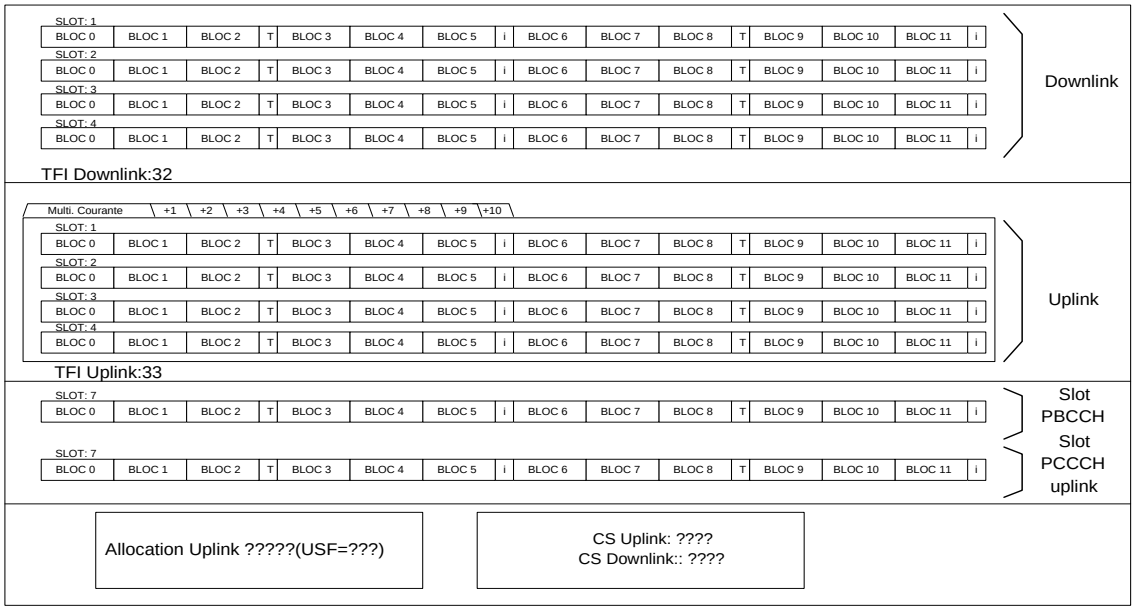


FIG. 4.2 – Interface destinée à illustrer l'allocation des ressources sur la voie radio dans GPRS

- Un TBF DOWNLINK est en cours.
- Un TBF UPLINK et un TBF DOWNLINK sont en cours.

Enfin, il faut remarquer que la notion de canaux logiques perd de son sens dans le système GPRS. En effet, on ne sait plus à l'avance où se trouvent les canaux logiques dédiés. Le mobile ou le réseau le découvre en déchiffrant le bloc RLC/MAC. Le type du paquet est indiqué dans l'en-tête de chaque bloc et c'est en fonction de ce type qu'on peut déterminer à quel canal logique appartient le bloc. La notion de canaux logiques a été maintenue pour la beauté de la norme et reste cependant valide pour certains canaux communs descendants comme le PBCCH (*Packet Broadcast Control CHannel*).

4.3 Une interface pour illustrer l'allocation des ressources radio

La figure 4.2 représente notre proposition d'interface pour illustrer l'allocation des ressources sur la voie radio dans GPRS. Cet écran peut comporter jusqu'à 8 représentations de la structure de multitrames à 52. Les 4 premiers sont ceux représentant la voie descendante, une représentation par slot ou le mobile peut être à l'écoute [3], les quatres suivantes représentent ceux de la voie montante , l'avant-dernier est la représentation du slot contenant

[3]On fait la supposition que les mobiles multislots capables de gérer plus de 4 slots n'apparaîtront pas avant longtemps, même si en théorie, la norme permet l'utilisation de 8 slots simultanément.

le PBCCH(s'il est utilisé)[4] et le dernier la représentation du slot contenant le PCCCH[5] (*Packet Common Control CHannel*) .

Dans cette interface, nous proposons d'appliquer le conventions suivantes :
- Mettre une case représentant une bloc en vert signifie que ce bloc est réservé à un usage futur par le mobile de trace.
- Mettre un bloc en rouge signifie qu'il vient de recevoir ou d'émettre un message qui a été transmis dans ce bloc. Le slot utilisé détermine dans quelle structure de multitrames on fait passer le bloc en couleur. L'emplacement du bloc dans la multitrame est déterminable au moyen du Frame Number.
- Les labels de chaque bloc vont changer en fonction des informations dont on dispose sur le canal logique qu'il véhicule. Dans le cas où on ne dispose pas d'informations sur le canal logique du bloc, l'intitulé pourrait rester "bloc n", où n représente le numéro du bloc dans la multitrame. Le fait d'afficher le canal logique du bloc permet d'illustrer quel type de données il transporte : de données ou de contrôle, dédiées ou communes ....

Les multitrames qui apparaissent normalement à l'écran sont les multitrames courantes mais les onglets au-dessus des multitrames uplink permettent de visualiser les allocations de blocs sur les multitrames futures.

Le champ TFI UPLINK (DOWNLINK) indique la valeur actuelle du TFI sur la voie montante (descendante). Si aucun TBF UPLINK (DOWNLINK) n'est en cours la valeur affichée pour le TFI UPLINK (DOWNLINK) est " ? ? ?".

Le champs *Allocation Uplink* permet d'afficher le type d'allocation Uplink, dynamique ou fixe (comme expliqué au point 4.2).

Le champs *USF* indique la valeur de l'USF si le mécanisme d'allocation courant est dynamique. Dans le cas d'une allocation fixe, ce champs n'apparaît pas.

Enfin, les champs *CS UPLINK* et *CS DOWNLINK* permettent de préciser le schéma de codage en usage actuellement pour les blocs de données. Pour rappel, le schéma de codage pour la signalisation est toujours CS-1.

4.4 Mode de spécification du comportement de l'interface

Pour spécifier le comportement de l'interface du point précédent, on s'est basé sur la spécification en CSN.1 du protocole RLC/MAC (se reporter au point 3.2 pour une description de CSN.1). Nous nous sommes tout particulièrement penchés sur les blocs de contrôle qui portent les informations d'allocation. Le bloc dénommé *PACKET UPLINK*

[4] A noter : si le PBCCH est utilisé, seules 3 représentations de multitrames DOWNLINK peuvent encore être utilisées, étant donné que le PBCCH occupe un slot DOWNLINK. Attention, ce slot, en plus du PBCCH peut aussi transporter des données.

[5] Meme remarque que pour le slot PBCCH, mais pour les représentations de multitrame UPLINK

ASSIGNMENT comprend par exemple dans sa définition les champs suivants (ceci est une version simplifiée de la vraie syntaxe) :

```
< TBF Starting Time : < Starting framenumber Description IE >

< ALLOCATION_BITMAP_LENGTH :bit(7)>

< ALLOCATION_BITMAP : bit (val(ALLOCATION_BITMAP_LENGTH)) > ;
```

Le champs *ALLOCATION_ BITMAP* décrit sous forme d'un bitmap l'allocation des slots pour une certain nombre, dépendant de la longueur du champ, de trames. Ce groupe de trames commence à la trame dont le *Frame Number* est désigné par la valeur du champs *TBF Starting Time*.

Pour définir le comportement de l'interface en réception d'un bloc *PACKET UPLINK ASSIGNMENT*, nous avons donc précisé de quelle façon l'affichage devait se modifier en fonction des valeurs de ces champs. Cette spécification requiert la compréhension de l'interface radio GPRS.

Le document complet de spécification du comportement de l'interface se trouve en annexe mais nous en avons recopié les parties les plus intéressantes au point 4.5.

L'utilisation de cette spécification devrait se révéler assez simple. En effet, si l'on génère au moyen du compilateur du chapitre 3 un décodeur à partir de la spécification CSN.1 des blocs RLC/MAC, on peut utiliser ce décodeur pour générer pour chaque bloc tracé par le mobile une structure de décodage telle que décrite au point 3.4.1.

Et à partir de cette structure, il est extrêmement facile de vérifier la présence ou l'absence d'un champs sur base de son nom et d'en obtenir le contenu. Par extrêmement facile, on entend qu'il existe déjà une méthode, portant sur cette structure, qui est capable de retrouver un champs et sa valeur sur base de son label.

Mais pour arriver à animer l'interface évoquée ci-dessus, il faut tout d'abord compiler la spécification en CSN.1 du format des blocs RLC/MAC. Ce processus est évoqué au point 4.6.

4.5 Spécification du comportement de l'interface

Cette spécification de la façon dont l'écran doit réagir à la réception d'un bloc RLC/MAC s'appuie sur leur définition en CSN.1. Comme nous l'avons expliqué plus en détail au chapitre 3, il est possible à partir d'une syntaxe en CSN.1 de créer un programme qui pourra déterminer si une chaine de bits qu'on lui soumet appartient à l'ensemble de chaînes de bits défini par la syntaxe et qui, le cas échéant, peut isoler chacune des sous-chaînes nommées dans cette syntaxe. L'idée est donc de spécifier les réactions du programme en fonction des valeurs de ces sous-chaînes. Bien sûr, dans le cas où une de ces sous-chaînes ne

serait pas présente, le programme ne doit avoir aucune réaction relative à sa valeur, sauf contre-indication explicite.

Comportement général à l'écran

paquets downlink : En fonction du type d'allocation sur la voie montante, le comportement en réception de chaque bloc est différent :

- mode d'allocation dynamique : si l'USF du paquet est celui du mobile sur ce PDCH, mettre en vert la trame suivante sur la voie montante du même pdcch.
- mode d'allocation statique : ne pas tenir compte de l'USF du paquet.

De plus, dès qu'un paquet downlink est tracé, le bloc correspondant passe en rouge et le précédent bloc mis en rouge sur la voie descendante retourne à sa couleur originelle.

Dans la suite de cette section, nous allons décrire la réaction de l'interface à quelques blocs RLC/MAC de contrôle assez intéressants, c.-à-d. provocant des changements importants dans l'interface. Nous adopterons la structure suivante : nous ferons correspondre à chaque nom de bloc la liste des champs qui comportent des informations d'allocation de ressource sur la voie radio. Nous décrirons alors pour chaque champ le mécanisme d'allocation correspondant et la réaction de l'interface en réception d'un tel bloc.

Packet PDCH Release :

Timeslots_Available : Bitmap de 8 bits indiquant les slot assignés à GPRS sur l'ARFCN courant (*Absolute Radio Frequency Channel Number*, un numéro désignant une fréquence dans la bande de fréquence allouée à GSM). Le bit 8 indique le statut du slot 0, le bit 7 indique le statut du slot 1... Si le bit vaut 0, le slot n'est pas assigné à GPRS et vice-versa. *A l'écran* : afficher une structure de multitrame par slot utilisé par GPRS, faire correspondre les numéros de slots.

Packet TBF Release :

GLOBAL TFI IE : Pour identifier le mobile auquel ce message s'adresse.

UPLINK_RELEASE : Champ de 1 bit. S'il est mis à 1, ce message va provoquer la fin du TBF uplink. *A l'écran* : effacer la valeurs de TFI, réinitialiser les blocs relatifs au TBF uplink, effacer la valeur de CS.

DOWNLINK_RELEASE : Champ de 1 bit. S'il est mis à 1, ce message va provoquer la fin du TBF downlink. BF A L'ÉCRAN : effacer la valeurs de TFI, réinitialiser les blocs relatifs au TBF downlink, effacer la valeur de CS.

Packet Uplink Assignment :
Ce paquet contient la description de l'allocation de ressources pour la transmission par le mobile. Trois cas sont possibles :

- L'allocation fixe
- L'allocation dynamique
- L'allocation d'un bloc unique (par exemple pour envoyer un paquet Packet Resource Request)

La présence de certains champs est dépendante du type d'allocation envisagé. Nous présentons donc d'abord les champs dont la présence ne dépend pas du type d'allocation et ensuite, les autres champs, par type d'allocation.

Global TFI IE : Champs de 6 bits. S'il commence par un 0, les 5 bits suivants représentent le UPLINK_TFI, s'il commence par un 1, ces bits représentent le DOWNLINK_TFI. Ce champs sert à désigner le destinataire de ce message.

Et maintenant, les champs dont la présence dépend du type d'allocation :

- Allocation Statique :

  UPLINK_TFI_ASSIGNMENT : Contient le TFI uplink du mobile pour un futur TBF uplink. *A l'écran* : afficher cette valeur en décimal dans le champ "TFI Uplink". Ce champ n'est pas redondant avec le champs GLOBAL TFI IE (quand celui-ci contient un TFI Uplink) qui sert lui à désigner le destinataire de ce message-ci. Les deux TFI peuvent être différents.

  FINAL_ALLOCATION : Champ de 1 bit. S'il est mis à 1, ce message est la dernière allocation de ressources pour le TBF en cours. Il faut donc dans le programme surveiller ce bit. Pas d'effet direct à l'écran.

  DOWNLINK_CONTROL_TIMESLOT : Champ de 3 bits contenant le numéro du slot contenant le PACCH downlink (*Packet Associated Control CHannel*). Ce champ pourrait éventuellement être traité par le programme.

  BLOCKS_OR_BLOCK_PERIODS : Champ de 1 bit. Mis à 1 si l'allocation se fait pour tous les slots alloués en même temps. Mis à 0, si l'allocation se fait indépendamment pour chaque slot alloué . Si ce champs n'est pas présent, on considère que l'allocation se fait indépendamment pour chaque slot alloué.

  ALLOCATION_BITMAP_LENGTH : Champ de 7 bits. Longueur du champs ALLOCATION_BITMAP. Si le champs ALLOCATION_BITMAP_LENGTH n'est pas présent, c'est que l'ALLOCATION_BITMAP occupe toute la fin du bloc.

  TBF_STARTING_TIME : indique le frame number du début de l'allocation. Peut-être exprimé de manière absolue sous la forme d'un champs de 16 bits(Absolute Frame Number Encoding) ou de manière relative au Frame Number de la première trame TDMA du bloc contenant ce champs sous la forme d'un champs de 13 bits.

  ALLOCATION_BITMAP : Champ de longueur variable de taille maximale de 128

bits. Chaque bit représente un bloc ou un groupe de plusieurs blocs[6](en fonction de la valeur du bit BLOCKS_OR_BLOCK_PERIODS). Deux cas sont donc possible :

– BLOCKS_OR_BLOCK_PERIODS = 1 : Dans ce cas, l'allocation se fait pour tous les slots alloués à la fois. Le bitmap décrit un tableau unidimensionnel. Pour chaque bit du bitmap, autant de blocs sont alloués aux mobiles qu'il a eu de slots assignés par le champ TIMESLOT_ALLOCATION. Il s'agit de blocs "parallèles", c'est à dire partageant les mêmes trames TDMA. Le tableau est indexé comme suit :
$bloc[z]\ \forall z, tel\ que\ 0 \leq z \leq L$
avec

L : (nombre de bits -1)dans l'ALLOCATION_BITMAP.

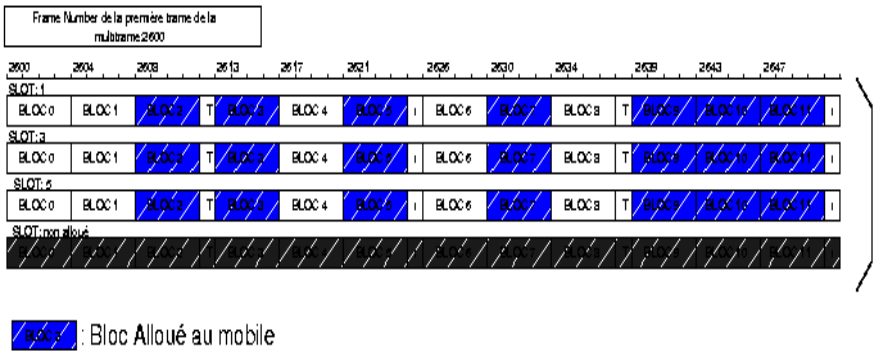
z : le numéro de bloc relatif au TBF_STARTING_TIME.

A l'écran : mettre en vert les blocs qui sont réservés au mobile(y compris dans les 10 onglets).

Exemple : Voici un champs ALLOCATION_BITMAP :

1	1	0	1	0	1	0	1	1	1	1	0	0	1	0	1

Comme le bit BLOCKS_OR_BLOCK_PERIODS est mis à 1, ce tableau représente l'allocation par bloc, pour tous les slots désignés dans le bitmap TIMESLOT_ALLOCATION. Etant donné que le champs TBF Starting Time valait 2608 voici ce que ça devrait donner à l'écran(en supposant que seuls 3 slots ont été alloués au mobile).



– BLOCKS_OR_BLOCK_PERIODS = 0 : Dans ce cas, l'allocation se fait bloc par bloc et slot par slot. Le bitmap décrit un tableau bidimensionnel de blocs. Le nombre de colonnes dans le tableau est variable et est égal au nombre de slots alloués dans le TIMESLOT_ALLOCATION. Le tableau est indexé comme suit :

[6]en parallèle sur plusieurs slots

$bloc[x, y]$
$x = (L - n) \div NTS \ \forall n, tel \ que \ 0 \leq n \leq L,$
$y = (L - n) mod \ NTS \ \forall n, tel \ que \ 0 \leq n \leq L,$

avec

x : le numéro de bloc relatif au TBF_STARTING_TIME.

y : le numéro de slot parmi les slots assignés dans le TIMESLOT_BITMAP.

L : (nombre de bits -1)dans l'ALLOCATION_BITMAP.

n l'index des bits dans l'ALLOCATION_BITMAP.

NTS le nombre de slots assignés dans le bitmap TIMESLOT_ALLOCATION($1 \leq NTS \leq 8$).

A l'écran : mettre en vert les blocs qui sont réservés au mobile(y compris dans les 10 onglets).

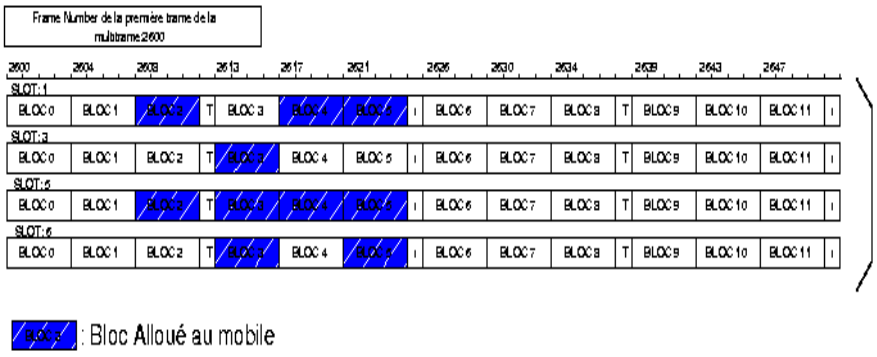
Exemple : Voici un champs ALLOCATION_BITMAP :

1	1	0	1	0	1	0	1	1	1	1	0	0	1	0	1

En supposant que le mobile s'est vu alloué 4 slots, on peut réorganiser le champ sous la forme suivante :

	4ème slot	3ème slot	2ème slot	1er slot
bloc 3	1	1	0	1
bloc 2	0	1	0	1
bloc 1	1	1	1	0
bloc 0	0	1	0	1

Ce qui donne à l'écran, en supposant que le champs TBF Starting Time valait 2608.



– Allocation dynamique : Dans le cas de l'allocation dynamique, le mobile se voit allouer un identifiant USF (*Uplink State Flag*) par slot qu'il va gérer. Il va se mettre à l'écoute sur la voie descendante sur tous les slots pour lesquels il a reçu un USF. Quand il décode son USF sur un de ces slots (intégré dans l'en-tête d'un PDU RLC/MAC), il sait qu'il peut transmettre sur la voie montante un ou quatre PDU RLC/MAC (en fonction de la valeur du dernier paramètre USF_GRANULARITY reçu). Le mobile commence sa transmission dans le bloc montant suivant celui où il a décodé son USF.

A l'écran : mettre ce ou ces blocs en vert (quand le message contenant l'USF a été tracé, ce qui ne sera sans doute pas toujours le cas, ce sera fonction du mobile de trace utilisé).

TBF Starting Time contient le Frame Number de la trame pendant laquelle le TBF alloué peut commencer.Peut-être exprimé de manière absolue sous la forme d'un champs de 16 bits(Absolute Frame Number Encoding) ou de manière relative au Frame Number de la première trame TDMA du bloc contenant ce champs.(sous la forme d'un champs de 13 bits)

USF_GRANULARITY champ de 1bit. Si ce bit est mis à 1, l'allocation se fait par 4 blocs à la fois(=16 slots). S'il est mis à 0, l'allocation se fait par 1 bloc à la fois.

USF_TNi (avec i=0...7) Champs optionnels de 3 bits. Pour chaque slot alloué au mobile, un USF lui est transmis. Pourrait être affiché à côté du numéro de slot sur la voie montante.

Remarque : l'allocation dynamique risque d'être peu utilisée à cause de la plus grande consommation électrique qu'elle entraîne en forçant le mobile à rester à l'écoute en permanence.

– Allocation d'un bloc unique :

TIMESLOT_NUMBER Champ de 3 bits. Le numéro de slot sur lequel le mobile va pouvoir transmettre son bloc.

TBF Starting Time Le Frame Number de la trame TDMA ou le mobile pourra commencer à émettre le bloc unique .Peut-être exprimé de manière absolue sous la forme d'un champs de 16 bits(Absolute Frame Number Encoding) ou de manière relative au Frame Number de la première trame TDMA du bloc contenant ce champs.(sous la forme d'un champs de 13 bits) A l'écran : faire passer le bloc désigné par cette allocation en vert.

4.6 Compilation de la syntaxe CSN.1 de RLC/MAC

Pour compiler la spécification CSN.1 du format des paquets RLC, on a utilisé le compilateur que nous avons décrit au chapitre 3.

Contre toute attente, cette spécification comportait plusieurs erreurs[7] de syntaxe dans la version utilisée[ETSa].

Par exemple, dans cet extrait de la table12.28.a1 de [ETSa] :

```
< LSA Parameters IE > ::= < NR_OF_FREQ_OR_CELLS : bit (5) >:

{<LSA ID information : < LSA ID information struct >>
```

[7] Plus d'une quinzaine

```
* (val(NR_OF_FREQ_OR_CELLS))

};
```

On peut remarquer la présence d'un signe " :" en trop au bout de la première ligne.

A l'aide du compilateur, nous avons pu aisément repérer les erreurs de syntaxe dans les spécifications CSN.1 et nous nous sommes étonnés que les concepteurs de la norme n'aient pas eu recours à de tels procédés.

Pour générer un décodeur à partir de ces spécifications, il a donc fallu les corriger. Mais si certaines corrections, comme celle de l'exemple ci-dessus sont assez triviales, d'autres, tel que l'ajout d'une accolade fermante, sont plus délicates, vu que l'emplacement de cette accolade peut changer la sémantique de la syntaxe.

Lors de la compilation, nous avons rencontré un autre problème : seuls les contenus des paquets de contrôle est spécifié en CSN.1. Les en-tête d'une taille allant de 1 à 4 octets ne sont pas spécifiées en CSN.1. Pour réaliser un décodeur, nous avons du remédier à cet oubli. La figure 4.3 représente cette spécification CSN.1.

Une fois ces différents problème résolus, nous avons obtenu un programme de décodage des paquets du protocole RLC/MAC. Malheureusement, il semble que nos corrections n'aient pas toutes été justes et ce décodeur s'est avéré incapable de décoder tous les blocs RLC/MAC complets même s'il fonctionne sur nombre de cas isolés.

Nous expliquerons au point 4.8 comment nous avons tout de même pu valider la démarche de spécification et d'utilisation de la spécification CSN.1 évoquée au point 4.3.

4.7 Adaptation de J-GSM-Show à GPRS

Une fois que l'on dispose d'un décodeur des blocs RLC/MAC, il reste à récupérer de tels blocs dans les trames séries qu'un mobile de trace GPRS peut émettre. Le mobile de trace GPRS dont nous disposons est le SAGEM OT96MGPRS.

A cette fin, nous avons réalisé une nouvelle classe instanciant *FlowParser* et une autre instanciant *MobileManager* (voir chapitre 2) à partir de la spécification du protocole utilisé par le mobile sur la ligne série. Ce travail fut relativement ardu vu que contrairement au mobile Orbitel que nous utilisions jusque là, le mobile SAGEM OT96MGPRS utilise un protocole de communication relativement compliqué avec contrôle d'erreur sur les trames séries et une délimitation des trames séries par des drapeaux et un champs longueur. Ce protocole est spécifié dans le document [SAG00].

Attardons nous quelque peu sur ce processus de décodage : Comme nous l'avons vu au point 4.2 la notion de canal logique perd quelque peu de son sens dans RLC/MAC. En effet, on ne peut en général connaître le canal logique sur lequel est transmis un paquet

```
<RLC/MAC Block>::=<Downlink RLC/MAC Block>|<Uplink RLC/MAC Block>;

<Downlink RLC/MAC Block>::= <Downlink RLC/MAC control block>|<Downlink RLC/MAC data bloc>;
<Downlink RLC/MAC control block>::=
 {
    <Payload Type : 01>
    <RRBP : bit(2)>
    <S/P : bit>
    <USF : bit(3)> -- 8bits
    |
    <Payload Type : 10>
    <RRBP : bit(2)>
    <S/P : bit>
    <USF : bit(3)> -- 8bits
    <RBSN : bit>
    <RTI : bit(5)>
    <FS : bit>
    <AC : bit>    -- 16bits
    <PR : bit(2)>
    <TFI : bit(5)>
    <D: bit>      -- 24bits
 }
 < Downlink RLC/MAC control message >
 < padding bits >
 |
 <Payload Type : 11>
 <RRBP : bit(2)>
 <S/P : bit>
 <USF : bit(3)>  -- 8 bits
 < padding bits >;
<Downlink RLC/MAC data bloc>::=
    <Payload Type : 00>
    <RRBP : bit(2)>
    <S/P : bit>
    <USF : bit(3)>       --8bits

    <PR : bit(2)>
    <TFI : bit(5)>
    <FBI : bit>          -- 16 bits

    <BSN : bit (7)>
    <E: bit>
    < padding bits >;

<Uplink RLC/MAC Block>::=<Uplink RLC/MAC control block>|<Uplink RLC/MAC data block>;
<Uplink RLC/MAC control block>::=
 <Payload Type : 01>
 < spare : bit(5)>
 < R: bit>
 < Uplink RLC/MAC control message >
 < padding bits >;
<Uplink RLC/MAC data bloc>::=
 <Payload Type : 00>
 <Countdown Value: bit(4)>
 <SI : bit>
 <R: bit>               -- 8bits
 <spare: bit(2)>
 <TFI : bit(5)>
 <TI : bit>             -- 16 bits
 <BSN : bit(7)>
 <E : bit(1)>           --24 bits
 {<RLC/MAC data bloc data> ((val(E) +1) % 2)}
 < padding bits >;
```

FIG. 4.3 – Spécification en CSN.1 du format binaire de l'en-tête des blocs RLC/MAC

qu'au moment où, en décodant ce paquet, on découvre de quel type il est et on en déduit le canal logique par lequel il a transité. Par conséquent, le canal logique n'est pas précisé dans l'en-tête du paquet. En fait, seuls le Frame Number et le sens de la communication nous intéressent vraiment dans l'en-tête ajoutée par le mobile pour un bloc GPRS.

Le Frame Number nous intéresse parce qu'il permettra de déterminer dans quels slots a été transmis le bloc (qui, on le rappelle, occupe 4 slots).

Le sens de la communication (montant ou descendant) présente un grand intérêt, en effet, il faut savoir que GPRS est un protocole asymétrique, avec le réseau d'un côté et la MS de l'autre. Le nombre de messages de signalisation possible sur la voie descendante est environ trois plus important que sur la voie montante. C'est logique, vu que c'est le réseau qui détient la majeure partie de l'information utile.

De plus, étant donné qu'il est impossible pour une MS de confondre un message descendant avec un message montant, la norme permet que deux messages identiques au bit près aient des significations différentes en fonction du sens de la communication.

Par conséquent, le sens de la communication est un élément d'information essentiel. Et nous avons pu scinder la spécification CSN.1 des blocs du protocole en deux parties distinctes : messages montants et messages descendants.

Les deux décodeurs générés à partir de ces spécifications sont utilisées par le *MobileManager* pour générer une structure de décodage, telle que définie au point 3.4.1, pour chaque bloc RLC/MAC.

Nous avons ensuite défini une nouvelle structure de donnée destinée à contenir le résultat du décodage des trames séries et des blocs RLC/MAC qu'elles contiennent : *GPRSBlock*. On trouvera à la figure 4.2 une description des champs de cette structure.

Ensuite, nous avons créé un nouveau type d'évènements, *GPRSBlockEvent* et un nouveau type d'observateur, *GPRSBlockListener*, tous deux gérés par une instance de la classe *Dispatcher*, reprenant le modèle d'observateur décrit au point 2.2.2.

La figure 4.4 illustre ces différents concepts.

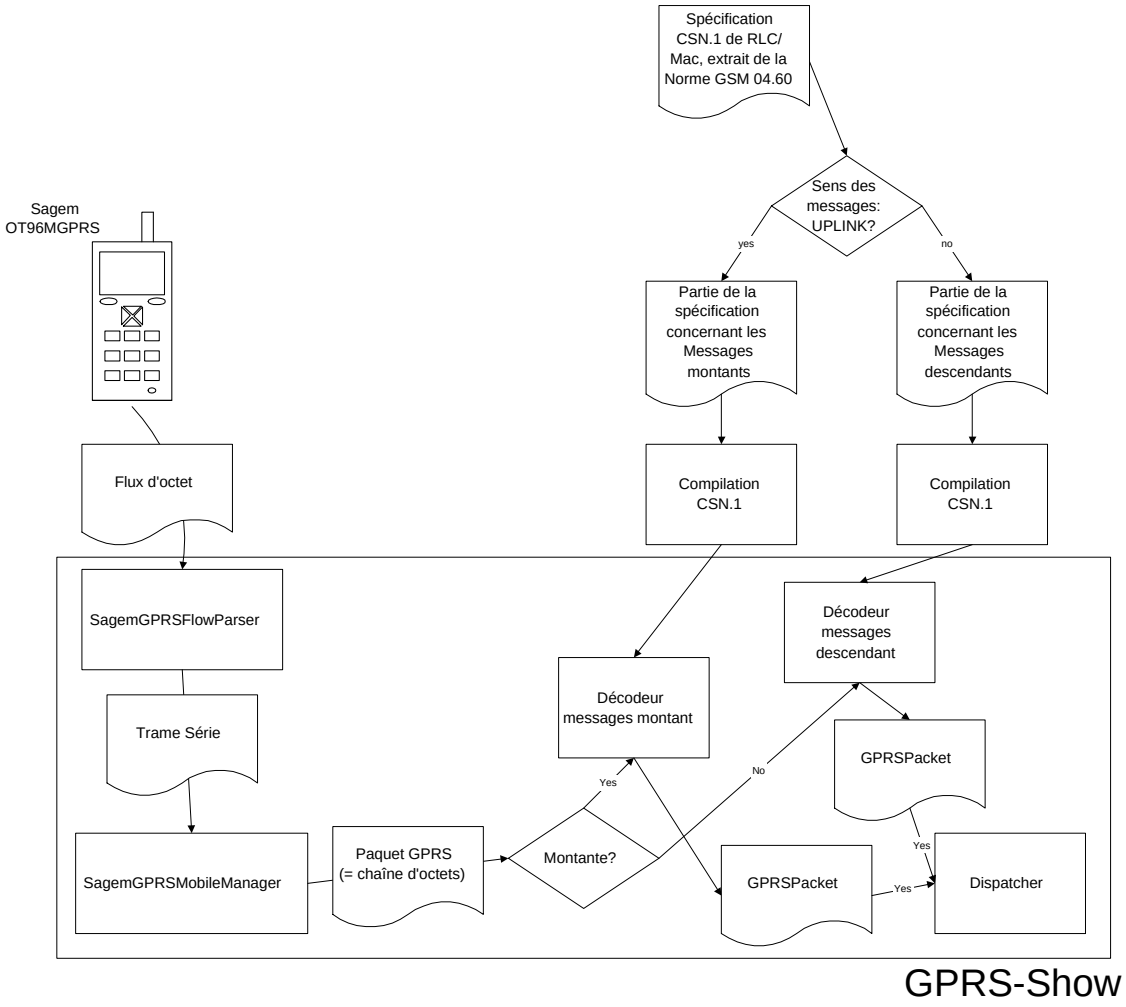


FIG. 4.4 – Diagramme des flux d'information dans GPRS-Show

Structure de GPRSBlock :		
Type	Nom du Champ	Commentaire
Int	FrameNumber	Frame Number du premier slot du bloc
int	direction	Sens de transmission de la trame : UPLINK ou DOWNLINK
int	frequence	Numéro de fréquence ARFCN sur laquelle le message est reçu(de 0 à 1023)
int	channel	Canal logique
int	longueur	Longueur en nombre d'octets du contenu
int[]	contenu	Contenu octet par octet ou champ par champ
String	message	Contenu de la trame série correspondant au bloc
String	nom_RLC	Nom du message RLC/MAC (voir norme)
csn1ConcreteNode	racineDecodage	Racine de la structure de donnée représentant le résultat du décodage du bloc

TAB. 4.2 – Les champs de la classe GPRSBlock

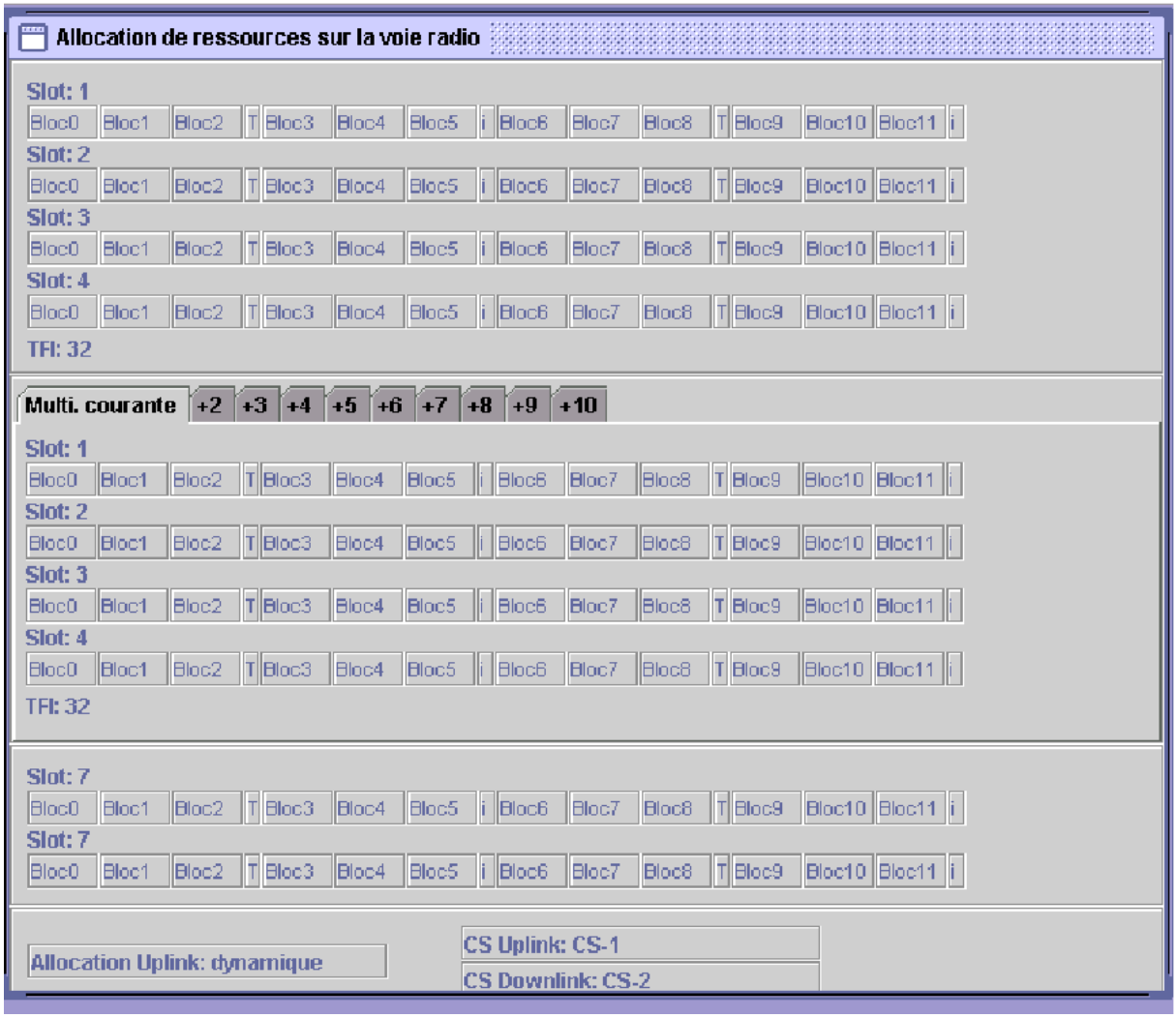


FIG. 4.5 – Réalisation concrète de l'interface décrite au point 4.3

4.8 Validation de la démarche

A cause des erreurs dans la norme [ETSa] et par manque de temps, il ne nous a pas été possible d'implémenter l'ensemble de la spécification évoquée au point 4.3. Nous avons implémenté l'interface elle même sans pouvoir la rendre active. On trouvera une illustration de la réalisation concrète de cette interface à la figure 4.5. Pour donner un ordre d'idée, la réalisation intelligente d'une telle interface demande de 1 à 2 jours.

Cependant pour valider la démarche entreprise, nous avons réalisé une interface simple similaire à celles créés pour illustrer le niveau 2 ou le niveau 3 des protocoles GSM. Cette interface affiche pour chaque bloc RLC/MAC tracé par le mobile le sens de la communication, le type de bloc, de contrôle ou de donnée, et s'il s'agit d'un bloc de contrôle le

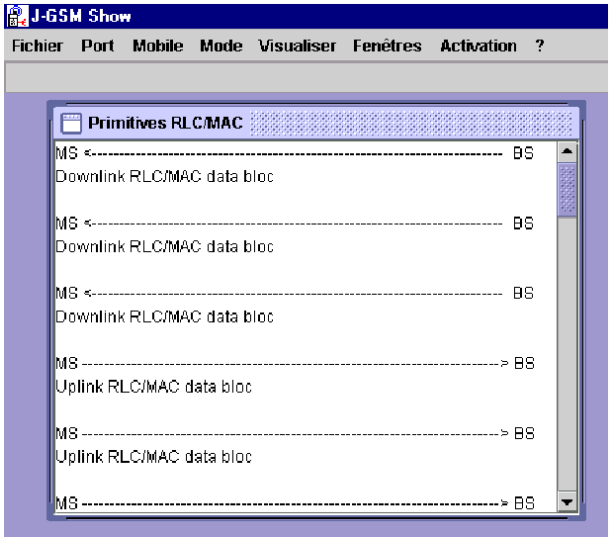


FIG. 4.6 – Interface simple d'illustration de l'échange de primitives du protocole RLC/MAC

nom exact du message de contrôle. Elle est illustrée à la figure 4.6. Pour arriver à faire fonctionner cette interface, nous avons du utiliser un spécification simplifiée des blocs qui ne décrit précisément que jusqu'à 5 octets des blocs RLC/MAC(sur 22 octets).

Le fonctionnement de cette interface prouve la validité de l'approche utilisée et le bon fonctionnement du compilateur. L'interface en elle-même présente un certain intérêt pour illuster le fonctionnement de RLC/MAC.

On trouvera les 2 (UPLINK et DOWNLINK) spécifications CSN.1 simplifiées des blocs RLC/Mac à la figure 4.3

4.9 Conclusion

Dans la réalisation de ce projet, nous nous sommes heurtés à quatre difficultés majeures :

Tout d'abord, la complexité et le manque d'information sur GPRS. En effet en novembre 2000, hormis un chapitre du livre de Xavier Lagrange [LGT00], aucun ouvrage ne s'attardait vraiment sur le fonctionnement de RLC/MAC et notamment des mécanismes d'allocation des ressources sur la voie radio au moyen de ce protocole. Il nous a donc fallu analyser le protocole directement à partir de la norme [ETSa] qui si elle est exhaustive ne peut pas prétendre à une grande lisibilité.

Ensuite, les erreurs de la norme dans la syntaxe CSN.1 qui nous ont fait perdre beaucoup de temps. Temps qui aurait pu servir à implémenter l'interface du point 4.3.

Et finalement, le manque de données GPRS à proprement parler. En effet, à l'heure où

nous écrivons, aucun réseau GPRS n'est encore ouvert au public, ni à Rennes, lieu du stage préparatoire à ce mémoire, ni en Belgique. Pour tester l'ensemble du programme, nous avons disposé en tout et pour tout de deux fichiers de traces au moyen desquels nous avons pu rejouer des scénarios enregistrés au moyen du mobile de trace GPRS dans des réseaux expérimentaux. Le premier de ces deux fichiers provenait d'une version différente du mobile de trace et les données qu'il contenait étaient partiellement inutilisables et le deuxième ne nous est parvenu que le 8 mai 2001.

Néanmoins, nous sommes parvenu à implémenter un programme capable à partir des traces fournies par un mobile GPRS d'afficher le nom et le sens de ces paquets. Nous avons pu tester ce programme à partir des 2 fichiers dont nous disposions. Si cette réalisation est relativement modeste, elle prouve la validité de la démarche qui la sous-tend et qui à terme, pourrait aboutir à des réalisations bien plus impressionnantes.

De plus, l'ajout de modules permettant la gestion d'un autre mobile que celui dont nous disposions à l'origine a permis de prouver la souplesse de l'architecture du programme.

Enfin, nous aimerions souligner l'important effort d'analyse qui a été nécessaire pour réaliser la spécification évoquée au point 4.3 à partir de la norme [ETSa] qui est particulièrement ardue à comprendre. Cette spécification a requis une compréhension en profondeur des mécanismes d'allocation de ressources sur la voie radio dans le cadre du protocole RLC/MAC.

Conclusion

Ce mémoire se base sur la compréhension des mécanismes de fonctionnement de GSM et GPRS, surtout sur la voie radio. Nombre des concepts employés se rattachent plus à une formation d'ingénieur que d'informaticien mais la convergence actuelle des télécommunications et de l'informatique devrait amener de plus en plus de gens à acquérir cette double compétence.

Nous allons tout d'abord évoquer ce qui a déjà été réalisé, ensuite, nous expliquerons ce que selon nous, il reste à faire et nous conclurons sur le futur du programme.

Ce travail nous a tout d'abord amené à réaliser le portage et l'amélioration d'un programme destiné à illustrer le fonctionnement de GSM sur la voie radio. Les choix d'architecture sont notre contribution la plus originale pour cette partie. Ils devraient permettre une adaptation facile du programme à toutes sortes d'améliorations dans le futur.

Ensuite notre travail a consisté en l'ajout au programme existant d'une partie destinée à l'illustration des protocoles GPRS sur la voie radio. Cette technologie encore récente et par conséquent peu documentée s'est révélée d'une assez grande complexité. L'analyse des différents protocoles et la réalisation des interfaces destinées à les illustrer constituent une partie de notre contribution à ce projet. Nous avons pu implémenter un prototype destiné à illustrer GPRS qui nous a permis de prouver la validité de l'architecture et de la démarche employée pour décoder les messages du protocole au coeur de GPRS sur la voie radio : RLC/MAC.

Mais le plus original de notre travail pour cette deuxième partie a consisté en la programmation du compilateur CSN.1 (*Concrete Syntax Notation 1*).La spécification du protocole RLC/MAC utilise CSN.1. Cette notation permet de définir le format de messages binaires de manière assez lisible. Elle présente surtout l'avantage d'être compilable, ce qui veut dire qu'une spécification de format binaire en CSN.1 peut être transformée automatiquement en un décodeur des messages binaires qu'elle décrit. Cette transformation, c'est le compilateur qui l'effectue. Nous avons créé de cette façon un décodeur pour les messages du protocole RLC/MAC que nous avons pu utiliser pour illustrer le fonctionnement de ce protocole. Outre cette utilisation immédiate, le compilateur pourrait servir à bien d'autres projets, comme par exemple à faciliter la réalisation de serveurs utilisant de nouveaux protocoles .

Au final, en dehors du texte en lui-même, ce mémoire se constitue d'environ 7000 lignes de code java et des documentations de ses différentes parties.

Il reste encore plusieurs travaux à réaliser : il faudrait tout d'abord implémenter les trois écrans de GSM-Show qui ne sont pas encore présents dans J-GSM-Show. Ensuite, on pourrait implémenter l'écran d'allocation des ressources dans le cadre de GPRS. Et finalement on pourrait aussi optimiser le fonctionnement du compilateur CSN.1 et des décodeurs qu'il génère.

En ce qui concerne l'avenir de ce projet, le programme résultant de ce travail devrait permettre un enseignement très vivant et concret du fonctionnement des protocoles GSM et GPRS. Il est possible au moyen d'un tel logiciel de réaliser des travaux pratiques qui vont faciliter l'acquisition par les élèves d'une connaissance opérationnelle des deux systèmes. En outre, même si ce n'est pas le but premier de ce logiciel, il pourrait permettre à un opérateur GSM de surveiller le bon fonctionnement du réseau au niveau des utilisateurs, ce qui est très important en terme d'image de marque.

Bibliographie

[AU79] Alfred V. Aho and Jeffrey D. Ullman. *Principles of compiler designs.* Addison-Wesley Publishing Company, California, 1979.

[BW97] Götz Brasche and Bernard Walke. Concepts, Services, and Protocols of the New GSM Phase 2+ General Packet Radio Service. *IEEE Communications Magazine*, août 1997.

[CG97] Jian Cai and David J. Goodman. General Packet Radio Service in GSM. *IEEE Communications Magazine*, octobre 1997.

[ETSa] ETSI. Digital cellular telecommunications system (Phase 2+) ; General Packet Radio Service (GPRS) ; Mobile Station (MS) - Base Station System (BSS) interface ; Radio Link Control/Medium Access Control (RLC/MAC) protocol (gsm 04.60 version 8.4.1 release 1999).

[ETSb] ETSI. Digital cellular telecommunications system(Phase 2+) ; General Packet Radio Service(GPRS) ; Mobile Station(MS) - Serving GPRS Support Node(SGSN) ; Subnetwork Dependent Convergence Protocol(SNDCP) (GSM 04.65 version 8.0.0 release 1999).

[ETSc] ETSI. Digital cellular telecommunications system(Phase 2+) ; General Packet Radio Service(GPRS) ; Mobile Station(MS) - Serving GPRS Support Node(SGSN) ; Logical Link Control(LLC) layer specification (GSM 04.64 version 8.4.0 release 1999).

[FS97] Martin Fowler and Kendall Scott. *UML distilled.* Addison Wesley Longman, Inc., 1997.

[GHJV97] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns.* Addison-Wesley Publishing Company, 1997.

[HFAA99] Scott E. Hudson, Frank Flannery, C. Scott Ananian, and Andrew W. Appel. *Cup User's Manual.* http ://www.cs.princeton.edu/ appel/modern/java/CUP, 1999.

[Kle01] Gerwin Klein. *JFlex User's Manual.* http ://www.cs.princeton.edu/ appel/modern/java/JFLEX, 2001.

[LGT00] Xavier Lagrange, Philippe Godlewski, and Sami Tabbane. *Réseaux GSM.* Hermes science Publication, Paris, 2000.

[Mou00] Michel Mouly. *CSN.1 Specification, Version 2.0.* Cell & Sys, Palaiseau, 2000.

[SAG00] SAGEM. Serial link trace interface specifications for trace mobile M42, 2000.

[SM01] Inc. Sun Microsystems. *Java(tm) Communications API Read Me, Version 2.0.* http ://www.sun.com, 2001.

[Tan96] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, Inc., 1996.

[Zeg00] Djamal Zeghlache. *Méthodes d'accès*, pages 235–272. Hermes, Paris, 2000.

Table des figures

Liste des tableaux